НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
комп'ютерних наук

Голуб Б.Л., доц., к.т.н.

(підпис) (ПІБ, вчене звання і ступінь)

«___»_____2025 р

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Програмне забезпечення інтернет-магазину для продажу меблів»

Спеціальність 121 «Інженерія програмного забезпечення»

Гарант освітньої програми

к.т.н. доцент _____ Вайганг Г.О.
(Науковий ступень та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи

_____ _____ Бородкін Г.О.
(Науковий ступень та вчене звання) (підпис) (ПІБ)

Виконав _____ Ломако Я.М.
(підпис) (ПІБ)

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ

Завідувач кафедри
комп'ютерних наук

_____ / Голуб Б.Л., доцент, к.т.н /

підпис

"__"_____2025 р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

студентці Ломако Ярослав Миколайович

Спеціальність 121 «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи: Програмне забезпечення інтернет-магазину для продажу меблів

Затверджена наказом ректора НУБіП України від 16.12.2024 № 2248 "С"

Термін подання завершеної роботи на кафедру _____

(рік, місяць, число)

Вихідні дані до роботи: опис програмного забезпечення

Перелік питань , які потрібно розробити:

Аналіз проблемної області, вибір та обґрунтування засобів для розробки системи, проектування інформаційної системи.

Дата видачі завдання "16" 12 2024р.

Керівник бакалаврської кваліфікаційної роботи

_____ _____ Бородкін Г.О

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання _____ Ломако Я.М.

(підпис) (ПІБ студента)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД - База даних

ПЗ – Програмне забезпечення

ІБ – Інформаційна база

ВМ - Віртуальна машина

ОС – Операційна система

СР – Середовище розробки

ВСТУП

В наш час для підприємств є обов'язковим мати свій сайт для забезпечення зростання бізнесу, тому не зважаючи на те що мебельна промисловість є одним із самим високим попитом, також не оминула ця потреба мати свій персональний сайт для автоматизації купівлі товарів, щоб полегшити навантаження на персонал і також зменшити кількість персоналу для роботи, полегшення купівлі товарів в асортиментів, щоб користувачам не приходилось кудись йти щоб придбати той чи інший товар не виходячи з дома, а прибрати його, наприклад, з ліжка, а також для поширення пізнаваності магазину і легкому розширення без потреби постійно відкривати нові магазини в нових місцях.

Також не стоїть забувати, що де є можливість збільшити прибуток для свого підприємства і одночасно зменшити затрати і впізнаність свого підприємство, то цим обов'язково скористаються конкуренти, що може бути зробити катастрофічним наслідком для підприємства бо конкуренти просто затьмарити вас без можливості на отримання хоча би якогось "світла" і з цим нічого не можна зробити бо так влаштований наш світ, де той хто не може пристосовуватися до сучасних тенденцій і технологій, той ніколи не зможе провести своє підприємство до процвітання.

Тому так важливо любому підприємству, не тільки для магазинів меблів, а й любим другим магазинам і навіть заводом, і тому подібному, мати свій сайт де вони можуть реалізувати це все і не відставати від інших конкурентів цієї галузі.

Тому в цій бакалаврської кваліфікаційної роботі на тему "Програмне забезпечення інтернет-магазину для продажу меблів" було розроблено, як видно із назви цієї роботи, інтернет магазин для продажу меблів, щоб вирішити цю проблему і додатково получити навички роботі в сфері розробки різноманітних сайтів.

Для того щоб досягти цієї мети були вирішені наступні завдання:

- Проаналізовано предметну область щодо функціонування і різних потреб для створення інтернету магазину
- Побудова багатьох і різноманітних моделей по принципу UML діаграм
- Обрано необхідні інструменти для розробки системи
- Розробити саму бекенду системи
- Розробити фронденду системи
- Підключення і створено структури БД
- Проведення тестування системи
- Складання пояснювальної записці

Основним завданням ПЗ є: додавання і видалення товарів, зміна даних товарів включаючи кількість, ціну, знижки і інші, можливість авторизації і реєстрації, управління кошику товарів з можливістю додавати, видаляти і змінювати кількість товарів в ньому, управління категоріями, управління замовленнями з можливістю змінювати стан замовлення, функціональна панель адміністрації з різними ролями, пошук товарі, фільтрація, сортування товарів і з можливістю замовлення потрібних для користувачеві товарів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Загальний опис системи:

Система реалізує онлайн-продажу меблів. Ця система буде містить пропоновані товари які користувачі можуть купити. Користувач може буде переглядати каталог товарів, інформацію конкретного товару, добавлення товару в кошик і оплачувати їх. Також система буде мати адміністративну панель де менеджер сайту може буде управляти даними товарів.

Основні ролі користувачів:

Покупець:

Може переглядати товар, додавати його в корзину і купувати. Також в нього є можливість фільтрації товару і його пошуку в пошуковій системи. Може переглядати історію своїх покупок

Менеджер:

Керує даними товарів. Додає нові товари і видаляє старі. Може керувати акціями товарів і їхніми цінами, тощо

Працівник складу:

Керує даними замовлення і відправляє товар покупцям. Також може добавляти новий товар в систему

1.2 Аналіз вимог до програмної системи

Функціональні вимоги:

Функціонал для покупця:

- Реєстрація та авторизація – додавання нового користувача в систему і здійснення входу в систему
- Перегляд каталогу – перегляд наявних в системі товари користувачами

- Пошук товарів – здійснення пошуку товарів по пошукувачу по вказаним словам
- Фільтр товару – здійснювати фільтрацію наявних товарів по вибраним можливими критеріями
- Управління своїм кошиком – можливість керувати своїм кошиком додаючи, видаляючи або змінювання свого товару
- Оформлення покупки – здійснення покупки попередньо вибраних в кошику товарів
- Підтвердження отримання товару – можливість підтвердити отримання замовлення
- Управління даними кабінету користувача – здійснювати зміну своїх власних даних в профілю користувача

Менеджер:

- Додавання товару – можливість додавання нового товару до системи магазину меблів
- Управління товару – зміна даних конкретного товара магазину, таких як: кількість, ціна, знижка, тощо.
- Видалення товару – видалення конкретного товару в системи магазину якщо цей товар вже не потрібний
- Управління акціями – управління поточної акції конкретного товара і подальшої зміни його ціни

Працівник складу:

- Управління статусів замовлень – управляє статусами всіх замовлень які були зроблені користувачами
- Відправка замовлення – відправляє конкретне замовлення до конкретного користувача, який замовив його

Нефункціональні вимоги:

- Адаптивний інтерфейс – створення інтерфейсу який б адаптувався до пристрої користувачів
- Зрозумілий інтерфейс – створення інтерфейсу який б був легко-зрозумілим
- Зміна мови сайту – можливість змінити мову сайту за потребою користувачів
- Безпека сайту – захист сайту від різноманітних хакерських атак і виправлення уразливості сайту

Очікуваний результат:

Готова для використання система онлайн-продажу меблів. Де буде можливо покупцям купувати товар онлайн, користуватись пошуками, фільтрами, управління кошиком, перегляд власних покупок , а менеджерам додавати новий товар, категорію або змінювати їх і видаляти. Також сайт буде мати адаптивність що дозволяє користуватись ним і через телефон.

1.3 Моделювання предметної області

Для того щоб краще розробити сайт і для зменшення помилок в ході розробки систем потрібно з початку змоделювати її предметну область через різноманітні моделі таких як UML

Моделювання предметної області — це один із найважливіших етапів розробки програмного забезпечення. Воно дозволяє зрозуміти, що саме потрібно реалізувати, ще до початку програмування.

Моделювання предметної області потрібен для:

1. Краще розуміння задачі

Модель показує, як працює реальний бізнес або процес, які об'єкти є, як вони взаємодіють. Це дає чітке уявлення і для замовника, і для розробника.

2. Формалізація вимог

Замість абстрактного «зробіть сайт», ми маємо чітку структуру: які є класи, дії, атрибути, зв’язки тощо. Це зменшує ризик помилок і непорозумінь.

3. Економія часу та грошей

Краще знайти логічну помилку на схемі, ніж у вже написаному коді. Моделювання допомагає вчасно виявити недоліки в логікі системи.

4. Полегшує командну роботу

Модель зрозуміла всім: аналітикам, розробникам, тестувальникам. Це спільна «мова» проекту.

5. Планування архітектури

Модель — це основа для побудови бази даних, API, інтерфейсу тощо. Вона допомагає структуровано реалізовувати систему.

Один з самих популярних методів побудови цих діаграм е мова побудови моделей UML

UML (Unified Modeling Language) — це уніфікована мова моделювання, яка використовується для візуалізації, опису та проектування програмних систем.

Призначення UML

Малюнок вартий тисячі слів, це абсолютно підходить для обговорення UML. Об'єктно-орієнтовані концепції були представлені набагато раніше UML. Тому на той час не було стандартних методологій для організації та консолідації об’єктно-орієнтованої розробки. У той момент з’явився UML. Існує кілька цілей для розробки UML, але найважливішою є:

- Щоб визначити деяку мову моделювання загального призначення, яку можуть використовувати всі модельєри, а також її потрібно зробити простою для розуміння та використання.
- Створено для розробників, а також для бізнес-користувачів, простих людей і всіх, хто хоче зрозуміти систему.
    - Система може бути програмною і непрограмною.
    - Має бути зрозуміло, що UML не є методом розробки, а супроводжує процеси для створення успішної системи.

мета UML може бути визначена як простий механізм моделювання для моделювання всіх можливих практичних систем у сучасному складному середовищі.

«UML є мовою моделювання загального призначення. Спочатку він був запущений для запису поведінки складного програмного забезпечення та непрограмної системи, а тепер він став загальним стандартом.

UML надає елементи та компоненти для підтримки вимог складних систем. UML слідує об'єктно-орієнтованим концепціям і методології. Таким чином, об’єктно-орієнтовані системи зазвичай моделюються за допомогою графічної мови.

Діаграми UML створюються з різних точок зору, таких як проектування, реалізація, розгортання тощо. На завершення UML можна визначити як мову моделювання для охоплення архітектурних, поведінкових і структурних аспектів системи.

Об’єкти є ключем до цього об’єктно-орієнтованого світу. Основна вимога об'єктно-орієнтованого аналізу та проектування полягає в тому, щоб ідентифікувати об'єкт ефективно. Після цього розподіляються обов’язки по об’єктах. Після завершення цього завдання проект виконується з використанням вхідних даних аналізу.

UML відіграє важливу роль у цьому ООП-аналізі та проектуванні; Діаграми UML використовуються для моделювання дизайну. Тому UML відіграє важливу роль»[7].

І під час розробки даної системи було розроблено деякі діаграми моделювання, першим з яких була діаграма прецедентів (Рис. 1)

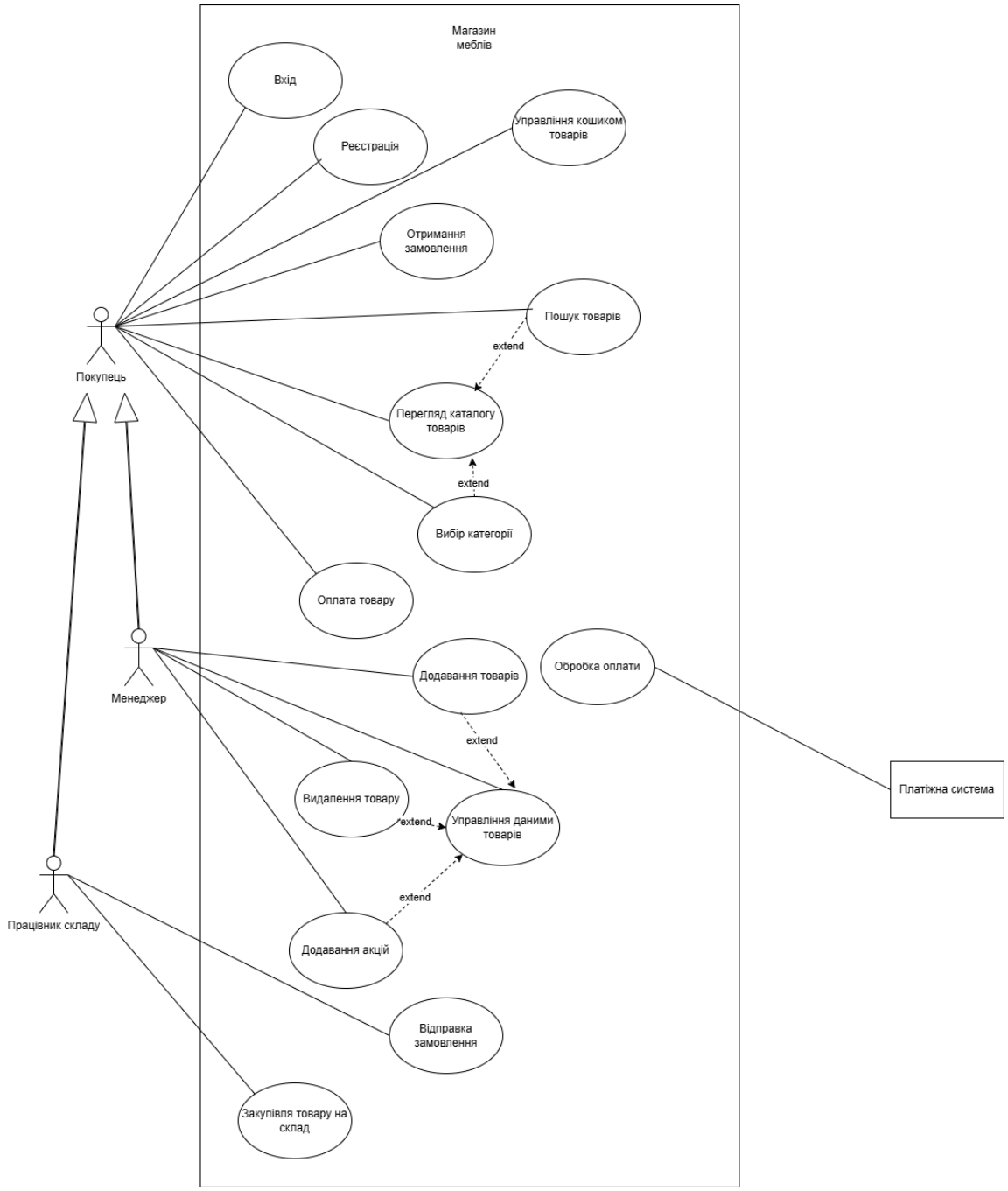


Рис. 1 Діаграма прецедентів

В цій діаграмі описуються ролі в системи і можливості(прециденти) які вони можуть виконувати в системі.

Після чого була розроблена діаграма діаграма послідовності(Рис. 2) для дипломної системи.

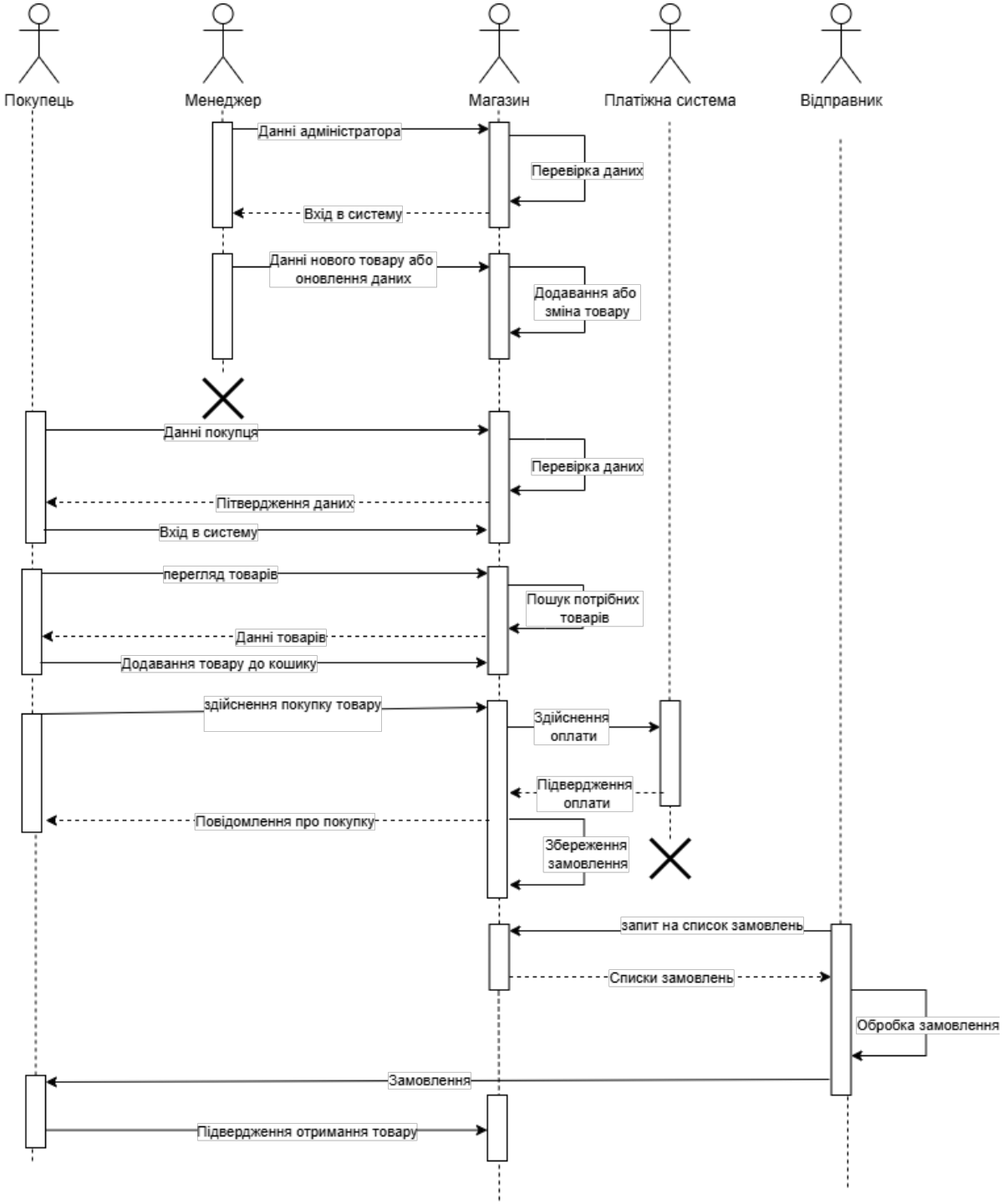


Рис. 2 Діаграма послідовності

Ця діаграма описує послідовність процесу які проходять між акторами яку потрібно виконати для конкретної мети.

Остання діаграма яка була змодельована для предметної області була діаграма активності(Рис. 3).

