МІНІСТЕРСТВО ОСВІТИ И НАУКИ УКРАЇНИ
ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І. МЕЧНИКОВА
Факультет математики, фізики та інформаційних технологій
Кафедра математичного забезпечення комп'ютерних систем

КУРСОВИЙ ПРОЕКТ

з дисципліни

«Проектування інформаційних систем»

на тему:

«База даних продуктового магазину»

студента III курсу

групи ___I_____

спеціальності Інформаційні системи та технології

_____

Керівник:_____

Захищено «___»_____202___р.

з оцінкою _____

Комісія:

_____
(ПІБ) (Підпис)

_____
(ПІБ) (Підпис)

_____
(ПІБ) (Підпис)

Одеса – 2022

АНОТАЦІЯ

Метою даного курсового проєкту є проєктування та реалізація інформаційної системи для організації та автоматизації процесів управління продуктовим магазином. Основними користувачами системи є працівники магазину, які здійснюють облік товарів, формування замовлень, контроль постачань та обробку повернень продукції. Реалізація системи виконана з використанням мови програмування JavaScript на базі фреймворку Next.js, ORM-бібліотеки Prisma для роботи з базою даних, а також системи управління базами даних PostgreSQL.

У створеній системі передбачено повний життєвий цикл товарообігу: від прийому товару на склад, формування накладних та квитанцій, до продажу кінцевим споживачам та обробки повернень. Реалізовано функціонал ведення обліку персоналу, постачальників, категорій товарів та історії транзакцій. Система побудована за архітектурною моделлю MVC, що забезпечує чітке розмежування рівнів обробки даних, бізнес-логіки та відображення інформації. Особлива увага приділена цілісності даних та захищеності бізнес-логіки, що дозволяє мінімізувати можливість помилок з боку користувачів під час виконання операцій.

Результатом виконаної роботи є сучасна, зручна у користуванні інформаційна система обліку продуктового магазину, що забезпечує ефективне управління торгово-складськими процесами та може бути адаптована до потреб підприємств роздрібної торгівлі.

ЗМІСТ

МІНІСТЕРСТВО ОСВІТИ И НАУКИ УКРАЇНИ 1
ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І. МЕЧНИКОВА 1
АНОТАЦІЯ 2
ЗМІСТ 3
ВСТУП 5
ПОСТАНОВКА ЗАДАЧІ 7
ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ 10
ІНФОРМАЦІЙНЕ МОДЕЛЮВАННЯ ПРЕДМЕТНОЇ ОБЛАСТІ 12
ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ СТВОРЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ 15
ПРОГРАМНА МОДЕЛЬ ЗАСТОСУНКА 17
СТВОРЕННЯ БАЗИ ДАНИХ 19
ЗАПИТИ ДО БАЗИ ДАНИХ ДЛЯ ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ 22
ПРОГРАМНА РЕАЛІЗАЦІЯ ІНТЕРФЕЙСУ 25
БЕЗПЕКА ІНФОРМАЦІЙНОЇ СИСТЕМИ 34
ІНСТРУКЦІЯ КОРИСТУВАЧА 36
10.1 Авторизація з першого погляду 36
10.2 Інтерфейс адміністратора 37
10.3 Інтерфейс продавця 45
ВИСНОВОК 46
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 47
ДОДАТОК А 48
Задачі користувачів ІС «База даних продуктового магазину» 48
ДОДАТОК В 51
Опис сутностей предметної області 51
ДОДАТОК С 56
Запити на створення доменів та таблиць бази даних 56

ДОДАТОК D 59
Запити на створення додаткових функцій і тригерів збереження цілістності ІС 59
ДОДАТОК E 64
Запити на створення представлень для вирішення задач користувачів 64
ДОДАТОК F 65
Список привілеїв ролей на об’єкти БД 65
ДОДАТОК G 66
Створення ролей та призначення їм привілеїв 66

ВСТУП

Ведення обліку товарів у продуктовому магазині є складним і відповідальним процесом, який потребує постійного контролю за залишками продукції, актуальністю цін, дотриманням термінів придатності, обробкою накладних, обліком продажів і повернень. Вручну цей процес супроводжується значними витратами часу та людськими помилками, що може призводити до фінансових втрат або порушення роботи магазину. Для підвищення ефективності роботи та зменшення впливу людського фактора доцільним є впровадження спеціалізованої інформаційної системи автоматизованого обліку.

У сучасних умовах, коли торгові заклади мають справу з великими обсягами даних, зростає потреба у зручному, гнучкому та масштабованому інструменті, який дозволив би контролювати всі аспекти діяльності — від постачання товарів до реалізації кінцевому споживачу. При цьому інформаційна система повинна враховувати потреби різних категорій користувачів — адміністратора та продавця — і забезпечувати захист від несанкціонованого доступу та розмежування прав.

Незважаючи на наявність комерційних рішень для торговельного обліку, багато з них є дорогими, складними в освоєнні або не підходять для кастомізації під конкретні потреби малого магазину. У зв’язку з цим актуальним є створення власного веб-застосунку, який дозволить реалізувати функціональність, необхідну саме для конкретного магазину, з урахуванням специфіки його роботи.

Мета даного курсового проєкту – проектування і реалізація інформаційної системи обліку продуктового магазину у вигляді веб-застосунку з поділом на ролі адміністратора та продавця. Створена система має надати інструменти для ефективного управління товарами, категоріями, поставками, продажами та поверненнями. При цьому:

адміністратор отримує можливість контролювати весь торговий процес, керувати працівниками, базою товарів та аналітикою;

продавець — зручний інтерфейс для виконання щоденних операцій з обліку продажу та оформлення повернень;

а власник бізнесу — прозору картину щодо товарообігу, залишків і діяльності працівників.

Для досягнення мети потрібно розв’язати низку завдань:

1. виконати аналіз предметної області,
2. визначити ролі користувачів і їхні функціональні потреби, обрати архітектурний підхід (у цьому випадку — MVP або MVVM),
3. спроектувати структуру бази даних PostgreSQL,
4. обрати стек технологій для фронтенду (React, Next.js),
5. реалізувати авторизацію, систему прав доступу, а також забезпечити захист даних на рівні застосунку та бази даних.

1 ПОСТАНОВКА ЗАДАЧІ

У процесі проєктування інформаційної системи етап постановки задачі є критично важливим, оскільки він передбачає збір і аналіз вимог до структури даних та до функціональності системи з боку різних категорій користувачів. Від правильно сформульованих задач та обґрунтованого поділу ролей користувачів залежить ефективність розроблюваної ІС. На основі аналізу вимог визначаються основні функціональні потреби системи та структура бази даних, яка буде використовуватись у застосунку.

Інформаційна система обліку товарів продуктового магазину покликана розв’язувати такі задачі:

1. Облік товарів, їх кількості, категорій, одиниць виміру, виробників та постачальників.
2. Ведення накладних на поставку товарів до магазину з деталізацією за кожною позицією.
3. Формування чеків на продаж товарів із зазначенням продавця, дати продажу та переліку проданих продуктів.
4. Оформлення повернень з можливістю повернення як усього чека, так і окремих товарів, із фіксацією відповідального працівника.
5. Перегляд історії продажів, поставок, повернень і фільтрація за критеріями (дати, працівник, тип чека тощо).
6. Контроль залишків товарів на складі з автоматичним списанням при продажі.
7. Управління працівниками магазину: створення, редагування, призначення посад.
8. Забезпечення розмежування прав доступу між працівниками різних посад (продавець, адміністратор).
9. Забезпечення цілісності та захисту даних на рівні бази даних та застосунку.

Категорії користувачів

Інформаційна система передбачає три основні категорії користувачів, які відображають організаційну структуру магазину та рівень доступу до даних:

1. Продавець

- Доступні функції:
  - Перегляд наявних товарів.
  - Перегляд історії продажів.
  - Створення нових чеків.
  - Оформлення повернень.
  - Перегляд детальної інформації про поставки та продукти.
- Обмеження:
  - Немає доступу до редагування довідкової інформації (товари, категорії, працівники, постачальники).
  - Не може видаляти чеки або повернення.
  - Не має доступу до змін у налаштуваннях системи.

2. Адміністратор

Доступні функції:

- Повний доступ до всієї інформації в системі.
- Додавання, редагування та видалення даних про:
  - товари,
  - категорії,
  - постачальників,
  - працівників.
- Створення та контроль поставок.
- Управління обліковими записами користувачів.
- Перегляд усіх операцій у системі.
- Проведення аналітики товарообігу та ефективності працівників.

3. Гість

- Доступні функції:
  - Перегляд довідкової таблиці користувачів (read-only).
- Обмеження:
  - Немає доступу до жодних інших таблиць чи функцій.
  - Не може змінювати або додавати дані.
  - Має мінімальні права лише з метою ознайомлення або аудиту.

- Кожна категорія має доступ лише до тих функцій, які відповідають її ролі.

Це забезпечує:

- Зручність користування для персоналу.
- Безпеку обробки даних завдяки контрольованому доступу.
- Інформаційну цілісність у межах обов’язків користувача.

Усі задачі користувачів перелічені в Додатку А.

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

При аналізі структури необхідного додатку обрано три-рівневу архітектуру.

Трирівнева архітектура — це архітектурна модель, що передбачає в її складі три взаємопов’язані компоненти:

– **Клієнт,** реалізований на Next.js (React), що відповідає за інтерфейс користувача;

– **Сервер**, реалізований через API-роути Next.js із використанням Prisma ORM, що обробляє бізнес-логіку;

– **Сервер** бази даних — PostgreSQL, у якому зберігаються всі дані додатку.

Рисунок 2.1 – Трирівнева архітектура

У такій архітектурі кожний компонент створює свій логічний рівень: клієнт відповідає за рівень представлення, сервер додатку за рівень бізнес-логіки та сервер баз даних за рівень доступу до даних [1] (рис. 2.2).

Рисунок 2.2 – Логічні рівні додатку

У даній архітектурі окремі компоненти чітко розділені за обов’язками, що підвищує гнучкість, масштабованість та полегшує супровід додатку. Кожен компонент функціонує незалежно, що дозволяє окремим командам розробників працювати над відповідними рівнями.

Архітектура поділена на три рівні:

1. **Клієнт (Frontend)** – верхній рівень, реалізований за допомогою React (через Next.js). Клієнт відповідає лише за відображення інтерфейсу та збір дій користувача. Уся бізнес-логіка винесена на сервер. Клієнт взаємодіє із сервером через API-ендпоінти (app/api/.../route.ts), не маючи прямого доступу до бази даних.
2. **Веб-сервер (Backend)** – середній рівень, реалізований у вигляді серверних функцій Next.js (файли route.ts). Тут реалізовано бізнес-логіку, наприклад:
   - створення чеків, повернення товарів;
   - перевірка прав доступу через сесію (getServerSession);
   - обробка даних, перевірка введених параметрів;
   - маршрутизація логіки на основі ролі користувача через getPrismaByRole.
3. **Сервер бази даних (PostgreSQL)** – нижній рівень, що відповідає за збереження, обробку та цілісність даних. Завдяки Prisma, логіка звернення до бази чітко типізована. Крім того, деякі перевірки або обмеження (наприклад, через тригери або унікальні ключі) реалізовані безпосередньо на рівні бази.

