Міністэрства адукацыі Рэспублікі Беларусь

Установа адукацыі
«БЕЛАРУСКІ ДЗЯРЖАЎНЫ УНІВЕРСІТЭТ
ІНФАРМАТЫКІ І РАДЫЁЭЛЕКТРОНІКІ»

Факультэт камп'ютарных сістэм і сетак

Кафедра электронных вылічальных машын

Дысцыпліна: Праграмаванне на мовах высокага ўзроўню

ДА АБАРОНЫ ДАПУСЦІЦЬ

_____*Я. В. Богдан*

ТЛУМАЧАЛЬНАЯ ЗАПІСКА
курсавога праекту
па тэме
СІСТЭМА ЗАКАЗАЎ Ў ІНТЭРНЭТ-КРАМЕ

БДУІР КП 1–40 02 01 215 ПЗ

Студэнт: Капейкіна В.А.

Кіраўнік: Асістэнт кафедры ЭВМ
Богдан Я.В.

Мінск 2023

Установа адукацыі
«Беларускі дзяржаўны універсітэт інфарматыкі
і радыёэлектронікі»

Факультэт камп'ютарных сістэм і сетак

СЦВЯРДЖАЮ
Загадчык кафедрай ЭВМ

_____
(подпіс)
_____2023 г.

ЗАДАННЕ

па курсавому праектаванню

Студэнту *Капейкінай Вікторыі Анатольеўне*

Тэма праекта «*Сістэма заказаў у інтэрнэт-краме*»

2. Час здачы студэнтам закончанага праекту *15 снежня 2023 г.*

3. Зыходныя дадзеныя да праекту *Мова праграмавання – C++, асяроддзе распрацоўкі – Qt-Creator*

4. Змест разлікова-тлумачальнай запіскі (пералік пытанняў, якія падлягаюць распрацоўцы)

1. Ліст задання.

2. Уводзіны.

3. Агляд літаратуры.

3.1. Агляд метадаў і алгарытмаў рашэння пастаўленай задачы.

4. Функцыянальнае праектаванне.

4.1. Структура ўваходных і выходных дадзеных.

4.2. Распрацоўка дыяграмы класаў.

4.3. Апісанне класаў.

5. Распрацоўка праграмных модуляў.

5.1. Распрацоўка схем алгарытмаў (два найбольш важных метаду).

5.2. Распрацоўка алгарытмаў (апісанне алгарытмаў па кроках, для двух метадаў).

6. Вынікі працы.

7. Заключэнне

8. Літаратура

9. Прыкладанне

5. Пералік графічнага матэрыялу (з дакладным пазначэннем абавязковых чарцяжоў і графікаў)

1. Дыяграма класаў.

2. Схема метаду on_load().

3. Схема метаду sell_order().

6. Кансультант па праекту (з пазначэннем раздзелаў праекта) Я.В. Богдан

7. Дата выдачы задання *15.09.2023 г.*

8. Каляндарны графік працы над праектам на ўвесь перыяд праектавання (з пазначэннем тэрмінаў выканання і працаёмкасці асобных этапаў):

1. Выбар задання. Распрацоўка зместу тлумачальнай запіскі.

Пералік графічнага матэрыялу – 15 %;

раздзелы 2, 3 – 10 %;

раздзелы 4 да – 20 %;

раздзелы 5 да – 35 %;

раздзел 6,7,8 – 5 %;

раздзел 9 да – 5%;

афармленне тлумачальнай запіскі і графічнага матэрыялу да 15.12.22 – 10 %

Абарона курсавога праекту з 21.12 да 28.12.23г.

КІРАЎНІК Я.В. Богдан _____
(подпіс)

Заданне прыняў да выканання _____ *Капейкіна В.А.*
(дата и подпіс студэнта)

ЗМЕСТ

УВОДЗІНЫ

C++ дазваляе з лёгкасцю працаваць з памяццю, кіраваць рэсурсамі і выкарыстоўваць сучасныя падыходы да распрацоўкі ПЗ. З некалькімі дзесяцігоддзямі гісторыі і актыўнай падтрымкай з боку супольнасці распрацоўшчыкаў, C++ застаецца эфектыўным выбарам для распрацоўкі прыкладнога праграмнага забеспячэння. Невялікае апісанне сучасных магчымасцяў, даступных пры распрацоўцы прыкладанняў з выкарыстаннем C++:

- Стандартная бібліятэка. C++ пастаўляецца з багатай стандартнай бібліятэкай, якая ўключае ў сябе кантэйнеры, алгарытмы, увод/вывад, шматструменнасць і многія іншыя кампаненты. Гэта дазваляе распрацоўнікам ствараць прыкладанні больш эфектыўна, выкарыстоўваючы гатовыя рашэнні;
- Высокая прадукцыйнасць. C++ вядомы сваёй высокай прадукцыйнасцю дзякуючы блізкаму да апаратнага ўзроўню кіраванні памяццю і аптымізацыям, якія прадастаўляюць кампілятары;
- Падтрымка мноства платформаў. C++ падтрымліваецца на розных аперацыйных сістэмах і архітэктурах, што робіць яго ўніверсальным выбарам для распрацоўкі;
- Сучасны стандарт мовы. Апошнія стандарты C++ унеслі мноства паляпшэнняў і новых магчымасцяў у мову, уключаючы разумныя ўказальнікі, дыяпазоны і многае іншае;
- Бяспека. Сучасныя стандарты C++ таксама надаюць увагу бяспецы, падаючы сродкі для памяншэння рызыкі памылак у кодзе, такія як праверка межаў масіваў.

Апісаныя вышэй магчымасці дазваляюць праграмістам распрацоўваць складанае і прадукцыйнае ПЗ з выкарыстаннем C++.

Qt Creator з'яўляецца магутным інтэграваным асяроддзем распрацоўкі (IDE), спецыяльна распрацаваным для стварэння прыкладанняў з выкарыстаннем фрэймворка Qt.

Qt Creator прадастаўляе распрацоўнікам зручнае і эфектыўнае працоўнае асяроддзе, якое аб'ядноўвае ў сабе мноства інструментаў для распрацоўкі, адладкі і прафілявання прыкладанняў. Ён падтрымлівае розныя мовы праграмавання, уключаючы C++, QML і Python, што дазваляе ствараць кросплатформенныя прыкладання з прывабным карыстацкім інтэрфейсам.

1 ПАСТАНОЎКА ЗАДАЧЫ

Неабходна распрацаваць функцыянальную сістэму заказаў для інтэрнэт-крамы, якая прадаставіць карыстальнікам магчымасць заказваць тавары онлайн, забяспечваючы зручнасць і эфектыўнасць працэсу. Праграма «сістэма заказаў у інтэрнэт-краме павінна ўключаць наступны набор функцый:

- Рэгістрацыя ў асабістым кабінеце. Стварэнне асабістага ўліковага запісу з унікальным лагінам і паролем.
- Пошук тавару па фільтры і назве. Пошук тавараў з выкарыстаннем фільтраў, такіх як катэгорыі, кошт і іншыя параметры, а таксама па назве тавара.
- Даданне і выдаленне тавараў з кошыку. Зарэгістраваныя карыстальнікі могуць дадаваць тавары ў кошык і выдаляць іх па сваім меркаванні.
- Разлік агульнага кошту замовы. Сістэма павінна аўтаматычна вылічаць агульную кошт заказу на аснове дададзеных тавараў і іх колькасці.

Распрацоўка і ўкараненне дадзенай сістэмы заказаў інтэрнэт-крамы забяспечыць наступныя перавагі для карыстальнікаў і ўладальнікаў крамы:

- Зручнасць для пакупнікоў. Карыстальнікі змогуць лёгка знаходзіць патрэбныя тавары, кіраваць кошыкам, выбіраць спосаб дастаўкі, і бачыць агульную суму замовы, што зробіць працэс пакупак больш зручным і празрыстым.
- Эфектыўнасць абслугоўвання. Уладальнікі крамы змогуць аўтаматызаваць працэс апрацоўкі заказаў, разліку скідак і ўліку тавараў у кошыку, што дазволіць ім прадастаўляць больш эфектыўнае абслугоўванне кліентаў.
- Павелічэнне продажаў. Зручны працэс замовы можа стымуляваць кліентаў да здзяйснення пакупак і павелічэнню сярэдняга чэку.

2 АГЛЯД ЛІТАРАТУРЫ

2.1 Агляд метадаў і алгарытмаў рашэння пастаўленай задачы

Для ўнікальнасці аб'екту выкарыстоўваецца шаблон template < class T>. Гэты шаблон мае метад класу, які вяртае спасылку на адзіны асобнік аб'екта тыпу T, які рэалізуе стварэнне і вяртанне адзінага асобніка аб'екта. Таксама дадзены клас змяшчае аператар прысвойвання, каб прадухіліць прысвойванне асобнікаў.

Лістынг коду праграмы знаходзіцца ў дадатку А.

2.2 Аналіз існуючых аналагаў

Тэма курсавога праекту была абраная з мэтай асваення навыкаў распрацоўкі інтэрнэт-крамы з выкарыстаннем фреймворку Qt і мовы SQL для працы з базай дадзеных.

Інтэрнэт-крамы з'яўляюцца актуальнымі і запатрабаванымі ў сучасным свеце, паколькі ўсё больш людзей аддаюць перавагу рабіць пакупкі анлайн. Для стварэння карэктна працуючай інтэрнэт-крамы з выкарыстаннем Qt і SQL неабходна мець уяўленне аб існуючых аналагах.

Трэба таксама прааналізаваць існуючыя рашэнні інтэрнэт-крам, рэалізаваных з выкарыстаннем Qt і SQL, каб вызначыць, якія функцыянальнасці і асаблівасці могуць быць карысныя ў створаным дадатку.

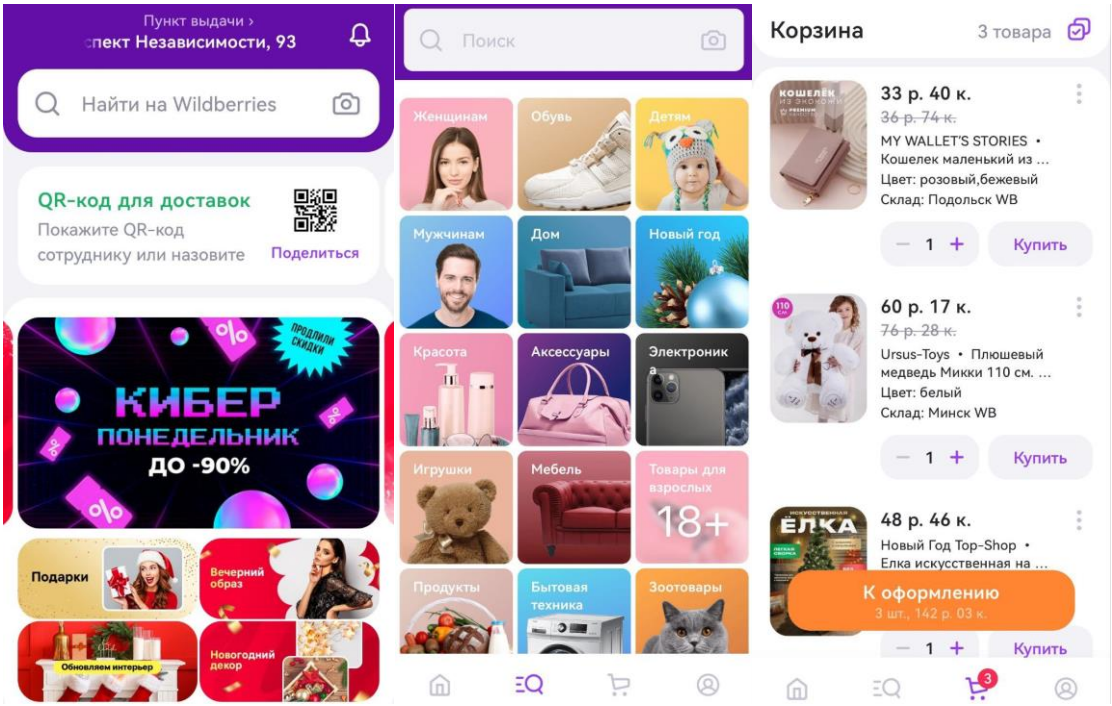
2.2.1 Дадатак Wildberries

Wildberries — гэта шматфункцыянальнае прыкладанне для мабільных прылад, якое забяспечвае зручную платформу для пакупак у Інтэрнэце. З дапамогай прыкладання Wildberries карыстальнікі могуць лёгка знаходзіць і заказваць розныя тавары з шырокага асартыменту.