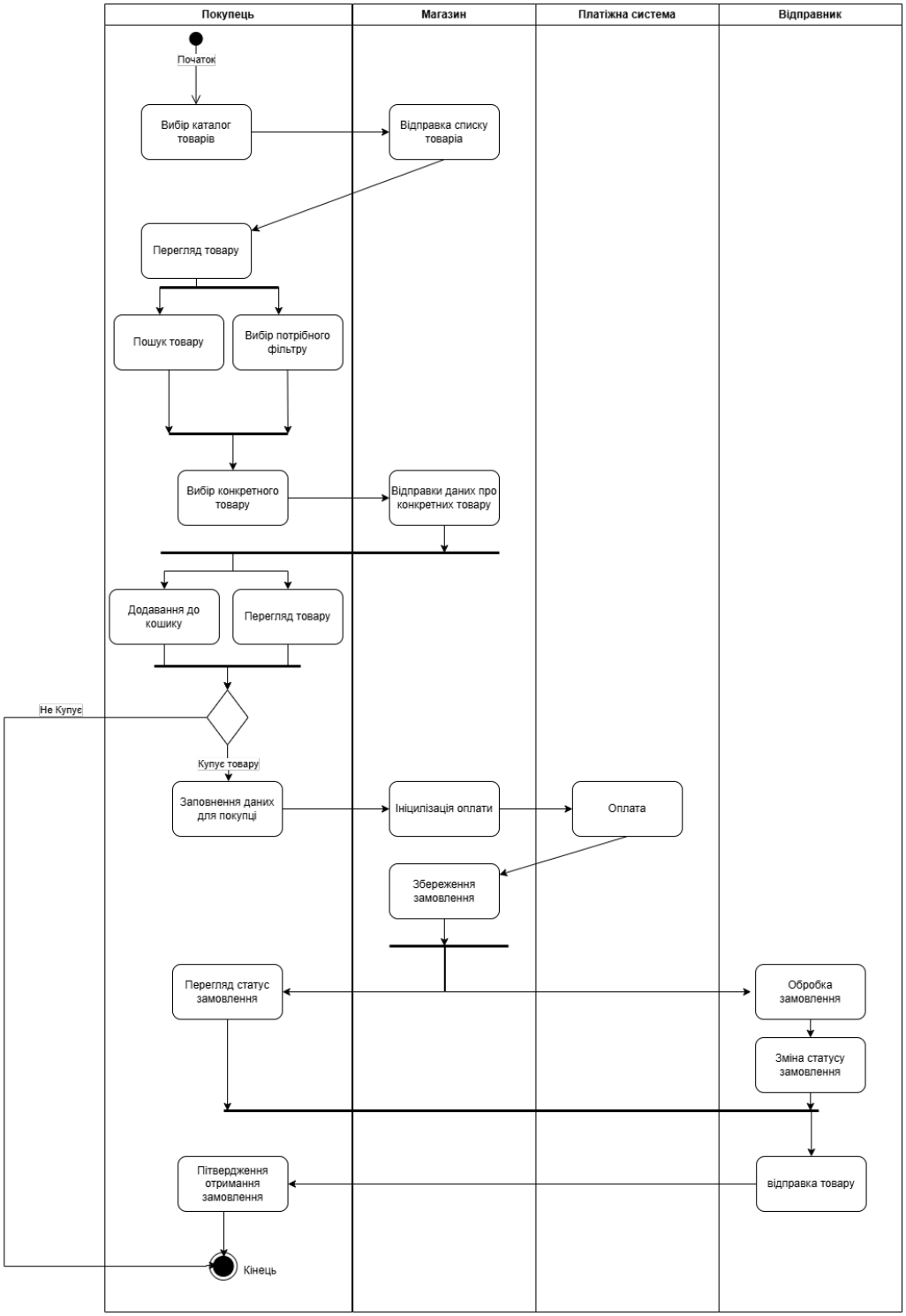


Рис. 3 Діаграма активності

Ця діаграма описує логіку виконання процесу або алгоритму. Вона показує послідовність дій, умовні розгалуження, цикли, паралельні гілки тощо. Вона дещо схожа на блок-схему, який показує, як система або користувач виконує завдання крок за кроком.

1.4 Огляд інформаційних джерел та існуючих рішень

Для допомоги в розробки системи був використаний офіційна документація Django, де є багато існуючих рішень щодо реалізації проекту.

Також були використані письмові і онлайн гайди для навчання з інструментами за допомогою яких був реалізований проект.

Якщо вникали якісь проблеми під час розробки, які важко було вирішити за допомогою вище перерахованих варіантів, був використаний популярний сайт для розробників Stuk Owerflow, де програмісти можуть написати питання і проблеми щоб програмісти з всього світу могли допомогти тобі вирішити її.

1.5 Постановка завдання

Розробити інтернет магазин для продажу меблів, щоб вирішити цю проблему і додатково получити навички роботі в сфері розробки різноманітних сайтів.

Для розробки сайту були вибрані декілька мов програмування які будуть відповідати за “бекенд” сайту і “фронтенд” частину сайту.

Для бекенду була вибрана високорівнева мова програмування Python і один з популярних його фреймворків по створенню сайтів Django. Ця мова програмування відповідає за всю логіку системи і взаємодією з серверами баз даних і кешу.

Для фронтеду була вибрані стек мов програмувань які спеціалізуються на створення цієї часті сайту: HTML, CSS, JavaScript. HTML відповідає за основну обгортку сайту і розміщення основних компонентів інтерфейсу користувача. CSS придає обгортці, створеному через HTML, стилі щоб розкрасити сайт і придати

йому придатний вигляд. JavaScript відповідає за динамічні зміни інтерфейсу користувача не перезавантажуючи повторно сторінку бо HTML і CSS цього не можуть. Також для JavaScript був додана бібліотека JQuery, яка полегшує розробку компонентів створених через JavaScript.

Також для полегшення і прискорення розробки інтерфейс користувача була додана бібліотека до проекту – Bootstrap. Це бібліотека вже з готовими елементами інтерфейсів, які легко налаштовувати і вже мають деякі функціїї, наприклад адаптації до екрана користувача і тому не потрібно додатково втрачати час на адаптацію інтерфейсу. Хоча під час розробки все рівно приходилось трохи зніматись адаптацією.

Обов'язковим компонентом любих сайтів магазинів і не тільки є наявність сховище даних де будуть зберігатись різноманітні дані для сайту. В даному випадку була використана об'єктна-реляційна база даних PostgreSQL як сховище всіх даних сайту. Також була використана NoSQL база даних Redis для кешування запити які часто використовуються, але які не часто змінюються. Це потрібно щоб зменшити навантаження на основну БД.

Також в проекті був використаний Docker для зберігання баз даних в середині контейнерів і для подальшої контейнеризації проекту для бистрого розгортання на інших серверах.

1.6 Висновки до розділу 1

Під час опрацювання даного розділу було проаналізовано предметну область, були визначенні функціональні і нефункціональні вимоги, які необхідні системи, розроблена і побудовані діаграми UML для подальшої розробки системи і полегшення процесу.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

2.1 Логічна модель даних у вигляді ER-діаграми

Для того щоб правільно побудувати логічну модель і запобігти помилкам в ході розробки даних потрібно спочатку її попередньо спроектувати.

Самим популярною моделі побудові даних є ER діаграми.

«Діаграма взаємозв'язку сутностей (ER) — це тип блок-схеми, яка ілюструє, як «сутності», такі як люди, об'єкти чи поняття, пов'язані між собою в системі.

ER-діаграми найчастіше використовуються для проектування або налагодження реляційних баз даних у сферах розробки програмного забезпечення, бізнес-інформаційних систем, освіти та досліджень.

Також відомі як ERD або ER Models, вони використовують певний набір символів, таких як прямокутники, ромби, овали та з'єднувальні лінії, для зображення взаємозв'язку сутностей, зв'язків та їхніх атрибутів.

Вони віддзеркалюють граматичну структуру з сутностями як іменниками та зв'язками як дієсловами»[8].

Основні цілі використання ER-діаграми:

- **Проектування бази даних** — допомагає зрозуміти, як дані будуть взаємодіяти та зберігатися в базах даних. Також допомагає визначити типи даних
- **Візуалізація структури даних** — дає змогу, через графічні моделі, легко побачити взаємозв'язки між елементами даних і їхні типи.
- **Комунікація між розробниками** — є зрозумілим інструментом для обговорення і визначення логіки зберігання даних між аналітиками, розробниками БД і програмістами.

- **Документація системи** — використовується як частина технічної документації до проекту.
- **Аналіз бізнес-процесів** — дозволяє побачити, які дані важливі для бізнесу, як вони пов’язані та як їх можна оптимізувати.
- **Усунення несправностей бази даних** - Діаграми ER використовуються для аналізу існуючих баз даних, щоб знайти та вирішити проблеми в логікі або розгортанні системи. Розроблена діаграма має можливість виявити, де й які помилки виникли в системи.

Тому для цього дипломної роботі було побудована в інструменті Draw.io ER діаграму(Рис 4) для цього проекту

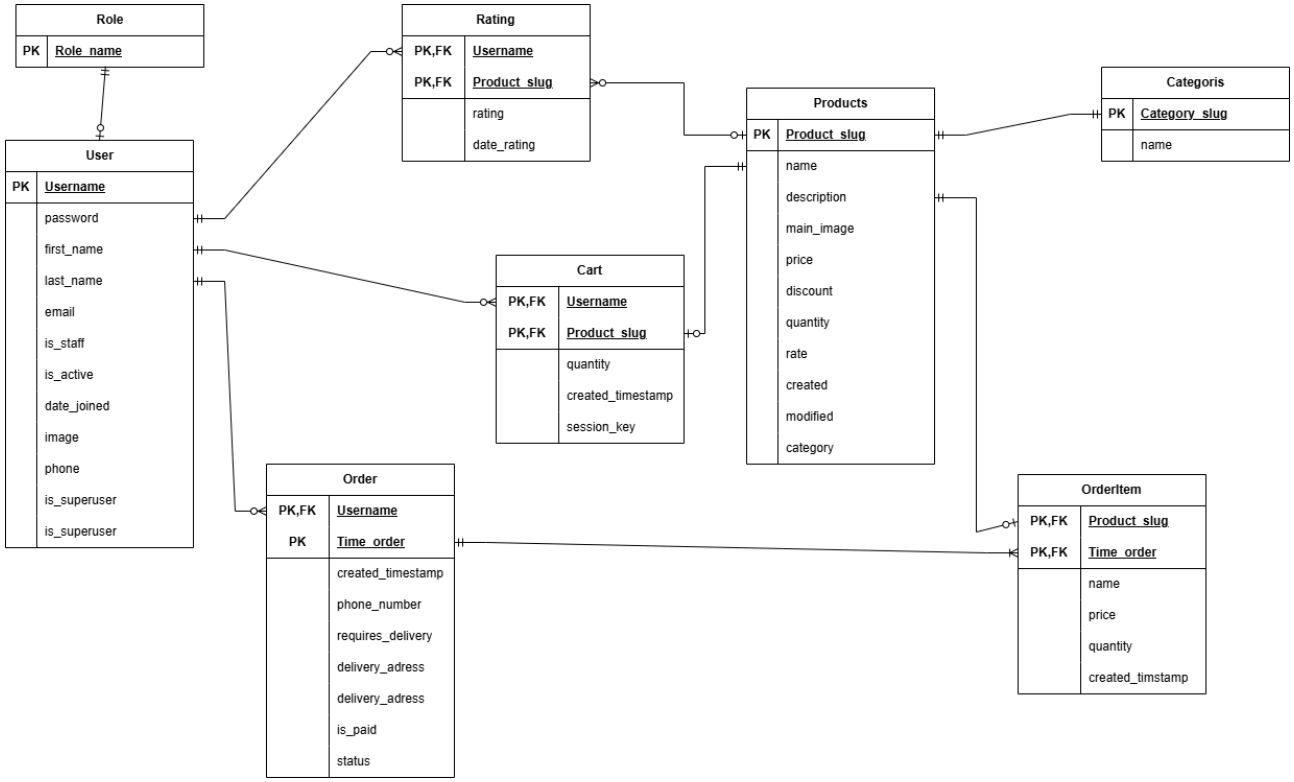


Рис. 4 ER Діаграма

На ній зображена логічна модель даних яка використовується в системі.

На ній зображені показує, які сутності (entity) існують у системі, які атрибути мають ці сутності, та які між ними зв’язки (relationship).

Ця діаграма буде задіяна для подальшої створення БД .

2.2 Діаграма класів та кооперацій

2.2.1 Діаграма класів

Діаграма класів (Class Diagram) — це один із основних типів діаграм в UML (Unified Modeling Language), який використовується для моделювання структури об'єктно-орієнтованої системи.

«Діаграми класів є одним із найкорисніших типів діаграм в UML, оскільки вони чітко відображають структуру конкретної системи шляхом моделювання її класів, атрибутів, операцій і зв’язків між об’єктами.

Популярні серед розробників програмного забезпечення для документування архітектури програмного забезпечення, діаграми класів є типом структурної діаграми, оскільки вони описують те, що має бути присутнім у системі, що моделюється»[6].

Вона полегшує розробку систем бо допомагає розробнику подивитись на програму в общем перспективі.

Для чого використовується діаграма класів:

- **Проектування об'єктно-орієнтованого програмного забезпечення** — дозволяє описати, які класи будуть у системі, їхні атрибути та методи.
- **Візуалізація структури коду** — показує залежності, наслідування, асоціації, агрегації та композиції між класами.
- **Аналіз системи** — допомагає зрозуміти як взаємодіють між собою об'єкти до написання коду.
- **Документація системи** — діаграма класів може бути частиною технічної документації, яка буде допомогать під час розробки чи супроводження системи.

Що включає в себе клас на діаграмі:

Назва класу – назва класу, яка описує чим цей клас знімається і що він робить.

Атрибути (властивості) – це данні які зберігає в себе той чи інший атрибути який має конкретний клас.

Методи (операції) – це ті функції що може здійснювати в системі конкретний клас.

І в ході розробки проекту також була змодельована діаграма класів(Рис. 5) для цієї системи.

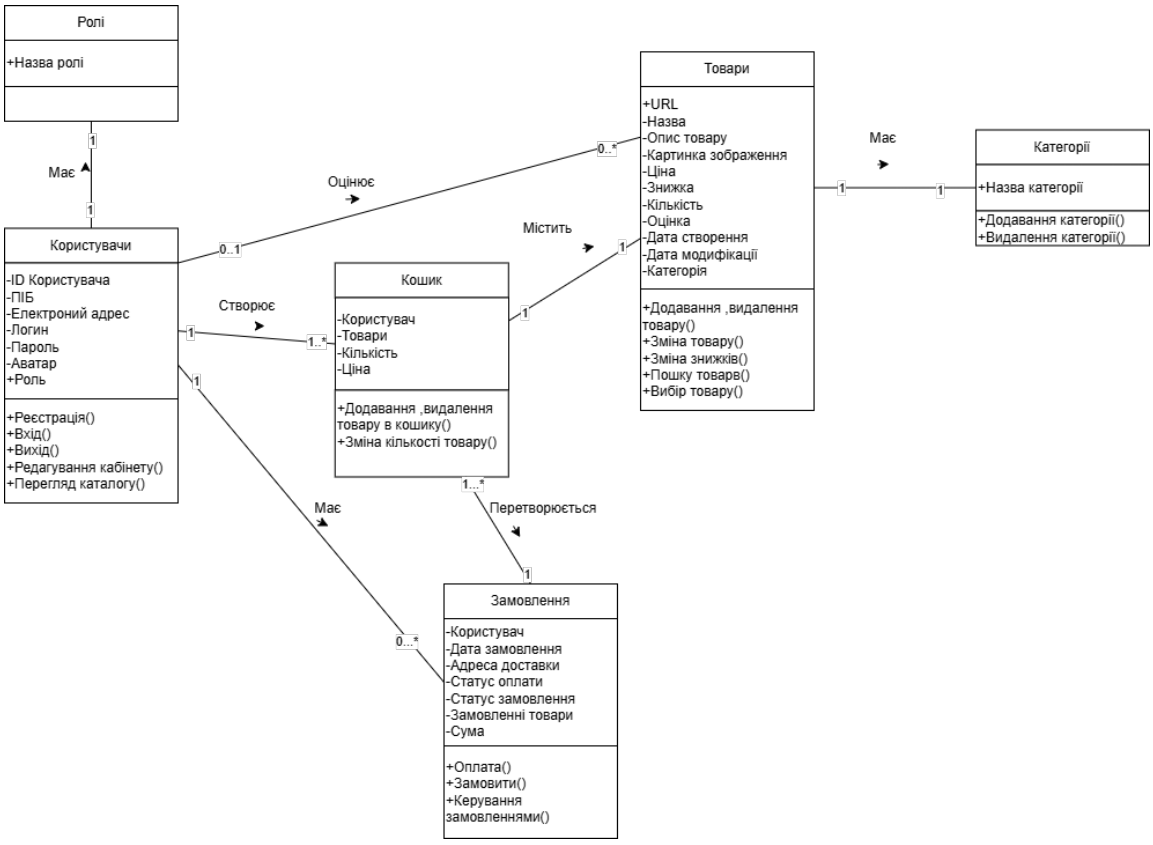


Рис. 5 Діаграма класів

Вона описує класи їх атрибути і методі які будуть задіяні під час розробки системи.

2.2.2 Діаграма кооперацій

«Діаграма співпраці, також відома як діаграма зв’язку, є ілюстрацією зв’язків і взаємодії між програмними об’єктами в уніфікованій мові моделювання (UML). Розробники можуть використовувати ці діаграми, щоб зобразити динамічну поведінку конкретного випадку використання та визначити роль кожного об’єкта»[4].

Щоб створити діаграму співпраці, спочатку визначте структурні елементи, необхідні для виконання функціональних можливостей взаємодії. Потім побудуйте модель, використовуючи зв’язки між цими елементами. Кілька постачальників пропонують програмне забезпечення для створення та редагування діаграм співпраці.

Для чого використовується діаграма кооперацій (комунікацій):

- Щоб показати, як об'єкти взаємодіють між собою під час виконання конкретного сценарію.
- Вона поєднує інформацію про структуру (об’єкти) і повідомлення (виклики методів), які передаються між об’єктами.
- Це альтернатива діаграмі послідовностей, але більше акцентується на структурі зв’язків між об’єктами, ніж на хронології подій в системі.

І в ході розробки систем були розроблені декілька діаграм кооперацій які допоможуть в створенні системі:

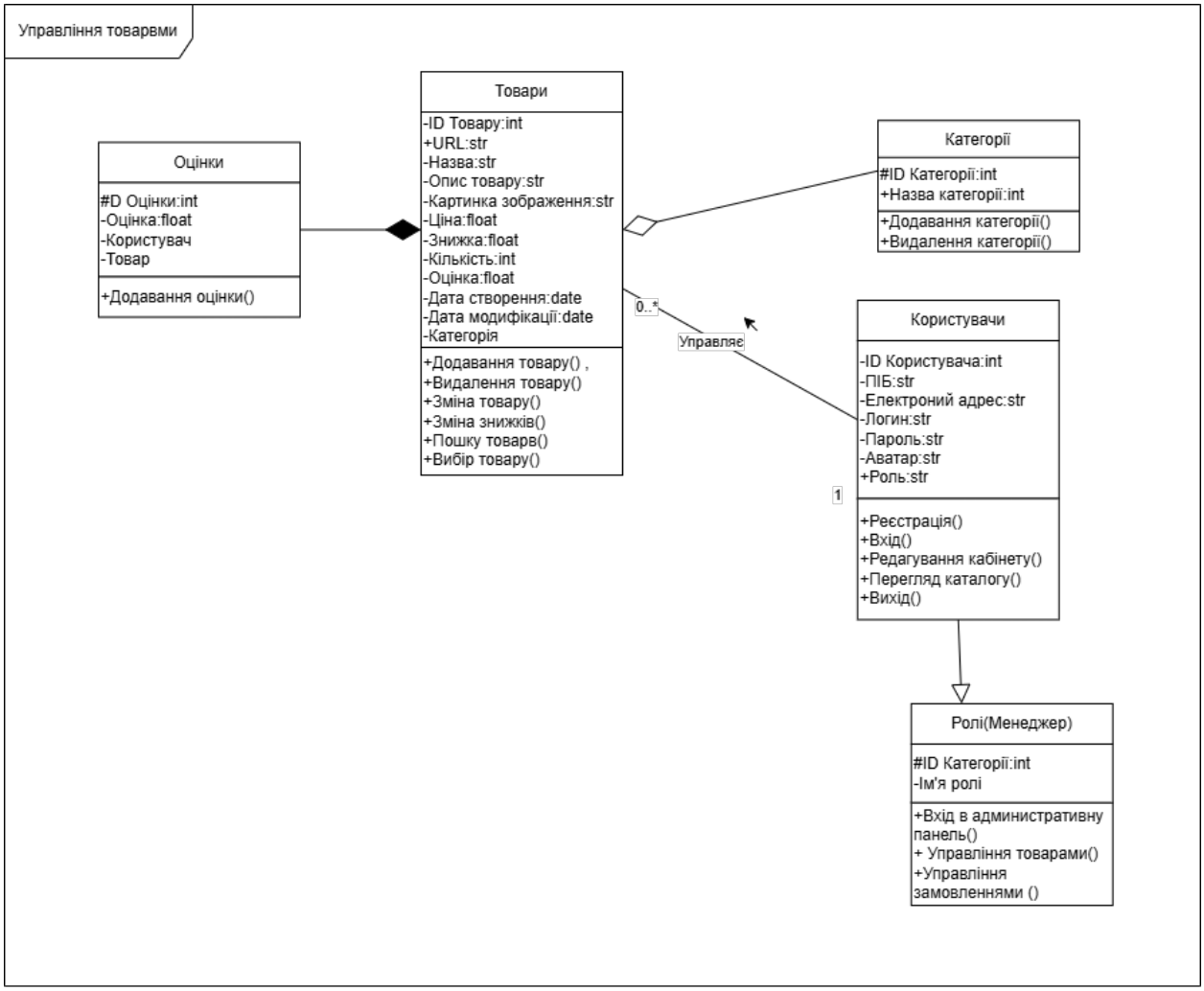


Рис.6 Діаграма кооперацій управління товарами

Вона описує структури взаємодії для управління даними про товар в системі.