Рисунок 2.3 – Модель MVC

3 ІНФОРМАЦІЙНЕ МОДЕЛЮВАННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

Сутності та їх атрибути з описом обмежень, що потрібні для розв’язання поставлених задач, наведено в додатку В.

В базі даних інформаційної системи «База даних продуктового магазину» існують 2 види зв’язку між сутностями, а саме:

1) один-до-багатьох;
2) один-до-багатьох умовний

Розглянемо кожний зі зв’язків по черзі:

1) зв’язок один-до-багатьох формалізується додаванням первинного ключа сутності, що відповідає одиничному зв’язку у якості зовнішнього ключа до сутності зі сторони множинного зв’язку. Такий вид зв’язку наявний між таблицями:

a) Categories → Products;
b) Products → ProductsToOrder;
c) Position → Employee;
d) Employee → Invoice;

2) Invoice → Provider, для формалізації яких встановлено умовний зв’язок. Його реалізовано шляхом додавання зовнішнього ключа provider_id у таблицю Invoice, що посилається на первинний ключ таблиці Provider. Проте зв’язок є умовним, оскільки постачальник (provider) обов’язково вказується лише для певних типів накладних. Для типу 'Списання' зв’язок із постачальником відсутній (provider_id IS NULL). Такий підхід реалізовано через обмеження CHECK, що забезпечує логічну цілісність даних згідно з бізнес-правилами. Сутність Invoice залежно від значення invoice_type може мати або не мати пов’язаний запис у сутності Provider.

Розглянемо зв’язок між сутностями Position та Employee. Зрозуміло, що кожний працівник може мати лише одну посаду, при цьому кожна посада може мати безліч працівників. Отже маємо класичний зв’язок один-до-багатьох.

Наступний приклад — розглянемо зв’язок між сутностями Invoice та Provider. На перший погляд, кожна накладна може бути пов’язана з певним постачальником, що відповідає зв’язку один-до-багатьох: один постачальник може мати безліч накладних, але кожна накладна — лише одного постачальника. Проте в інформаційній системі передбачено умовний зв’язок, оскільки не кожна накладна обов’язково пов’язана з постачальником. Наприклад, для накладних типу «Списання» постачальник не вказується взагалі. Таким чином, наявність зв’язку з Provider залежить від значення атрибута invoice_type. Такий тип зв’язку називається умовним і забезпечується за допомогою обмеження CHECK, яке дозволяє зберігати логічну цілісність даних залежно від контексту.

Усі вказані таблиці знаходяться у формі Бойса – Кодда.

ER-діаграма отримана в результаті формалізації зв’язків інформаційної системи база даних продуктового магазину приведена на рисунку 3.1. Для її побудування використано веб застосунок Draw.io з використанням нотації «вороняча лапка».

Рисунок 3.1 – ER-діаграма інформаційної системи «База даних продуктового магазину»

4 ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ СТВОРЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Інформаційна система ‘База даних продуктового магазину’ реалізована у форматі веб-застосунку, що дозволяє взаємодіяти з системою безпосередньо через браузер.

Для реалізації було використано такі технології:

1. Система управління базами даних – PostgreSQL, що забезпечує надійне зберігання, швидкий доступ та потужну рольову систему контролю доступу;
2. Основна мова Back-End частини – TypeScript;
3. Front-End – побудований з використанням Next.js (фреймворк поверх React), що дозволяє використовувати SSR (Server-Side Rendering) для оптимізації швидкості завантаження;
4. CSS – реалізовано з використанням Tailwind CSS.
5. Доступ до бази даних – забезпечується ORM Prisma, яка дозволяє легко взаємодіяти з базою даних PostgreSQL через автогенеровані класи моделей;
6. Інтегроване середовище розробки – Visual Studio Code.

СУБД PostgreSQL обрана через:

- відкриту безкоштовну ліцензію;
- високу продуктивність та масштабованість;
- об’єктно-реляційну модель даних, що полегшує зв’язок із класами Prisma;
- гнучке управління ролями та правами доступу;
- активну спільноту і велику кількість доступної документації.

Фреймворк Next.js дозволяє реалізовувати як серверну логіку, так і клієнтський інтерфейс в одному середовищі, забезпечуючи зручну маршрутизацію, оптимізацію SEO, SSR/SSG функціонал та API-ендпоінти “з коробки”.

Для доступу до бази використано ORM Prisma, що надає механізм генерації моделей на основі вже наявної структури бази даних (Database-First або Schema-First підхід). Prisma автоматично генерує типи та CRUD-запити, що

дозволяє розробнику працювати з базою даних як зі звичайними класами у TypeScript.

Завдяки такій архітектурі система є масштабованою, зручною для підтримки та легко інтегрується з іншими сервісами (наприклад, системами авторизації, платіжними API тощо).

5 ПРОГРАМНА МОДЕЛЬ ЗАСТОСУНКА

У цьому розділі розглянуто модель програми, що була розроблена в рамках курсового проєкту — веб-застосунку для управління продуктовим магазином. Програма створена з використанням стеку технологій Next.js, Prisma та PostgreSQL.

Структура застосунку реалізована відповідно до принципів модульності та об'єктно-орієнтованого підходу. У процесі моделювання застосунок було декомпозовано на підсистеми: управління продуктами, категоріями, рахунками, чеками, працівниками та постачальниками. Такий підхід забезпечує гнучкість, розширюваність і масштабованість інформаційної системи.

Логіка програми структурована за схемою MVC. Рівень управління ресурсами (Model) реалізовано через ORM Prisma — для зручної роботи з базою даних PostgreSQL. На рівні прикладної логіки (Controller/Presenter) використано API-обробники Next.js, які забезпечують взаємодію між клієнтською частиною і базою даних, а також виконують валідацію, перевірку прав доступу і бізнес-логіку. Представлення (View) реалізовано у вигляді React-компонентів, сторінок і форм з використанням Tailwind CSS.

Важливою частиною проєкту є реалізація ролей користувачів, зокрема адміністратора і продавця. Для кожної ролі реалізовані окремі підсистеми з відповідними інтерфейсами, які відображають тільки ті функції, що доступні даному типу користувача.

Модель рівня представлення (View) побудована у вигляді ієрархії веб-сторінок. Для адміністратора передбачені сторінки: головна панель керування, управління працівниками, категоріями, постачальниками, продуктами, рахунками, аналітикою. Продавець має доступ до сторінок: створення продажу (чек), перегляд повернень, пошук продуктів, перегляд чеків.

Перехід між сторінками реалізовано через вбудований маршрутизатор Next.js, що дозволяє здійснювати навігацію з урахуванням контексту дій користувача. Інтерфейс розроблено таким чином, щоб мінімізувати кількість переходів і спростити виконання бізнес-операцій.

Завдяки такій організації модель програми є простою для розуміння,

зручною для підтримки і придатною до подальшого розширення функціональності.

Рисунок 5.1 – Ієрархія веб-сторінки ІС «База даних продуктового магазину»

6 СТВОРЕННЯ БАЗИ ДАНИХ

Кожна база даних складається таких основних компонентів як таблиці, послідовності, представлення, функції, збережені процедури та тригери.

Для створення таблиці `products` потрібно виконати наступний SQL-запит:

```
CREATE TABLE products (
    products_id smallserial PRIMARY KEY,
    products_name VARCHAR(255) NOT NULL,
    category_id INTEGER NOT NULL,
    unit VARCHAR(50) NOT NULL,
    company VARCHAR(100) NOT NULL,
    storage_temperature text,

    CONSTRAINT fk_category
        FOREIGN KEY (category_id)
        REFERENCES categories(categories_id)
        ON DELETE CASCADE
);
```

Для створення таблиці використовується команда `CREATE TABLE`, після якої вказується назва таблиці. Для кожного поля вказується тип даних, що представляє інформацію, яку він зберігає. Поле `products_id` є первинним ключем (`PRIMARY KEY`) та має тип `SMALLSERIAL`, що насправді не є типом даних, а представляє собою механізм використання такого компоненту баз даних як `SEQUENCE`, або послідовність, що збільшується на одиницю кожний раз, коли видається нове значення за замовчуванням. Обмеження `NOT NULL` означає, що поле не може приймати порожні значення.

Для прикладу створимо таблицю сутності, що посилається на таблицю `products`.

```
CREATE TABLE products_to_order (
    products_to_order_id smallserial PRIMARY KEY,
    product_id INTEGER NOT NULL,
    quantity INTEGER NOT NULL,
    order_date DATE DEFAULT CURRENT_DATE NOT NULL ,
    CONSTRAINT fk_product
        FOREIGN KEY (product_id)
        REFERENCES products(products_id)
        ON DELETE CASCADE
);
```

Поле `product_id` посилається на поле `products_id` таблиці `products`, виступаючи зовнішнім ключем з метою забезпечення зв’язку та цілісності. Пункт `DEFAULT` у стовпці `order_date` означає те, якщо при

створенні нового рядку у таблиці сутності дане поле буде порожнім, то за замовчуванням воно буде встановлено значенням `CURRENT_DATE`, тобто сьогоднішньою датою. Після вказання стовпця йде умова, що визначає дії, які виконується зі значенням поля у разі певних подій. У даному випадку умовою виступає `ON DELETE CASCADE`, що означає, що при видаленні запису у батьківській таблиці, пов'язані з ним записи у дочірній таблиці також будуть автоматично видалені.

Для створення представлення використовується команда `CREATE VIEW`, після якої вказується його назва. Представлення – це віртуальна поіменована похідна таблиця [3], тобто її не існує у фізичному вигляді, а замість читання таблиці у базі виконується SQL-запит, який представляє це представлення.

Представлення для отримання поточних продуктів в наявності

```
CREATE OR REPLACE VIEW public.current_product_stock
 AS
 SELECT pfi.product_id,
    p.product_name AS products_name,
    sum(
        CASE
            WHEN i.invoice_type = 'Отримання'::invoice_type_enum AND i.status::text = 'Завершено'::text THEN pfi.quantity::numeric
            WHEN (i.invoice_type = ANY (ARRAY['Списання'::invoice_type_enum, 'Повернення'::invoice_type_enum])) AND i.status::text = 'Завершено'::text THEN (- pfi.quantity)::numeric
            ELSE 0::numeric
        END) AS invoice_quantity,
    max(pfi.product_price) AS product_price
   FROM products_for_invoice pfi
     JOIN invoice i ON i.invoice_id = pfi.invoice_id
     JOIN products p ON p.product_id = pfi.product_id
  GROUP BY pfi.product_id, p.product_name;
```

Дане представлення надає можливість переглянути продукти, що є в наявності, їх назву та кількість.