Дадатак Wildberries прапануе мноства пераваг. Ён забяспечвае зручны інтэрфейс і зразумелую навігацыю, што робіць працэс пакупак простым і прыемным. Карыстальнікі могуць лёгка праглядаць тавары па катэгорыях, выкарыстоўваць фільтры для ўдакладнення вынікаў пошуку і сартаваць тавары па розных параметрах.

Дадзены дадатак таксама прапануе зручныя спосабы аплаты, уключаючы анлайн-плацяжы і аплату пры атрыманні тавару. Гэта забяспечвае гнуткасць і зручнасць пры здзяйсненні пакупак.

Сінхранізацыя файлаў і дадзеных паміж прыладамі на базе Android з'яўляецца яшчэ адным перавагай прыкладання Wildberries. Гэта дазваляе карыстальнікам захоўваць свае перавагі, гісторыю заказаў і кошык пакупак на ўсіх сваіх прыладах, забяспечваючы бяспеку і зручнасць выкарыстання.



Малюнак 2.1 — Скрыншоты Wildberries

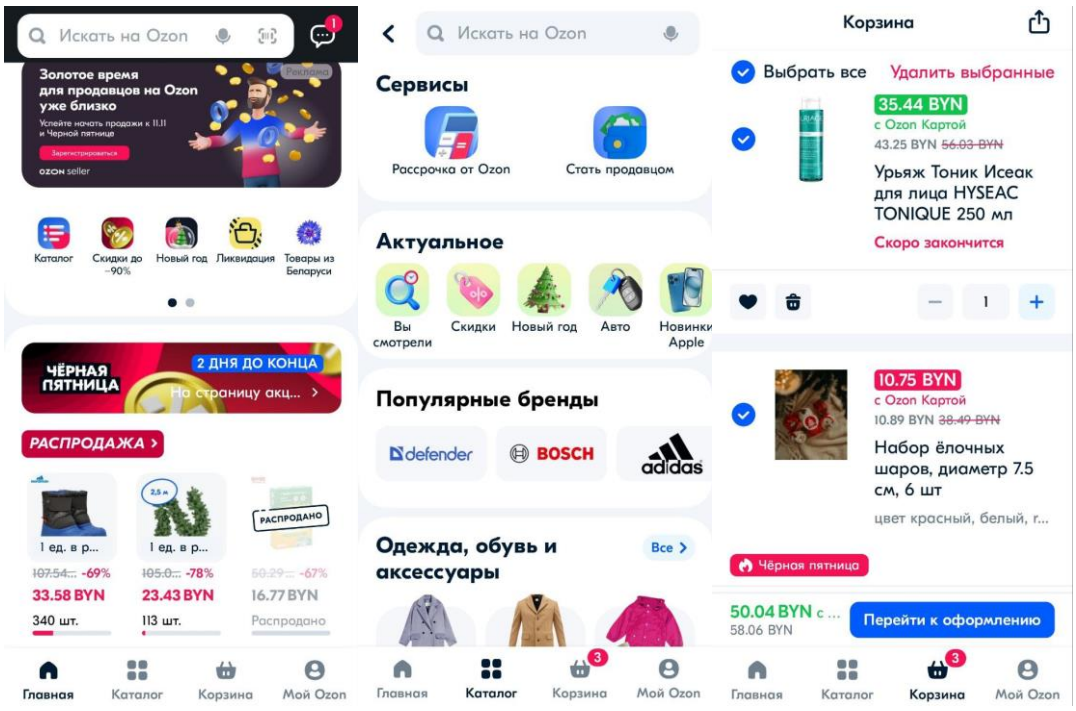
2.2.2 Дадатак Ozon

Ozon — інавацыйнае прыкладанне для мабільных прылад, якое прадастаўляе шырокі выбар тавараў і зручную платформу для анлайн-пакупак. З дапамогай прыкладання Ozon карыстальнікі могуць лёгка знаходзіць і заказваць разнастайныя тавары з розных катэгорый.

Дадатак Ozon прапануе зручны і інтуітыўна зразумелы інтэрфейс, які забяспечвае лёгкую навігацыю і прыемны вопыт пакупак. Карыстальнікі могуць выкарыстоўваць зручныя функцыі фільтрацыі, каб удакладніць вынікі пошуку і хутка знайсці патрэбныя тавары. Акрамя таго, функцыянальнасць "рэкамендацыі" дапамагае карыстальнікам адкрываць новыя тавары і прапановы, грунтуючыся на іх перавагах і гісторыі пакупак.

Адным з ключавых пераваг Ozon з'яўляецца просты і бяспечны працэс афармлення замовы і аплаты. Карыстальнікі могуць выбіраць зручныя спосабы аплаты, уключаючы онлайн-плацяжы і аплату пры атрыманні тавару. Акрамя таго, Ozon прапануе надзейную сістэму дастаўкі, што забяспечвае аператыўнае і надзейнае атрыманне замоўленых тавараў.

Дадзены дадатак таксама прапануе праграму лаяльнасці, якая дазваляе карыстальнікам атрымліваць дадатковыя бонусы, зніжкі і спецыяльныя прапановы. Гэта стварае стымул для рэгулярных пакупак і забяспечвае дадатковыя выгады для карыстальнікаў.



Малюнак 2.2 — Скрыншоты Ozon

2.2.3 Дадатак OZ.by

OZ.by — гэта інтэрнэт-крама, якая прапануе шырокі асартымент тавараў, уключаючы кнігі, гульні, касметыку, тавары для дому, творчасці, падарункі і прадукты. Яна таксама прапануе дастаўку па Беларусі. OZ.by з'яўляецца надзейным і зручным месцам для куплі тавараў анлайн.

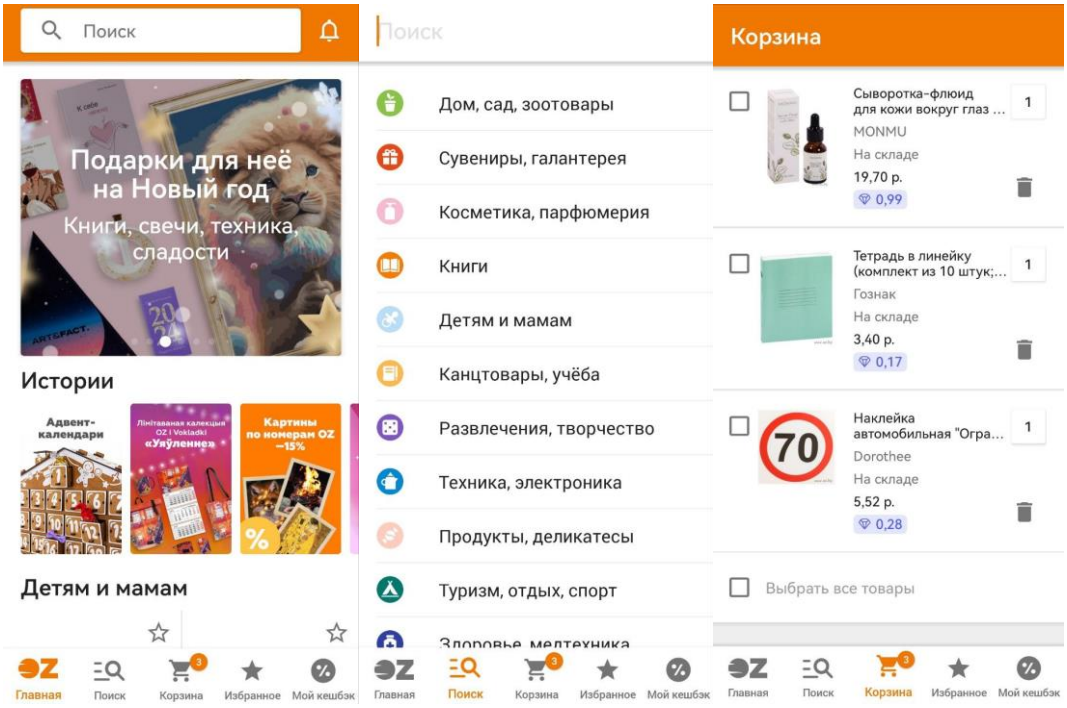
OZ.by прапануе зручны інтэрфейс і простую загрузку кніг і іншых тавараў. Вы можаце лёгка знайсці і набыць патрэбныя тавары, выбраўшы з шырокага асартыменту. OZ.by таксама прапануе сінхранізацыю файлаў для Android-прылад, што робіць выкарыстанне інтэрнэт-крамы яшчэ больш зручным.

У інтэрнэт-краме OZ.by вы знойдзеце разнастайныя кнігі, уключаючы мастацкую літаратуру, нехудожественную літаратуру, бізнес-літаратуру і дзіцячую літаратуру. Вы можаце выбраць кнігі розных жанраў, уключаючы замежную і рускую літаратуру, фантастыку, дэтэктывы і многае іншае. OZ.by прапануе навінкі кніжнага свету і топавыя кнігі, каб задаволіць любыя літаратурныя перавагі.

Акрамя кніг, OZ.by таксама прапануе шырокі выбар тавараў для дома. Вы можаце набыць посуд, кухонныя прыналежнасці, прадметы інтэр'еру, тавары для саду, гаспадарчыя тавары і многае іншае. OZ.by прапануе розныя брэнды і варыянты тавараў, каб задаволіць патрэбы ў хатнім уладкаванні.

У інтэрнэт-краме OZ.by вы таксама знойдзеце разнастайныя тавары касметыкі і парфумерыі. Тут прадстаўлены сродкі для догляду за тварам, целам, валасамі, рукамі і нагамі. Вы можаце выбраць дэкаратыўную

касметыку, парфумерыю і іншыя тавары, каб падкрэсліць сваю прыгажосць і клапаціцца пра сябе.



Малюнак 2.3 — Скрыншоты OZ.by

2.3 Патрабаванні да працы праграмы

Пасля разгляду аналагаў розных інтэрнэт-крам становіцца зразумела, што падобнага роду прыкладання звычайна ўключаюць у сябе асноўныя функцыі:

- Адлюстраванне каталогу тавараў, прадстаўленых у інтэрнэт-краме.
- Апрацоўка і адлюстраванне інфармацыі аб кожным тавары, уключаючы назву, апісанне, цану, малюнкі і іншыя характарыстыкі.
- Прадастаўленне магчымасці дадання тавараў у кошык для наступнага афармлення замовы.
- Адлюстраванне інфармацыі аб даступнасці тавараў, зніжках або акцыях.
- Прадастаўленне інфармацыі пра спосабы аплаты, дастаўцы і вяртанні тавараў.
- Розныя спосабы камунікацыі з падтрымкай кліентаў, уключаючы онлайн-чат, электронную пошту ці тэлефон.
- Адаптыўны дызайн, які забяспечвае зручнасць выкарыстання інтэрнэт-крамы на розных прыладах, уключаючы камп’ютары, планшэты і смартфоны.

Для рэалізацыі інтэрнэт-крамы быў абраны мова праграмавання C++ з выкарыстаннем фреймворка Qt і мовы SQL.

Перавагі C++ уключаюць яго высокую прадукцыйнасць і шырокія магчымасці. Дзякуючы нізкаўзроўневым магчымасцям мовы, C ++ забяспечвае больш прамы доступ да сістэмных рэсурсаў і дазваляе кіраваць памяццю і працэсарнымі рэсурсамі больш эфектыўна. Гэта асабліва важна для інтэрнэт-крамы, які можа мець вялікі аб'ём дадзеных і высокую нагрузку.