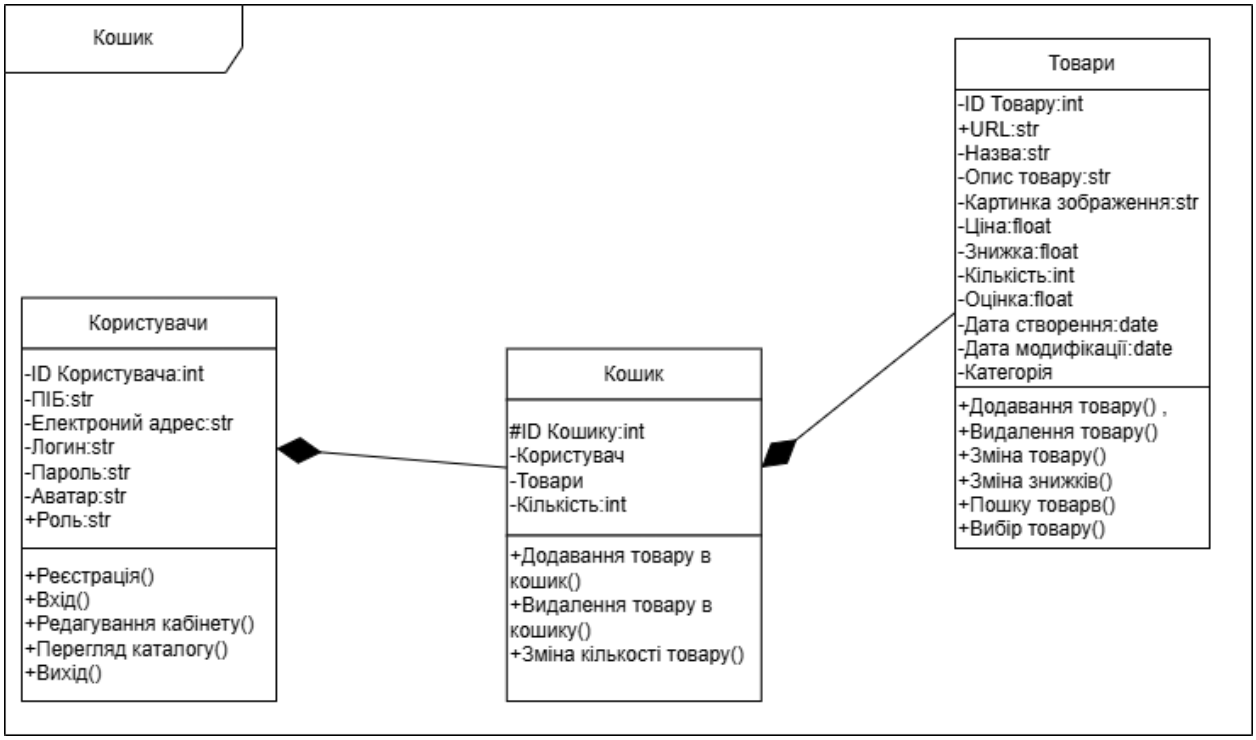


Рис.7 Діаграма кооперацій управління кошиком.

Вона показує взаємодію для управління даними про кількість товару в кошику користувача.

Повна діаграма кооперацій де можна її розглянути доступна в Додатку В.

2.3 Діаграма пакетів

Діаграма пакетів (Package Diagram) — це тип діаграми в UML, який використовується для організації великих систем у логічні групи (пакети). Вона допомагає побачити залежності між частинами програмної системи.

- **Для чого використовується діаграма пакетів:**
- **Структуризація проекту** — показує, як класи або модулі згруповані у пакети в середині системи.
- **Упорядкування коду** — допомагає спроектувати архітектуру так, щоб зменшити зв’язність і підвищити модульність в системі.
- **Візуалізація залежності** — показує, які пакети залежать один від одного, що важливо для супроводження, рефакторингу та масштабування.
- **Документація** — слугує частиною загального опису архітектури системи.

І вході розробки системи також була розроблена діаграма пакетів(Рис. 8) яка показує залежності між частинами програмної системи.

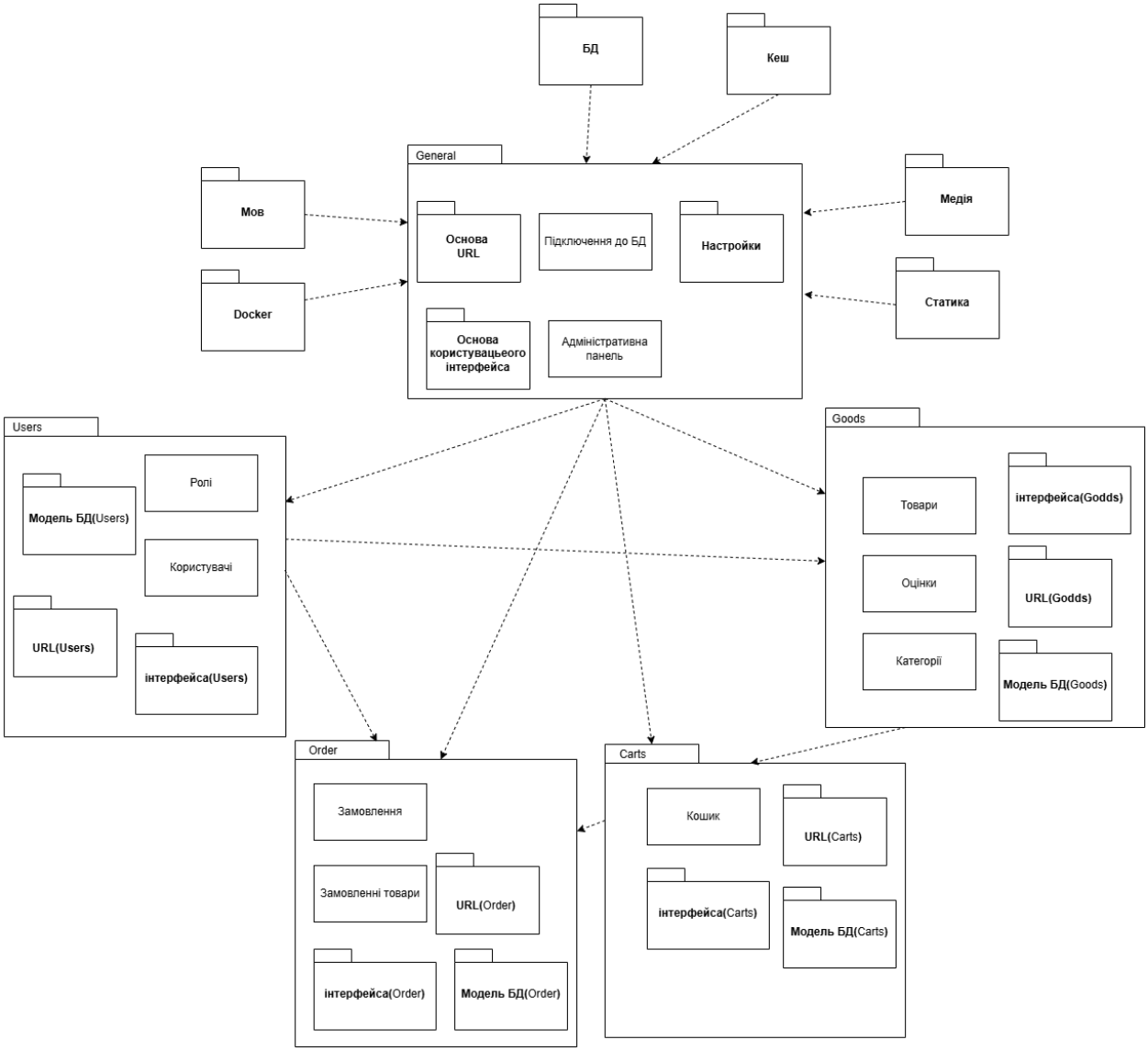


Рис. 8 Діаграма пакетів

Також ця діаграма зображує компоненти є в розробленої системи і взаємодію між ними.

2.4 Діаграма компонентів

Діаграма компонентів (Component Diagram) — це UML-діаграма, яка показує фізичні частини програмної системи — тобто компоненти, з яких вона складається, і як ці компоненти взаємодіють між собою.

«Діаграми компонентів UML надають концептуальну картину взаємодії між різними системами. Можуть бути присутні як аспекти логічного, так і фізичного моделювання. Крім того, компоненти автономні. Це модульний системний елемент в UML, який можна замінити на альтернативні. Вони містять конструкції будь-якої складності та є самодостатніми. Лише через інтерфейси вкладені частини взаємодіють з іншими компонентами. Крім того, компоненти мають свої інтерфейси, але вони також можуть отримати доступ до операцій і служб інших компонентів за допомогою їхніх інтерфейсів. На діаграмі компонентів інтерфейси також показують зв'язки та залежності в архітектурі програмного забезпечення»[1].

Для чого використовується діаграма компонентів:

- Опис архітектури системи на високому рівні
- Візуалізація компонентів програми та їхніх інтерфейсів
- Планування структури системи перед розробкою
- Комунікація між командами (розробка, тестування, DevOps)
- Документація для технічного супроводження та масштабування

І вході розробки системи для дипломного проекту було змодельована діаграма пактів(Рис. 10).

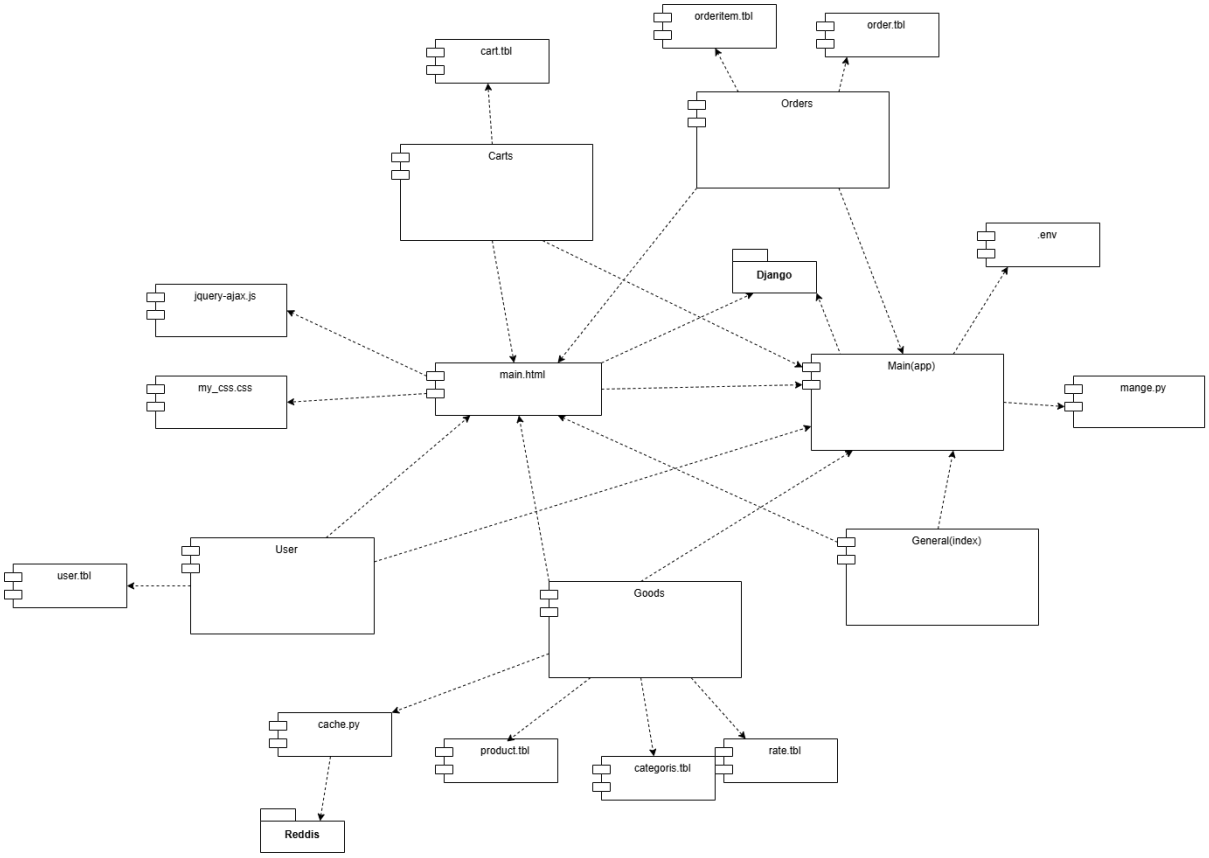


Рис. 10 Діаграма компонентів

В цій діаграми описуються і показуються компоненти фізичні компоненти в розробленої дипломної роботі і взаємодії між ними.

Також в ній вказано які розширення має той чи інший компонент в даної системі.

Із за того що фізичних компонентів за багато було прийнято рішення деяких з них об’єднати в один компонент, які мають спільний напрям функціоналу, чи в один пакет, тому вони не мають власного розширення.

2.5 Висновки до розділу 2

Під час опрацювання даного розділу була визначена структура системи, з чого вона буде складатись і як зберігати данні, і відповідно до них були побудовані відповідні UML діаграми, які показують структуру системи.

3 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Система управління інформаційною базою

Для того щоб розробити повноцінний програмний продукт потрібно вибрати інформаційну базу яка б управляла даними.

І після аналізу предметної області було прийняте рішення вибрати одну SQL БД для зберігання сесій і даних і одну додаткову NoSQL БД для зберігання хешированих даних щоб не завантажувати основну БД.

3.1.1 Вибір SQL бази даних

SQL БД (SQL база даних) — це система зберігання та обробки даних, яка використовує мову SQL (Structured Query Language) для управління інформацією.

«Структурована мова запитів (SQL) — це мова програмування для зберігання та обробки інформації в реляційній базі даних. Реляційна база даних зберігає інформацію в табличній формі, у якій рядки та стовпці представляють різні атрибути даних і різні зв’язки між значеннями даних. Ви можете використовувати оператори SQL для зберігання, оновлення, видалення, пошуку та отримання інформації з бази даних. Ви також можете використовувати SQL для підтримки та оптимізації продуктивності бази даних.

Структурована мова запитів (SQL) — це популярна мова запитів, яка часто використовується в усіх типах програм. Аналітики даних і розробники вивчають і використовують SQL, оскільки він добре інтегрується з різними мовами програмування. Наприклад, вони можуть вбудовувати запити SQL за допомогою мови програмування Java для створення високопродуктивних програм обробки даних з основними системами баз даних SQL, такими як Oracle або MS SQL Server. SQL також досить легко вивчити, оскільки він використовує загальні англійські ключові слова у своїх оператора»[11].

І для того щоб побудувати ефективне ПЗ потрібно підібрати підходящу БД і для цього потрібно розглянути декілька варіантів баз даних, і їх переваги і недоліки.

MySQL

MySQL — це система управління реляційною базою даних (RDBMS), яка використовує SQL для взаємодії з даними.

Вона є однією з найпопулярніших у світі і безкоштовною з відкритим кодом (open source). Входить до складу таких популярних стеків як LAMP (Linux, Apache, MySQL, PHP).

Для чого використовуються MySQL

- Веб-сайти (WordPress, Joomla);
- Інтернет-магазини (Magento, OpenCart);
- Мобільні додатки;
- Системи управління контентом;
- ERP/CRM рішення.

Переваги MySQL:

- Безкоштовна та з відкритим кодом;
- Можна використовувати без ліцензійних витрат;
- Швидка робота з великими об’ємами даних;
- Оптимізована для читання даних, ідеальна для веб-додатків;
- Проста у встановленні та використанні;
- Інтуїтивно зрозуміла, легко інтегрується з мовами програмування (PHP, Python, Java тощо) ;
- Кросплатформеність;
- Працює на Windows, Linux, macOS;
- Надійність та стабільність;

- Підтримує транзакції, реплікацію, резервне копіювання;
- Гарна документація і велика спільнота;
- Легко знайти допомогу та рішення для проблем;
- Підтримка індексів, тригерів, збережених процедур, подій.

Недоліки MySQL:

- Не завжди найкраще для дуже складних запитів або аналітики;
- У порівнянні з PostgreSQL чи спеціалізованими системами, аналітичні запити можуть працювати повільніше;
- Обмежена підтримка стандарту SQL;
- Не всі SQL-функції реалізовані повністю відповідно до стандарту;
- Менша гнучкість в роботі з великими транзакціями;
- Хоча підтримує транзакції (через InnoDB), у PostgreSQL вона глибша й потужніша;
- Наявність декількох типів таблиць (MyISAM, InnoDB) може збивати з пантелику;
- Не всі типи підтримують транзакції або зовнішні ключі;
- Закритий розвиток;
- Хоча MySQL — open-source, компанія Oracle керує розробкою, що викликає побоювання щодо майбутнього проєкту.

MariaDB

MariaDB — це система управління реляційними базами даних (RDBMS), яка базується на коді MySQL. Її створили розробники оригінального MySQL після того, як компанію MySQL придбала Oracle. Головна мета — залишити базу даних повністю відкритою та незалежною від великих корпорацій.

Для чого використовуються MariaDB

- Вебсайти, CMS (WordPress, Joomla) ;
- Електронна комерція (наприклад, Magento);

- ERP-системи;
- Хмарні сервіси;
- Аналітичні системи.

Переваги MariaDB:

- Повністю відкритий код (100% open source) ;
- Розробляється прозоро, без ризику “закриття” функцій, як у MySQL (Oracle) ;
- Сумісність з MySQL;
- Можна легко перейти з MySQL на MariaDB: ті ж запити, API, команди, драйвери;
- Зберігає зворотну сумісність з MySQL 5.x. ;
- Вища продуктивність у деяких сценаріях;
- Оптимізовано для кращої роботи з певними типами запитів, особливо з великими об’ємами даних;
- Більше функцій «з коробки» ;
- Наприклад, покращені рушії зберігання (Aria, ColumnStore, MyRocks), додаткові типи індексів, JSON-функції, віртуальні стовпці;
- Активна спільнота та постійний розвиток;
- Часто додаються нові фічі раніше, ніж у MySQL;
- Сумісність з Galera Cluster;
- Вбудована підтримка для створення кластерів високої доступності (HA).

Недоліки MariaDB:

- Поступова втрата повної сумісності з новими версіями MySQL;
- Нові версії MySQL (8.0 і далі) мають функції, які MariaDB може не підтримувати повністю;
- Менша кількість офіційних ресурсів від великих хостинг-провайдерів;
- Хоча MariaDB дуже популярна, іноді MySQL підтримується краще;

- Не всі сторонні інструменти одразу оптимізовані під MariaDB;
- Деякі фреймворки або програми можуть працювати краще з MySQL за замовчуванням;
- Може бути складніше знайти розробників, які спеціалізуються саме на MariaDB.

PostgreSQL

PostgreSQL — це потужна об’єктно-реляційна система управління базами даних (ORDBMS) з відкритим кодом. Вона підтримує як класичні реляційні дані (таблиці), так і нестандартні типи даних — JSON, масиви, XML тощо.

Це одна з найстабільніших, найгнучкіших і найфункціональніших СУБД з відкритим кодом, яку часто називають «The world's most advanced open source database».

Для чого використовуються PostgreSQL

- Банківські системи;
- Аналітика та обробка великих даних;
- Веб- та мобільні додатки;
- BI-системи;
- ГІС-додатки (географічні дані).

Переваги PostgreSQL:

- Повністю відкритий код, без комерційного впливу;
- Розробляється глобальною спільнотою, не належить жодній компанії;
- Висока відповідність стандартам SQL;
- Один з лідерів у дотриманні SQL-стандартів;
- Підтримка складних типів даних;
- Підтримка JSON, XML, масиви, геодані (PostGIS), UUID, кастомні типи;
- Потужна підтримка транзакцій;

- ACID-сумісність, підтримка MVCC (одночасна робота багатьох користувачів без блокувань) ;
- Підтримка збережених процедур і тригерів на різних мовах: PL/pgSQL, Python, Perl, SQL;
- Розширюваність;
- Можна створювати власні функції, типи даних, індекси;
- Паралельна обробка запитів;
- Ефективна робота з великими даними та аналітичними запитами;
- Активна спільнота, велика документація;
- Легко знайти рішення, підтримку, розширення.

Недоліки PostgreSQL:

- Складніша крива навчання для новачків;
- Не така інтуїтивна, як MySQL/MariaDB;
- Вища вимогливість до ресурсів;
- Потребує більше пам’яті та обчислювальної потужності (особливо при масштабуванні) ;
- Швидкість при простих запитах трохи нижча, ніж у MySQL;
- Якщо система не потребує складних операцій, PostgreSQL може бути «зайвим» ;
- Менше підтримки в деяких хостингах за замовчуванням.