Тексти команд SQL-запитів, які використовуються для створення всіх таблиць БД наведені в додатку С.

Деякі допоміжні тригери та функції виконують необхідні завдання автоматизації процесів технічного процесу та слідкують за правилами цілістності ІС.

Серед них, наприклад, тригери, що слідкують за правильним відніманням кількості продуктів при продажі, чи перевірка на наявність необхідної кількості продуктів при додаванні їх до накладної.

Тексти команд SQL-запитів для створення допоміжних функцій та тригерів наведено у додатку D.

7 ЗАПИТИ ДО БАЗИ ДАНИХ ДЛЯ ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ

Для вирішення задач інформаційної системи «База даних продуктового магазину» необхідні різноманітні SQL-запити, функції, збережені процедури та тригери.

Усі таблиці, що використовуються для виконання задач користувачів подані у додатку С.

1) Для вирішення задач М1, М3, М5, М7, М8, М14, М16, М19, М22, М23, М24, К1, К2 виконується запит типу `INSERT INTO таблиця(список_стовпців) VALUES (список_стовпців)` з підстановкою відповідних імен таблиць та списку стовпців:

- для вирішення задачі М1 – ім'я таблиці `Categories`, список стовпців – `category_name;`
- для задачі М3 – ім'я таблиці `Products`, список стовпців – `product_name, category_id,unit, company;`
- для задачі М5 – ім'я таблиці `ProductsToOrder`, список стовпців – `product_id, quantity, order_date;`
- для задачі М7 – ім'я таблиці `ProductsForInvoice`, список стовпців –`invoice_id, product_id, quantity, product_price, date_of_manufacture,use_by_date;`
- для задачі М8 – ім'я таблиці `Invoice`, список стовпців – `employee_id, provider_id, createt_at, completed_at,invoice_type, status;`
- для задачі М14 – ім'я таблиці `Position`, список стовпців – `position_name;`
- для задачі М16 – ім'я таблиці `Employee`, список стовпців – `first_name, last_name, middle_name, phone_number, email, password, hire_date, dismissial_date, position_id;`

- для задачі М19 – ім’я таблиці `Providers`, список стовпців – `fist_name, last_name, company_name, phone_number, email;`
- для задачі М22 – ім’я таблиці `Receipt`, список стовпців – `receipt_create_date, receipt_date, employee_id, receipt_type;`
- для задачі М23 – ім’я таблиці `Receipt`, список стовпців – `receipt_create_date, receipt_date, employee_id, receipt_type;`
- для задачі М24 – ім’я таблиці `ProductsForReceipt`, список стовпців – `receipt_id, employee_id, produc_id, quantity, price;`
- для задачі К1 – ім’я таблиці `Receipt`, список стовпців – `receipt_create_date, receipt_date, employee_id;`
- для задачі К2 – ім’я таблиці `ProductsForReceipt`, список стовпців – `receipt_id, employee_id, produc_id, quantity, price;`

2) Для вирішення задач М10, М12, М17 виконується запит типу `UPDATE таблиця SET стовпець_1 = значення_1 […, стовпець_n = значення_n] [WHERE умова]` з підстановкою відповідних імен таблиць та списку необхідних стовпців.

- для вирішення задачі М10 – ім’я таблиці `ProductsForInvoice`, список стовпців – `date_of_manufacture, use_by_date` умова – співпадіння `products_for_invoice_id`;
- для задачі М12 – ім’я таблиці `Invoice`, список стовпців – `compledet_at, status,` умова – співпадіння `invoice_id`;
- для задачі М17 – ім’я таблиці `Employee`, список стовпців – `first_name, second_name, middle_name, phone_number, email, dismissal_date,` умова – співпадіння `employee_id;`

3) Для вирішення задач М2, М4, М6, М11, М13, М15, М18, М20 виконується запит типу `DELETE FROM таблиця WHERE id_таблиці = значення [додаткова умова]` з підстановкою відповідних імен таблиць

та стовпців.

- для вирішення задачі М2 – ім’я таблиці `Categories`, стовпець ідентифікатора – `category_id`;
- для задачі М4 – ім’я таблиці `Products`, стовпець ідентифікатора – `product_id`;
- для задачі М6 – ім’я таблиці `ProductsToOrder`, стовпець ідентифікатора – `products_to_order_id`;
- для задачі М11 – ім’я таблиці `ProductsForInvoice`, стовпець ідентифікатора – `products_for_invoice_id`;
- для задачі М13 – ім’я таблиці `Invoice`, стовпець ідентифікатора – `invoice_id`;
- для задачі М15 – ім’я таблиці `Position`, стовпець ідентифікатора – `position_id`;
- для задачі М18 – ім’я таблиці `Employee`, стовпець ідентифікатора – `employee_id`;
- для задачі М20 – ім’я таблиці `Provider`, стовпець ідентифікатора –`provider_id;`

4) Для вирішення задач М9, М21, виконується запит типу `SELECT список_стовпців FROM таблиця [WHERE умова]` з підстановкою відповідних імен таблиць, списку стовпців та умов, коли вони необхідні:

- для задачі М9 – ім’я таблиці `ProductsForInvoice`, список стовпців – `product_id,` за умовою збігу `invoice_id;`
- для задачі М21 – ім’я таблиці `ProductsForReceipt`, список стовпців – `product_id,` за умовою збігу `receipt_id;`

8 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНТЕРФЕЙСУ

Додаток побудовано за шаблоном MVC (Model-View-Controller), тому взаємодія між користувачем, сервером та базою даних відбувається наступним чином:

1. Графічний інтерфейс (View) реалізований за допомогою компонентів Next.js, які генерують сторінки у форматі SSR (Server-Side Rendering) або CSR (Client-Side Rendering). Користувач взаємодіє з додатком через веб-браузер, заповнюючи форми, переглядаючи списки даних, виконуючи пошук тощо.

2. Запити до сервера (Controller) ініціюються при взаємодії користувача з інтерфейсом. Для цього використовуються HTTP-запити (методи GET, POST, PUT, DELETE), які обробляються на серверній стороні за допомогою API-роутів Next.js або серверних функцій. Контролери приймають дані від користувача, здійснюють необхідну перевірку та формують запити до бази даних.

3. Робота з базою даних (Model) здійснюється через ORM-бібліотеку Prisma, яка надає типізований доступ до таблиць бази даних. Prisma автоматично генерує класи моделей для кожної таблиці на основі описаної схеми. При зверненні до бази даних Prisma формує SQL-запити та повертає результати у вигляді об’єктів моделей.

4. Серверна функція обробляє отримані з бази дані, у разі потреби проводить додаткову обробку, сортування чи фільтрацію, після чого передає ці дані до компонентів представлення (View), які динамічно формують HTML-код сторінки для користувача.

5. Після генерації сторінки або відповіді API, користувач отримує актуальні дані у зручному вигляді. Цикл взаємодії завершується до наступної дії користувача.

Звернення до бази даних здійснюється за допомогою Prisma Client — він створює з’єднання з базою та повертає об’єкти відповідних моделей. Кожна модель у Prisma (наприклад, Employee) представляє собою окрему таблицю у базі даних, а її поля відповідають стовпчикам цієї таблиці. Окрім того, Prisma

дозволяє виконувати фільтрацію, сортування, обмеження вибірки та обробку зв’язків між таблицями без написання SQL-запитів вручну.

Приклад класу моделі таблиці Employee у схемі Prisma наведено у лістингу:

```
model employee {
  employee_id Int @id @default(autoincrement()) @db.SmallInt
  first_name           String                @db.VarChar(100)
  last_name            String                @db.VarChar(100)
  middle_name          String                @db.VarChar(100)
  phone_number         String                @db.VarChar(20)
  email                String                @db.VarChar(100)
  password             String                @db.VarChar(100)
  hire_date            DateTime              @db.Date
  dismissal_date       DateTime?             @db.Date
  position_id          Int
  db_role              String?
  position              position                @relation(fields:
[position_id],   references:   [position_id],   onDelete:   SetNull,
onUpdate: NoAction, map: "fk_position")
  invoice              invoice[]
  products_for_receipt products_for_receipt[]
  receipt              receipt[]
}
```

Лістинг 8.1 – модель `Employee`

Приклад виведення всіх користувачів наведено у лістингу 8.2.

```
<Table>
              <TableHeader>
                <TableRow>
                  <TableHead className="w-[30px] text-
center">Номер</TableHead>
                  <TableHead>Прізвище</TableHead>
                  <TableHead>Ім'я</TableHead>
                  <TableHead>По батькові</TableHead>
                  <TableHead>Телефон</TableHead>
                  <TableHead>Пошта</TableHead>
                  <TableHead>Дата початку</TableHead>
                  <TableHead>Дата завершення</TableHead>
                  <TableHead>Посада</TableHead>
                  <TableHead className="text-
center">Редагувати</TableHead>
                  <TableHead className="text-
right">Видалити</TableHead>
                </TableRow>
              </TableHeader>
              <TableBody>
                {isLoading ? (
                  <>
                    {[...Array(5)].map((_, index) => (
```

```
                      <TableRow key={index}>
                        {Array.from({ length: 11 }).map((_,
cellIndex) => (
                          <TableCell
                            key={cellIndex}
                            className={
                              cellIndex === 0 ? "font-medium" :
"text-right"
                            }
                          >
                            <Skeleton className="h-8" />
                          </TableCell>
                        ))}
                      </TableRow>
                    ))}
                  </>
                ) : (
                  <>
                    {employee.length === 0 ? (
                      <TableRow></TableRow>
                    ) : (
                      employee.map((emp) => (
                        <TableRow
                          key={emp.employee_id}
                          className="transition-transform"
                        >
                          <TableCell className="font-medium text-
center">
                            {emp.employee_id}
                          </TableCell>
                          <TableCell>{emp.last_name}</TableCell>
                          <TableCell>{emp.first_name}</TableCell>
                          <TableCell>{emp.middle_name}</TableCell>
                          <TableCell>{emp.phone_number}</TableCell
>
                          <TableCell>{emp.email}</TableCell>
                          <TableCell>
                            {String(emp.hire_date).slice(0, 10)}
                          </TableCell>
                          <TableCell>
                            {String(emp.dismissal_date).slice(0,
10)}
                          </TableCell>
                          <TableCell>{emp.position_name}</TableCel
l>
                          <TableCell className="text-center">
                            <Button
                              onClick={() =>
handleEditEmployee(emp)}
                              className="p-3 bg-primary text-white
hover:bg-primary/90"
                            >
                              Редагувати
                            </Button>
                          </TableCell>
                          <TableCell className="text-right">
                            <Button
```

```
                                onClick={() =>
handleDeleteUser(emp.employee_id)}
                                className="p-3"
                              >
                                <Trash2Icon size={16} />
                              </Button>
                            </TableCell>
                          </TableRow>
                        ))
                      )}
                    </>
                  )}
                </TableBody>
              </Table>
```