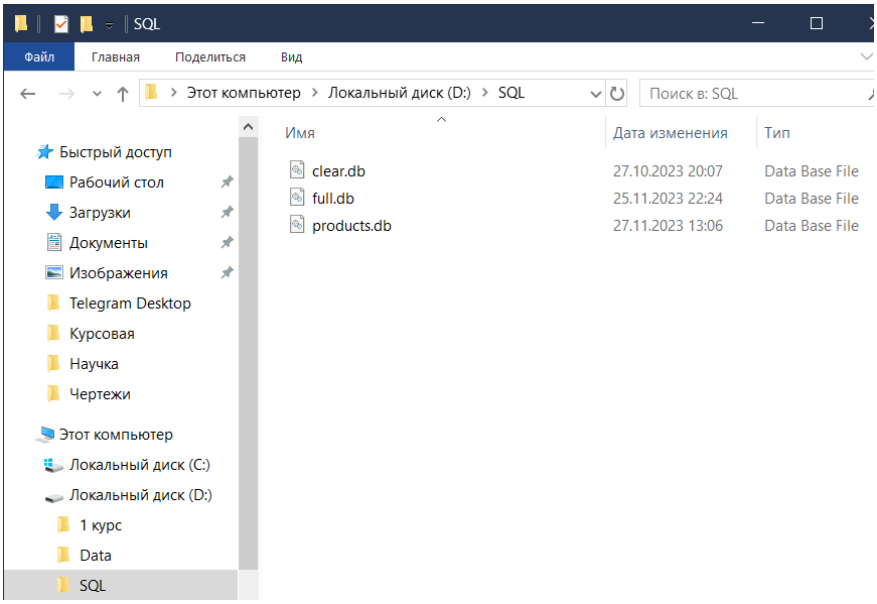
Фреймворк Qt прадастаўляе набор інструментаў для распрацоўкі графічнага інтэрфейсу карыстальніка. Ён забяспечвае крос-платформеннасць, што дазваляе ствараць прыкладанні, якія працуюць на розных аперацыйных сістэмах, такіх як Windows, macOS і Linux. Qt таксама дае зручныя сродкі для працы з сеткай, базамі дадзеных і іншымі функцыянальнымі магчымасцямі, неабходнымі для рэалізацыі інтэрнэт-крамы.

Мова SQL (Structured Query Language) выкарыстоўваецца для працы з базамі дадзеных. Ён дазваляе ствараць, змяняць і кіраваць дадзенымі ў базе дадзеных. SQL з'яўляецца стандартным мовай запытаў для мноства баз дадзеных, што робіць яго ўніверсальным і магутным інструментам для працы з інфармацыяй у інтэрнэт-краме.

3 ФУНКЦЫЯНАЛЬНАЕ ПРАЕКТАВАННЕ

3.1 Структура ўваходных і выходных дадзеных

Уваходнымі дадзенымі ў дадатку з'яўляюцца файлы фармату db, якія мы загружаем з памяці камп’ютара.



Малюнак 3.1 — Уваходныя файлы фармату db

Далей дадзеныя, якія змяшчаюцца ў файле, запісваюцца ў SQLite у такім выглядзе:

Табліца 3.1 — Структура інфармацыі аб тавары ў базе дадзеных

users	products	solt_products	products_order
kojom23 vichka25 lerka18	Перац Малако Чай	Вада Сукенка Пудра	Цацка Кніга

3.2 Распрацоўка дыяграмы класаў

Дыяграма класаў — гэта структурны тып дыяграмы, які выкарыстоўваецца ў аб'ектна-арыентаваным праграмаванні (ААП), каб паказаць класы, іх атрыбуты і ўзаемадзеяння паміж імі. Яна з'яўляецца асноўным інструментам аналізу і праектавання сістэмы ў метадалогіі Unified Modeling Language (UML).

Дыяграма класаў складаецца з прастакутнікаў, якія ўяўляюць класы. Унутры прастакутнікаў паказваюцца назва класа, Яго атрыбуты і метады. Атрыбуты — гэта зменныя, якія захоўваюць дадзеныя, звязаныя з класам, А

метады — гэта функцыі або аперацыі, якія могуць быць выкананы класам. Дыяграма класаў прадстаўлена ў дадатку Б.

3.3 Апісанне класаў

3.3.1 Каталог пакупніка

Клас `CatalogBuyer` уяўляе сабой галоўнае акно каталогу тавараў для пакупніка. Гэты клас успадкаваны ад `'QMainWindow'`, што робіць яго асноўным акном прыкладання. Дадзены клас служыць асноўным кантролерам для ўзаемадзеяння пакупніка з каталогам тавараў, забяспечваючы кіраванне дадзенымі, замовамі і інтэрфейсам.

Метад `on_login()` апрацоўвае падзею ўваходу карыстальніка ў сістэму. Дадзены метад выклікае дыялогавае акно для ўводу ўліковых дадзеных і запускае працэс аўтэнтыфікацыі.

Метад `auth_ok(User::Role)` прызначаны для апрацоўкі паспяховай аўтэнтыфікацыі карыстальніка. Дадзены метад выконвае дзеянні, звязаныя з узроўнем доступу карыстальніка (роляй), напрыклад, налада інтэрфейсу ў залежнасці ад яго правоў.

Метад `on_updated()` выкарыстоўваецца для абнаўлення дадзеных у каталогу. Дадзены метад звяртаецца да базы дадзеных, каб атрымаць актуальную інфармацыю аб таварах.

Метад `on_add()` апрацоўвае падзею дадання прадуктаў у заказ. Дадзены метад выклікае дыялогавае акно для выбару прадуктаў і дадае іх у адпаведны заказ.

Метад `on_order()` прызначаны для адлюстравання замовы пакупніка. Дадзены метад адкрывае новае акно для прагляду бягучага стану замовы.

Таксама ёсць указальнік ui на аб'ект класу `CatalogBuyer`, што дае доступ да элементаў карыстальніцкага інтэрфейсу. Гэта дазваляе праграмна кіраваць элементамі інтэрфейсу, змяняць іх уласцівасці і абнаўляць, якая адлюстроўвае інфармацыю.

Указальнік `order_buyer` звязаны з аб'ектам класу `OrderBuyer`, які прадстаўляе заказ пакупніка. Гэта дазваляе ўзаемадзейнічаць з замовай, дадаваць у яе прадукты, адсочваць яго стан і магчыма рэдагаваць інфармацыю аб замове.

3.3.2 Каталог мэнэджара

Клас `CatalogManagement` прызначаны для цэнтралізаванага кіравання каталогам, уключаючы кіраванне карыстальнікамі, даданне/выдаленне прадуктаў, кіраванне замовамі і генерацыю справаздач аб продажах. Ён выкарыстоўвае іншыя класы і аб'екты для канкрэтных функцый, такіх як `Users`, `AddProduct`, `OrderManagement` і `SalesReport`. Клас распрацаваны для

ўзаемадзеяння з карыстальніцкім інтэрфейсам (Ui::CatalogManagement) для графічнага прадстаўлення аперацый кіравання каталогам.

Метад on_user_management() выклікаецца пры націску кнопкі кіравання карыстальнікамі. Ён адлюстроўвае акно для кіравання карыстальнікамі.

Метад on_report() выклікаецца пры націску кнопкі адлюстравання справаздач. Ён адлюстроўвае акно для прагляду справаздач.

Метад on_product_remove() выклікаецца пры націску кнопкі выдалення тавараў. Ён атрымлівае выбраныя элементы з табліцы і выдаляе адпаведныя тавары з базы дадзеных.

Метад on_product_add() выклікаецца пры націску кнопкі дадання тавараў. Ён адлюстроўвае акно для дадання новых тавараў.

Метад on_order_management() выклікаецца пры націску кнопкі кіравання замовамі. Ён адлюстроўвае акно для кіравання замовамі.

Метад on_product_added(Product product) выклікаецца пры даданні новага тавару. Ён правярае ці існуе тавар з такім жа артыкулам у базе дадзеных, і калі няма, то дадае новы тавар.

Метад on_updated() выклікаецца пры абнаўленні дадзеных у базе дадзеных прадуктаў. Ён атрымлівае спіс прадуктаў з базы дадзеных і абнаўляе табліцу ў карыстальніцкім інтэрфейсе з інфармацыяй аб прадуктах.

У класе catalog Management выкарыстоўваюцца ўказальнікі на аб'екты карыстальніцкага інтэрфейсу (Ui::Catalog Management), аб'екты для кіравання карыстальнікамі (Users), дадання прадукту (AddProduct), кіравання замовамі (OrderManagement) і справаздач (SalesReport). Паказальнікі выкарыстоўваюцца для стварэння асобнікаў гэтых класаў і кіравання іх адлюстраваннем і ўзаемадзеяннем з іншымі кампанентамі прыкладання.

3.3.3 Апрацоўка заказаў

Клас OrderManagement з'яўляецца падкласам QWidget і ўяўляе сабой кампанент прыкладання для кіравання замовамі. Дадзены клас прадастаўляе зручны і зразумелы інтэрфейс для кіравання замовамі ў дадатку. Ён дазваляе загружаць, выдаляць, прадаваць і атрымліваць выбраныя заказы, забяспечваючы эфектыўнае кіраванне замовамі і павялічваючы агульны ўзровень выгоды выкарыстання прыкладання.

Метад on_load() загружае спіс заказаў з базы дадзеных і адлюстроўвае іх у табліцы на карыстальніцкім інтэрфейсе. Ён выклікаецца пры запуску прыкладання і пры абнаўленні спісу заказаў.

Метад on_remove() выдаляе выбраныя заказы з базы дадзеных і абнаўляе спіс заказаў на карыстальніцкім інтэрфейсе. Ён выклікаецца пры націску на кнопку "Выдаліць".

Метад on_sell() пазначае выбраныя заказы як прададзеныя ў базе дадзеных і абнаўляе спіс заказаў на карыстальніцкім інтэрфейсе. Ён выклікаецца пры націску на кнопку "Прадаць".

Метад `selected_orders()` вяртае спіс выбраных заказаў з табліцы на карыстацкім інтэрфейсе. Ён выкарыстоўваецца ў метадах `on_remove()` і `on_cell()` для атрымання выбраных заказаў і выканання адпаведных аперацый з імі.

3.3.4 Афармленне замовы

Клас `OrderBuyer` з'яўляецца кампанентам прыкладання, якія адказваюць за функцыянальнасць пакупкі тавараў і стварэнне заказаў. Дадзены клас `OrderBuyer` прадастаўляе функцыянальнасць для куплі тавараў і стварэння заказаў. Ён дазваляе загружаць спіс даступных прадуктаў, ствараць заказы, выдаляць прадукты з замовы і атрымліваць інфармацыю аб бягучым стане замовы.

Метад `load(set < string> product_articles)` загружае набор артыкулаў прадуктаў, якія ўяўляюць даступныя прадукты, якія могуць быць дададзеныя ў заказ. Ён дадае артыкулы прадуктаў у набор `product_articles` і абнаўляе табліцу ў карыстальніцкім інтэрфейсе для адлюстравання загружаных прадуктаў.

Метад `current_order()` вяртае бягучы набор артыкулаў прадуктаў у замове. Ён выкарыстоўваецца для атрымання інфармацыі аб выбраных прадуктах у замове.

Слот `create_order()` выклікаецца калі карыстальнік націскае кнопку "`ok`" у карыстальніцкім інтэрфейсе. Ён атрымлівае ўвод электроннай пошты карыстальніка, правярае яго на карэктнасць, атрымлівае бягучую дату, здабывае выбраныя прадукты з набору `product_articles` і дадае заказ у базу дадзеных прадуктаў.

Слот `remove_product()` выклікаецца, калі карыстальнік націскае кнопку "remove" у карыстальніцкім інтэрфейсе. Ён атрымлівае выбраныя радкі з табліцы, здабывае адпаведныя артыкулы прадуктаў і выдаляе іх з набору `product_articles`. Потым ён загружае абноўлены набор прадуктаў у табліцу.

Указальнік `Ui::OrderBuyer *ui` выкарыстоўваецца для доступу да элементаў карыстальніцкага інтэрфейсу і іх змены.

Поле `set < string> product_articles` захоўвае артыкулы прадуктаў, якія ў цяперашні час знаходзяцца ў замове. Яно выкарыстоўваецца для адсочвання выбраных прадуктаў у замове.

3.3.5 Аўтарызацыя

Клас `AuthWidget` уяўляе сабой віджэт, які выкарыстоўваецца для аўтэнтыфікацыі карыстальнікаў. Дадзены клас прадастаўляе карыстальніцкі інтэрфейс для аўтэнтыфікацыі карыстальнікаў і праверкі іх уліковых дадзеных. Ён дазваляе карыстальнікам уводзіць свае ўліковыя дадзеныя, правярае іх правільнасць і паведамляе аб паспяховай аўтэнтыфікацыі, перадаючы ролю карыстальніка.