Тому опираючись на аналіз розглянутих SQL БД було прийняте рішення вибрати об’єктно-реляційну базу даних PostgreSQL і ось декілька причин:

1. Відкритий код та безкоштовність

PostgreSQL є відкритим програмним забезпеченням, що означає:

- Безкоштовний доступ: ви не платите за ліцензії;

- Гнучкість: можна змінювати та налаштовувати код відповідно до своїх потреб.

2. Повна підтримка ACID

PostgreSQL підтримує ACID (Atomicity, Consistency, Isolation, Durability) – це гарантія цілісності даних при будь-яких збої в системі. Завдяки цьому бази даних у PostgreSQL надійно зберігають правильність даних навіть під час збоїв або відключень.

3. Розширена підтримка типів даних

PostgreSQL підтримує безліч вбудованих типів даних, зокрема:

- JSON і JSONB для зберігання структурованих даних;
- XML, UUID, масиви, Hstore (для пар ключ-значення);
- Географічні типи (через розширення PostGIS).

Це робить PostgreSQL ідеальним для роботи з різноманітними даними, від структурованих до неструктурованих

4. Висока продуктивність

PostgreSQL забезпечує високу продуктивність завдяки:

- Індексації (підтримка різних типів індексів, зокрема B-tree, GiST, GIN, BRIN);
- Кешуванню: дані зберігаються в пам'яті для швидшого доступу;
- Оптимізації запитів: розумне планування виконання запитів.

5. Повнотекстовий пошук

PostgreSQL має вбудовану підтримку повнотекстового пошуку, що дозволяє швидко і ефективно знаходити записи на основі текстових запитів, навіть у великих базах даних.

6. Надійність та масштабованість

Реплікація: PostgreSQL підтримує як фізичну, так і логічну реплікацію, що дозволяє створювати резервні копії та розподіляти навантаження на кілька серверів.

Масштабованість: PostgreSQL підтримує горизонтальне масштабування через партиціювання таблиць та шардинг.

7. Безпека

PostgreSQL надає потужні механізми безпеки:

- Шифрування з'єднань через SSL;
- Ролі та привілеї для детального управління доступом;
- Підтримка автентифікації через LDAP, Kerberos та інші методи.

8. Гнучкість у роботі з транзакціями

PostgreSQL підтримує:

- Транзакції з відкотом (rollback) при невдачі;
- Збережені процедури та тригери, які дозволяють автоматизувати процеси.

9. Активна спільнота

PostgreSQL має дуже активну та підтримуючу спільноту, що дає можливість:

- Швидко знаходити рішення на проблеми;
- Оновлювати систему.

10. Найбільш сумісна з мовою програмування Python

PostgreSQL має високу сумісність з мовою програмування Python і найбільш сумісна БД з фреймворком Django, який якраз написаний на мові програмування Python і через якого був розроблений проект

11. Об’єктна-реляційна база даних

Об'єктно-реляційна база даних (Object-Relational Database, ORDBMS) — це система, яка поєднує можливості реляційної моделі (таблиці, SQL, ключі) з концепціями об’єктно-орієнтованого програмування (класи, наслідування, інкапсуляція).

PostgreSQL являється об’єктна реляційною БД що дає йому деякі переваги такі як:

1.Найбільша сумісність з ORM системи, одна з яких був використаний в проекті

2.Підтримка складних типів даних. Можна зберігати масиви, JSON, геодані (наприклад, координати), зображення, аудіо, структури.

3.Наслідування. Таблиці можуть успадковувати структуру одна від одної, як класи у мовах програмування.

4.Інкапсуляція логіки. Можна створювати збережені процедури, функції, тригери — логіка БД виконується «всередині», як методи класів.

5.Підтримка об'єктних конструкцій. Створення власних типів даних, агрегатів, операторів, індексів.

6.Краща відповідність до об'єктно-орієнтованого коду. Полегшує роботу з БД для розробників, які працюють з мовами типу Java, Python, C#.

7.Потужна аналітика. Можна ефективно обробляти великі обсяги складної, структурованої інформації.

8.Розширюваність. Можна додавати нові можливості до БД без зміни ядра (розширення, плагіни).

9.Гнучкість. Підходить як для класичних табличних даних, так і для сучасних застосунків (змішані структури, напівструктуровані дані).

Тому основуючись на ці переваги і була вибрана ця об’єктна-реляційна база даних.

3.1.2 Вибір NoSQL бази даних

Із за того що під час роботи сайту при оновлені сторінки чи переході на іншу система постійно відправляє запит на поучення даних до БД, що сильно навантажує саму базу даних, змушуючи постійно виконувати одні й ти самі запити хоча відповіді не відрізняються. Що призводить до того що БД починає виконувати лишню роботу, що зменшує її працездатність

І тому для вирішення цієї проблеми і використовують хешування і NoSQL БД заточених по це.

Ці БД дозволяють зберігати данні в хеш-таблицях і получати їх через ключи, що забезпечує швидку отримання даних.

Їм не потрібно пробігати всю БД щоб знайти якийсь елемент, їм достатньо перейти до потрібних даних через хеш-ключ

Що робить їх хорошим вибором для додаткової БД, щоб зберігати популярні запроси до БД розгрузивши основну базу даних і прискоривши роботи всієї системи.

І ось декілька варіантів NoSQL БД для розгляду:

Reddis

Redis (REmote DIctionary Server) — це база даних типу key-value, яка зберігає дані в оперативній пам’яті (RAM), а не на диску. Вона дуже швидка,

легка і часто використовується як кеш, черга повідомлень, або навіть тимчасове сховище даних.

Redis підтримує різні типи структур даних: рядки, списки, множини, хеші, упорядковані множини, бітові поля тощо.

Для чого використовуються Redis

- Кешування результатів запитів;
- Зберігання сесій користувачів;
- Реалізація черг завдань (наприклад, у фонових процесах) ;
- Рейтинг-системи, лічильники ;
- Pub/Sub системи (реактивні повідомлення в реальному часі).

Переваги Redis:

- **Дуже висока швидкість**. Працює з оперативною пам'яттю, що в тисячі разів швидше за дискові БД;
- **Простота**. Надзвичайно простий синтаксис: ключ → значення;
- **Підтримка різноманітних структур даних**. Списки, множини, хеші, черги, тощо — можна будувати складні логіки;
- **Кешування**. Ідеальний для кешу, підтримує TTL (час життя ключа), автоочищення;
- **Pub/Sub система**. Підтримує механізм «публікація-підписка», що корисно для чатів, повідомлень;
- **Зменшення навантаження на основну БД**. Наприклад, збереження часто запитуваних даних;
- **Підтримка атомарних операцій**. Безпечна обробка одночасних змін (важливо для лічильників, транзакцій);
- **Можливість збереження на диск (опційно).** Хоча Redis — in-memory, дані можна зберігати на диск у фоновому режимі (RDB, AOF).

Недоліки Redis:

- **Обмежена кількість пам’яті**. Працює в оперативній пам’яті, тому не підходить для великих обсягів даних без додаткової оптимізації;
- **Не реляційна БД**. Не підтримує складні SQL-запити, зв’язки між даними, транзакції як у класичних БД;
- **Тимчасове сховище (в більшості випадків)**. Часто використовується як кеш, а не як постійна база даних;
- **Дані можуть бути втрачені**. Якщо не налаштоване збереження на диск, при перезапуску Redis — дані зникають;
- **Вимагає ретельного моніторингу та налаштування**. Щоб не “переповнити” пам’ять або не втратити важливі дані;

MongoDB

MongoDB — це нереляційна база даних (NoSQL), яка зберігає дані у вигляді документів формату BSON (розширений JSON). Це означає, що замість таблиць з рядками та стовпцями, як у SQL-базах, MongoDB використовує гнучку структуру — документи (об’єкти) зі змінними полями.

Для чого використовуються MongoDB

- Веб-додатки (Node.js, React, Angular);
- Розподілені системи;
- Big Data та аналітика;
- Мобільні додатки;
- IoT (IoT), стрімінгові сервіси;
- Платформи з динамічними або складними структурами даних.

Переваги MongoDB:

- **Гнучка схема (schema-less).** Дані не прив’язані до фіксованої структури — кожен документ у колекції може мати різну структуру. Дуже зручно при змінюваних або складних моделях даних;
- **Висока швидкість при роботі з великими обсягами.** Швидке читання/запис документів без складних зв’язків;
- **Масштабованість.** Легко масштабується горизонтально (через шардинг).
- **Зберігання складних вкладених об’єктів.** Можна зберігати масиви, вкладені документи — ідеально для зберігання JSON-подібних структур;
- **Зручна інтеграція з сучасними мовами програмування.** Добре поєднується з JavaScript, Node.js, Python тощо;
- **Реплікація та відмовостійкість.** Підтримує replica sets для забезпечення безперебійної роботи;
- **Вбудовані механізми агрегації.** Потужна pipeline-агрегація для обробки даних.

Недоліки MongoDB:

- **Відсутність транзакцій.** У старих версіях не підтримувались повноцінні ACID-транзакції. Нові — підтримують, але не так потужно як SQL;
- **Відсутність зв’язків типу JOIN.** Немає нативної підтримки JOIN'ів, як у реляційних БД (можна симулювати через $lookup, але це повільніше) ;
- **Надлишковість даних.** Через денаормалізацію (повторення даних), може бути перевитрата пам’яті та місця;
- **Складніша валідація.** Відсутність чіткої схеми може спричинити помилки при неправильному введенні даних;
- **Не завжди підходить для фінансових або строго структурованих систем.** Наприклад, банківські системи краще реалізовувати у класичних SQL-БД.

Elasticsearch

Elasticsearch — це розподілена пошукова та аналітична база даних з відкритим кодом, яка дозволяє дуже швидко зберігати, шукати та аналізувати великі обсяги текстової або логічної інформації в реальному часі.

Для чого використовуються Elasticsearch:

- Логи (журнали подій) ;
- Тексти, документи;
- Телеметрію з пристроїв (IoT);
- Веб-аналітику;
- Дані моніторингу, метрики;
- JSON-документи (подібно до MongoDB).

Переваги Elasticsearch:

- **Блискавично швидкий пошук.** Підтримує повнотекстовий пошук (full-text search) із релевантністю, підсвіткою, автодоповненням тощо;
- **Потужна аналітика.** Агрегації, фільтри, групування, часова аналітика — у реальному часі;
- **Масштабованість.** Розподілена архітектура — легко працює на багатьох вузлах (кластері) ;
- **JSON API.** Комунікація через REST API з даними у форматі JSON — легко інтегрується з будь-якою мовою програмування;
- **Інтеграція з Logstash, Beats, Kibana (ELK-стек).** Повна екосистема для збирання, обробки та візуалізації даних;
- **Гнучка схема даних.** Не потрібно заздалегідь жорстко описувати структуру;
- **Пошук у будь-якому полі, фільтрація, autocomplete, fuzzy search.** Надзвичайно зручний пошук навіть у великих масивах неструктурованих даних.

Недоліки Elasticsearch:

- **Високі вимоги до ресурсів.** Потребує багато пам’яті (RAM) та CPU, особливо при великих обсягах даних або агрегаціях;
- **Зберігання не оптимальне для критичних даних.** Elasticsearch не є транзакційною БД — не гарантує 100% збереження даних, як SQL-БД;
- **Складність налаштування.** Потрібно ретельно налаштовувати кластер, індекси, реплікації, мапінги;
- **Може втратити дані при аваріях.** Без правильної конфігурації кластера і реплік — ризик втрати даних;
- **Повільне оновлення документів.** Сильніше оптимізований для читання/ додавання, ніж для змін (update/delete) ;
- **Комерційна частина.** Elastic має ліцензії для деяких розширених функцій, не все — open source;

Тому опираючись на аналіз розглянутих NoSQL БД було прийняте рішення вибрати об’єктно-реляційну базу даних Redis і ось декілька причин:

1. **Легкість.** Redis являється простою БД, яка немає великого функціонала що робить її легкою в плані споживання ресурсів. А так як NoSQL БД буде використовуватися лише для хешування тому функціонала redis достатньо;
2. **Надзвичайно висока швидкість.** Redis зберігає всі дані в оперативній пам’яті (RAM), що забезпечує відгук у мілісекундах. Підходить для задач, де критична швидкість — наприклад, кешування результатів запитів, лічильники кліків, сесії користувачів;
3. **Простота моделі даних**: key-value. Кожен запис — це пара ключ → значення, як у звичайному словнику. Дуже легко працювати, особливо в мікросервісах, API або реальному часі;
4. **Підтримка багатьох типів даних.** Redis — це не просто key-value: він підтримує багаті структури, це дозволяє використовувати Redis для складної логіки — черг, кешів, лідербордів, тощо;

5. **Підтримка TTL (Time to Live).** Можна задати час життя для кожного ключа. Після закінчення TTL ключ автоматично видаляється — ідеально для кешу або тимчасових токенів, які використовуються в системі;
6. **Атомарність операцій.** Redis гарантує, що кожна команда виконується атомарно. Це важливо при обробці одночасних запитів, наприклад для контролю транзакцій;
7. **Підтримка реплікації і кластеризації.** Redis може масштабуватися горизонтально, використовуючи кластери. Підтримує майстер-репліка моделі, автоматичне перемикання в разі відмови (Redis Sentinel) ;
8. **Легка інтеграція з різними мовами.** Redis має інтеграцію майже для всіх популярних мов: Python, Node.js, Java, Go, PHP, .NET тощо;
9. **Зменшення навантаження на основну базу даних.** Redis чудово працює як проксі-кеш для SQL або NoSQL баз — запити, що часто повторюються, обробляються з RAM, не навантажуючи основну систему.

Так що для розробки системи були обрані БД PostgreSQL для основної бази даних, яка буде відповідати за всі запити і транзакцій, і Redis як другорядну БД для кеширування популярних запити, для того щоб зменшити навантаження на основну БД.

3.2 Розробка інформаційної бази

Для того щоб облегшити розробку системи і зробити її більш захищеною, для розробки ІБ була використана ORM система.

«Об’єктно-реляційне відображення (ORM) — це спосіб узгодження програмного коду зі структурами бази даних . ORM використовує дескриптори метаданих для створення рівня між мовою програмування та реляційною базою даних. Таким чином, він з’єднує код об’єктно-орієнтованої програми (ООП) з базою даних і спрощує взаємодію між реляційними базами даних і мовами ООП.

Ідея ORM базується на абстракції . Механізм ORM дає змогу адресувати об’єкти , отримувати доступ до них і маніпулювати ними, не враховуючи, як ці

об’єкти пов’язані зі своїми джерелами даних. ORM дозволяє програмістам підтримувати узгоджене уявлення про об’єкти протягом тривалого часу, навіть якщо змінюються джерела, які їх доставляють, приймачі, які їх отримують, і програми, які отримують до них доступ»[2].

Переваги ORM:

- **Простота використання.** Замість написання SQL-запитів — працює з об’єктами та методами;
- **Менше рутини (boilerplate-коду).** Створення, оновлення, видалення записів — простими методами через код. Автоматично генеруються запити до БД;
- **Безпечніше — захист від SQL-ін’єкцій.** ORM автоматично екранує значення — не потрібно вручну турбуватися про це.
- **Абстракція над БД.** Можна змінити СУБД (наприклад, з SQLite на PostgreSQL) — без змін у логікі коду. ORM ізолює від специфіки SQL-діалектів, які мають кожна БД, що дозволяє не заучувати кожну БД;
- **Легке відображення зв’язків між таблицями.** Зв’язки типу один до багатьох, багато до багатьох — через властивості, а не через JOIN'и;
- **Тестованість.** ORM сприяє unit-тестуванню, бо не потрібно взаємодіяти напряму з SQL;
- **Міграції.** Більшість ORM підтримують автоматичні або напівавтоматичні міграції (створення та оновлення структури БД із коду).

Недоліки ORM:

- **Продуктивність.** ORM — це додатковий шар абстракції, тому складні запити можуть бути повільніші, ніж написані вручну на SQL. Іноді він генерує неоптимальний SQL запити;
- **«N+1» проблема.** Часто ORM виконує багато маленьких запитів замість одного JOIN-запиту, що знижує продуктивність;

- **Обмежена гнучкість.** У складних SQL-запитах з багатьма JOIN, агрегатами, підзапитами — ORM може бути обмеженим або незручним.
- **Складність дебагу.** ORM приховує SQL — важко зрозуміти, що саме пішло не так або який запит був виконаний, додатково потрібно підключати інші інструменти що вирішити цю проблему;
- **Необхідність навчання.** У кожного ORM — своя філософія, API, міграції, синтаксис — потрібен час щоб вивчити це все;
- **Втрата контролю над SQL.** Іноді потрібно «опуститися на рівень SQL», бо ORM не підтримує всіх можливостей СУБД (наприклад, специфічні індекси, тригери, CTE).

ORM — це чудовий інструмент для:

- Швидкого старту проекту;
- Стандартних CRUD-операцій;
- Читабельного та безпечного код.

Але У проектах з високими вимогами до продуктивності або складною логікою SQL-запитів, ORM краще поєднувати з ручними SQL-запитами або ORM з «raw SQL» підтримкою.

Для створення ІБ в проекті було обрано ORM, яка вже є в фреймворке Django, а конкретно Django ORM. Ця ORM дозволяє працювати з базою даних через Python-класи замість написання SQL-запитів вручну.

Django ORM перетворює класи Python на таблиці БД, а об’єкти на рядки в таблицях, і дозволяє взаємодіяти з ними через Python-код.

Для того щоб почати створювати інформаційну базу на основі баз даних через ORM потрібно розробити саму БД і для цього був використаний Docker. Він дозволяє запускати локальні машини, які в Docker називаються контейнерами, позволяючи встановлювати декілька БД на основну машину не

“засмічуючи” її. А також позволяє потім швидко і легко розвертати і масштабувати контейнери на інші сервера лише пару, або навіть однією командою.

Так що спочатку я створюю файл docker-compose.yml де вказуються параметри контейнера(Рис. 11).

```
db:
  container_name: ${POSTGRESQL_CONT_NAME}
  ports:
    - "${POSTGRESQL_PORT}:5432"
  environment:
    - POSTGRES_PASSWORD=${POSTGRESQL_PASSWORD}
    - POSTGRES_USER=${POSTGRESQL_USER}
  image: postgres:${POSTGRESQL_IMAGE}

  volumes:
    - ./bddata:/var/lib/postgresql/data
```

Рис. 11 docker-compose.yml для встановлення PostgreSQL

Де:

- **Container_name** –це назва контейнера.
- **Ports** – це перекидання портів з основної машини до контейнера. Порт 5432 – стандартний порт для PostgreSQL, який також встановлена в БД всередині контейнера.
- **Environment** – це змінні середовища контейнера. Наприклад в даному контейнері туди вписуються root користувача і його пароль. І під час створення БД цей користувач створиться автоматично.
- **Image** – це образ на основі якого буде створений контейнер. Образ був взятий офіціальний образ PostgreSQL, яка находилась в каталоге Docker.

- **Volumes** – це перекидання файлів з каталогу в середині Docker в каталог на основною машиною і навпаки. Що дозволяє просто маніпулювати цими даними не через віртуальну машину в Docker, а через основну машину, що дозволяє розробникам або іншим, кому потрібні ці файли, не лізти в ВМ.

Після того як була створена БД і створено root акаунт, потрібно з цією БД якось взаємодіяти бо в контейнері докера немає ніякого адмінера для керування нею, а якщо й була то користуватись нею було б складно бо в контейнерах докера зазвичай встановлений ОС з необхідним мінімум для повноцінного функціювання БД і лише його.

Тому для того щоб взаємодіяти з БД в редакторе коду, на якому було розроблена система, була додана розширення DatabaseClient. Це розширення дозволяє підключатись до віддаленого БД і керувати ним через редактора коду. Це дозволяє швидко переключатись між кодом і БД не використовуючи сторонні програми.

Недоліком цього розширення є те що в неї є premium, яка обмежує базовий функціонал безплатної версії. Наприклад обмеження кількість підключень до SQL і NoSQL БД. Але для проекту достатньо базового функціонала.

Отже після встановлення і запуска PostgreSQL в контейнере, підключаю її до редактора коду через розширення(Рис. 12).