Лістинг 8.2 – Реалізація виведення користувачів

Створення нового користувача виконується шляхом запису даних до форми, та подальшого передавання інформації до бази даних. Реалізація форми наведена у лістингу 8.3.

```
<div className="mt-5">
            <Title
              text="Форма для створення користувача"
              size="lg"
              className="mb-4"
            />
            <form
              onSubmit={async (e) => {
                e.preventDefault();

                // Перевірка паролів
                if (
                  (password1 || confirmPassword1) &&
                  password1 !== confirmPassword1
                ) {
                  toast.error(
                    "Паролі не співпадають або одне з полів не
заповнене!"
                  );
                  return;
                }

                // Знаходимо position_name по position_id
                const selectedPosition = position.find(
                  (pos) =>
                    pos.position_id.toString() ===
newEmployee.position_id
                );

                const position_name =
selectedPosition?.position_name || "";

                // Формування даних для відправки
                const payload = {
```

```
                ...newEmployee,
                hire_date: newEmployee.hire_date
                  ? new Date(newEmployee.hire_date)
                  : null,
                password: password1 || undefined,
                position_name, // додаємо сюди назву посади
              };

              const response = await fetch("/api/employee", {
                method: "POST",
                headers: {
                  "Content-Type": "application/json",
                },
                body: JSON.stringify(payload),
              });

              if (response.ok) {
                toast.success("Користувач успішео створений", {
icon: "✅" });
                setNewEmployee({
                  last_name: "",
                  first_name: "",
                  middle_name: "",
                  phone_number: "",
                  email: "",
                  hire_date: "",
                  position_id: "",
                  position_name: "",
                });
                setPassword1("");
                setConfirmPassword1("");
                fetchUsers();
              } else {
                toast.error("Помилка при створенні
користувача");
              }
            }}
            className="flex flex-col items-end space-y-4"
          >
            <input
              className="w-full border px-2 py-1 rounded-sm
border-primary"
              type="text"
              value={newEmployee.last_name}
              onChange={(e) =>
                setNewEmployee({ ...newEmployee, last_name:
e.target.value })
              }
              placeholder="Прізвище"
            />
            <input
              className="w-full border px-2 py-1 rounded-sm
border-primary"
              type="text"
              value={newEmployee.first_name}
              onChange={(e) =>
```

```
                  setNewEmployee({ ...newEmployee, first_name:
e.target.value })
                }
                placeholder="Ім'я"
              />
              <input
                className="w-full border px-2 py-1 rounded-sm
border-primary"
                type="text"
                value={newEmployee.middle_name}
                onChange={(e) =>
                  setNewEmployee({
                    ...newEmployee,
                    middle_name: e.target.value,
                  })
                }
                placeholder="По батькові"
              />
              <input
                className="w-full border px-2 py-1 rounded-sm
border-primary"
                type="text"
                value={newEmployee.phone_number}
                onChange={(e) =>
                  setNewEmployee({
                    ...newEmployee,
                    phone_number: e.target.value,
                  })
                }
                placeholder="Телефон"
              />
              <input
                className="w-full border px-2 py-1 rounded-sm
border-primary"
                type="email"
                value={newEmployee.email}
                onChange={(e) =>
                  setNewEmployee({ ...newEmployee, email:
e.target.value })
                }
                placeholder="Email"
              />
              <input
                className="w-full border px-2 py-1 rounded-sm
border-primary"
                type="password"
                value={password1}
                onChange={(e) => setPassword1(e.target.value)}
                placeholder="Пароль"
              />
              <input
                className="w-full border px-2 py-1 rounded-sm
border-primary"
                type="password"
                value={confirmPassword1}
                onChange={(e) =>
setConfirmPassword1(e.target.value)}
```

```
                placeholder="Підтвердження паролю"
              />
              <input
                className="w-full border px-2 py-1 rounded-sm
border-primary"
                type="date"
                value={newEmployee.hire_date}
                onChange={(e) =>
                  setNewEmployee({ ...newEmployee, hire_date:
e.target.value })
                }
              />

              <select
                className="w-full border px-2 py-1 rounded-sm
border-primary"
                value={newEmployee.position_id || ""}
                onChange={(e) =>
                  setNewEmployee({
                    ...newEmployee,
                    position_id: e.target.value, // конвертуємо в
число
                  })
                }
              >
                <option value="" disabled>
                  Оберіть посаду
                </option>
                {position.map((pos) => (
                  <option key={pos.position_id}
value={pos.position_id}>
                    {pos.position_name}
                  </option>
                ))}
              </select>

              <div className="flex justify-end gap-2 rounded-sm
border-primary">
                <button
                  type="submit"
                  className="px-4 py-2 bg-primary text-white
rounded-sm"
                >
                  Створити
                </button>
              </div>
            </form>
          </div>
```

Лістинг 8.3 – Реалізація форми для додавання нового користувача

При підтвердженні відправки форми виконується код для відправки даних в базу даних. Реалізація даного коду представлена в лістингу 8.4.

```
export async function POST(req: Request) {
  try {
    const {
      first_name,
      last_name,
      middle_name,
      phone_number,
      password,
      email,
      hire_date,
      position_id,
      position_name, // наприклад, "Адміністратор" або "Касир"
    } = await req.json();

    // Визначаємо роль за назвою посади
    const role =
      position_name === "Адміністратор" ? "admin_role" :
"seller_role";

    // 🔐 Використовуємо суперкористувача для створення нового
працівника
    await prismaDefault.$executeRawUnsafe(
      `
      INSERT INTO employee
        (first_name, last_name, middle_name, phone_number, email,
password, hire_date, position_id, db_role)
      VALUES
        ($1, $2, $3, $4, $5, $6, $7, $8, $9)
      `,
      first_name,
      last_name,
      middle_name,
      phone_number,
      email,
      password,
      new Date(hire_date),
      Number(position_id),
      role
    );

    // 🖥️ Створення PostgreSQL-юзера
    await prismaDefault.$executeRawUnsafe(`
    DO $$
    DECLARE
      user_email text := '${email}';
      user_password text := '${password}';
    BEGIN
      IF NOT EXISTS (SELECT FROM pg_catalog.pg_roles WHERE rolname
= user_email) THEN
        EXECUTE format('CREATE ROLE %I LOGIN PASSWORD %L;',
user_email, user_password);
      END IF;
    END
    $$;
  `);
```

```
    // 🎓 Призначення рольової групи (admin_role або seller_role)
    await prismaDefault.$executeRawUnsafe(`GRANT ${role} TO
"${email}";`);

    return NextResponse.json({
      message: "Працівника та роль створено успішно",
    });
  } catch (error) {
    console.error("POST /employee error:", error);

    if (error instanceof Prisma.PrismaClientKnownRequestError) {
      if (error.code === "P2010") {
        const message = error.message || "";
        if (
          message.includes("42501") ||
          (error.meta &&
JSON.stringify(error.meta).includes("42501"))
        ) {
          return NextResponse.json({ message: "Forbidden" }, {
status: 403 });
        }
      }
    }

    return NextResponse.json(
      { message: "Помилка при створенні працівника" },
      { status: 500 }
    );
  }
}
```

Лістинг 8.4 – Додавання користувача за допомогою Prisma.

9 БЕЗПЕКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

Безпека на рівні БД забезпечується шляхом створення ролей і наділення їх відповідними привілеями. В ІС «База даних продуктового магазину » реалізовано три ролі:

Адміністратор, Продавець і Гість. У додатку G наведені привілеї ролей на таблиці БД. При цьому використані такі позначення: C (Create) – створення (додавання) даних, R (Read) – читання даних, U (Update) – оновлення даних, D (Delete) – видалення даних, E (Execute) – виконання процедури (функції).

Створення ролей і наділення їх привілеями відповідно до наведеної вище таблиці здійснюється за допомогою наступних SQL-запитів (лістинги 9.1-9.3):

```
CREATE ROLE    guest_user WITH LOGIN PASSWORD ‘guest_user’;
GRANT SELECT ON employee, position  TO guest_user;
```

Лістинг 9.1 – Створення і наділення привілеями ролі Гість

```
CREATE ROLE seller_role WITH LOGIN PASSWORD 'seller_role123' NOSUPERUSER NOCREATEDB NOCREATEROLE;
GRANT INSERT, UPDATE ON TABLE receipt TO seller_role;
GRANT SELECT ON TABLE products_in_stock TO seller_user;
GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE products_for_receipt TO seller_role;
```

Лістинг 9.2 – Створення і наділення привілеями ролі Співробітник

```
CREATE ROLE admin_role WITH LOGIN PASSWORD ‘123456’ NOSUPERUSER NOCREATEDB NOCREATEROLE;
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO admin_role;
GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA public TO admin_role;
```

Лістинг 9.3 – Створення і наділення привілеями ролі Адміністратор

Також створена службова роль guest_user, від імені якої виконується підключення

до БД для виконання наступних дій:

- перевірки існування в таблиці `employee` облікового запису користувача, що проходить процедуру автентифікації в застосунку;

- визначення того, яка роль з перерахованих вище відповідає цьому

обліковому запису.

Роль guest_user не володіє ніякими привілеями, крім можливості здійснювати вибірку даних з таблиці employee.

10 ІНСТРУКЦІЯ КОРИСТУВАЧА

10.1 Авторизація з першого погляду

Кожний відвідувач сайту при вході потрапляє на форму авторизації (рис. 10.1.1) у яку користувач може ввести дані свого облікового запису – логін та пароль, якщо він їх звичайно має. Потрібно підмітити, що SQLad– корпоративна система та використовується виключно співробітниками компанії. Якщо дані облікового запису введені невірно, користувач отримує повідомлення на формі (рис. 10.1.2).

Рисунок 10.1.1 – Форма авторизації користувача

Рисунок 10.1.2 – Форма авторизації користувача з помилкою входу

10.2 Інтерфейс адміністратора

Авторизуючись під груповою роллю адміністратора, користувачу доступний певний набір сторінок (див. мал. 5.1). Навігаційна панель кадровика зображена на рис. 10.2.1.

Рисунок 10.2.1 – Навігаційна панель адміністратора

Після авторизації адміністратор може вибрати декілька опцій.

Перегляд сторінки категорій.

Рисунок 10.2.2 – Сторінка для керування категоріями

Адміністратор має можливість переглядати список всіх категорій, шукати необхідні категорії за їх назвою, видаляти не потрібні категорії. При додаванні категорії необхідно вказати назву категорії та натиснути кнопку “Додати”, після додавання категорія буде додана та з’явиться повідомлення про успішне додавання.

Перегляд сторінки продуктів.