Слот `enter()` выклікаецца пры націску на кнопку "`Увайсці`" (`ui->enter`). У гэтым метадзе адбываецца атрыманне ўведзенага лагіна і пароля, праверка правільнасці ўведзеных дадзеных і адпраўка сігналу `auth_ok` з інфармацыяй пра ролю карыстальніка, калі дадзеныя верныя.

Сігнал `void auth_ok(User::Role)` адпраўляецца пры паспяховай аўтэнтыфікацыі карыстальніка. Перадае інфармацыю пра ролю карыстальніка (`User::Role`).

3.3.6 Даданне тавару

Клас `AddProduct` ўяўляе сабой віджэт, які выкарыстоўваецца для дадання новага прадукту. Дадзены клас прадастаўляе карыстальніцкі інтэрфейс для дадання новага прадукту і адпраўкі інфармацыі пра яго. Ён дазваляе карыстальнікам ўводзіць інфармацыю аб прадукце, правярае яе на карэктнасць і адпраўляе сігнал з інфармацыяй аб прадукце для далейшай апрацоўкі.

Сігнал `product(Product product)` адпраўляе сігнал з інфармацыяй аб дададзеным прадукце.

Слот `on_add()` выклікаецца пры націску на кнопку "`Дадаць`". Ён правярае запоўненыя ці ўсе палі ўводу, стварае аб'ект `Product` з дадзенымі палёў уводу і адпраўляе сігнал product з гэтым аб'ектам.

3.3.7 Справаздача па продажам

Клас `SalesReport` уяўляе сабой віджэт, які выкарыстоўваецца для адлюстравання справаздач аб продажах. Ён з'яўляецца падкласам `QWidget` і дае карыстальніцкі інтэрфейс для ўзаемадзеяння з дадзенымі справаздачы.

Метад `on_calc()` выклікаецца пры націску на кнопку "`Разлічыць`". Ён атрымлівае значэння даты пачатку і заканчэння перыяду з адпаведных палёў уводу і выкарыстоўвае іх для выкліку метадаў `count()` і sum() класа `PRODUCT_DB`. Метад `count()` вяртае колькасць прадуктаў, прададзеных у паказаным перыядзе, а метад `sum()` вяртае суму продажаў за гэты перыяд. Затым атрыманыя значэнні колькасці і сумы адлюстроўваюцца ў адпаведных палях на карыстальніцкім інтэрфейсе.

3.3.8 Кіраванне карыстальнікамі

Клас `Users` уяўляе сабой віджэт, які выкарыстоўваецца для кіравання спісам карыстальнікаў. Ён успадкоўваецца ад класа `QWidget` і дае карыстальніцкі інтэрфейс для дадання, выдалення і абнаўлення карыстальнікаў. Клас `Users` прадастаўляе карыстальніцкі інтэрфейс для ўзаемадзеяння з дадзенымі пра карыстальнікаў, такіх як іх імёны, адрасы электроннай пошты і іншая інфармацыя. Ён можа ўтрымліваць элементы

інтэрфейсу, такія як табліцы, палі ўводу і кнопкі, для зручнага кіравання спісам карыстальнікаў.

Метад `on_remove()` атрымлівае спіс выбраных элементаў у табліцы карыстальнікаў; здабывае нумары радкоў выбраных элементаў і захоўвае іх у мноства `selected_rows`; для кожнай абранай радкі атрымлівае лагін карыстальніка і дадае яго ў мноства `removed_users`; праходзіць па кожным лагіне ў `removed_users` і выдаляе адпаведнага карыстальніка з базы дадзеных.

Метад `on_add()` атрымлівае лагін і пароль новага карыстальніка з адпаведных палёў уводу; правярае, што лагін і пароль не пустыя; правярае, што лагін не заняты іншым карыстальнікам у базе дадзеных; вызначае ролю новага карыстальніка ў залежнасці ад стану сцяжка "`Суперкарыстальнік`"; дадае новага карыстальніка ў базу дадзеных з названымі лагінам, паролем і роляй.

Метад `on_updated()` атрымлівае спіс усіх карыстальнікаў з базы дадзеных; ачышчае табліцу карыстальнікаў; для кожнага карыстальніка стварае новы радок у табліцы і запаўняе ячэйкі дадзенымі аб лагіне і ролі карыстальніка.

3.3.9 Мадэль карыстальніка ў сістэме

Клас `User` уяўляе сабой мадэль карыстальніка ў сістэме. Ён змяшчае інфармацыю аб лагіне, паролі і ролі карыстальніка. Асноўная задача класа `User` — даць функцыянал для працы з дадзенымі карыстальніка і яго роляй у сістэме.

Метад `on_remove()` апрацоўвае падзея націску на кнопку "`Выдаліць`"; атрымлівае спіс выбраных элементаў у табліцы карыстальнікаў; здабывае інфармацыю аб выбраных карыстачах і дадае іх у мноства `removed_users`; правярае, ці з'яўляецца абраны карыстальнік бягучых карыстальнікам. Калі так, то прапускае яго выдаленне. Таксама выдаляе выбраных карыстальнікаў з базы дадзеных.

Метад `on_add()` апрацоўвае падзея націску на кнопку "`Дадаць`"; атрымлівае ўведзеныя значэння лагіна і пароля з тэкставых палёў; правярае, што лагін і пароль не пустыя; правярае, што лагін не заняты іншым карыстальнікам у базе дадзеных; вызначае ролю карыстальніка ў залежнасці ад абранага сцяжка "`isSuper`"; дадае новага карыстальніка ў базу дадзеных.

Метад `on_updated()` апрацоўвае падзея абнаўлення базы дадзеных карыстальнікаў; атрымлівае спіс усіх карыстальнікаў з базы дадзеных; ачышчае табліцу карыстальнікаў; дадае кожнага карыстальніка ў табліцу.

3.3.10 Мадэль тавару

Клас `Product` уяўляе сабой мадэль тавару і змяшчае інфармацыю аб яго характарыстыках.

3.3.11 Дата продажу прадукту

Клас `SoltProduct` з'яўляецца вытворным класам ад класа `Product`. Ён дадае новае поле `realization_date` (дата рэалізацыі або продажу прадукту) і атрымлівае ў спадчыну ўсе астатнія поля і метады класа `Product`.

3.3.12 Мадэль замовы

Клас `Order` уяўляе сабой мадэль замовы, якая змяшчае інфармацыю пра дату замовы, электроннай пошце заказчыка і спісе ідэнтыфікатараў прадуктаў у замове.

3.3.13 Фасад базы дадзеных

Клас `DBFacade` уяўляе сабой фасад базы дадзеных і выконвае ролю прамежкавага пласта паміж прыкладаннем і базай дадзеных. Ён дае зручны інтэрфейс для працы з базай дадзеных і хавае дэталі рэалізацыі.

Метад exec выконвае `SQL`-запыт да базы дадзеных. Прымае радок з `SQL`-запытам у якасці параметру. Калі выкананне запыту не ўдалося, выкідваецца выключэнне `ExecException`.

Метад `qs(QString)` прымае радок у якасці параметру і вяртае гэты радок, складзеную ў адзінарныя двукоссі. Гэты метад выкарыстоўваецца для правільнага фарматавання радковых значэнняў у `SQL`-запытах.

Метад `qs(string)` прымае string у якасці параметру, пераўтворыць яго ў `QString` і выклікае метад `qs(QString)` для фарматавання радка.

Сігнал `updated` выпускае, калі база дадзеных абнаўляецца.

Клас `DBFacade` таксама змяшчае два карыстальніцкіх выключэння.

Клас `OpenDBException` — гэта выключэнне выкідваецца, калі не ўдалося адкрыць злучэнне з базай дадзеных. Яно прымае імя базы дадзеных у якасці параметру.

Клас `ExecException` — гэта выключэнне выкідваецца, калі выкананне `SQL`-запыту не ўдалося. Яно прымае сам запыт у якасці параметру.

3.3.14 Інтэрфейс для працы з базай дадзеных

Дадзены клас `ProductDB` з'яўляецца падкласам класа `DBFacade` і ўяўляе сабой інтэрфейс для працы з базай дадзеных прадуктаў. Асноўная задача класа гэта прадастаўленне метадаў для дадання, выдалення і атрымання інфармацыі пра карыстальнікаў, прадуктах і замовах з базы дадзеных.

Метад `is_login_busy(QString login)` правярае, заняты ці ўказаны лагін карыстальнікам.

Метад `add_user(User user)` дадае новага карыстальніка ў базу дадзеных.

Метад `users()` здабывае і вяртае спіс карыстальнікаў з базы дадзеных.

Метад `remove_user(string login)` выдаляе карыстальніка з базы дадзеных на аснове названага лагіна.

Метад `get_user(qstring login)` атрымлівае інфармацыю пра карыстальніка з базы дадзеных на аснове лагіна.

Метад `products()` здабывае і вяртае спіс прадуктаў з базы дадзеных.

Метад `add_product(Product product)` дадае новы прадукт у базу дадзеных.

Метад `product_by_article(string article, Product &product)` здабывае інфармацыю аб прадукце з базы дадзеных на аснове артыкула.

Метад `product_by_id(int id, Product &product)` здабывае інфармацыю аб прадукце з базы дадзеных на аснове ID.

Метад `remove_product(string article)` выдаляе прадукт з табліцы на аснове зададзенага артыкула.

Метад `add_order(string date, string email, vector<Product> products)` дадае заказы ў кошык.

Метад `orders()` здабывае і вяртае спіс заказаў з базы дадзеных.

Метад `remove_order(string date, string email)` выдаляе заказы, якія маюць пэўную дату і электронную пошту.

Метад `sell_order(string date, string email)` апрацоўвае заказы на продаж тавараў.

Метад `count(QString from, QString to)` падлічвае колькасць тавараў, якія былі прададзеныя ў зададзеным прамежку часу.

Метад `sum(QString from, QString to)` вылічае суму кошту тавараў, прададзеных у зададзеным прамежку часу.

Метад `createTables()` стварае чатыры табліцы ў базе дадзеных.

3.3.15 Шаблон для адзінага асобніка

Клас `Singleton` прызначаны для стварэння і выкарыстання адзінага асобніка аб'екта. Асноўная задача гэтага класа - гарантаваць наяўнасць толькі аднаго асобніка класа і прадастаўляць глабальную кропку доступу да гэтага асобніку.

Метад `instance()` з'яўляецца статычным метадам і рэалізуе стварэнне і вяртанне адзінага асобніка аб'екта. Пры першым выкліку метаду `instance()` асобнік будзе створаны, а пры наступных выкліках будзе вяртацца спасылка на гэты ўжо створаны асобнік. Адзіны асобнік класа ствараецца толькі ўнутры самога класа і не можа быць зменены або створаны іншымі класамі.

Выкарыстанне дадзенага класа дазваляе ствараць адзіныя аб'екты, якія могуць быць даступныя і выкарыстаныя з любога месца праграмы праз выклік статычнага метаду `instance()`.

4 РАСПРАЦОЎКА ПРАГРАМНЫХ МОДУЛЯЎ

4.1 Распрацоўка схем алгарытмаў

Схема метаду on_load () прыведзена ў дадатку В. – метад для загрузкі заказаў з базы дадзеных і адлюстравання інфармацыі аб замовах ў выглядзе табліцы ў карыстацкім інтэрфейсе.

Схема метаду sell_order () прыведзена ў дадатку Г. – метад для апрацоўкі заказаў на продаж тавараў.

4.2 Распрацоўка алгарытмаў

4.2.1 Алгарытм загрузкі інфармацыі аб замовах з баз дадзеных і адлюстраванне іх у табліцы

Для алгарытму па кроках разгледжаны метад on_load класу OrderManagement.