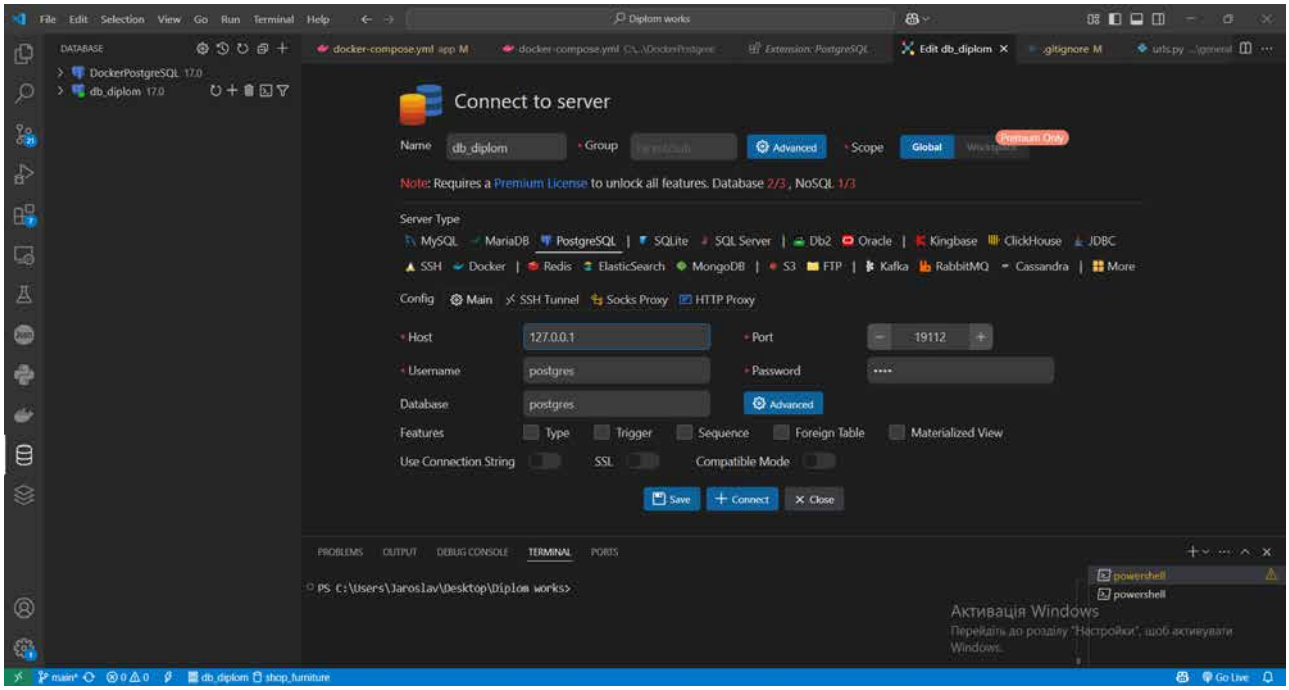


Рис. 12 Підключення до сервера

Після чого через редактор коду можна взаємодіяти з сервером через спеціальний інтерфейс розширення (Рис. 13).

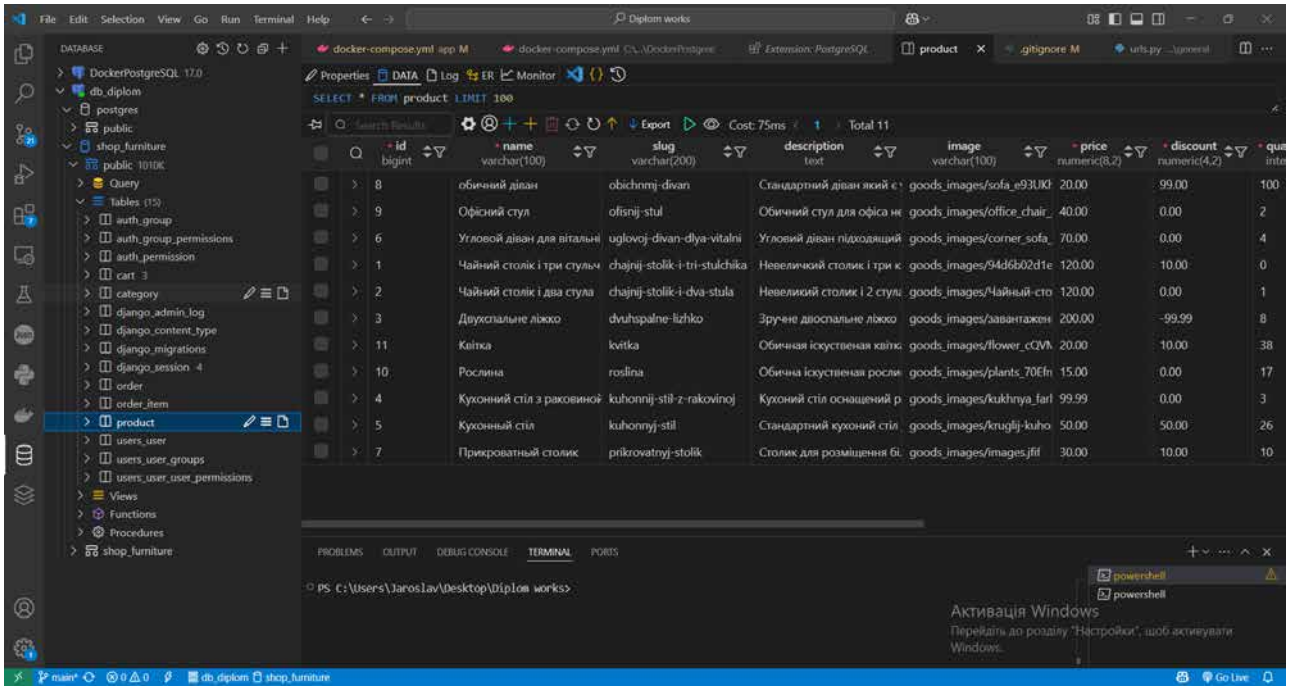


Рис. 13 Інтерфейс розширення

Після того як була встановлений PostgreSQL і підключення адмінера до нього тепер потрібно створити роль для конкретної БД (Рис. 14) щоб

розмежувати можливість взаємодії з основним сервером під час підключень і для цього якраз і був використаний встановлене розширення.

```
CREATE ROLE meneger WITH
    LOGIN
    SUPERUSER
    CREATEDB
    CREATEROLE
    INHERIT
    NOREPLICATION
    CONNECTION LIMIT -1
    PASSWORD '123456789';
```

Рис. 14 Створення ролі для майбутньої БД

Після того як роль для майбутньої БД була створена можна вже створити і саму базу даних через SQL запит (Рис. 15).

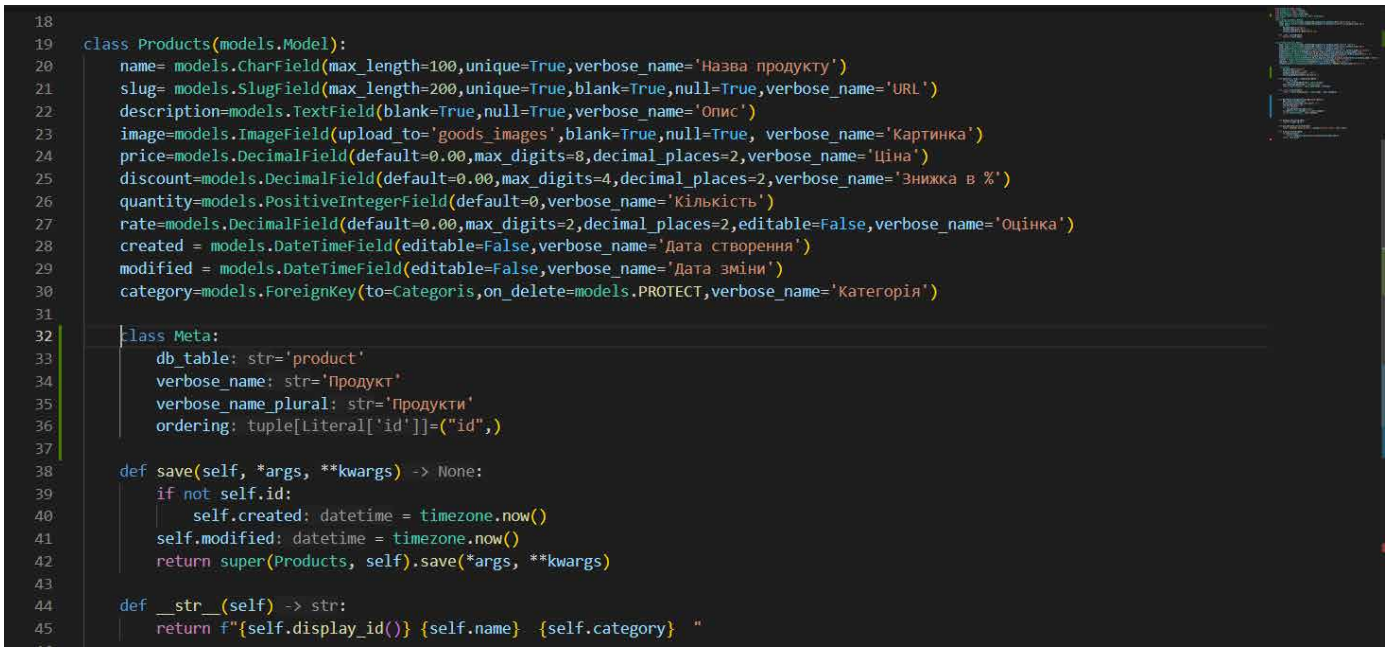
```
CREATE DATABASE shop_furniture
    WITH
    OWNER = meneger
    ENCODING = 'UTF8'
    LOCALE_PROVIDER = 'libc'
    CONNECTION LIMIT = -1
    IS_TEMPLATE = FALSE;
```

Рис. 15 Створення ролі для майбутньої БД

В цьому SQL запроси створюється основна БД для проекту, власником якої є попередньо створена роль.

Після чого в проекті створюються моделі таблиць які потім будуть мігрувати до БД. Ця модель забезпечить зв’язок між представленням в образі моделі і БД.

Наприклад для проекту потрібна таблиця яка б описувала товари(Рис. 16) і їх категорії. І замість того щоб напряму створювати запроси ORM системи дозволяють створювати їх не через SQL код а й на інших мовах.



```python

class Products(models.Model):
    name= models.CharField(max_length=100,unique=True,verbose_name='Назва продукту')
    slug= models.SlugField(max_length=200,unique=True,blank=True,null=True,verbose_name='URL')
    description=models.TextField(blank=True,null=True,verbose_name='Опис')
    image=models.ImageField(upload_to='goods_images',blank=True,null=True, verbose_name='Картинка')
    price=models.DecimalField(default=0.00,max_digits=8,decimal_places=2,verbose_name='Ціна')
    discount=models.DecimalField(default=0.00,max_digits=4,decimal_places=2,verbose_name='Знижка в %')
    quantity=models.PositiveIntegerField(default=0,verbose_name='Кількість')
    rate=models.DecimalField(default=0.00,max_digits=2,decimal_places=2,editable=False,verbose_name='Оцінка')
    created = models.DateTimeField(editable=False,verbose_name='Дата створення')
    modified = models.DateTimeField(editable=False,verbose_name='Дата зміни')
    category=models.ForeignKey(to=Categoris,on_delete=models.PROTECT,verbose_name='Категорія')

    class Meta:
        db_table: str='product'
        verbose_name: str='Продукт'
        verbose_name_plural: str='Продукти'
        ordering: tuple[Literal['id']]=("id",)

    def save(self, *args, **kwargs) -> None:
        if not self.id:
            self.created: datetime = timezone.now()
        self.modified: datetime = timezone.now()
        return super(Products, self).save(*args, **kwargs)

    def __str__(self) -> str:
        return f"{self.display_id()} {self.name}  {self.category}  "
```

Рис. 15 Модель таблиці продуктів

Де клас продуктів описує саму таблицю і їх полів. Клас Мета відповідає за мета дані такі як: назву таблиці в БД, назву системи в системі, наявнсть автоматичного сортування. Функції які є в середині класу відповідають за обробку даних в моделі.

Також крім цієї моделі також було створено моделі які відповідають за замовлення(Рис. 16).

```python
class ENUM:
    OrderStatus: tuple[tuple[str, str], tupl...=(
        ('В обробці' , "В обробці"),
        ('Відправлено' , "Відправлено"),
        ('Доставлено' , "Доставлено"),
        ('Получено' , "Получено")
    )

class Order(models.Model):
    user=models.ForeignKey(to=User,on_delete=models.SET_DEFAULT,null=False,blank=True, verbose_name="Користувач",default=None)
    created_timestamp = models.DateTimeField(auto_now_add=True,verbose_name="Дата замовлення")
    phone_number=models.CharField(max_length=20,verbose_name="Номер телефона")
    requires_delivery=models.BooleanField(default=False,verbose_name="Потрібна доставка")
    delivery_adress=models.TextField(null=True,blank=True,verbose_name="Адрес доставкі")
    payment_on_get=models.BooleanField(default=False,verbose_name="Оплата при доставці")
    is_paid=models.BooleanField(default=False,verbose_name="Заплачено")
    status=models.CharField(max_length=50, choices=ENUM.OrderStatus, default=ENUM.OrderStatus[0], verbose_name="Статус замовлення"


    class Meta:
        db_table: str="order"
        verbose_name: str="Замовлення"
        verbose_name_plural: str="Замовлення"

    def __str__(self) -> str:
        return f"Замовлення номер {self.pk} | Покупець {self.user.first_name} {self.user.last_name}"
```

Рис. 16 Модель таблиці Замовлень

А також модель яка відповідає за кошики товарів(Рис. 17) які мають користувачі і в яких є товар.

```python
class Cart(models.Model):

    user = models.ForeignKey(to=User,on_delete=models.CASCADE, blank=True, null=True, verbose_name='Користувач')
    product = models.ForeignKey(to=Products,on_delete=models.CASCADE, verbose_name='Продукт')
    quantity = models.PositiveSmallIntegerField(default=0, verbose_name='Кількість')
    created_timestamp=models.DateField(auto_now_add=True, verbose_name='Дата додавання')
    session_key=models.CharField(max_length=32,null=True,blank=True)

    class Meta:
        verbose_name: str="Корзину"
        verbose_name_plural: str="Корзини"
        db_table: str='cart'

    objects: Manager=CartQueryset().as_manager()
    def __str__(self) -> str:
        if self.user:
            return f'Корзина {self.user.username} | Товар {self.product.name} | Кількість {self.quantity}'
        else:
            return f'Анонимна корзина | Товар {self.product.name} | Кількість {self.quantity}| Сесіоний ключ {self.session_key}'


    def products_total_price(self) -> Any:
        return round(self.product.d_price() * self.quantity,2)
```

Рис. 17 Модель таблиці Кошика

Та інші моделі. Повний список моделі їх атрибутах і їх функціях доступний в додатку А.

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

3.3.1 Вибір мови програмування

Для того щоб створити хоча б якійсь проект потрібна інструментарії, де й буде створюватись прикладна програмне середовище.

Першу чергу потрібно з'ясувати на якій мові програмування буде написаний проект. Є багато різних мов програмування і в кожного з них є свої плюси і недоліки. І ось декілька мов програмування які були розглядані.

JavaScript/TypeScript

«JavaScript — це мова сценаріїв або мова програмування, яка дозволяє вам реалізовувати складні функції на веб-сторінках — щоразу, коли веб-сторінка робить щось більше, ніж просто стоїть там і відображає статичну інформацію, на яку ви можете дивитися — відображає своєчасні оновлення вмісту, інтерактивні карти, анімовану 2D/3D-графіку, музичні автомати з прокручуванням відео тощо — ви можете посперечатися, що JavaScript, ймовірно, задіяний»[10].

Також ця мова використовується не тільки для фронденд програмування, але й для бекенда в деяких фреймворках таких як Node.js. Що робить його універсальною мовою програмування сайтів.

В свою чергою TypeScript являється надбудовою JavaScript, яка добавляє типізацію коду в цьому мові програмування. Що збільшує читання коду і зменшує виниканню помилок під час розробки системи.

Переваги JavaScript:

- **Працює прямо в браузері.** Не потребує додаткового встановлення — достатньо браузера. Дає змогу одразу бачити результат.
- **Кросплатформеність.** Працює в усіх сучасних браузерах і на будь-якій операційній системі.

- **Інтерактивність**. Дає змогу реалізовувати анімації, форми, динамічне оновлення контенту без перезавантаження сторінки (через AJAX, Fetch API).
- **Велика спільнота.** Безліч бібліотек і фреймворків: React, Vue, Angular, jQuery. Багато прикладів, документації, підтримка на форумах.
- **Асинхронність.** Підтримка callback-функцій, промісів, async/await — для зручної роботи з асинхронними операціями.
- **Можна використовувати і на сервері.** Завдяки Node.js, JavaScript можна використовувати для бекенду.

Недоліки JavaScript:

- **Динамічна типізація.** Типи змінних не перевіряються під час написання коду, що може призводити до помилок на етапі виконання.
- **Менше контролю над помилками.**
- **Безпека.** Як мова, що виконується на стороні клієнта, може бути вразливою до атак, якщо не дотримуватися стандартів безпеки (наприклад, XSS).
- **Несумісність між браузерами (інколи).** Іноді нові можливості мови не підтримуються старими браузерами.
- **Складність масштабування.** Без додаткових інструментів структура коду великого проекту може стати неорганізованою.

JavaScript — це потужна мова для розробки веб-інтерфейсів, яка дозволяє створювати динамічні та інтерактивні сторінки. Його основна сила — простота старту та широке використання, а основні слабкі сторони — динамічна типізація та менша передбачуваність у великих проектах.

C#

C# — це мова програмування, розроблена Microsoft, яка найчастіше використовується разом із ASP.NET для створення багатофункціональних веб-

додатків. Її часто застосовують для розробки корпоративних сайтів, CRM-систем, порталів, API та інших серверних частин.

Переваги C#:

- **Потужна платформа — ASP.NET.** ASP.NET Core — сучасний, швидкий і кросплатформений фреймворк для веб-розробки;
- **Підтримка** MVC, Razor Pages, Blazor (фреймворк для фронтенду на C#);
- **Статична типізація.** Більше контролю над кодом. Помилки знаходяться на етапі компіляції;
- **Продуктивність і масштабованість.** Добре підходить для високонавантажених систем;
- **Сильна інтеграція з Microsoft-екосистемою.** Ідеально підходить для роботи з Azure, MS SQL Server, Active Directory, Office API тощо;
- **Інструменти розробки.** Visual Studio — один з найпотужніших IDE з автодоповненням, відладчиком, дизайнерами форм тощо;
- **Безпека.** Платформа ASP.NET має вбудовані механізми захисту від XSS, CSRF, SQL-ін’єкцій;
- **Підтримка багатопоточності.** Зручно працювати з асинхронними операціями (через async/await) ;

Недоліки C#:

- **Вища складність старту.** Порівняно з JavaScript чи PHP, почати розробку може бути складніше через конфігурацію проєкту;
- **Ресурсоємність.** Додатки на .NET можуть вимагати більше пам’яті і ресурсів, ніж легші рішення на PHP або Node.js;
- **Менша популярність у вебі.** Не такий великий пул веб-фреймворків або шаблонів, як у JavaScript;
- **Більше налаштувань.** Потрібно конфігурувати сервер, залежності, середовище, особливо при деплої;

- **Залежність від Microsoft.** Хоча .NET Core кросплатформений, частина інструментів або сервісів все ще оптимізована під Windows.

C# — відмінний вибір для серйозних, стабільних, корпоративних сайтів або веб-сервісів, які потребують високої продуктивності, безпеки та підтримки великого коду. Але для маленьких проектів або швидких MVP може бути трохи важким.

Python

Python — це високорівнева мова програмування, відома своєю простотою, читабельністю та широким спектром застосування: від вебу до машинного навчання. У веб-розробці Python найчастіше використовується разом із фреймворками Django або Flask.

Переваги Python:

- **Простий і зрозумілий синтаксис.** Код легко читати і писати, що пришвидшує розробку;
- **Потужні веб-фреймворки.** Django — фреймворк "все в одному" з авторизацією, ORM, безпекою, шаблонізатором тощо. Flask — мінімалістичний, гнучкий фреймворк для швидких проектів і API;
- **Швидкий старт.** Мінімум коду для запуску простого сайту або API;
- **Велика кількість бібліотек.** Багато пакетів для баз даних, безпеки, REST API, інтеграції з іншими сервісами;
- **Активна спільнота.** Багато туторіалів, документації, рішень на Stack Overflow;
- **Чудова інтеграція з наукою/AI.** Якщо треба інтегрувати машинне навчання, аналітику чи скрипти — Python ідеальний вибір;
- **Кросплатформеність.** Працює на Windows, Linux, macOS.