Рисунок 10.2.3 – Сторінка для керування продуктами

Ми маємо дві додаткові опції, це – “Список продуктів” та “Продукти для замовлення”. Кожну з них ми розглянемо детально. На сторінці “Список продуктів” ми можемо спостерігати список всіх продуктів які зареєстровані в базі даних, їх опис. Кожен продукт має дві кнопки: “Додати” та “Видалити”.

При натисканні на Додати відкривається модальне вікно в якому потрібно вказати кількість та підтвердити. Після підтвердження продукт додається в список продуктів для замовлення, які можна переглянути на рисунку 10.2.4.

Також ми можемо додавати продукти, де потрібно вказати назву продукту, обрати категорію з випадаючого списку, вказати одиниці вимірювання продукту, компанію та умови зберігання. Після додавання продукту в виведеться даний продукт та буде виведене повідомлення про успішне додавання.

Продукти для замовлення.

Рисунок 10.2.4 – Сторінка продуктів для замовлення

На даній сторінці ми маємо можливість пошуку продуктів за назвою, переглядати інформацію, видаляти продукти з даного списку, також є кнопка “Додати”. При натисканні на кнопку в нас з’являється модальне вікно, рис 10.2.5.

Рисунок 10.2.5 – Модальне вікно

Ми використовуємо дане вікно для додавання продуктів, які необхідно замовити, до накладних. В нас відображається назва продукту, надаються накладні які зараз активні, вказуємо кількість та ціну, при спробі додавання більше продуктів чим вказано в таблиці для замовлення, з’являється помилка, рис 10.2.6.

Рисунок 10.2.6 – Помилка при додаванні

Накладні.

Рисунок 10.2.7 – Сторінка з накладними

На даній сторінці також є додаткова опція, а саме продукти для накладних. На сторінці “Накладні” ми маємо можливість пошуку накладних за вказаним параметром на вибір, рис 10.2.8.

Рисунок 10.2.8 – Список можливих параметрів

На сторінці відображається вся інформація про накладні, з можливістю перегляду вмісту накладної, видаленням та редагуванням. Функція редагування та перегляду буде продемонстровано на рис 10.2.9 та 10.2.10 відповідно.

Рисунок 10.2.9 – Перегляд вмісту накладної

Як ми бачимо з рисунку продукти також мають функціонал, за допомогою кнопки “Редагувати продукт” ми можемо, за допомогою модального вікна, змінити дату виробництва та дату вжити до, а також ціну, дивитися рис. 10.2.11.

Рисунок 10.2.11 – Редагування продукту в накладній

Рисунок 10.2.10 – Редагування накладної

Тут ми вказуємо дату отримання накладної, а також змінюємо її статус.

Сторінка “Продукти для накладної”.

Рисунок 10.2.12 – Перегляд сторінки з продуктами з накладних

Маємо можливість пошуку за назвою продукту та за типом накладної до якої належить продукт. Реалізовано за допомогою випадаючого вікна та представлено на рисунках 10.2.13 та 10.2.14.

Рисунок 10.2.13 – Пошук за типом накладної

Рисунок 10.2.14 – Параметр за яким буде відбуватися пошук

Також є кнопка “Додати”. За допомогою неї ми можемо додати продукт до накладної яка має тип “Списання” або “Повернення”.

Рисунок 10.2.15 – Повернення продукту

Сторінка “Персонал” має 3 додаткові сторінки в собі.

Рисунок 10.2.16 – Перегляд сторінки персоналу

На сторінці “Посади” ми переглядаємо існуючі посади, можемо їх видалити або створити нову у відповідному вікні. Також ми маємо можливість пошуку за назвою посади.

На сторінці “Працівники” ми переглядаємо персонал.

Рисунок 10.2.17 – Перегляд сторінки з працівниками

Відображається вся інформація, є можливість пошуку за одним із списку можливих параметрів на вибір, рис 10.2.18.

Рисунок 10.2.18 – Список параметрів для пошуку

При натисканні на кнопку “Редагувати” відкривається модальне вікно з формою редагування працівника, рис 10.2.19.

Рисунок 10.2.19 – Форма для редагування працівника

Також маємо форму для створення працівника, на ній зображені необхідні поля для заповнення, рис 10.2.20

Рисунок 10.2.20 – Форма для створення працівника

На сторінці “Постачальник” маємо аналогічний функціонал який представлений на рисунках 10.2.21, 10.2.22 та 10.2.23.

Рисунок 10.2.21 – Перегляд сторінки постачальників

Рисунок 10.2.22 – Критерії пошуку серед постачальників

Рисунок 10.2.23 – Форма для додавання нового постачальника

Сторінка “Чеки” представлена на рисунку 10.2.24. Вона містить 3 додаткові сторінки.

Рисунок 10.2.24 – Перегляд сторінки чеків

Дана сторінка відображає всю інформацію про чеки, також має 2 кнопки, а саме : “Переглянути” та “Повернути”. При натисканні “Переглянути” в нас відображається вміст даного чеку, рис 10.2.25. При натисканні “Повернути”, в нас відкривається чек з можливістю повернути один або декілька продуктів, також ми можемо вказати кількість продуктів які повертаємо, рис 10.2.26.

Рисунок 10.2.25 – Перегляд вмісту чека

Рисунок 10.2.26 – Повернення всього чеку

Також реалізовано пошук чеків за вказаними критеріями, рис 10.2.27.

Рисунок 10.2.27 – Критерії пошуку чеку

Сторінка “Продукти для чеків” містить в собі всі продукти які належать всім чекам, рис 10.2.28. Також на сторінці реалізований пошук продаж за кожним працівником, що надає зможу мати точні звіти продаж та повернення за кожним працівником, рис 10.2.29. Можемо вказати дату від та до, за яким часом робити пошук продуктів, можемо обрати працівника для якого будемо знаходити список, також можна вибрати тип чека: “Продаж” або “Повернення”.

Також реалізований пошук продуктів за двома критеріями на вибір, рис 10.2.30.

Рисунок 10.2.28 – Перегляд сторінки з чеками

Рисунок 10.2.29 – Пошук продуктів для кожного працівника

Рисунок 10.2.29 – Критерії пошуку продуктів

Сторінка “Створити чек”, на ній ми маємо натиснути кнопку “Створити чек” для початку роботи, рис 10.2.30. Після цього нам відкривається доступ до створення чека, рис 10.2.31. Нам потрібно вибрати продукт з випадаючого списку, за необхідності ми можемо скористатися пошуком, вказати кількість продукту. При натисканні “Додати продукт”, обраний продукт додається до даного чека, та стає доступною кнопка “Продати”, рис 10.2.32. Після продажу все повторюється.

Рисунок 10.2.30 – Початкова сторінка створення чека

Рисунок 10.2.31 – Функціональна частина створення чека

Рисунок 10.2.32 – Поява кнопки “Продати”

На сторінці “Статистика” в нас є два вікна: “Продажі” та “Топ продаж”. На першій сторінці в нас відслідковується середній дохід за тиждень, є дата та порівняння з попереднім тижнем, рис 10.2.33. На другій сторінці ми отримує

найпопулярніші продукти по продажам для кожної категорії, рис 10.2.34.

Рисунок 10.2.33 – Середньотижневий дохід

Рисунок 10.2.34 – Найпопулярніші продукти

10.3 Інтерфейс продавця

Взаємодія продавця з системою обмежується лише створенням чеку, та має таку початкову сторінку, як на рисунку 10.3.1. Весь функціонал створення чеку був описаний вище з візуальними прикладами, рисунок 10.2.30 – 10.2.32. В ІС функціонал продавця відповідає частині функціоналу адміністратора.

Рисунок 10.3.1 – Початкова сторінка продавця

ВИСНОВОК

У ході виконання даного проєкту було розроблено інформаційну систему для автоматизації облікових процесів діяльності продуктового магазину. В процесі роботи спроєктовано та реалізовано реляційну базу даних, що забезпечує збереження, обробку та ефективне управління інформацією про товари, категорії, постачальників, працівників, накладні, чеки, повернення товарів тощо. Як серверна частина системи використано сучасний технологічний стек, що включає фреймворк Next.js, систему управління базами даних PostgreSQL та ORM Prisma для зручної роботи з базою даних. Було реалізовано функціонал взаємодії користувачів із системою залежно від їх ролей, що дозволяє обмежувати доступ до функціональних можливостей відповідно до рівня прав доступу.

Розроблена система дозволяє виконувати повний спектр операцій: додавання, редагування та видалення даних, формування чеків та повернень, а також здійснювати пошук та фільтрацію інформації за різними параметрами. Система забезпечує інтеграцію фронтенд та бекенд складових через API-запити, що гарантує оперативність обробки даних та зручність користування.

Отримані результати можуть бути застосовані в реальних торгових підприємствах для автоматизації облікових процесів, покращення контролю товарних залишків, оптимізації роботи персоналу та покращення якості обслуговування клієнтів. Розроблена інформаційна система може слугувати основою для подальшого розширення функціоналу, зокрема впровадження аналітичних модулів, систем прогнозування попиту, інтеграції з фінансовими системами та бухгалтерськими комплексами.

Таким чином, поставлені в роботі завдання виконані в повному обсязі, а розроблена система відповідає вимогам до сучасних інформаційних систем управління для торгових підприємств.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1) Коротко про типи архітектур програмного забезпечення: Трирівнева архітектура [Електронний ресурс] – Режим доступу: https://spark.ru/startup/1cloud/blog/42506/kratko-o-tipah-arhitektur-programmnogo-obespecheniya-i-perehode-na-mikroservisi
2) What is Entity Framework? [Електронний ресурс] – Режим доступу: https://www.entityframeworktutorial.net/what-is-entityframework.aspx
3) Малахов Є.В. Основи проектування БД: Конспект лекцій. [Електронний ресурс] – Режим доступу: http://library.opu.ua.
4) Adam Freeman, Pro ASP.NET Core 3, 8th Edition – Published by Apress, 2020. – 1086 p.
5) Claims-based authorization in ASP.NET Core [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/aspnet/core/security/authorization/claims?view=aspnetcore-6.0

ДОДАТОК А

Задачі користувачів ІС «База даних продуктового магазину**»**

Таблиця А.1 – Список задач користувачів ІС **«**База даних продуктового магазину**»**

Номер	Задача	Вхідні дані	Вихідні дані
Адміністратор			
М1	Додати категорію	Назва категорії	Нова створена категорія
М2	Видалити категорію	Номер категорії	Видалена категорія
М3	Додати продукт	Назва продукту, категорія, одиниці виміру, компанія, температура зберігання	Створено новий продукт
М4	Видалити продукт	Номер продукту	Видалений продукт
М5	Додавання продукту в таблицю для замовлення	Кількість	Доданий продукт з вказаною кількістю
М6	Видалити продукт з таблиці продуктів для замовлення	Номер продукту	Видалений продукт
М7	Додавання продукту до накладної	Номер продукту, номер накладної, кількість, ціна	Доданий продукт до накладної з власною кількістю та ціною.
М8	Створення накладної	Тип накладної, номер постачальника	Створення накладної з вказаним типом та постачальником
М9	Перегляд вмісту накладної	Номер накладної	Відображення продуктів які належать даній накладній.
М10	Редагування продукту в накладній	Номер продукту, дата виготовлення, дата вжити до, ціна	Оновлені дані для вказаного продукту, дата виготовлення, дата вжити до, ціна
М11	Видалення продукту з накладної	Номер продукту	Видалення вказаного продукту з накладної