Крок 1. Атрыманне спісу заказаў з базы дадзеных;

Крок 2. Ачыстка табліцы перад загрузкай новых дадзеных;

Крок 3. Атрыманне бягучага індэксу радка табліцы;

Крок 4. Устаўка новага радка ў табліцу;

Крок 5. Запаўненне ячэйкі табліцы з датай замовы;

Крок 6. Запаўненне ячэйкі табліцы з электроннай поштай заказчыка;

Крок 7. Стварэнне радка са спісам ідэнтыфікатараў прадуктаў замовы;

Крок 8. Змесціва sstr счытваецца ў зменную ids_str з дапамогай getline(). Гэта дазваляе атрымаць радок, якая змяшчае ідэнтыфікатары прадуктаў, падзеленыя прабеламі;

Крок 9. Запаўненне ячэйкі табліцы са спісам ідэнтыфікатараў прадуктаў;

Крок 10. Завяршэнне выканання метаду.

4.2.2 Алгарытм функцыяналу па апрацоўцы аперацый продажу прадуктаў і ўзаемадзеянню з базай дадзеных прадуктаў

Для алгарытму па кроках разгледжаны метад sell_order класу ProductDB.

Крок 1. Атрыманне бягучай даты ў фармаце ISODate і пераўтварэнне яе ў радок з фарматам "гггг.мм. дд";

Крок 2. Пабудова запыту для выбаркі ідэнтыфікатараў прадуктаў з табліцы products_order, дзе дата роўная перададзенай даце і электроннай пошце роўнай перададзенай;

Крок 3. Выкананне запыту да базы дадзеных;

Крок 4. Стварэнне вектара для захоўвання ідэнтыфікатараў прададзеных прадуктаў;

Крок 5. Перабор вынікаў запыту і даданне ідэнтыфікатараў ў вектар;

Крок 6. Для кожнага ідэнтыфікатара прададзенага прадукту выкананне наступных дзеянняў;

Крок 7. Атрыманне інфармацыі аб прадукце па яго ідэнтыфікатару;

Крок 8. Пабудова запыту для дадання інфармацыі аб прададзеным прадукце ў табліцу sold_products;

Крок 9. Выкананне запыту да базы дадзеных;

Крок 10. Пабудова запыту для выдалення прадукту з табліцы products па яго ідэнтыфікатару;

Крок 11. Выкананне запыту да базы дадзеных;

Крок 12. Выдаленне замовы з табліцы products_order па перададзенай даце і электроннай пошце;

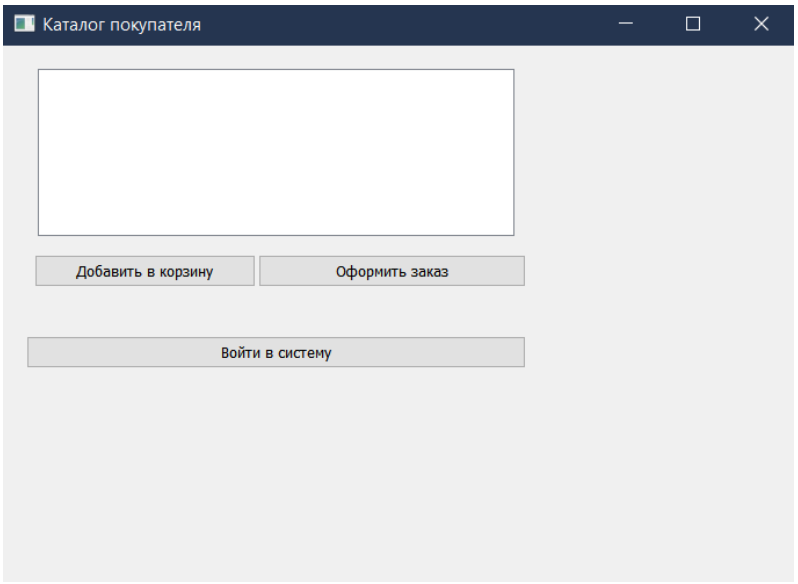
Крок 13. Генерацыя сігналу аб абнаўленні дадзеных;

Крок 14. Завяршэнне выканання метаду.

5 ВЫНІК РАБОТЫ

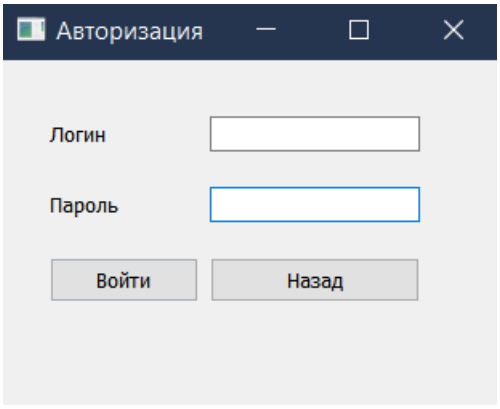
5.1 Выкарыстанне прыкладання

Для запуску праграмы неабходна адкрыць файлы зыходнага кода ў Qt Creator і сабраць праект. Інфармацыя па стварэнні праекту знаходзіцца ў файле product.db. Пасля гэтага адкрыецца акно праграмы з каталогам пакупніка, дзе будзе адлюстроўвацца спіс дададзеных тавараў (малюнак 5.1).



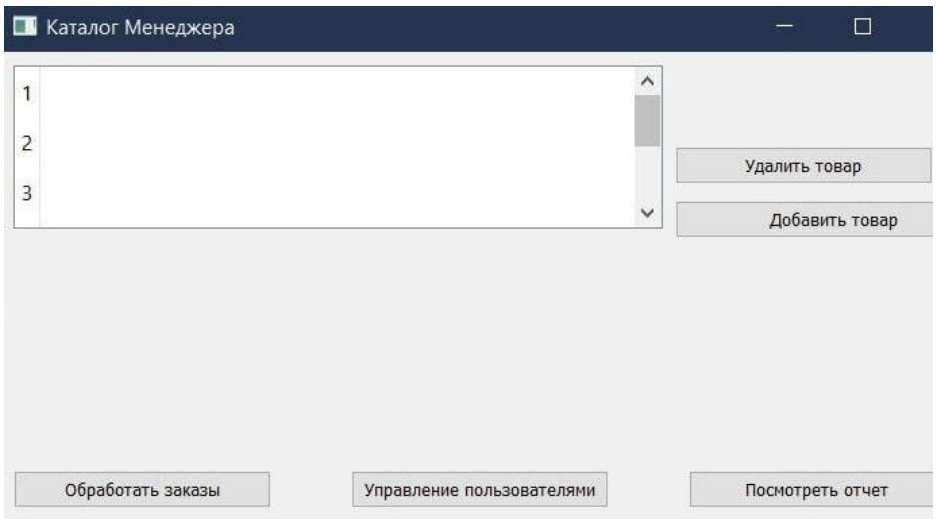
Малюнак 5.1 — Каталог пакупніка

Пры націску на кнопку ўваходу адкрыецца акно, дзе будзе запытаны лагін і пароль карыстальніка (сістэма правярае ці з'яўляецца карыстальнік адміністратарам або простым пакупніком) (малюнак 5.2).



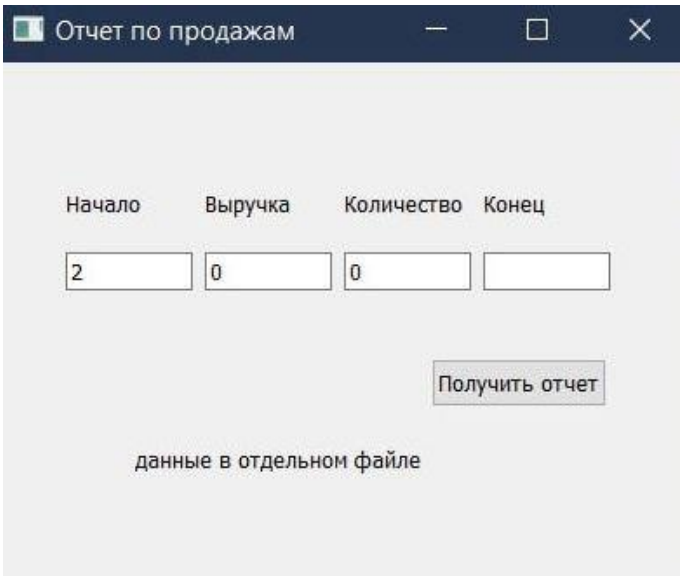
Малюнак 5.2 – Аўтарызацыя

Пасля ўваходу адміністратара адкрываецца каталог мэнэджара, у якім можна дадаваць тавары, апрацоўваць заказы, выдаляць/дадаваць тавары, кіраваць карыстальнікамі і праглядаць справаздачу (малюнак 5.3).



Малюнак 5.3 – Каталог мэнэджара

Пры націску на кнопку прагляду рахунку ўзнікае акно справаздачы па продажах (малюнак 5.4).



Малюнак 5.4 – Справаздача па продажах

ЗАКЛЮЧЭННЕ

У рамках дадзенага курсавога праекту была вывучана і паспяхова рэалізавана «Сістэма заказаў у інтэрнэт-краме» з выкарыстаннем Qt Creator. Мэта складалася ў стварэнні надзейнай і эфектыўнай сістэмы, здольнай кіраваць працэсам заказаў, забяспечваць зручнасць пакупнікоў і палягчаць працу адміністратараў.

У працэсе працы над праектам было выкарыстана мноства магчымасцяў Qt Creator, такіх як стварэнне карыстальніцкага інтэрфейсу з дапамогай візуальнага рэдактара, арганізацыю структуры кода, працу з базай дадзеных, апрацоўку падзей і многае іншае. Дзякуючы гэтым інструментам і функцыянальнасці Qt Creator, была распрацавана сістэма заказаў, якая адпавядае сучасным патрабаванням інтэрнэт-крам.

Дадзены праграмны прадукт не толькі забяспечвае функцыянальнасць для пакупнікоў, але таксама дапамагае адміністратарам эфектыўна кіраваць працэсам заказаў і забяспечваць высокі ўзровень абслугоўвання кліентаў.

Важна адзначыць, што прыкладанне сістэмы заказаў ў інтэрнэт-краме было распрацавана з выкарыстаннем мовы праграмавання C++. Выбар C++ для распрацоўкі дадзенага прыкладання абумоўлены яго перавагамі, такімі як высокая прадукцыйнасць, эфектыўнае выкарыстанне рэсурсаў кампутара, багаты набор бібліятэк і шырокія магчымасці для аб'ектна-арыентаванага праграмавання.

Qt Creator, як інтэграванае асяроддзе распрацоўкі, прадастаўляе магутныя інструменты і бібліятэкі, якія сумяшчальныя з C++ і дапамаглі паскорыць працэс распрацоўкі прыкладання. Таксама была выкарыстаная функцыянальнасць Qt для стварэння карыстацкага інтэрфейсу, апрацоўкі падзей, працы з базай дадзеных і іншых задач, звязаных з сістэмай заказаў ў інтэрнэт-краме.

Створаная сістэма заказаў ўяўляе сабой надзейнае і гнуткае рашэнне для інтэрнэт-крам, здольнае задаволіць патрэбы як пакупнікоў, так і адміністратараў.

СПІС ЛІТАРАТУРЫ

1. Метад. Ўказанні па К65 курсавому праектаванню для студ. І-40 02 01 "вылічальныя машыны, сістэмы і сеткі" для ўсіх формаў навучыў. / уклад. А.В. Бушкевіч, А. М. Кавальчук, І. В. Лук'янава. - Мінск: БДУІР, 2009.

2. Вучоба. Дапаможнік / Ю. А.Луцык , В. М. Комлічэнка. - Мінск: БДУІР, 2008.

3. Qt Documentation — [Электронны рэсурс] – Рэжым доступу: https://doc.qt.io/all-topics.html — Дата звароту: 10.11.2023.

4. Ўвядзенне ў SQL — [Электронны рэсурс] – Рэжым доступу: https://www.academia.edu — Дата звароту: 10.11.2023.

5. C ++ GUI Programming with Qt 4, — Jasmin Blanchette, Mark Summerfield, 2015 — Дата звароту: 10.11.2023.

6. The C Programming Language. 2nd Edition, — Dennis Ritchie, Brian Kernighan, 1978.

7. Праца з базамі дадзеных у Qt — [Электронны рэсурс] – Рэжым доступу: https://habr.com/ru/articles/51650 / — Дата звароту: 10.11.2023.

8. Урокі Qt5 – [Электронны рэсурс] – Рэжым доступу: https://ravesli.com/uroki-po-qt5 / — Дата звароту: 10.11.2023.