Недоліки Python:

- **Менша швидкодія.** У порівнянні з C# чи Java, Python працює повільніше (інтерпретована мова);
- **Не працює з фронтендом.** Використовується лише для бекенду, для клієнтської частини потрібен JavaScript;
- **Проблеми з масштабування.** Django добре масштабується, але Python в цілому не так оптимізований під високі навантаження, як Go чи Node.js;
- **Проблеми з багатопоточністю (GIL).** Обмеження Global Interpreter Lock ускладнює реалізацію деяких високопродуктивних задач;
- **Менша популярність серед хостингів.** У порівнянні з PHP або Node.js, підтримка Python у бюджетних хостинг-провайдерів менш.

Python — чудовий вибір для створення сайтів, REST API, внутрішніх сервісів або проектів із машинним навчанням. Якщо тобі потрібен швидкий старт, чистий код і багатофункціональний фреймворк, — Django або Flask на Python стануть чудовим варіантом.

Отже розглянувши мови програмування були вибрані такі мови програмування як Python для програмування бекенда, і мова програмування JavaScript для програмування фронтенда.

Також для фреймворка бекенда був вибраний framework Django. А для JavaScript був вибраний JQuery.

«Django — це високорівневий веб-фреймворк Python, який забезпечує швидку розробку безпечних веб-сайтів, які зручно підтримувати. Створений досвідченими розробниками, Django бере на себе більшу частину клопоту веб-розробки, тому можна зосередитися на написанні свого додатка без потреби винаходити колесо. Він безкоштовний із відкритим кодом, має процвітаючу та активну спільноту, чудову документацію та багато варіантів безкоштовної та платної підтримки»[3].

Переваги Django:

- **Все в одному.** Вбудовані засоби для авторизації, ORM, панелі адміністратора, роутінгу, шаблонів, форм, безпеки тощо;
- **Швидкий старт.** Можна створити повноцінний сайт або API дуже швидко, навіть з мінімальним кодом;
- **ORM (Object-Relational Mapping).** Зручна робота з базами даних без прямого SQL — через Python-класи;
- **Панель адміністратора.** Потужна і настроювана адмінка, яка генерується автоматично;
- **Безпека.** Вбудований захист від XSS, CSRF, SQL-ін’єкцій, клієнтських підробок запитів тощо;
- **Хороша документація.** Одна з найкращих серед фреймворків. Навіть офіційний туторіал дуже доступний;
- **Масштабованість.** Підходить як для маленьких сайтів, так і для великих веб-проектів;
- **Активна спільнота.** Велика кількість плагінів (пакетів), готових рішень, розширень.

Недоліки Django:

- **Менш гнучкий, ніж Flask.** Через "монолітну" структуру може бути важко кастомізувати під нетипові архітектури;
- **"Магія" фреймворку.** Частина речей працює "автоматично", що може заплутати початківця при налагодженні;
- **Важкий для дуже простих проектів.**
- **Менше контроль над SQL.** ORM зручний, але іноді обмежує гнучкість при складних запитах (хоча є можливість писати "сирий" SQL) ;
- **Нав’язана структура.** Django має чітку архітектуру (apps, models, views, templates), і її треба дотримуватись, навіть якщо в тебе дуже нетиповий кейс.

jQuery — це швидка, невелика та багатофункціональна бібліотека JavaScript. Це значно спрощує такі речі, як обхід HTML-документів і маніпуляції з ними, обробка подій, анімація та Ajax завдяки простому у використанні API, який працює в багатьох браузерах. Завдяки поєднанню універсальності та розширюваності jQuery змінив спосіб написання JavaScript мільйонами людей.

Переваги jQuery

- **Простота синтаксису.** Короткі, зрозумілі команди — замінює складні JS-операції;
- **Кросбраузерність.** jQuery забезпечує однакову поведінку в різних браузерах (особливо було важливо до появи сучасних стандартів) ;
- **Широкий вибір плагінів.** Безліч готових плагінів для галерей, слайдерів, модальних вікон, форм тощо;
- **AJAX спрощено.** Дуже легка реалізація запитів до серверу.
- **Підтримка старих браузерів.** ;
- **Малий поріг входу.** Ідеальний для новачків або швидких правок на сайті.

Недоліки jQuery

- **Велика вага (відносно функцій).** 80+ КБ (мінімізований) — зайве, якщо використовуєш 2–3 функції;
- **Погано масштабується.** jQuery — це не фреймворк, тож для великих застосунків логіку важко структурувати;
- **Мікс HTML + JS.** Часто веде до "спагеті-коду", де логіка, розмітка і стилі змішані;
- **Немає реактивності.** На відміну від React/Vue, jQuery не має автоматичної реакції на зміну стану чи даних;
- **Необхідність у сторонніх рішеннях.** Для складних речей потрібні сторонні бібліотеки, які не завжди оновлюються.

Також для створення розгортки фронтенду були вибрані HTML і CSS

HTML — це мова розмітки, яка використовується для створення структури веб-сторінок. Вона визначає, що буде на сторінці: заголовки, параграфи, зображення, таблиці, форми, посилання тощо.

CSS — це мова стилів, яка описує, як виглядає HTML-структура. Вона задає кольори, шрифти, відступи, анімації, розміщення елементів тощо.

3.3.2 Вибір середи розробки і інструментів

Так як основна мова програмування була вибрана Python є дві основних редактора коду: PyCharm і Visual Studio Code.

Але в проекті використовується не тільки Python а й інші мови програмування такі як: JavaScript, HTML, CSS. Тому PyCharm яка заточена лише на мові програмування не підійде, тому залишається Visual Studio Code

VS Code — це безкоштовний, легкий, але потужний редактор коду, розроблений Microsoft. Підтримує велику кількість мов програмування (JS, TS, Python, C#, Java тощо) та має величезну екосистему розширень.

Переваги VS Code

- **Безкоштовний і кросплатформений**. Працює на Windows, macOS та Linux безкоштовно;
- **Легкий і швидкий.** Запускається набагато швидше, ніж повноцінні IDE (типу Visual Studio або IntelliJ) ;
- **Потужна система розширень.** Є маркетплейс з тисячами плагінів: GitHub Copilot, Prettier, ESLint, Python, Docker, C#, Live Server тощо;
- **Вбудована підтримка Git.** Можна робити коміти, пушити, мерджити — прямо в редакторі;
- **Інтелектуальне автодоповнення (IntelliSense).** Підказки по коду, типам, методам, змінним;
- **Підтримка налагодження (debugging).** Можна запускати і відлагоджувати код без виходу з редактора;

- **Термінал всередині редактора.** Вручну виконувати команди без перемикання між вікнами;
- **Широка підтримка мов.** Один редактор — для всього: веб, бекенд, мобільна розробка, DevOps, ML тощо.

Недоліки VS Code

- **Не повна IDE.** Для деяких завдань (наприклад, складної розробки на C++ або Java) краще використовувати "повноцінну" IDE;
- **Потребує налаштувань.** З коробки редактор базовий. Потрібно ставити розширення для зручної роботи з Python, Django, React тощо;
- **Може сповільнюватися з великою кількістю плагінів.** Особливо на слабких машинах або з великими проектами;
- **Менше вбудованих інструментів, ніж у IDE.** Наприклад, відсутні графічні дизайнерські інструменти для UI;
- **Плагіни не завжди стабільні.** Деякі плагіни — аматорські або погано підтримуються.

Також для VS Code були завантажені плагіни для полегшення розробки такі як:

- **CSS Peek**-Дозволити перегляд ідентифікатора css і рядків класу як визначень із файлів html до відповідного CSS. Дозволяє переглядати та переходити до визначення;
- **Dev Containers**- Дозволяє використовувати контейнер Docker як повнофункціональне середовище розробки;
- **Black Formatter**- Підтримка форматування для файлів Python за допомогою Black formatter;
- **Django**- Підтримка форматування Django;
- **Docker**- Спрощує створення, керування та налагодження контейнерних програм;

- **eCSStractor for VSCode** - Плагін VSCode для отримання назв класів із HTML і створення таблиці стилів CSS для подальшої роботи;
- **HTML Boilerplate**- Базовий генератор шаблонних фрагментів HTML5;
- **HTML CSS Support** - CSS Intellisense для HTML;
- **IntelliCode** - Розширення Visual Studio IntelliCode надає функції розробки за допомогою штучного інтелекту для розробників Python, TypeScript/JavaScript і Java у Visual Studio Code, з уявленнями на основі розуміння контексту коду в поєднанні з машинним навчанням;
- **IntelliCode API Usage Examples -** Дає змогу побачити реальні приклади того, як інші розробники використовували певну функцію. Показані приклади взято з публічних сховищ із відкритим кодом на GitHub;
- **IntelliSense for CSS class names in HTML** - Доповнення імені класу CSS для атрибута класу HTML на основі визначень, знайдених у робочій області;
- **Isort -** Підтримка організації імпорту для файлів Python за допомогою isort;
- **Live Server -** Запуск локальний сервер розробки з функцією живого перезавантаження для статичних і динамічних сторінок;
- **Pylance -** Продуктивний, багатофункціональний мовний сервер для Python у VS Code;
- **Python -** Підтримка мови Python із розширеними точками доступу для IntelliSense (Pylance), налагодження (Python Debugger), лінтування, форматування, рефакторингу, модульних тестів тощо;
- **Python Debugger -** Розширення Python Debugger за допомогою debugpy.
- **Jinja -** Підтримка мови шаблону Jinja для Visual Studio Code;
- **Python Indent –** Допомагає робити правильні відступ Python;
- **autoDocstring - Python Docstring Generator -** Автоматично генерує рядки документів Python;
- **Auto Rename Tag -** Автоматичне перейменування парних тегів HTML/XML;

- **PostgreSQL -** Клієнт PostgreSQL для коду Visual Studio;
- **Composer -** Універсальна інтеграція композитора, швидкі дії, команди, автоматичне встановлення, завдання, кодові лінзи, діагностика та composer.json IntelliSense;

Також для бистрого розгортання системи був використаний Docker

Docker — це платформа, яка дозволяє створювати, запускати і керувати ізольованими середовищами, які називаються контейнерами.

Контейнер — це, по суті, "міні-комп'ютер" всередині основного комп'ютера. У ньому є все, що потрібно для запуску додатку: код, бібліотеки, залежності, конфігурації — незалежно від того, на якій ОС ти запускаєш його.

«Docker — це відкрита платформа для розробки, доставки та запуску програм. Docker дозволяє відокремити ваші програми від інфраструктури, щоб можна швидко доставляти програмне забезпечення. За допомогою Docker можна керувати інфраструктурою так само, як і іншими програмами. Використовуючи переваги методології Docker для доставки, тестування та розгортання коду, можна значно скоротити затримку між написанням коду та його запуском у виробництві.

Docker надає можливість пакувати та запускати програму в слабко ізольованому середовищі, яке називається контейнером. Ізоляція та безпека дозволяють запускати багато контейнерів одночасно на певному хості. Контейнери легкі та містять усе необхідне для запуску програми, тому вам не потрібно покладатися на те, що встановлено на хості. Ви можете ділитися контейнерами під час роботи, і будьте впевнені, що всі, з ким ви ділитеся, отримають той самий контейнер, який працює однаково»[9].

Також для того щоб під час розробки не виникли б ситуації такі як: видалення важливої часті коду, коли “наворотив” в коді так що вона перестала працювати, а ти не знаєш чому, якщо прилад в якому програма зламався і вся робота навічно

пропала, і якщо потрібно спосіб для спільної розробки системи, обов'язково знадобиться програма контролю версій такій як Git і хаб де б зберігались всі версії проекту віддалено, наприклад GitHub, який являється самим популярним представником свого роду.

Що таке Git?

«Git — це система контролю версій, яка використовується для відстеження змін у файлах комп'ютера. Зазвичай він використовується для керування вихідним кодом у розробці програмного забезпечення»[5].

Для чого використовується Git:

- Git використовується для відстеження змін у вихідному коді;
- Розподілений інструмент контролю версій використовується для керування вихідним кодом;
- Це дозволяє кільком розробникам працювати разом;
- Він підтримує нелінійний розвиток через тисячі паралельних гілок;

Його особливості:

- Відстежує історію
- Безкоштовний і відкритий код
- Підтримує нелінійний розвиток
- Створює резервні копії
- Масштабований
- Підтримує співпрацю
- Розгалуження легше
- Розподілена розробка

Тому важливо в кожному проекти підключати Git навіть якщо розробник працює один. І даний проект також не став винятком.

Інструмент і сховище Git був обраний GitHub – самий популярний представник свого роду.

GitHub — це веб-платформа для зберігання, спільної роботи та керування проєктами, що використовують систему контролю версій Git.

Для інтеграції GitHub був використаний вже вмонтований інструментарій в VSCode(Рис.18).

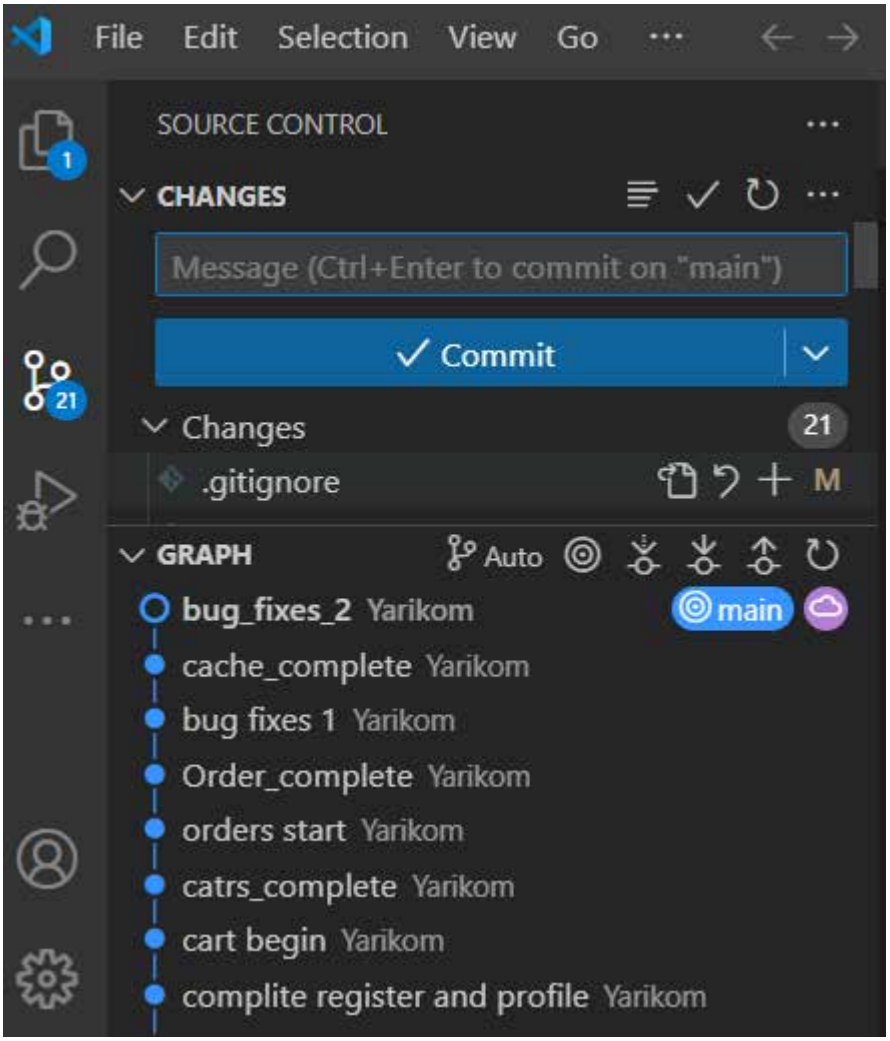


Рис. 18 Вмонтований інструментарій контролю версій на основі Git в VSCode

Для того щоб в GitHub не попали чутливі данні або файли і не попали тимчасові файли такі як кеш чи медіа, до проекту потрібно додати спеціальний який б вказував Git які файли потрібно ігнорувати і не відправляти в хаб де інші

люди могли б цим скористатись, або щоб не смітити хаб лишніми даними, а конкретно додати файл .gitignore(Рис. 19).

Також для того щоб відокремити чутливі данні від інших файлів їх зазвичай розміщують в спеціальному файлі . env. Там як правило зберігаються всі чутливі данні які не можна відсилати в інтернет.

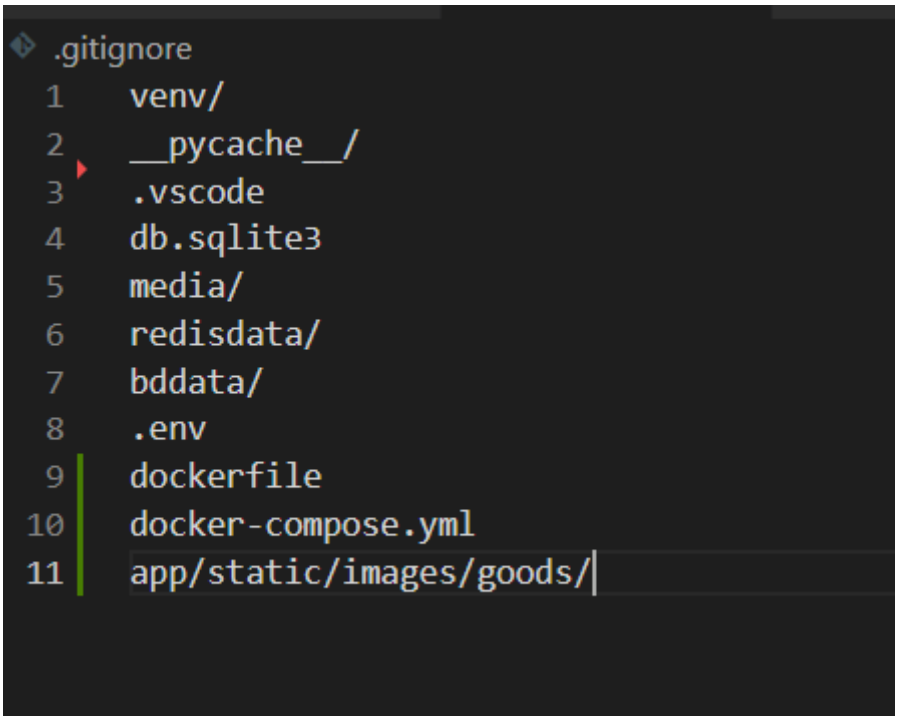


Рис. 19 файл .gitignore в проекті

3.4 Алгоритмізація та програмування програмних модулів

Для реалізації бекенд системи був вибрано фреймворк Django на мові програмування Python, а для фронтенд були вибрані HTML, CSS, JavaScript.

В ході розробки проекту були реалізовані алгоритми які виконують бізнес логіку і бекенд до них.

І по прикладу одного з частин системи, а конкретно верифікація, буде показано як був побудований проект.