Продовження таблиці А.1

Номер	Задача	Вхідні дані	Вихідні дані
М12	Редагування накладної	Номер накладної, дата завершення дії накладної, статус	Оновлена інформація в певній накладній: Дата завершення, статус.
М13	Видалення накладної	Номер накладної	Видалена накладна з вказаним номером
М14	Додати посаду	Назва посади	Додана нова посада
М15	Видалити посаду	Номер посади	Видалена вказана посада
М16	Створення працівника	Прізвище, ім’я, по батькові, номер телефону, пошта, пароль, дата початку роботи, посада.	Створення нового працівника з вказаними параметрами.
М17	Редагування працівника	Нові один або декілька параметрів на вибір: прізвище, ім’я, по батькові, номер телефону, пошта, пароль, дата початку роботи, дата звільнення.	Оновлені дані працівника.
М18	Видалити працівника	Номер працівника	Видалений працівник за номером.
М19	Створення нового постачальника	Ім’я, прізвище, назва компанії, номер телефону, пошта.	Створений постачальник з вказаними параметрами.
М20	Видалення постачальника	Номер постачальника	Видалений постачальник за вказаним номером.
М21	Переглянути вміст чека	Номер чека	Список продуктів які належать даному чеку.
М22	Повернути чек	Або номер чеку для повернення всього чеку, або номер продукту для	Створений новий чек з типом “Повернення” з

		чеку з вказаною кількістю для повернення обраного числа продуктів.	призначеними для нього продуктами.
M23	Створити чек	Номер працівника, дата	Створений новий чек з вказаним працівником, датою та за замовчуванням з типом “Продаж”
M24	Додавання продукту до чеку	Номер продукту, кількість	Додавання нового продукту до чеку з параметрами: Назва продукту, кількість, ціна, продавець.
Продавець			
K1	Створити чек	Номер працівника, дата	Створений новий чек з вказаним працівником, датою та за замовчуванням з типом “Продаж”
K2	Додавання продукту до чеку	Номер продукту, кількість	Додавання нового продукту до чеку з параметрами: Назва продукту, кількість, ціна, продавець.

ДОДАТОК В

Опис сутностей предметної області

Таблиця В.1 – Опис сутностей предметної області «База даних продуктового магазину»

Ім’я атрибута	Призначення атрибута	Обмеження
Categories(Категорії)		
category_id	Ідентифікатор котегорії	Первинний ключ
category_name	Назва котегорії	Не порожнє, унікальне
Position(Посади)		
position_id	Номер посади	Первинний ключ
position_name	Назва посади	Не порожнє, унікальне
Employee(Працівник)		
employee_id	Ідентифікатор	Первинний ключ
first_name	Ім’я працівника	Не порожнє
last_name	Прізвище працівника	Не порожнє
middle_name	По батькові працівника	Не порожнє
phone_number	Номер телефону	не порожнє, унікальне

email	Пошта працівника	Не порожнє, унікальне
password	Пароль працівника	Не порожнє
hire_date	Дата взяття на роботу	Не порожнє
dismissal_date	Дата звільнення	Порожнє
position_id	Ідентифікатор посади працівника	Не порожнє, зовнішній ключ для зв’язку з сутністю Position (position_id)
Provider(Постачальник)		
provider_id	ідентифікатор постачальника	Первинний ключ
first_name	Ім’я постачальника	Не порожнє
last_name	Прізвище постачальника	Не порожнє
company_name	Назва компанії	Не порожнє
phone_number	Номер телефону постачальника	Не порожнє, унікальне
email	Пошта постачальника	Не порожнє, унікальне
Invoice(Накладна)		
invoice_id	Ідентифікатор накладної	Первинний ключ
employee_id	Ідентифікатор працівника	Не порожнє, зовнішній ключ для зв’язку з сутністю Employee(employee_id)
provider_id	Ідентифікатор постачальника	Може бути порожнє, зовнішній ключ для зв’язку з сутністю Provider(provider_id), залежно від типу накладної дане поле може бути пустим, при умові, що тип = Повернення

created_at	Дата створення накладної	Не порожнє, > поточну дату
completed_at	Дата завершення накладної	Не порожнє, має бути не менше за created_at
invoice_type	Тип накладної	Не порожнє, один з enum: Повернення, Списання, Отримання
status	Статус	Не порожнє
Products(Продукти)		
product_id	Ідентифікатор продукту	Первинний ключ
product_name	Назва продукту	Не порожнє, унікальне
category_id	Назва категорії	Не порожнє, зовнішній ключ для зв’язку з сутністю Categories(category_id)
unit	Одиниці вимірювання	Не порожнє
company	Назва компанії	Не порожнє
ProductsForInvoice(Продукти для накладних)		
products_for_invoice_id	Ідентифікатор продукту для накладної	Первинний ключ
invoice_id	Ідентифікатор накладної	Не порожнє, зовнішній ключ для зв'язку з сутністю Invoice (invoice_id)
product_id	Ідентифікатор продукту	Не порожнє, зовнішній ключ для зв’язку з сутністю Products (product_id)
quantity	Кількість	Не порожнє

product_price	Ціна продукту	Не порожнє, значення > 0
date_of_manufacture	Дата виготовлення продукту	Не порожнє
use_by_date	Дата вжити до	Не порожнє
Receipt(Чек)		
receipt_id	Ідентифікатор чеку	Первинний ключ
receipt_create_date	Дата створення чека	Не порожнє
receipt_type	Тип чека	Не порожнє
employee_id	Ідентифікатор працівника	Не порожнє, зовнішній ключ для зв’язку з сутністю Employee (employee_id)
ProductsForReceipt(Продукти для чека)		
products_for_receipt_id	Ідентифікатор продукта для чека	Первинний ключ
receipt_id	Ідентифікатор чека	Не порожнє, зовнішній ключ для зв’язку з сутністю Receipt (receipt_id)
employee_id	Ідентифікатор працівника	Не порожнє, зовнішній ключ для зв’язку з сутністю Employee (employee_id)
product_id	Ідентифікатор продукту	Не порожнє, зовнішній ключ для зв’язку з сутністю Products (product_id)
quantity	Кількість товарів	Не порожнє, >0
price	Ціна товару	Не порожнє, >0
ProductsToOrder(Продукти для замовлення)		

products_to_order_id	Ідентифікатор продукта для замовлення	Первинний ключ
product_id	Ідентифікатор продукту	Не порожнє, зовнішній ключ для зв’язку з сутністю Products (product_id)
quantity	Кількість продуктів	Не порожнє, > 0
order_date	Дата замовлення	Не порожнє, за замовчуванням CURRENT_DATE

ДОДАТОК С

Запити на створення доменів та таблиць бази даних

```
Enum:
CREATE TYPE invoice_type_enum AS ENUM ('Отримання',
'Повернення', 'Списання');
```

Таблиці:

```
CREATE TABLE categories (
    categories_id smallserial PRIMARY KEY,
    category_name VARCHAR(255) NOT NULL
);


CREATE TABLE products (
    products_id smallserial PRIMARY KEY,
    products_name VARCHAR(255) NOT NULL,
    category_id INTEGER NOT NULL,
    unit VARCHAR(50) NOT NULL,
    company VARCHAR(100) NOT NULL,
    storage_temperature text,

    CONSTRAINT fk_category
        FOREIGN KEY (category_id)
        REFERENCES categories(categories_id)
        ON DELETE CASCADE
);


CREATE TABLE products_to_order (
    products_to_order_id smallserial PRIMARY KEY,
    product_id INTEGER NOT NULL,
    quantity INTEGER NOT NULL,
    order_date DATE DEFAULT CURRENT_DATE NOT NULL ,
    CONSTRAINT fk_product
        FOREIGN KEY (product_id)
        REFERENCES products(products_id)
        ON DELETE CASCADE
);


CREATE TABLE position (
    position_id smallserial PRIMARY KEY,
    position_name VARCHAR(100) NOT NULL
);



CREATE TABLE employee (
    employee_id smallserial PRIMARY KEY,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    middle_name VARCHAR(100) NOT NULL,
    phone_number VARCHAR(20) NOT NULL,
```

```
        email VARCHAR(100) NOT NULL,
        password VARCHAR(100) NOT NULL,
        hire_date DATE NOT NULL,
        dismissal_date DATE,
        position_id INTEGER NOT NULL,
        db_role TEXT,

        CONSTRAINT fk_position
            FOREIGN KEY (position_id)
            REFERENCES position(position_id)
            ON DELETE SET NULL
    );


    CREATE TABLE provider (
        provider_id smallserial PRIMARY KEY,
        first_name VARCHAR(100) NOT NULL,
        last_name VARCHAR(100) NOT NULL,
        company_name VARCHAR(150) NOT NULL,
        phone_number VARCHAR(20) NOT NULL,
        email VARCHAR(100) NOT NULL
    );


    CREATE TABLE invoice (
        invoice_id SMALLSERIAL PRIMARY KEY,
        employee_id INT NOT NULL,
        provider_id INT,
        created_at DATE NOT NULL,
        completed_at DATE,
        status VARCHAR(50) NOT NULL,
        invoice_type invoice_type_enum NOT NULL,

        -- Зовнішній ключ на працівника
        CONSTRAINT fk_employee
            FOREIGN KEY (employee_id) REFERENCES
employee(employee_id),

        -- Зовнішній ключ на постачальника
        CONSTRAINT fk_provider
            FOREIGN KEY (provider_id) REFERENCES
provider(provider_id),

        -- CHECK: provider_id обов'язковий, крім типу 'Списано'
        CONSTRAINT check_provider_condition
            CHECK (
                (invoice_type = 'Списано' AND provider_id IS
NULL)
                OR
                (invoice_type IN ('Отримано', 'Повернено') AND
provider_id IS NOT NULL)
            )
    );


    CREATE TABLE products_for_invoice (
        products_for_invoice_id smallserial PRIMARY KEY,
```

```
        invoice_id INT NOT NULL,
        product_id INT NOT NULL,
        quantity INT NOT NULL CHECK (quantity > 0),
        product_price DECIMAL(10, 2) NOT NULL CHECK
(product_price > 0),
        date_of_manufacture DATE,
        use_by_date DATE,
        FOREIGN KEY (invoice_id) REFERENCES invoice(invoice_id)
ON DELETE CASCADE,
        FOREIGN KEY (product_id) REFERENCES products(products_id)
    );


    CREATE TABLE receipt (
        receipt_id smallserial PRIMARY KEY,
        receipt_create_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        receipt_type VARCHAR(50) NOT NULL,
        employee_id INTEGER NOT NULL,
        FOREIGN KEY (employee_id)
            REFERENCES employee(employee_id)
            ON DELETE CASCADE
    );


    CREATE TABLE products_for_receipt (
        products_for_receipt_id smallserial PRIMARY KEY,
        receipt_id INT NOT NULL,
        employee_id INT NOT NULL,
        product_id INT NOT NULL,
        quantity numeric(10,2) NOT NULL CHECK (quantity > 0),
        price numeric(10,2) NOT NULL CHECK (price > 0),
        FOREIGN KEY (receipt_id) REFERENCES receipt(receipt_id)
ON DELETE CASCADE,
        FOREIGN KEY (employee_id) REFERENCES
employee(employee_id)
    );
```