9. Qt для пачаткоўцаў. Урок 1. Найпростае GUI-прыкладанне і асноўныя фішкі — [Электронны рэсурс] – Рэжым доступу: http://knzsoft.ru/qt-bgr-ls1 / - Дата звароту: 10.11.2023.

ДАДАТАК А
Лістынг коду

```
//main.cpp

#include "catalogbuyer.h"
#include "catalogmanagement.h"
#include "users.h"
#include <QApplication>

#include "productdb.h"

Q_DECLARE_METATYPE(Product)

int main(int argc, char *argv[])
{
  QApplication a(argc, argv);
  qRegisterMetaType<Product>();
  PRODUCT_DB.createTables();
  CURRENT_USER = "";
  CatalogBuyer w;
  w.show();
  return a.exec();
}


//addproduct.h

#ifndef ADDPRODUCT_H
#define ADDPRODUCT_H

#include <QWidget>
#include "product.h"

namespace Ui
{
class Addproduct;
}

class Addproduct : public QWidget
{
  Q_OBJECT

public:
  explicit Addproduct(QWidget *parent = nullptr);
  ~Addproduct();

signals:
  void product(Product product);
public slots:
  void on_add();
```

```
private:
  Ui::Addproduct *ui;
};

#endif // ADDPRODUCT_H


//addproduct.cpp

#include "addproduct.h"
#include "ui_addproduct.h"
#include <QIntValidator>
#include <QRegExpValidator>

Addproduct::Addproduct(QWidget *parent) :
  QWidget(parent),
  ui(new Ui::Addproduct)
{
  ui->setupUi(this);
  ui->cost->setValidator(new QIntValidator(0, 9999999, this));
  ui->kind->setValidator(new QIntValidator(0, 10, this));
  QRegExpValidator*          dateValidator          =          new
QRegExpValidator(QRegExp("20\\d{2}\\.[01]?\\d\\.[0123]?\\d"),
this);
  ui->expiry_date->setValidator(dateValidator);
  ui->receipt_date->setValidator(dateValidator);
  connect(ui->add, SIGNAL(clicked()), SLOT(on_add()));
  connect(ui->back, SIGNAL(clicked()), SLOT(hide()));
}

Addproduct::~Addproduct()
{
  delete ui;
}

void Addproduct::on_add()
{
  auto all_fields = {ui->kind, ui->cost, ui->name, ui->article,
ui->name, ui->expiry_date, ui->receipt_date};
  for (auto field : all_fields)
  {
    if (field->text().isEmpty())
      return;
  }
  Product data;
  data.cost = ui->cost->text().toInt();
  data.kind = ui->kind->text().toInt();
  data.name = ui->name->text().toStdString();
  data.article = ui->article->text().toStdString();
  data.expiry_date = ui->expiry_date->text().toStdString();
  data.receipt_date = ui->receipt_date->text().toStdString();
  emit product(data);
}
```

```
//authwidget.h

#ifndef AUTHWIDGET_H
#define AUTHWIDGET_H

#include <QWidget>
#include "user.h"

namespace Ui
{
    class AuthWidget;
}

class AuthWidget : public QWidget
{
    Q_OBJECT
public:
    explicit AuthWidget(QWidget *parent = nullptr);
    ~AuthWidget();
private slots:
    void enter();
signals:
    void auth_ok(User::Role);
private:
    Ui::AuthWidget *ui;
};

#endif // AUTHWIDGET_H


//authwidget.cpp

#include "authwidget.h"
#include "ui_authwidget.h"
#include "productdb.h"
#include "user.h"

using namespace std;

AuthWidget::AuthWidget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::AuthWidget)
{
    ui->setupUi(this);
    connect(ui->enter, SIGNAL(clicked()), SLOT(enter()));
    connect(ui->back, SIGNAL(clicked()), SLOT(close()));
}

AuthWidget::~AuthWidget()
{
    delete ui;
```

```
}

void AuthWidget::enter()
{
  string login = ui->login->text().toStdString();
  string password = ui->password->text().toStdString();
  ui->login->text().clear();
  ui->password->text().clear();
  if  (PRODUCT_DB.is_login_busy(QString::fromStdString(login))  ==
false)
    return;
  User user = PRODUCT_DB.get_user(QString::fromStdString(login));
  if (user.password == password)
  {
    CURRENT_USER = user.login;
    emit auth_ok(user.role);
    this->close();
  }
}

//catalogbuyer.h

#ifndef CATALOG_H
#define CATALOGBUYER_H

#include <QMainWindow>
#include "user.h"
#include <set>
#include "orderbuyer.h"

namespace Ui
{
class CatalogBuyer;
}

class CatalogBuyer : public QMainWindow
{
  Q_OBJECT

public:
  explicit CatalogBuyer(QWidget *parent = 0);
  ~CatalogBuyer();

public slots:
  void on_login();
  void on_auth_ok(User::Role);
  void on_updated();
  void on_add();
  void on_order();

private:
  Ui::CatalogBuyer *ui;
```

```
  OrderBuyer* order_buyer;
};

#endif // CATALOGBUYER_H


//catalogbuyer.cpp

#include "catalogbuyer.h"
#include "ui_catalogbuyer.h"
#include "authwidget.h"
#include "catalogmanagement.h"
#include "productdb.h"

CatalogBuyer::CatalogBuyer(QWidget *parent) :
  QMainWindow(parent),
  ui(new Ui::CatalogBuyer)
{
  ui->setupUi(this);
  connect(ui->login, SIGNAL(clicked()), this, SLOT(on_login()));
  connect(ui->buy, SIGNAL(clicked()), this, SLOT(on_order()));
  connect(ui->add, SIGNAL(clicked()), this, SLOT(on_add()));
  on_updated();
  connect(&PRODUCT_DB, SIGNAL(updated()), SLOT(on_updated()));
  order_buyer = new OrderBuyer(nullptr);
  order_buyer->hide();
}

CatalogBuyer::~CatalogBuyer()
{
  delete ui;
}

void CatalogBuyer::on_login()
{
  AuthWidget* auth = new AuthWidget(nullptr);
  connect(auth,                        SIGNAL(auth_ok(User::Role)),
SLOT(on_auth_ok(User::Role)));
  auth->show();
}

void CatalogBuyer::on_auth_ok(User::Role role)
{
  this->hide();
  CatalogManagement   *catalog   =   new   CatalogManagement(role,
nullptr);
  catalog->show();
}

void CatalogBuyer::on_updated()
{
  vector<Product> products = PRODUCT_DB.products();
  ui->table->setRowCount(0);
```

```
  for (auto product : products)
  {
    int index = ui->table->rowCount();
    ui->table->insertRow(index);
    ui->table->setItem(index,                0,                new
QTableWidgetItem(QString::fromStdString(product.name)));
    ui->table->setItem(index,                1,                new
QTableWidgetItem(QString::fromStdString(product.article)));
    ui->table->setItem(index,                2,                new
QTableWidgetItem(QString::number(product.cost)));
    ui->table->setItem(index,                3,                new
QTableWidgetItem(QString::number(product.kind)));
    ui->table->setItem(index,                4,                new
QTableWidgetItem(QString::fromStdString(product.expiry_date)));
    ui->table->setItem(index,                5,                new
QTableWidgetItem(QString::fromStdString(product.receipt_date)));
  }
}

void CatalogBuyer::on_add()
{
  QList<QTableWidgetItem*>     selectedItems     =     ui->table-
>selectedItems();
  set<int> selected_rows;
  auto product_articles = order_buyer->current_order();
  for (auto item : selectedItems)
  {
    selected_rows.insert(item->row());
  }
  for (auto row : selected_rows)
  {
    auto article = ui->table->item(row, 1)->text().toStdString();
    product_articles.insert(article);
  }
  order_buyer->load(product_articles);
}

void CatalogBuyer::on_order()
{
  order_buyer->show();
}


//catalogmanagement.h

#ifndef CATALOGMANAGEMENT_H
#define CATALOGMANAGEMENT_H

#include <QWidget>
#include "user.h"
#include "users.h"
#include "addproduct.h"
#include "ordermanagement.h"
```

```
#include "salesreport.h"

namespace Ui
{
class CatalogManagement;
}

class CatalogManagement : public QWidget
{
  Q_OBJECT

public:
  CatalogManagement(User::Role role, QWidget *parent = nullptr);
  ~CatalogManagement();

public slots:
  void on_user_management();
  void on_report();
  void on_product_remove();
  void on_product_add();
  void on_order_management();
  void on_product_added(Product product);
  void on_updated();

private:
  Ui::CatalogManagement *ui;
  Users* users;
  Addproduct* add_product;
  OrderManagement* order_management;
  SalesReport *report;
};

#endif // CATALOGMANAGEMENT_H


//catalogmanagement.cpp

#include "catalogmanagement.h"
#include "ui_catalogmanagement.h"
#include "productdb.h"
#include <set>

CatalogManagement::CatalogManagement(User::Role   role,   QWidget
*parent) :
  QWidget(parent),
  ui(new Ui::CatalogManagement) {
  ui->setupUi(this);

  bool superEnabled = role == User::Role::CompanyManagement;

  ui->users_control->setEnabled(superEnabled);
  ui->report->setEnabled(superEnabled);
```

```
  connect(ui->add, SIGNAL(clicked()), SLOT(on_product_add()));
  connect(ui->remove,                          SIGNAL(clicked()),
SLOT(on_product_remove()));
  connect(ui->users_control,                   SIGNAL(clicked()),
SLOT(on_user_management()));
  connect(ui->report, SIGNAL(clicked()), SLOT(on_report()));
  connect(ui->orders,                          SIGNAL(clicked()),
SLOT(on_order_management()));

  users = new Users(nullptr);
  users->hide();

  add_product = new Addproduct(nullptr);
  add_product->hide();

  order_management = new OrderManagement(nullptr);
  order_management->hide();

  report = new SalesReport(nullptr);
  report->hide();

  connect(add_product,                   SIGNAL(product(Product)),
SLOT(on_product_added(Product)));
  connect(&PRODUCT_DB, SIGNAL(updated()), SLOT(on_updated()));

  on_updated();
}

CatalogManagement::~CatalogManagement() {
  delete ui;
}

void CatalogManagement::on_user_management() {
  users->show();
}

void CatalogManagement::on_report() {
  report->show();
}

void CatalogManagement::on_product_remove() {
  QList<QTableWidgetItem*>     selectedItems     =     ui->table-
>selectedItems();
  set<int> selected_rows;
  set<QString> removed_articles;

  for (auto item : selectedItems) {
    selected_rows.insert(item->row());
  }

  for (auto row : selected_rows) {
    auto article = ui->table->item(row, 1)->text();
    removed_articles.insert(article);
```

```
  }

  for (auto article : removed_articles) {
    PRODUCT_DB.remove_product(article.toStdString());
  }
}

void CatalogManagement::on_product_add() {
  add_product->show();
}

void CatalogManagement::on_order_management() {
  order_management->show();
}

void CatalogManagement::on_product_added(Product product) {
  Product other;
  if  (false  ==  PRODUCT_DB.product_by_article(product.article,
other))
    PRODUCT_DB.add_product(product);
}

void CatalogManagement::on_updated() {
  vector<Product> products = PRODUCT_DB.products();

  ui->table->setRowCount(0);
  for (auto product : products) {
    int index = ui->table->rowCount();
    ui->table->insertRow(index);

    ui->table->setItem(index,               0,               new
QTableWidgetItem(QString::fromStdString(product.name)));
    ui->table->setItem(index,               1,               new
QTableWidgetItem(QString::fromStdString(product.article)));
    ui->table->setItem(index,               2,               new
QTableWidgetItem(QString::number(product.cost)));
    ui->table->setItem(index,               3,               new
QTableWidgetItem(QString::number(product.kind)));
    ui->table->setItem(index,               4,               new
QTableWidgetItem(QString::fromStdString(product.expiry_date)));
    ui->table->setItem(index,               5,               new
QTableWidgetItem(QString::fromStdString(product.receipt_date)));
  }
}


//dbfacade.h

#ifndef DBFACADE_H
#define DBFACADE_H

#include <QObject>
#include <QtSql/QtSql>
```

```
#include <exception>

class DBFacade : public QObject
{
    Q_OBJECT
public:
    explicit DBFacade(QString databasename, QObject *parent = 0);
    ~DBFacade();

signals:
    void updated();

protected:
    void exec(QString);
    QString qs(QString);
    QString qs(std::string);
    QSqlDatabase m_db;
    QSqlQuery *m_query;
};

class OpenDBException: public std::exception
{
public:
    OpenDBException(const char* dbName) : m_dbname(dbName) {}

private:
    virtual const char* what() const throw()
    {
        return m_dbname;
    }
    const char* m_dbname;
};

class ExecException: public std::exception
{
public:
    ExecException(const char* request) : m_request(request) {}

private:
    virtual const char* what() const throw()