Наприклад на Рис. 20 показана блок схема реалізації верифікації

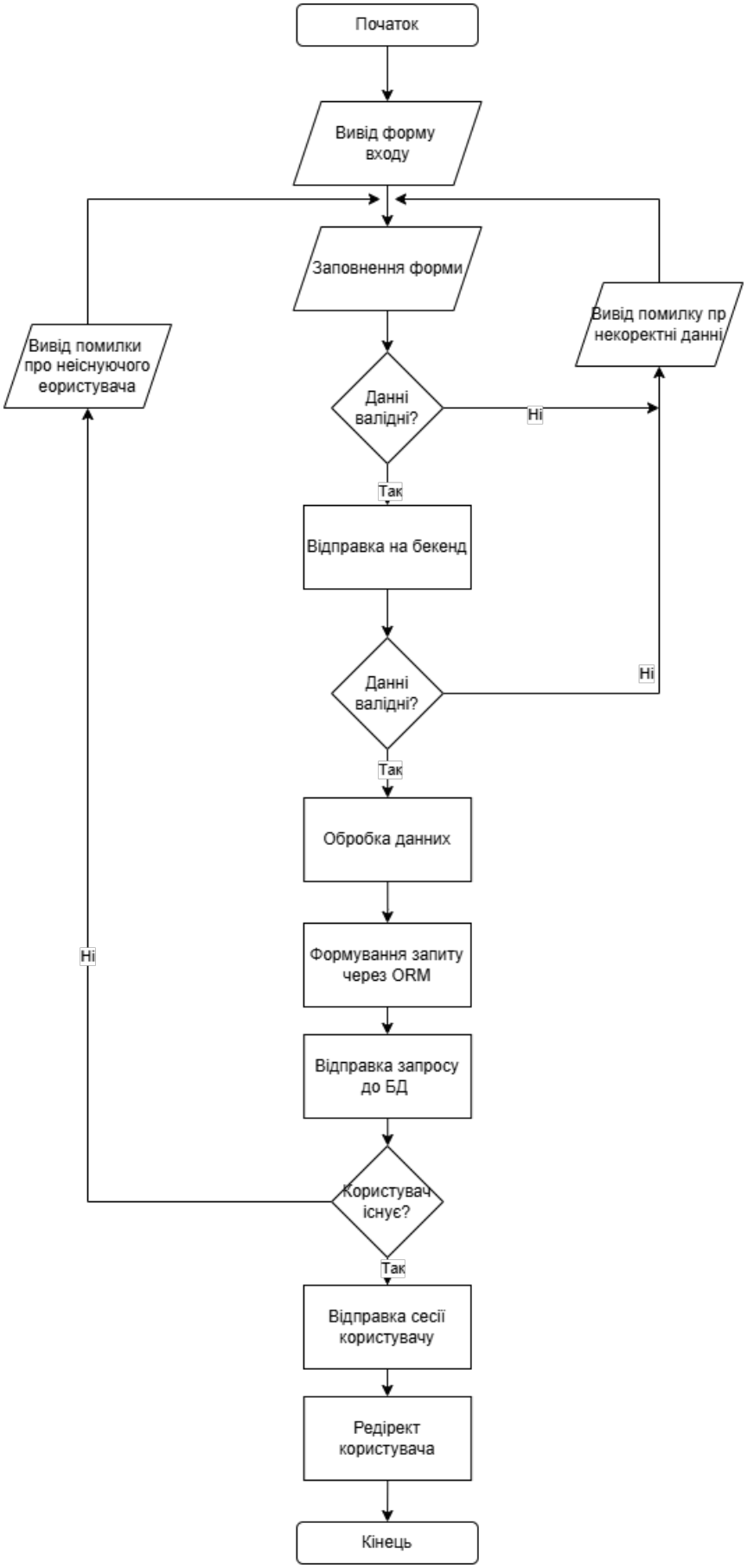


Рис. 20 блок-схема верифікації

На Рис 21 показаний код який реалізовує фронтенд частину.

```
{% extends "main.html" %}
{% load static %}

{% block cart %}
{% include "cart_button.html" %}
{% endblock cart %}



{% block content %}
<div class="row">
    <div class="container mt-5">
        <div class="row justify-content-center">
            <div class="col-md-6 bg-white p-4 mb-4 mx-3 rounded custom-shadow">
                <h2 class="text-center mb-4">Авторизація</h2>
                <form action = "{% url "user:login" %}" method="post">
                    {% csrf_token %}
                    {% if request.GET.next %}
                    <input type="hidden" name="next" value={{ request.GET.next }}>
                    {% endif %}
                    <div class="mb-3">
                        <label for="id_username"  class="form-label">Ім'я користувача</label>
                        <input type="text" name="username" class="form-control" value="{% if form.username.value  %}{{ form.userna
                    </div>
                    <div class="mb-3">
                        <label for="id_password" class="form-label">Пароль</label>
                        <input type="password" name="password" class="form-control" id="id_password" placeholder="Введіть пароль"
                    </div>
                    <button type="submit" class="btn btn-blue  btn-block">Увійти</button>
                </form>
                <div class="mt-3">
                    <a href="#">Забули пароль?</a> | <a href="{% url "user:registration" %}">Створити аккаунт</a>
                </div>
```

Рис. 21 Частина коду фронтенду верифікацій

Для того щоб відправити данні з фронтенд частини до бекенду потрібно створити для них спеціальну форму як на Рис. 22.

```
class UserLoginForm(AuthenticationForm):

    class Meta:
        model=User
        fields: list[str] =['username','password']

    username = forms.CharField()
    password = forms.CharField()
```

Рис. 22 Форма для верифікацій користувача

Після чого на бекенд частині виконується основна логіка і взаємодія з БД як показано на Рис.23. Якщо данні не потрібно нікуди відправляти то форма не потрібна.

```python
class UserLoginView(LoginView):
    template_name='users/login.html'
    form_class =UserLoginForm
    #success_url =reverse_lazy('general:index')
    def get_success_url(self) -> str | Any:
        redirect_page: str | None=self.request.POST.get('next',None)
        if redirect_page and redirect_page != reverse('user:logout'):
            return redirect_page
        return reverse_lazy('general:index')
    def form_valid(self, form) -> HttpResponseRedirect | None:
        session_key: Any=self.request.session.session_key

        user: User = form.get_user()
        cart: BaseManager[Cart]=Cart.objects.filter(session_key=session_key)
        if user:
            auth.login(self.request, user)
            if session_key and cart:
                forgot_carts: BaseManager[Cart]=Cart.objects.filter(user=user)
                if forgot_carts.exists():
                    forgot_carts.delete()
                Cart.objects.filter(session_key=session_key).update(user=user)

            messages.success(self.request, f"Вітаю {user.username}")
            return HttpResponseRedirect(self.get_success_url())
```

Рис. 23 Бекенд частина для верифікацій користувача

І по такому принципу і побудована багато модулів системи. Повний код програми і декілька блок схем для них можна знайти в Додатку Б.

3.5 Висновки до розділу 3

Під час опрацювання даного розділу було вибрано інструментарій (мови програмування, фреймворки, бібліотеки, редактор коду і розширення для нього) для розробки системи, була вибрана і інтегровані SQL і NOSQL БД. Була інтегрована ORM система і побудовані моделі для них. Було за допомогою вибраних інструментарію побудована основна система.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇСИСТЕМИ

4.1 Тестування системи

Для того щоб проект був робото-спроможним потрібно як слід його тестувати, щоби зменшити кількість помилок і багів, які можуть виникнути під час розробки.

Краще за все написати план тестування і самі тестування ще до розробки системи, так можна запобігти невеликі помилки ще на ранніх стадіях розробки, які б могли розростись до серйозних помилок і спричинити купу проблем, бо такі помилки вже важко виправити, і було б її дорого виправляти ніж могло б якщо раніше її виявити.

Також тестування бувають не тільки для виявлення помилок в коді програми, но й перевірки чи відповідає продукт вимогам і чи відповідає програма стандартам.

Для тестування розробленої системи був використаний Django unit test для тестування функцій системи і також був використаний інструмент для тестування UI сайту - selenium. Ця бібліотека дозволяє виконувати автоматичну дію з інтерфейсом користувача так ніби це взаємодіє реальний користувач, що дозволяє автоматично і вчасно знаходити помилки в інтерфейсі користувача.

4.2 Вимоги до апаратного та програмного забезпечення

Вимогою для розгортання системи є як мінімум 1 сервер, де буде виконуватись вся логіка системи і взаємодія з даними, мінімальна конфігурація якого:

- Intel Core i3+
- 8гб+ ОЗУ
- 120гб+ HDD

Але рекомендується для кращої роботи системи мати 3 сервера які кожні будуть відповідати за такі модулі системи як:

- Web-сервер(головний сервер де виконуються практично вся логіка)
- БД-сервер(сервер відповідає за збереження даних)
- Кеш-сервер(сервер для зберігання тимчасових і популярних даних)

Так як всі модулі системи будуть завернути в Docker контейнер і тому для розвертання на сервер достатньо встановлена операційна система з пакетним менеджером і встановлений Docker.

Покупцю, який буде користуватись системою через браузер, необхідна будь-яка операційна система, яка підримує графічні браузери.

Мінімальні вимоги для успішного користування системою під Windows

- Pentium 4 або новіший процесор, який підтримує SSE2
- 512МБ ОЗУ / 2ГБ ОЗУ для 64-бітної версії
- 200МБ вільного місця на диску

Мінімальні вимоги для успішного користування системою під Mac:

- Macintosh комп’ютер з Intel x86 процесором
- 512МБ ОЗУ
- 200МБ вільного місця на диску

Програмні вимоги для покупця:

- Наявність хоча б одного з більш-менш сучасних браузерів
- Для запуску під Linux платформи, потрібно встановити додаткові пакети

Приблизна схема розгортання представлена на Рис. 24

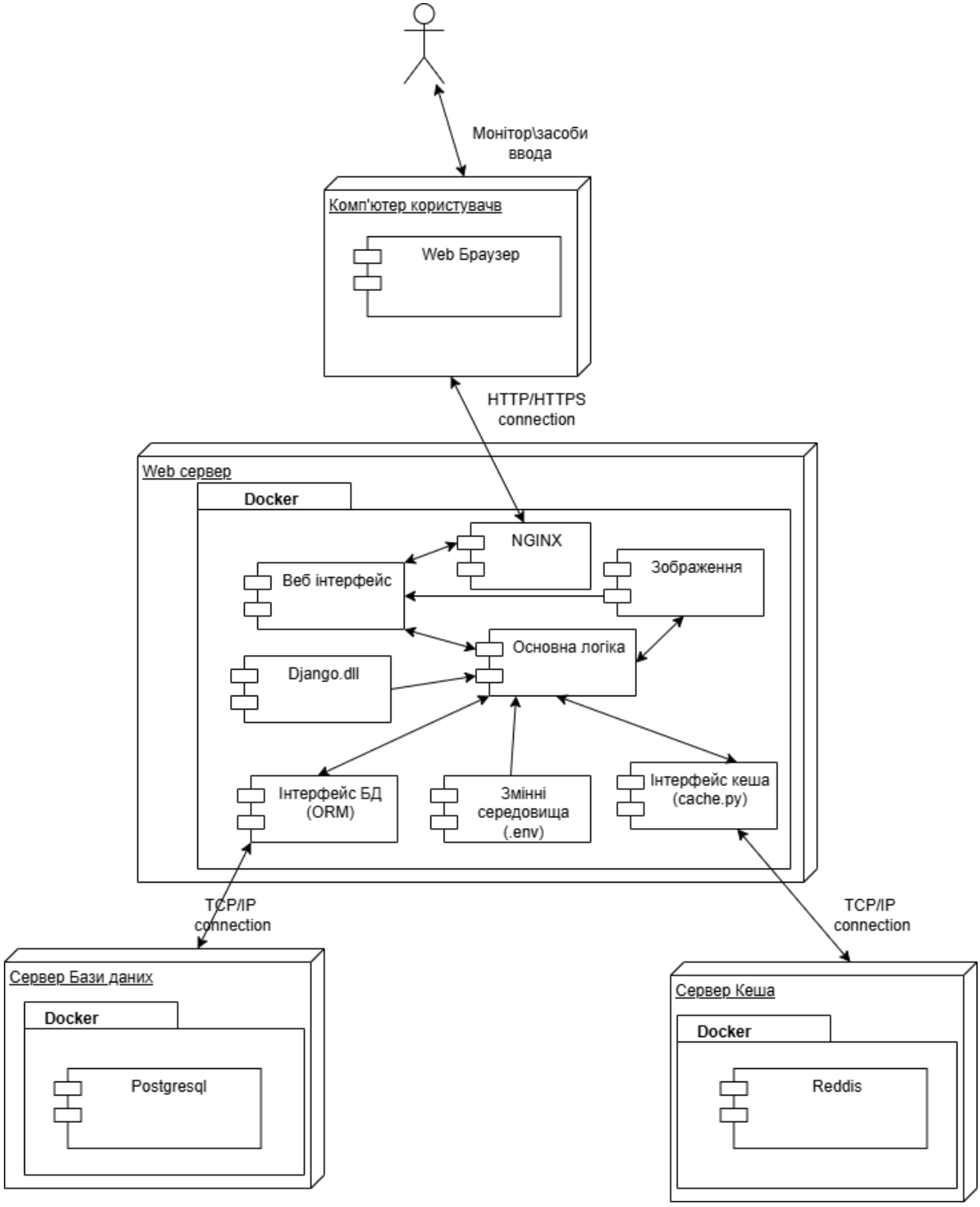


Рис 24. Діаграма розгортання

4.3 Склад інсталяційного пакету

- Так як система завернута в Docker контейнер всі потрібні файли і інструкція як їх встановлювати, і Docker сам встановлює систему по цим інструкціями. Тому для інсталяційного пакету нам потрібен встановлений Docker
- Docker контейнер з Web-server, БД, Кешу
- ОС для контейнера
- Інтерпретатор коду
- Django
- docker-compose – для автоматичного розгортання системи

Також для того щоб розгорнути в файли .env (Рис. 25) де зберігаються глобальні змінні, в змінні де вказана адрес локація чи хост потрібно ввести туди поточний адрес Web-server, БД, Кешу.

```
POSTGRESQL_ENGINE='django.db.backends.postgresql'
POSTGRESQL_NAME='shop_furniture'
POSTGRESQL_USER='meneger'
POSTGRESQL_PASSWORD='123456789'
POSTGRESQL_HOST='localhost'
POSTGRESQL_PORT='19112'
POSTGRESQL_IMAGE='17.0-alpine'
POSTGRESQL_CONT_NAME='db_diplom'


REDIS_HOST='localhost'
REDIS_PORT='19113'
REDIS_USERNAME='shop'
REDIS_USERNAME_PASSWORD='shoppass'
REDIS_PASSWORD='root'
REDIS_IMAGE='latest'
REDIS_CONT_NAME='redis_container'

SECRET_KEY = 'django-insecure-6hv-i1=qh&my0@%_!x24a&m6xqwuk2sp*oxs4i!sc@c8z2uxs2'
CACHES_BACKEND='django.core.cache.backends.redis.RedisCache'
CACHES_LOCATION='redis://shop:shoppass@localhost:19113'
DEBUG='True'
```

Рис. 25 .env файл проекту

Де в POSTGRESQL_HOST вказує на адрес БД, REDIS_HOST вказує на адрес кеша, CACHES_LOCATION = вказує місце де потрібно зберігати кеш.

Ці данні під час розгортання потрібно змінити на актуальні крім них змінювати не обов’язково. Крім DEBUG але він змінюється вже під кінець розробки, а не розгортання.

4.4 Опис роботи програми

Після того як користувач заходить на сайт його зустрічає вітальна сторінка, яка представлено на рис. 26. В ній користувач може взаємодіяти з головним меню, який знаходиться зверху і з кошиком та каталогом. Внизу сайту є посилання на git проекту.

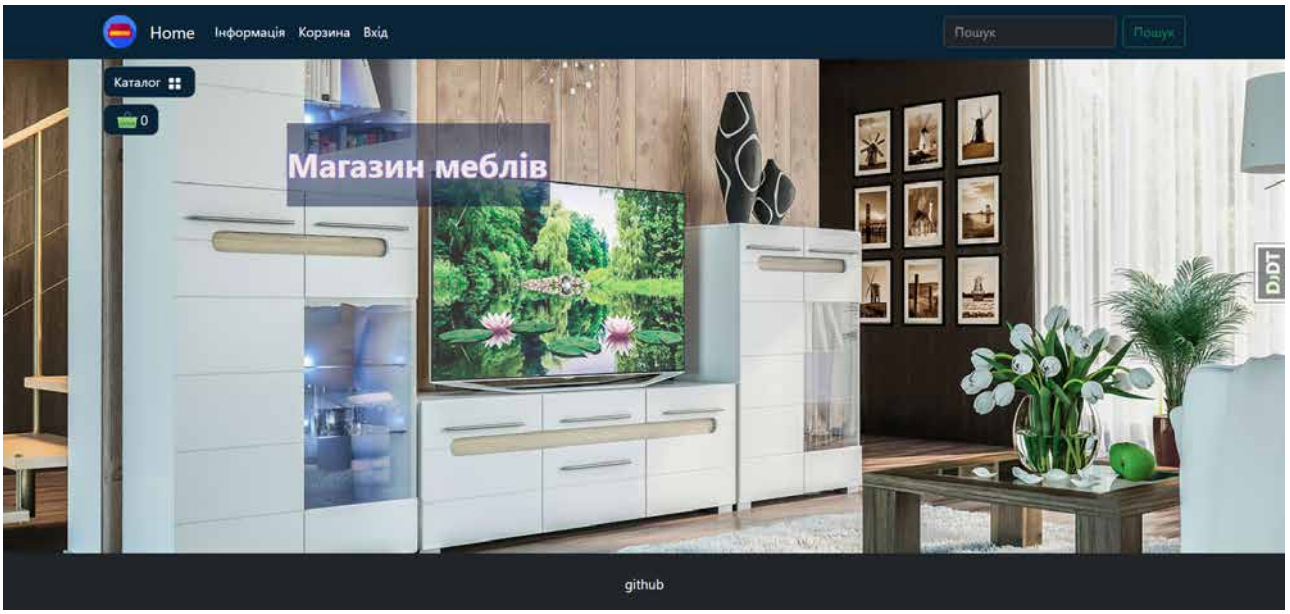


Рис. 26 Вітальна сторінка

Користувач може в меню каталогу, яка представлена на рис. 27, вибрати всі наявні товари або їхні категорії, які є на сайті щоб перейти до сторінки з списками наявних товарів (Рис. 28)

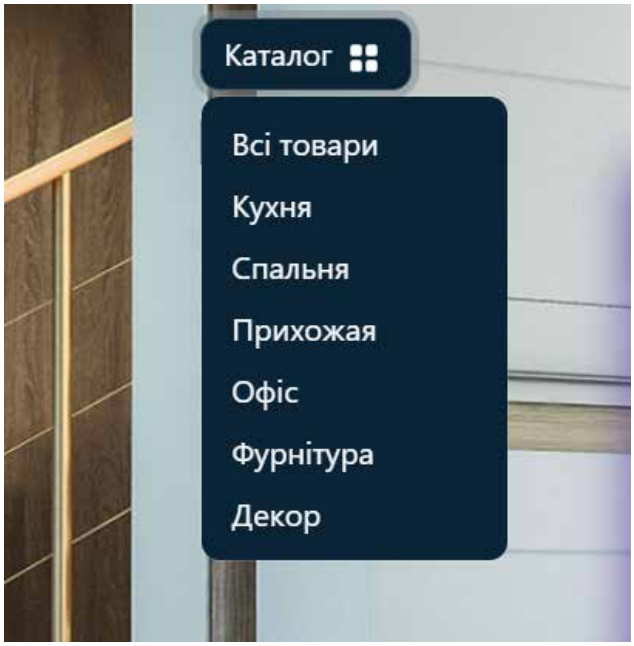


Рис. 27 Меню вибору категорії товару

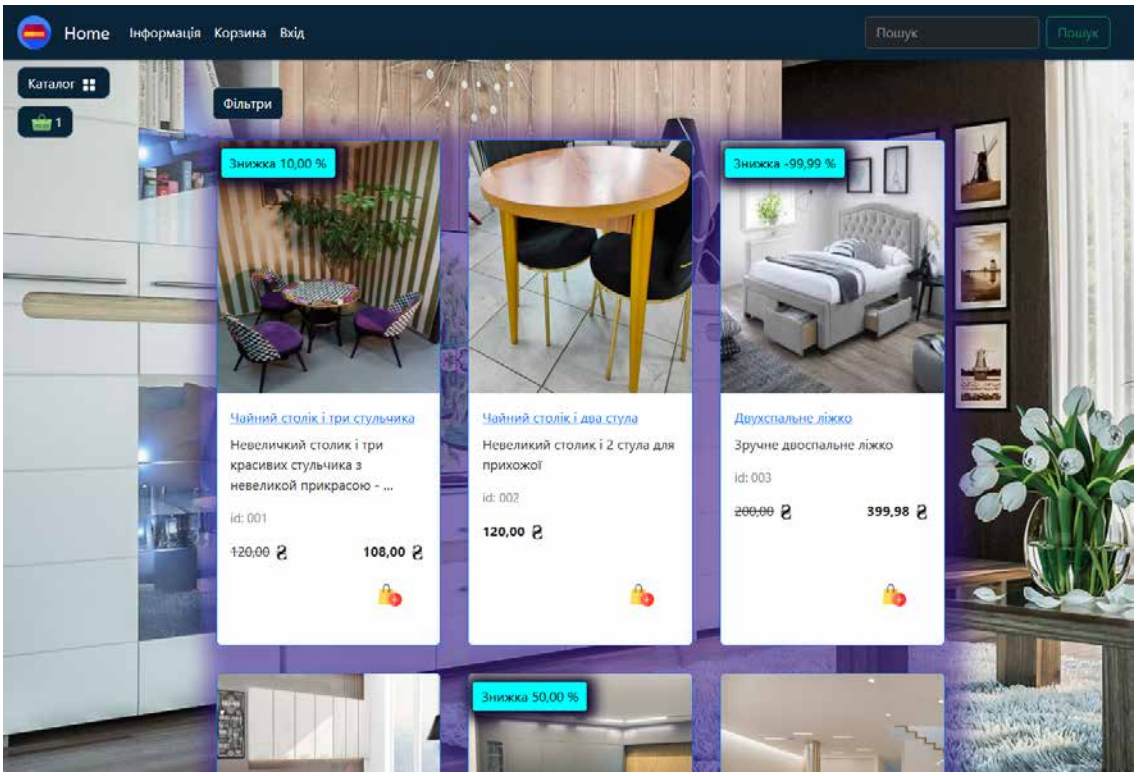


Рис. 28 Сторінка каталогу товару

На цій сторінці користувач може перейти до картки конкретного товару або відразу додати товар до кошику (Рис. 29), натиснувши на значок пакету. Товар можна додавати до кошику навіть не входячи до свого акаунта. Система сама збереже всі додані товари після авторизації.