ДОДАТОК D

Запити на створення додаткових функцій і тригерів збереження цілістності ІС

```
-- тригер який використовується під час додавання продукту в
-- таблицю products_for_receipt, перевіряє продукт на наявну
-- кількість та виконує логіку додавання нового запису якщо
-- такого не існує або оновлює кількість вже в інсуючому запису
CREATE OR REPLACE FUNCTION public.upsert_product_for_receipt()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
DECLARE
  existing_id INTEGER;
  existing_quantity INTEGER := 0;
  available_quantity INTEGER := 0;
  total_quantity INTEGER;
BEGIN
  -- Отримуємо кількість, яка вже є в чеку
  SELECT products_for_receipt_id, quantity INTO existing_id,
existing_quantity
  FROM products_for_receipt
  WHERE receipt_id = NEW.receipt_id AND product_id =
NEW.product_id;

  -- Отримуємо доступну кількість з представлення
  SELECT invoice_quantity INTO available_quantity
  FROM current_product_stock
  WHERE product_id = NEW.product_id;

  IF available_quantity IS NULL THEN
    RAISE EXCEPTION 'Продукт з id % відсутній на складі',
NEW.product_id;
  END IF;

  -- Обчислюємо загальну кількість, яка буде після додавання
  total_quantity := existing_quantity + NEW.quantity;

  -- Перевірка: чи не перевищуємо доступну кількість
  IF total_quantity > available_quantity THEN
    RAISE EXCEPTION
      'Недостатньо товару: є %, вже додано до чека %, намагаєтесь
додати %, що дасть загальну суму %, але максимум %',
      available_quantity, existing_quantity, NEW.quantity,
total_quantity, available_quantity;
  END IF;
```

```
   -- Якщо запис існує — оновлюємо
   IF existing_id IS NOT NULL THEN
     UPDATE products_for_receipt
     SET quantity = total_quantity
     WHERE products_for_receipt_id = existing_id;

     RETURN NULL; -- блокуємо вставку
   END IF;

   -- Якщо запису немає — вставляємо новий
   RETURN NEW;
 END;

 -- виконується під час додавання продукту до таблиці
 -- products_to_order, перевіряє чи існує запис з таким
 -- id продукту та оновлює його кількість, якщо існує, в іншому
 -- випадку створюється новий запис.
 CREATE OR REPLACE TRIGGER trg_upsert_product_for_receipt
     BEFORE INSERT
     ON public.products_for_receipt
     FOR EACH ROW
     EXECUTE FUNCTION public.upsert_product_for_receipt();

CREATE OR REPLACE FUNCTION public.handle_product_to_order_upsert()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
BEGIN
  -- спроба оновити існуючий запис
  UPDATE products_to_order
  SET quantity = quantity + NEW.quantity,
      order_date = CURRENT_DATE
  WHERE product_id = NEW.product_id;

  -- перевірка: якщо запис був оновлений — не вставляємо новий
  IF FOUND THEN
    RETURN NULL;
  END IF;

  -- інакше вставляємо новий запис
  RETURN NEW;
END;
CREATE OR REPLACE TRIGGER trg_upsert_product_to_order
    BEFORE INSERT
    ON public.products_to_order
    FOR EACH ROW
    EXECUTE FUNCTION public.handle_product_to_order_upsert();


-- Виконується після видалення елемента з таблиці
-- products_for_invoice, додає видалені продукти в таблицю
-- products_to_order в тій же кількості, створює новий запис
-- або оновлює кількість в уже створеному записі.
```

```
CREATE OR REPLACE FUNCTION
public.trg_restore_products_to_order_after_delete()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
DECLARE
  existing_order_id INTEGER;
BEGIN
  -- Перевіряємо, чи є вже такий продукт у таблиці products_to_order
  SELECT products_to_order_id
  INTO existing_order_id
  FROM products_to_order
  WHERE product_id = OLD.product_id
  LIMIT 1;

  IF FOUND THEN
    -- Якщо існує — оновлюємо кількість
    UPDATE products_to_order
    SET quantity = quantity + OLD.quantity
    WHERE products_to_order_id = existing_order_id;
  ELSE
    -- Якщо не існує — створюємо новий запис
    INSERT INTO products_to_order (product_id, quantity,
order_date)
    VALUES (OLD.product_id, OLD.quantity, NOW());
  END IF;

  RETURN OLD;
END;
CREATE OR REPLACE TRIGGER trg_after_delete_products_for_invoice
    AFTER DELETE
    ON public.products_for_invoice
    FOR EACH ROW
    EXECUTE FUNCTION
public.trg_restore_products_to_order_after_delete();


-- Викликається перед додаванням запису в таблицю
-- products_for_invoice, перевіряє кількість доданого продукту,
-- віднімає кількість доданих продуктів з таблиці
-- products_to_order, автоматично визнає ціну продукту,
-- та проводить перевірку на існування запису.
CREATE OR REPLACE FUNCTION
public.trg_products_for_invoice_insert()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
DECLARE
  total_available INTEGER;
  qty_to_deduct INTEGER := NEW.quantity;
  rec RECORD;
  existing_price NUMERIC;
BEGIN
```

```
  -- 1. Підрахунок загальної кількості товару
  SELECT COALESCE(SUM(quantity), 0)
  INTO total_available
  FROM products_to_order
  WHERE product_id = NEW.product_id;

  IF total_available < NEW.quantity THEN
    RAISE EXCEPTION 'Недостатня кількість товару для додавання до
накладної';
  END IF;

  -- 2. Віднімаємо з найстаріших записів
  FOR rec IN
    SELECT products_to_order_id, quantity
    FROM products_to_order
    WHERE product_id = NEW.product_id
    ORDER BY products_to_order_id
  LOOP
    IF rec.quantity >= qty_to_deduct THEN
      UPDATE products_to_order
      SET quantity = quantity - qty_to_deduct
      WHERE products_to_order_id = rec.products_to_order_id;

      DELETE FROM products_to_order
      WHERE products_to_order_id = rec.products_to_order_id AND
quantity = 0;

      qty_to_deduct := 0;
      EXIT;
    ELSE
      qty_to_deduct := qty_to_deduct - rec.quantity;
      DELETE FROM products_to_order
      WHERE products_to_order_id = rec.products_to_order_id;
    END IF;
  END LOOP;

  -- 3. Визначення існуючої ціни продукту (якщо є)
  SELECT pfi.product_price
  INTO existing_price
  FROM products_for_invoice pfi
  JOIN invoice i ON i.invoice_id = pfi.invoice_id
  WHERE pfi.product_id = NEW.product_id
    AND i.status != 'Списання'
  LIMIT 1;

  IF FOUND THEN
    NEW.product_price := existing_price; -- перезапис ціни
  END IF;

  -- 4. Якщо запис уже існує — оновлюємо кількість
  IF EXISTS (
    SELECT 1 FROM products_for_invoice
    WHERE product_id = NEW.product_id AND invoice_id =
NEW.invoice_id
  ) THEN
    UPDATE products_for_invoice
    SET quantity = quantity + NEW.quantity
```

```
    WHERE product_id = NEW.product_id AND invoice_id =
NEW.invoice_id;

    RETURN NULL;
  ELSE
    RETURN NEW;
  END IF;
END;
CREATE OR REPLACE TRIGGER trg_products_for_invoice_insert
    BEFORE INSERT
    ON public.products_for_invoice
    FOR EACH ROW
    EXECUTE FUNCTION public.trg_products_for_invoice_insert();


-- Викликається під час оновлення запису, оновлює ціну для
-- кожного елементу який відповідає вказаному id.
CREATE OR REPLACE FUNCTION
public.update_product_price_for_active_invoices()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
BEGIN
  -- Оновлюємо всі записи з таким же product_id,
  -- де накладна не завершена і не є типу "Повернення",
  -- і де інший запис (не той, що зараз оновлюється)
  UPDATE products_for_invoice
  SET product_price = NEW.product_price
  WHERE product_id = NEW.product_id
    AND products_for_invoice_id != NEW.products_for_invoice_id
    AND invoice_id IN (
      SELECT invoice_id
      FROM invoice
      WHERE invoice_type != 'Повернення'
    );

  RETURN NEW;
END;
CREATE OR REPLACE TRIGGER trigger_update_product_price
    AFTER UPDATE OF product_price
    ON public.products_for_invoice
    FOR EACH ROW
    WHEN (old.product_price IS DISTINCT FROM new.product_price)
    EXECUTE FUNCTION
public.update_product_price_for_active_invoices();
```

ДОДАТОК Е

Запити на створення представлень для вирішення задач користувачів

```
-- Представлення за допомогою якого ми отримуємо загальну
-- кількість продуктів, що є в наявності.
CREATE OR REPLACE VIEW public.products_in_stock
 AS
 WITH base_stock AS (
         SELECT pfi.product_id,
            p.product_name AS products_name,
            sum(
                CASE
                    WHEN i.invoice_type =
'Отримання'::invoice_type_enum AND i.status::text =
'Завершено'::text THEN pfi.quantity::numeric
                    WHEN (i.invoice_type = ANY
(ARRAY['Списання'::invoice_type_enum,
'Повернення'::invoice_type_enum])) AND i.status::text =
'Завершено'::text THEN (- pfi.quantity)::numeric
                    ELSE 0::numeric
                END) AS stock_quantity,
            max(pfi.product_price) AS product_price
           FROM products_for_invoice pfi
             JOIN invoice i ON i.invoice_id = pfi.invoice_id
             JOIN products p ON p.product_id = pfi.product_id
          GROUP BY pfi.product_id, p.product_name
        ), sold_products AS (
         SELECT pr.product_id,
            sum(pr.quantity) AS total_sold
           FROM products_for_receipt pr
             JOIN receipt r ON r.receipt_id = pr.receipt_id
          WHERE r.receipt_type::text = 'Продаж'::text
          GROUP BY pr.product_id
        ), returned_products AS (
         SELECT pr.product_id,
            sum(pr.quantity) AS total_returned
           FROM products_for_receipt pr
             JOIN receipt r ON r.receipt_id = pr.receipt_id
          WHERE r.receipt_type::text = 'Повернення'::text
          GROUP BY pr.product_id
        )
 SELECT bs.product_id,
    bs.products_name,
    bs.stock_quantity - COALESCE(sp.total_sold, 0::numeric) +
COALESCE(rp.total_returned, 0::numeric) AS quantity,
    bs.product_price
   FROM base_stock bs
     LEFT JOIN sold_products sp ON sp.product_id = bs.product_id
     LEFT JOIN returned_products rp ON rp.product_id =
bs.product_id
```