    {
        return m_request;
    }
    const char* m_request;
};

#endif // DBFACADE_H


//dbfacade.cpp

#include "dbfacade.h"
```

```
DBFacade::DBFacade(QString databasename, QObject *parent) :
QObject(parent)
{
    m_db = QSqlDatabase::addDatabase("QSQLITE", databasename);
    m_db.setDatabaseName(databasename);
    if (false == m_db.open())
        throw OpenDBException(databasename.toLatin1());
    m_query = new QSqlQuery(m_db);
}

DBFacade::~DBFacade()
{
    delete m_query;
}

QString DBFacade::qs(QString str)
{
    return "'" + str + "'";
}

QString DBFacade::qs(std::string str)
{
    return qs(QString::fromStdString(str));
}

void DBFacade::exec(QString str)
{
    if (false == m_query->exec(str))
        throw ExecException(str.toLatin1());
}


//order.h

#ifndef ORDER_H
#define ORDER_H

#include <string>
#include <vector>

using namespace std;
class Order
{
public:
  string date;
  string email;
  vector<int> product_ids;
};

#endif // ORDER_H


//orderbuyer.h
```

```
#ifndef ORDERBUYER_H
#define ORDERBUYER_H

#include <QWidget>
#include <set>
#include <string>
using namespace std;

namespace Ui
{
class OrderBuyer;
}

class OrderBuyer : public QWidget
{
  Q_OBJECT

public:
  explicit OrderBuyer(QWidget *parent = nullptr);
  ~OrderBuyer();

  void load(set<string> product_articles);
  set<string> current_order();

public slots:
  void create_order();
  void remove_product();

private:
  Ui::OrderBuyer *ui;
  set<string> product_articles;
};

#endif // ORDERBUYER_H


//orderbuyer.cpp

#include "orderbuyer.h"
#include "ui_orderbuyer.h"
#include "product.h"
#include "productdb.h"

OrderBuyer::OrderBuyer(QWidget *parent) :
  QWidget(parent),
  ui(new Ui::OrderBuyer)
{
  ui->setupUi(this);
  connect(ui->back, SIGNAL(clicked()), this, SLOT(hide()));
  connect(ui->ok, SIGNAL(clicked()), this, SLOT(create_order()));
  connect(ui->remove, SIGNAL(clicked()), SLOT(remove_product()));
}
```

```
OrderBuyer::~OrderBuyer()
{
  delete ui;
}

void OrderBuyer::load(set<string> additional_products)
{
  for (auto product : additional_products)
  {
    product_articles.insert(product);
  }
  ui->table->setRowCount(0);
  for (auto article : product_articles)
  {
    Product product;
    PRODUCT_DB.product_by_article(article, product);
    int index = ui->table->rowCount();
    ui->table->insertRow(index);
    ui->table->setItem(index,                0,                new
QTableWidgetItem(QString::fromStdString(product.name)));
    ui->table->setItem(index,                1,                new
QTableWidgetItem(QString::fromStdString(product.article)));
    ui->table->setItem(index,                2,                new
QTableWidgetItem(QString::number(product.cost)));
    ui->table->setItem(index,                3,                new
QTableWidgetItem(QString::number(product.kind)));
    ui->table->setItem(index,                4,                new
QTableWidgetItem(QString::fromStdString(product.expiry_date)));
    ui->table->setItem(index,                5,                new
QTableWidgetItem(QString::fromStdString(product.receipt_date)));
  }
}

set<string> OrderBuyer::current_order()
{
  return product_articles;
}

void OrderBuyer::create_order()
{
  QString email = ui->email->text();
  if  (email.contains('.')  ==  false  ||  email.contains('@')  ==
false)
    return;
  QString                      date_string                      =
QDate::currentDate().toString(Qt::ISODate);
  date_string.truncate(10);
  date_string.replace("-", ".");
  vector<Product> products;
  for (auto article : product_articles)
  {
    Product product;
```

```
    PRODUCT_DB.product_by_article(article, product);
    products.push_back(product);
  }
  PRODUCT_DB.add_order(date_string.toStdString(),
                       email.toStdString(),
                       products);
  hide();
}

void OrderBuyer::remove_product()
{
  QList<QTableWidgetItem*>     selectedItems     =     ui->table-
>selectedItems();
  set<int> selected_rows;
  for (auto item : selectedItems)
  {
    selected_rows.insert(item->row());
  }
  for (auto row : selected_rows)
  {
    auto article = ui->table->item(row, 1)->text().toStdString();
    product_articles.erase(article);
  }
  load(product_articles);
}


//ordermanagement.h

#ifndef ORDERMANAGEMENT_H
#define ORDERMANAGEMENT_H

#include <QWidget>
#include <vector>
#include "order.h"

using namespace std;

namespace Ui
{
class OrderManagement;
}

class OrderManagement : public QWidget
{
  Q_OBJECT

public:
  explicit OrderManagement(QWidget *parent = nullptr);
  ~OrderManagement();

public slots:
  void on_load();
```

```
  void on_remove();
  void on_sell();

  vector<Order> selected_orders();

private:
  Ui::OrderManagement *ui;
};

#endif // ORDERMANAGEMENT_H


//ordermanagement.cpp

#include "ordermanagement.h"
#include "ui_ordermanagement.h"
#include "productdb.h"
#include <sstream>

OrderManagement::OrderManagement(QWidget *parent) :
  QWidget(parent),
  ui(new Ui::OrderManagement)
{
  ui->setupUi(this);
  connect(ui->back, SIGNAL(clicked()), this, SLOT(hide()));
  connect(ui->sell, SIGNAL(clicked()), this, SLOT(on_sell()));
  connect(ui->remove,          SIGNAL(clicked()),          this,
SLOT(on_remove()));
  on_load();
}

OrderManagement::~OrderManagement()
{
  delete ui;
}

void OrderManagement::on_load()
{
  vector<Order> orders = PRODUCT_DB.orders();
  ui->table->setRowCount(0);
  for (auto order : orders)
  {
    int index = ui->table->rowCount();
    ui->table->insertRow(index);
    ui->table->setItem(index,               0,               new
QTableWidgetItem(QString::fromStdString(order.date)));
    ui->table->setItem(index,               1,               new
QTableWidgetItem(QString::fromStdString(order.email)));
    stringstream sstr;
    for (auto id : order.product_ids)
    {
      sstr << id << "  ";
    }
```

```
    string ids_str;
    getline(sstr, ids_str);
    ui->table->setItem(index,                2,                new
QTableWidgetItem(QString::fromStdString(ids_str)));
  }
}

vector<Order> OrderManagement::selected_orders()
{
  QList<QTableWidgetItem*>     selectedItems     =     ui->table-
>selectedItems();
  set<int> selected_rows;
  vector<Order> orders;
  for (auto item : selectedItems)
  {
    selected_rows.insert(item->row());
  }
  for (auto row : selected_rows)
  {
    auto date = ui->table->item(row, 0)->text().toStdString();
    auto email = ui->table->item(row, 1)->text().toStdString();
    auto   it_order   =   find_if(orders.begin(),   orders.end(),
[=](Order order)
    {
      return order.date == date && order.email == email;
    });
    if (it_order == orders.end())
    {
      Order order;
      order.date = date;
      order.email = email;
      orders.push_back(order);
    }
  }
  return orders;
}

void OrderManagement::on_remove()
{
  vector<Order> orders = selected_orders();
  for (auto order : orders)
  {
    PRODUCT_DB.remove_order(order.date, order.email);
  }
  on_load();
}

void OrderManagement::on_sell()
{
  vector<Order> orders = selected_orders();
  for (auto order : orders)
  {
    PRODUCT_DB.sell_order(order.date, order.email);
```

```
  }
  on_load();
}


//product.h

#ifndef PRODUCT_H
#define PRODUCT_H

#include <string>
using namespace std;

class Product
{
public:
  int id;
  string name;
  string article;
  int cost;
  int kind;
  string expiry_date;
  string receipt_date;
};
#endif // PRODUCT_H


//productdb.h

#ifndef PRODUCTDB_H
#define PRODUCTDB_H

#include "dbfacade.h"
#include "singleton.h"
#include <QObject>
#include <QList>
#include "order.h"
#include "product.h"
#include "soltproduct.h"
#include "user.h"
#include <set>

class ProductDB : public DBFacade
{
    Q_OBJECT
public:
    explicit ProductDB(QString dbFIlename = "products.sqlite",
QObject *parent = 0);
    bool is_login_busy(QString login);
    void add_user(User user);
    vector<User> users();
    void remove_user(string login);
    User get_user(QString login);
```

```
    vector<Product> products();
    void add_product(Product product);
    bool product_by_article(string article, Product &product);
    bool product_by_id(int id, Product &product);
    void remove_product(string article);
    void  add_order(string  date,  string  email,  vector<Product>
products);
    vector<Order> orders();
    void remove_order(string date, string email);
    void sell_order(string date, string email);
    int count(QString from, QString to);
    int sum(QString from, QString to);
    void createTables();

public slots:
};

#define PRODUCT_DB Singleton<ProductDB>::instance()

#endif // PRODUCTDB_H


//productdb.cpp

#include "productdb.h"
#include <QMap>

ProductDB::ProductDB(QString dbFIlename, QObject *parent)
    : DBFacade(dbFIlename, parent)
{
}

void ProductDB::createTables()
{
    if (false == m_db.tables().contains("users"))
    {
      exec("CREATE TABLE users"
           "("
              "login TEXT PRIMARY KEY, "
              "password TEXT NOT NULL, "
              "role TEXT"
           ");"
      );
    }

    if (false == m_db.tables().contains("products"))
    {
      exec("CREATE TABLE products"
           "("
              "id INTEGER PRIMARY KEY AUTOINCREMENT, "
              "name TEXT NUT NULL, "
              "article TEXT NUT NULL, "
              "cost INTEGER NUT NULL, "
```

```
                "kind INTEGER NUT NULL, "
                "expiry_date TEXT NOT NULL, "
                "receipt_date TEXT NOT NULL "
            ");"
      );
    }

    if (false == m_db.tables().contains("solt_products"))
    {
      exec("CREATE TABLE solt_products"
           "("
              "id INTEGER PRIMARY KEY, "
              "name TEXT NUT NULL, "
              "article TEXT NUT NULL, "
              "cost INTEGER NUT NULL, "
              "kind INTEGER NUT NULL, "
              "expiry_date TEXT NOT NULL, "
              "receipt_date TEXT NOT NULL, "
              "realization_date TEXT NOT NULL"
          ");"
      );
    }

    if (false == m_db.tables().contains("products_order"))
    {
      exec("CREATE TABLE products_order"
           "("
              "id_product INTEGER NOT NULL,"
              "date TEXT NOT NULL,"
              "email TEXT NOT NULL"
          ");"
      );
    }
}

bool ProductDB::is_login_busy(QString login)
{
  QString query = tr("SELECT login FROM users WHERE login = ") +
qs(login);
  exec(query);
  return m_query->first();
}

void ProductDB::add_user(User user)
{
  QString query = tr("INSERT INTO users(login, password, role)
VALUES (")
                        +  qs(QString::fromStdString(user.login))
+ ","
                        +
qs(QString::fromStdString(user.password)) + ","
                        +
qs(QString::fromStdString(user.role_to_string(user.role))) + ")";
```

```
  exec(query);
  emit updated();
}

vector<User> ProductDB::users()
{
  vector<User> users;
  QString query = tr("SELECT login, password, role FROM users ");
  exec(query);
  while (true == m_query->next())
  {
    QString login = m_query->value(0).toString();
    QString password = m_query->value(1).toString();
    QString role = m_query->value(2).toString();
    users.push_back(User(login.toStdString(),
                         password.toStdString(),

User::string_to_role(role.toStdString())));
  }
  return users;
}

User ProductDB::get_user(QString login)
{
  QString query = tr("SELECT password, role FROM users WHERE login
= ") + qs(login);
  exec(query);
  if (false == m_query->next())
  {
    throw string("wrong login");
  }
  QString password = m_query->value(0).toString();
  QString role = m_query->value(1).toString();
  return    User(login.toStdString(),    password.toStdString(),
User::string_to_role(role.toStdString()));
}

void ProductDB::remove_user(string login)
{
  QString  query  =  tr("DELETE  FROM  users  WHERE  login  =  ")  +
qs(login);
  exec(query);
  emit updated();
}

void ProductDB::add_product(Product product)
{
  QString query = tr("INSERT INTO products(name, article, cost,
kind, expiry_date, receipt_date) VALUES (")
                        + qs(product.name) + ","
                        + qs(product.article) + ","
                        + QString::number(product.cost) + ","
                        + QString::number(product.kind) + ","
```

```
                        + qs(product.expiry_date) + ","
                        + qs(product.receipt_date) + ")";
  exec(query);
  emit updated();
}

vector<Product> ProductDB::products()
{
  vector<Product> products;
  QString query = tr("SELECT id, name, article, cost, kind,
expiry_date, receipt_date FROM products ");
  exec(query);
  while (true == m_query->next())
  {
    Product data;
    data.id = m_query->value(0).toInt();
    data.name = m_query->value(1).toString().toStdString();
    data.article = m_query->value(2).toString().toStdString();
    data.cost = m_query->value(3).toInt();
    data.kind = m_query->value(4).toInt();
    data.expiry_date                  =                 m_query-
>value(5).toString().toStdString();
    data.receipt_date                  =                 m_query-
>value(6).toString().toStdString();

    products.push_back(data);
  }
  return products;
}

bool ProductDB::product_by_article(string article, Product&
product)
{
  QString query = tr("SELECT name, article, cost, kind,
expiry_date, receipt_date, id "
                     "FROM products WHERE article = ") +
qs(article);
  exec(query);
  if (false == m_query->next())
  {
    return false;
  }
  product.name = m_query->value(0).toString().toStdString();
  product.article = m_query->value(1).toString().toStdString();
  product.cost = m_query->value(2).toInt();
  product.kind = m_query->value(3).toInt();
  product.expiry_date                  =                 m_query-
>value(4).toString().toStdString();
  product.receipt_date                  =                 m_query-
>value(5).toString().toStdString();
  product.id = m_query->value(6).toInt();
  return true;
}
```

```
bool ProductDB::product_by_id(int id, Product& product)
{
  QString query = tr("SELECT name, article, cost, kind,
expiry_date, receipt_date, id "
                     "FROM products WHERE id = ") +
QString::number(id);
  exec(query);
  if (false == m_query->next())
  {
    return false;
  }
  product.name = m_query->value(0).toString().toStdString();
  product.article = m_query->value(1).toString().toStdString();
  product.cost = m_query->value(2).toInt();
  product.kind = m_query->value(3).toInt();
  product.expiry_date = m_query-
>value(4).toString().toStdString();
  product.receipt_date = m_query-
>value(5).toString().toStdString();
  product.id = m_query->value(6).toInt();
  return true;
}

void ProductDB::remove_product(string article)
{
  QString query = tr("DELETE FROM products WHERE article = ") +
qs(article);
  exec(query);
  emit updated();
}

void ProductDB::add_order(string date, string email,
vector<Product> products)
{
  for (auto product : products)
  {
    QString query = tr("INSERT INTO products_order(date, email,
id_product) VALUES (")
                       + qs(date) + ","
                       + qs(email) + ","
                       + QString::number(product.id) + ")";
    exec(query);
  }
}

vector<Order> ProductDB::orders()
{
  vector<Order> orders;
  QString query = tr("SELECT date, email, id_product FROM
products_order ");
  exec(query);
  while (true == m_query->next())
```

```
  {
    string date = m_query->value(0).toString().toStdString();
    string email = m_query->value(1).toString().toStdString();
    int id = m_query->value(2).toInt();
    auto   it_order   =   find_if(orders.begin(),   orders.end(),
[=](Order order){
      return order.date == date && order.email == email;
    });
    if (it_order != orders.end())
    {
      (*it_order).product_ids.push_back(id);
    }
    else
    {
      Order order;
      order.date = date;
      order.email = email;
      order.product_ids.push_back(id);
      orders.push_back(order);
    }
  }
  return orders;
}

void ProductDB::remove_order(string date, string email)
{
  QString query = tr("DELETE FROM products_order WHERE ") +
                     " date = " + qs(date) +
                     "AND email = " + qs(email);
  exec(query);
  emit updated();
}

void ProductDB::sell_order(string date, string email)
{
  QString                        date_string                        =
QDate::currentDate().toString(Qt::ISODate);
  date_string.truncate(10);
  date_string.replace("-", ".");
  QString query = tr("SELECT id_product FROM products_order WHERE
") +
                     " date = " + qs(date) +
                     "AND email = " + qs(email);
  exec(query);
  vector<int> sold_ids;
  while (true == m_query->next())
  {
    int id = m_query->value(0).toInt();
    sold_ids.push_back(id);
  }
  for (auto id : sold_ids)
  {
    Product product;
```

```
    product_by_id(id, product);
    query = tr("INSERT INTO solt_products")
         + tr(" (id, name, article, cost, kind, expiry_date,
receipt_date, realization_date)")
         + tr(" VALUES (")
         + QString::number(product.id) + ","
         + qs(product.name) + ","
         + qs(product.article) + ","
         + QString::number(product.cost) + ","
         + QString::number(product.kind) + ","
         + qs(product.expiry_date) + ","
         + qs(product.receipt_date) + ","
         + qs(date_string) + ")";
    exec(query);
    QString query = tr("DELETE FROM products WHERE id = ") +
QString::number(id);
    exec(query);
  }
  remove_order(date, email);
  emit updated();
}

int ProductDB::count(QString from, QString to)
{
    exec(
          tr("SELECT COUNT(*) FROM solt_products WHERE ") +
          tr("realization_date >= ") + qs(from) +
          tr(" AND realization_date <= ") + qs(to));
    m_query->next();
    return m_query->value(0).toInt();
}

int ProductDB::sum(QString from, QString to)
{
    exec(
          tr("SELECT SUM(cost) FROM solt_products WHERE ") +
          tr("realization_date >= ") + qs(from) +
          tr(" AND realization_date <= ") + qs(to));
    m_query->next();
    return m_query->value(0).toInt();
}


//salesreport.h

#ifndef SALESREPORT_H
#define SALESREPORT_H

#include <QWidget>

namespace Ui
{
class SalesReport;
```

```
}

class SalesReport : public QWidget
{
  Q_OBJECT

public:
  explicit SalesReport(QWidget *parent = nullptr);
  ~SalesReport();

public slots:
  void on_calc();

private:
  Ui::SalesReport *ui;
};

#endif // SALESREPORT_H


//salesreport.cpp

#include "salesreport.h"
#include "ui_salesreport.h"
#include "productdb.h"

SalesReport::SalesReport(QWidget *parent) :
  QWidget(parent),
  ui(new Ui::SalesReport)
{
  ui->setupUi(this);
  connect(ui->calc, SIGNAL(clicked()), SLOT(on_calc()));
  QRegExpValidator*          dateValidator          =          new
QRegExpValidator(QRegExp("20\\d{2}\\.[01]?\\d\\.[0123]?\\d"),
this);
  ui->start->setValidator(dateValidator);
  ui->end->setValidator(dateValidator);
}

SalesReport::~SalesReport()
{
  delete ui;
}

void SalesReport::on_calc()
{
  QString from = ui->start->text();
  QString to = ui->end->text();
  int count = PRODUCT_DB.count(from, to);
  int sum = PRODUCT_DB.sum(from, to);
  ui->count->setText(QString::number(count));
  ui->sum->setText(QString::number(sum));
}
```

```
//singleton.h

#ifndef SINGLETON_H
#define SINGLETON_H

#include <QObject>

template <class T>
class Singleton
{
public:
  static T& instance()
  {
    static T instance;
    return instance;
  }

private:
  Singleton();
  ~Singleton();
  Singleton(const Singleton &);
  Singleton& operator=(const Singleton &);
};

#endif


//soltproduct.h

#ifndef SOLTPRODUCT_H
#define SOLTPRODUCT_H

#include "product.h"

class SoltProduct : public Product
{
public:
  string  realization_date;
};

#endif // SOLTPRODUCT_H
//user.h

#ifndef USER_H
#define USER_H

#include <string>

using namespace std;

class User {
```

```
public:
  enum Role {
    ProductManagement, CompanyManagement
  };

  User(string _login, string _password, Role _role);

  static string role_to_string(Role role);
  static Role string_to_role(string);

  string login, password;
  Role role;
};

#define CURRENT_USER Singleton<string>::instance()

#endif // USER_H


//user.cpp

#include "user.h"

User::User(string _login, string _password, Role _role)
  : login(_login), password(_password), role(_role) {
}


string User::role_to_string(Role role) {
  if (role == ProductManagement) {
    return "Product";
  }
  return "Company";
}

User::Role User::string_to_role(string str) {
  if (str == "Product")
    return Role::ProductManagement;
  if (str == "Company")
    return Role::CompanyManagement;
  throw string("wrong role");
}

//users.h

#ifndef USERS_H
#define USERS_H

#include <QWidget>

namespace Ui
{
  class Users;
```

```
}

class Users : public QWidget
{
  Q_OBJECT

public:
  explicit Users(QWidget *parent = nullptr);
  ~Users();

public slots:
  void on_remove();a
  void on_add();
  void on_updated();

private:
  Ui::Users *ui;
};

#endif // USERS_H


//users.cpp

#include "users.h"
#include "productdb.h"
#include "ui_users.h"
#include "user.h"
#include <set>
using namespace std;

Users::Users(QWidget *parent) :
  QWidget(parent),
  ui(new Ui::Users)
{
  ui->setupUi(this);
  on_updated();
  connect(ui->add, SIGNAL(clicked()), SLOT(on_add()));
  connect(ui->remove, SIGNAL(clicked()), SLOT(on_remove()));
  connect(&PRODUCT_DB, SIGNAL(updated()), SLOT(on_updated()));
}

Users::~Users()
{
  delete ui;
}

void Users::on_remove()
{
  QList<QTableWidgetItem*>     selectedItems     =     ui->table-
>selectedItems();
  set<int> selected_rows;
  set<QString> removed_users;
```

```
  for (auto item : selectedItems)
  {
    selected_rows.insert(item->row());
  }
  for (auto row : selected_rows)
  {
    auto login = ui->table->item(row, 0)->text();
    removed_users.insert(login);
  }
  for (auto login : removed_users)
  {
    User user = PRODUCT_DB.get_user(login);
    if (user.login == CURRENT_USER)
    {
      continue;
    }
    PRODUCT_DB.remove_user(user.login);
  }
}

void Users::on_add()
{
  QString login = ui->login->text();
  QString password = ui->password->text();
  if (login.isEmpty() || password.isEmpty())
    return;
  if (PRODUCT_DB.is_login_busy(login))
    return;
  bool isSuper = ui->isSuper->isChecked();
  User::Role role;
  if (false == isSuper)
  {
    role = User::Role::ProductManagement;
  }
  else
  {
    role = User::Role::CompanyManagement;
  }
  PRODUCT_DB.add_user(User(login.toStdString(),
password.toStdString(), role));
}

void Users::on_updated()
{vector<User> users = PRODUCT_DB.users();
  ui->table->setRowCount(0);
  for (auto user : users)
  {int index = ui->table->rowCount();
    ui->table->insertRow(index);
    ui->table->setItem(index,                0,                new
QTableWidgetItem(QString::fromStdString(user.login)));
    ui->table->setItem(index,                1,                new
QTableWidgetItem(QString::fromStdString(User::role_to_string(use
r.role))));}}
```

ДАДАТАК Б
Дыяграма класаў

ДАДАТАК В

Схема метаду on_load()

ДАДАТАК Г

Схема метаду sell_order()

ДАДАТАК Д
Ведамасць дакументаў