ДОДАТОК F

Список привілеїв ролей на об’єкти БД

Таблиця К.1 – Список привілеїв ролей на об’єкти БД

Об’єкти БД	Ролі		
	Адміністратор (admin)	Продавець (seller)	Гість (guest)
Position	CRUD		R
Categories	CRUD		
Employee	CRUD	R	R
Invoice	CRUD		
Products	CRUD	R	
ProductsForInvoice	CRUD	R	
ProductsForReceipt	CRUD	CRD	
ProductsToOrder	CRUD		
Provider	CRUD		
Receipt	CRD	C	

ДОДАТОК G
Створення ролей та призначення їм привілеїв

```
-- Менеджер –

-------Таблиці-та-представлення---------
GRANT INSERT,SELECT,UPDATE,DELETE ON project TO manager_group;
GRANT INSERT,SELECT,UPDATE ON client TO manager_group;
GRANT SELECT ON project_stage TO manager_group;
GRANT SELECT ON stage TO manager_group;
GRANT SELECT ON planned_work TO manager_group;
GRANT SELECT ON appointment TO manager_group;
GRANT SELECT ON team TO manager_group;
GRANT SELECT ON stage_work_type TO manager_group;
GRANT SELECT ON work_type TO manager_group;
GRANT SELECT ON required_material TO manager_group;
GRANT SELECT ON material TO manager_group;
GRANT SELECT ON material_unit TO manager_group;
GRANT SELECT ON required_machine TO manager_group;
GRANT SELECT ON machine TO manager_group;
GRANT SELECT ON supplier TO manager_group;
GRANT SELECT ON team_member TO manager_group;
GRANT SELECT ON manager TO manager_group;
GRANT SELECT ON working_project_state TO manager_group;
GRANT SELECT ON project_estimate TO manager_group;
GRANT SELECT ON late_work TO manager_group;

------------Функції---та---процедури--------------
GRANT EXECUTE ON FUNCTION get_group TO manager_group;
GRANT EXECUTE ON FUNCTION get_project_by_status TO manager_group;
GRANT EXECUTE ON FUNCTION get_time_in_team TO manager_group;
GRANT   EXECUTE   ON   FUNCTION   get_last_appointment_id   TO
manager_group;
GRANT EXECUTE ON PROCEDURE change_password TO manager_group;

-- Адміністратор --

-------Таблиці-та-представлення---------
GRANT SELECT, UPDATE, DELETE, INSERT on position to admin_role
GRANT SELECT, UPDATE ON project TO planner_group;
GRANT INSERT,SELECT,DELETE ON project_stage TO planner_group;
GRANT SELECT ON stage TO planner_group;
GRANT INSERT,SELECT,UPDATE,DELETE
     ON planned_work TO planner_group;
GRANT SELECT ON team TO planner_group;
GRANT SELECT ON team_member TO planner_group;
GRANT SELECT ON stage_work_type TO planner_group;
GRANT SELECT ON work_type TO planner_group;
GRANT INSERT,SELECT,DELETE ON required_material TO planner_group;
GRANT SELECT ON material TO planner_group;
GRANT SELECT ON material_unit TO planner_group;
GRANT INSERT,SELECT,DELETE ON required_machine TO planner_group;
GRANT SELECT ON machine TO planner_group;
```

```
-----------Функції---та---процедури-------------
GRANT EXECUTE ON FUNCTION get_group TO planner_group;
GRANT EXECUTE ON FUNCTION get_project_by_status TO planner_group;
GRANT EXECUTE ON FUNCTION check_stage_planning TO planner_group;
GRANT EXECUTE ON FUNCTION get_free_teams TO planner_group;
GRANT EXECUTE ON FUNCTION check_empty_stages TO planner_group;
GRANT   EXECUTE   ON   FUNCTION   check_work_type_planning   TO
planner_group;
GRANT EXECUTE ON FUNCTION check_team_assignment TO planner_group;
GRANT EXECUTE ON PROCEDURE change_password TO planner_group;
```

-- Кадровик --

```
-------Таблиці-та-представлення---------
GRANT INSERT,SELECT ON appointment TO hr_group;
GRANT SELECT ON position TO hr_group;
GRANT INSERT,SELECT,UPDATE ON employee TO hr_group;
GRANT INSERT,SELECT,UPDATE ON team_member TO hr_group;
GRANT SELECT ON team TO hr_group;
GRANT SELECT ON person TO hr_group;
GRANT SELECT ON last_appointment TO hr_group;
GRANT SELECT ON worker TO hr_group;

-----------Функції---та---процедури-------------
GRANT EXECUTE ON FUNCTION get_group TO hr_group;
GRANT EXECUTE ON FUNCTION check_chief_existing TO hr_group;
GRANT EXECUTE ON FUNCTION get_last_appointment_id TO hr_group;
GRANT EXECUTE ON FUNCTION get_username_by_table_num TO hr_group;
GRANT EXECUTE ON FUNCTION get_role_by_position_id TO hr_group;
GRANT EXECUTE ON PROCEDURE change_password TO hr_group;
GRANT EXECUTE ON PROCEDURE create_new_user TO hr_group;
GRANT EXECUTE ON PROCEDURE set_user_role TO hr_group;
```

-- Комірник --

```
-------Таблиці-та-представлення---------
GRANT INSERT,SELECT,UPDATE,DELETE
     ON warehouse TO storekeeper_group;
GRANT INSERT,SELECT,UPDATE,DELETE
     ON material TO storekeeper_group;
GRANT INSERT,SELECT,UPDATE,DELETE
     ON material_type TO storekeeper_group;
GRANT INSERT,SELECT,UPDATE,DELETE
     ON material_unit TO storekeeper_group;
GRANT SELECT ON required_material TO storekeeper_group;
GRANT SELECT ON planned_work TO storekeeper_group;
GRANT SELECT ON project_stage TO storekeeper_group;
GRANT SELECT ON project TO storekeeper_group;
GRANT INSERT,SELECT,UPDATE,DELETE
     ON delivery TO storekeeper_group;
GRANT INSERT,SELECT,UPDATE,DELETE ON garage TO storekeeper_group;
GRANT INSERT,SELECT,UPDATE,DELETE ON machine TO storekeeper_group;
GRANT INSERT,SELECT,UPDATE,DELETE
```

```
     ON machine_type TO storekeeper_group;
GRANT INSERT,SELECT,DELETE ON used_machine TO storekeeper_group;
GRANT SELECT ON required_machine TO storekeeper_group;
GRANT INSERT,SELECT,UPDATE,DELETE
     ON supplier TO storekeeper_group;
GRANT INSERT,SELECT,UPDATE,DELETE
     ON supplied_material TO storekeeper_group;
GRANT INSERT,SELECT,UPDATE,DELETE
     ON supplied_machine TO storekeeper_group;
GRANT SELECT ON material_purchase TO storekeeper_group;
GRANT SELECT ON grouped_material TO storekeeper_group;
GRANT SELECT ON delivered_material TO storekeeper_group;
GRANT SELECT ON material_request TO storekeeper_group;
GRANT SELECT ON machine_rent TO storekeeper_group;
GRANT SELECT ON grouped_machine TO storekeeper_group;
GRANT SELECT ON machine_request TO storekeeper_group;

------------Функції---та---процедури--------------
GRANT EXECUTE ON FUNCTION get_group TO storekeeper_group;
GRANT EXECUTE ON FUNCTION get_use_price TO storekeeper_group;
GRANT EXECUTE ON FUNCTION get_free_rents TO storekeeper_group;
GRANT EXECUTE ON FUNCTION get_free_machines TO storekeeper_group;
GRANT EXECUTE ON FUNCTION get_active_material_requests
     TO storekeeper_group;
GRANT EXECUTE ON FUNCTION get_active_machine_requests
     TO storekeeper_group;
GRANT EXECUTE ON PROCEDURE change_password TO storekeeper_group;
```

-- Бригадир --

```
-------Таблиці-та-представлення---------
GRANT SELECT ON appointment TO chief_group;
GRANT SELECT ON team TO chief_group;
GRANT SELECT, UPDATE ON planned_work TO chief_group;
GRANT SELECT, UPDATE ON used_machine TO chief_group;
GRANT SELECT ON required_machine TO chief_group;
GRANT SELECT ON garage TO chief_group;
GRANT SELECT,UPDATE ON project_stage TO chief_group;
GRANT SELECT,UPDATE ON project TO chief_group;
GRANT SELECT ON required_material TO chief_group;
GRANT SELECT ON worker TO chief_group;
GRANT SELECT ON working_plan TO chief_group;
GRANT SELECT ON machine_request TO chief_group;
GRANT SELECT ON material_request TO chief_group;
GRANT SELECT ON delivered_material TO chief_group;

------------Функції---та---процедури--------------
GRANT EXECUTE ON FUNCTION get_group TO chief_group;
GRANT EXECUTE ON FUNCTION get_last_appointment_id TO chief_group;
GRANT EXECUTE ON FUNCTION get_active_material_requests
     TO chief_group;
GRANT EXECUTE ON FUNCTION get_active_machine_requests
     TO chief_group;
```

```
GRANT EXECUTE ON PROCEDURE change_password TO chief_group;
GRANT EXECUTE ON PROCEDURE try_close_machine_using TO chief_group;
GRANT EXECUTE ON PROCEDURE try_close_completed_stage
     TO chief_group;
GRANT EXECUTE ON PROCEDURE try_close_completed_project
     TO chief_group;
```

-- Робітник --

```
-------Таблиці-та-представлення---------
GRANT SELECT ON appointment TO worker_group;
GRANT SELECT ON team TO worker_group;
GRANT SELECT ON worker TO worker_group;
GRANT SELECT ON working_plan TO worker_group;

------------Функції---та---процедури--------------
GRANT EXECUTE ON FUNCTION get_group TO worker_group;
GRANT EXECUTE ON FUNCTION get_last_appointment_id TO worker_group;
GRANT EXECUTE ON PROCEDURE change_password TO worker_group;
```

-- Звичайний користувач --

```
------------Функції---та---процедури--------------
GRANT EXECUTE ON FUNCTION get_group TO default_group;
GRANT EXECUTE ON FUNCTION check_user TO default_group;
GRANT EXECUTE ON FUNCTION dblink(
text,text) TO default_group;
```