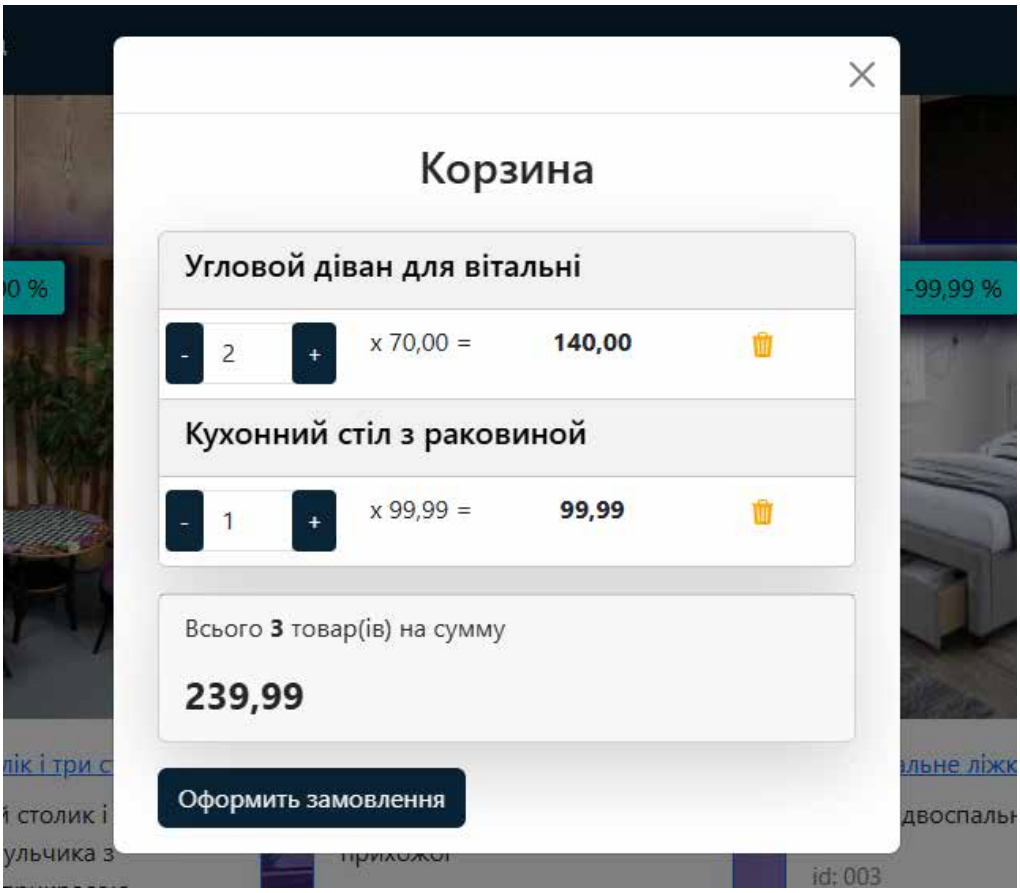


Рис. 29 Кошик товарів

Після того як користувач вибрав всі потрібні товари йому потрібно спочатку авторизуватись в системі або створити новий акаунт це можна зробити натиснувши на кнопку «Вхід» в меню щоб перейти на сторінку авторизації, яка представлена на Рис. 30, з цієї сторінці можна перейти на сторінку авторизації, яка представлена на Рис. 31 натиснувши на кнопку «створити акаунт».

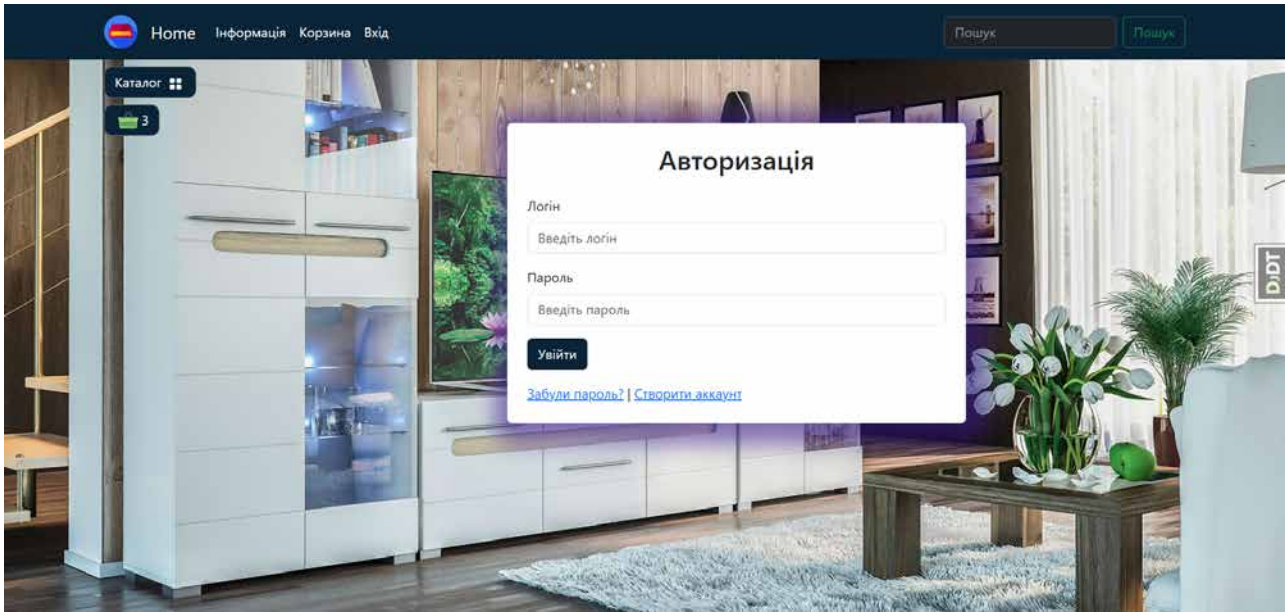


Рис. 31 Сторінка входу в акаунт

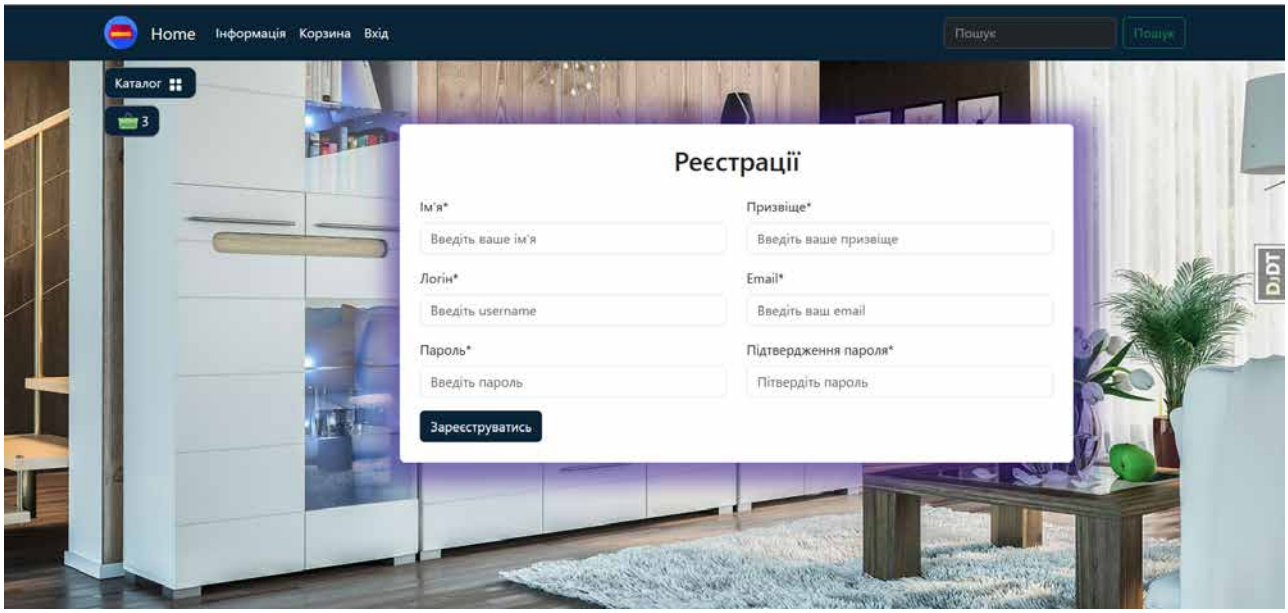


Рис. 32 Сторінка реєстрації

Після чого користувач в корзині може натиснути на кнопку «Оформити замовлення» щоб перейти на сторінку де можна створити нове замовлення, яка представлена на Рис. 33. В цій сторінці вказуються всі товари які на той час були в корзині і їхня загальна ціна, а під ними форма для введення всіх потрібних даних для оформлення нового замовлення. Після заповнення форми користувач натискає на кнопку «Оформити замовлення» і після перевірки і валідації додається до списку замовлень.

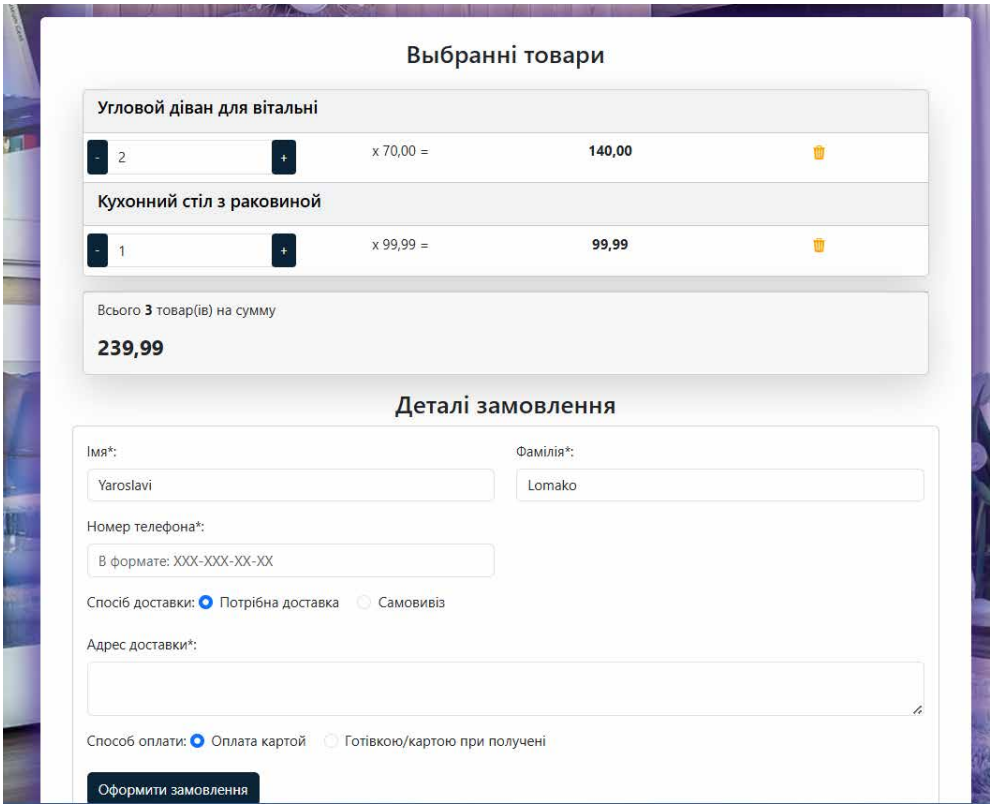


Рис. 33 Форма для оформлення замовлення

Після чого користувач може перевірити список замовлень в особистому кабінеті, яка представлена на Рис. 34, там користувач може змінити свої данні в профілі користувача Рис. 35.

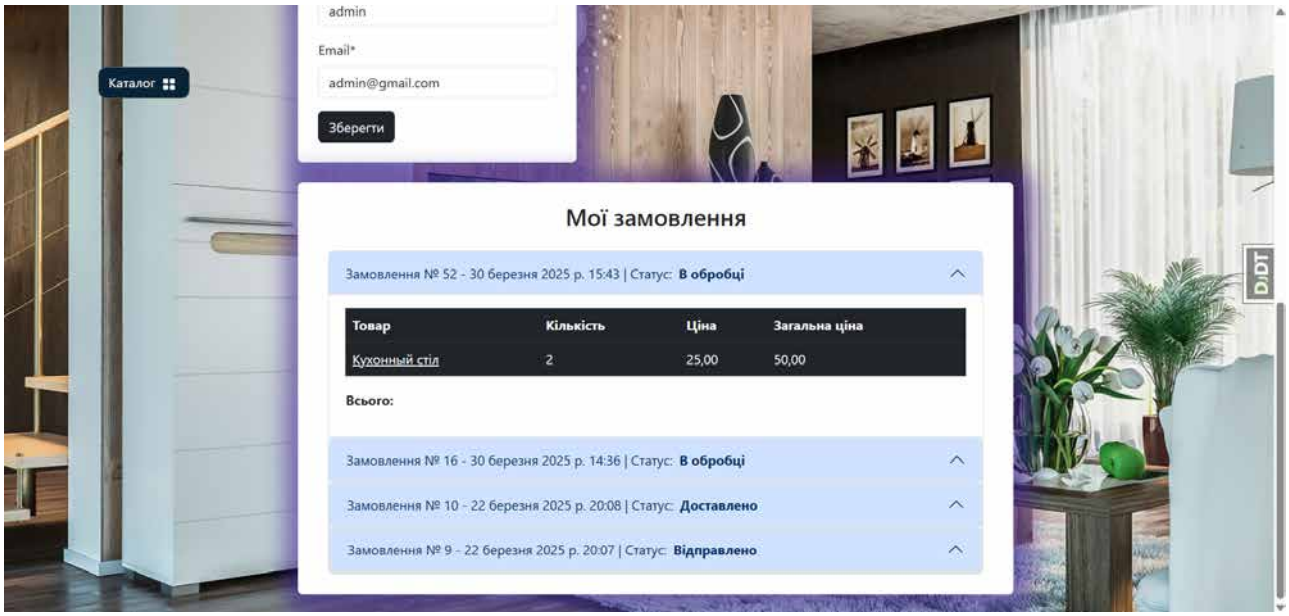


Рис. 34 Список замовлень в особистому кабінеті

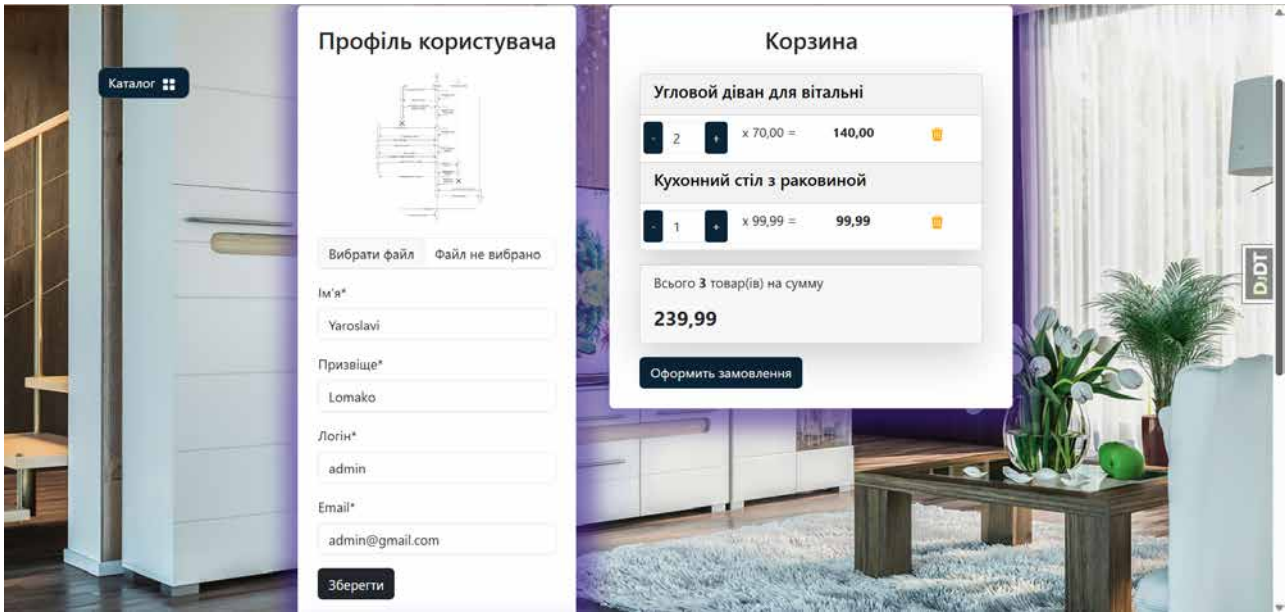


Рис. 35 Профіль користувача

Якщо у користувача є повноваження менеджера або працівника складу, то в головному меню буде відображатись кнопка для входу в адміністративну панель, яка звичайному користувачу не відображається. Після того як користувач з доступом переходить до адміністративної панелі його зустрічає головна сторінка адміністративної панелі, яка представлена на Рис. 36.

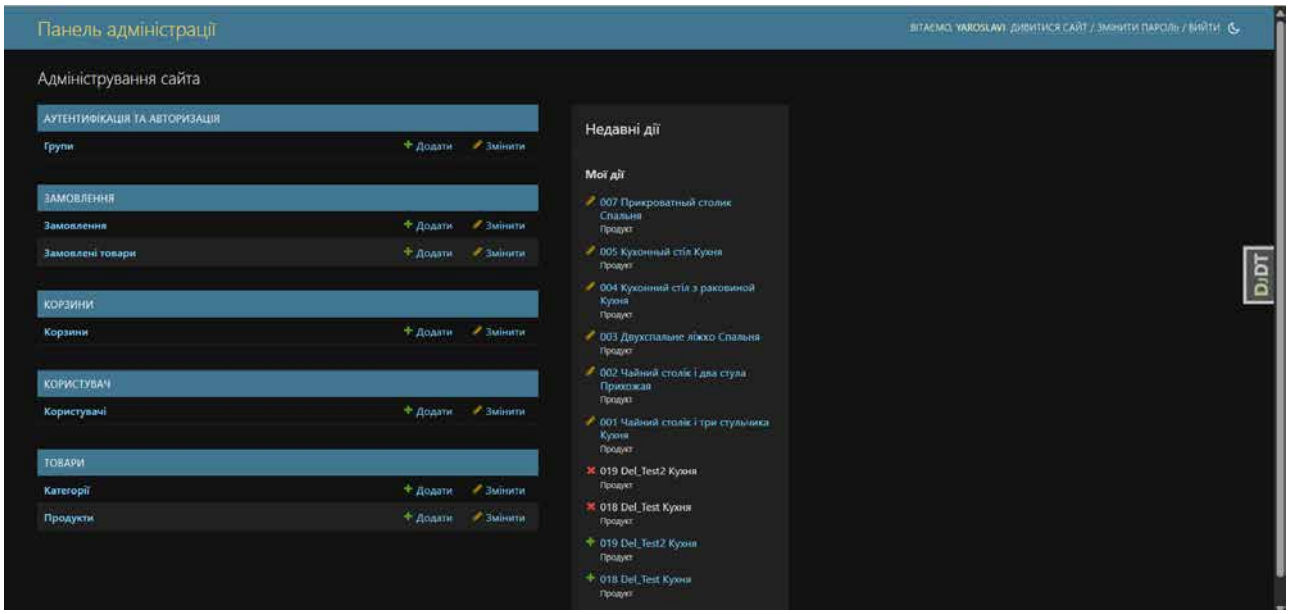


Рис. 36 Головна сторінка адміністративної панелі

Наприклад для того щоб додати новий товар потрібно натиснути на кнопку «Продукт» під вкладкою «Товари», тоді відкривається сторінка з всіма продуктами на сайті, яка представлена на Рис. 37.

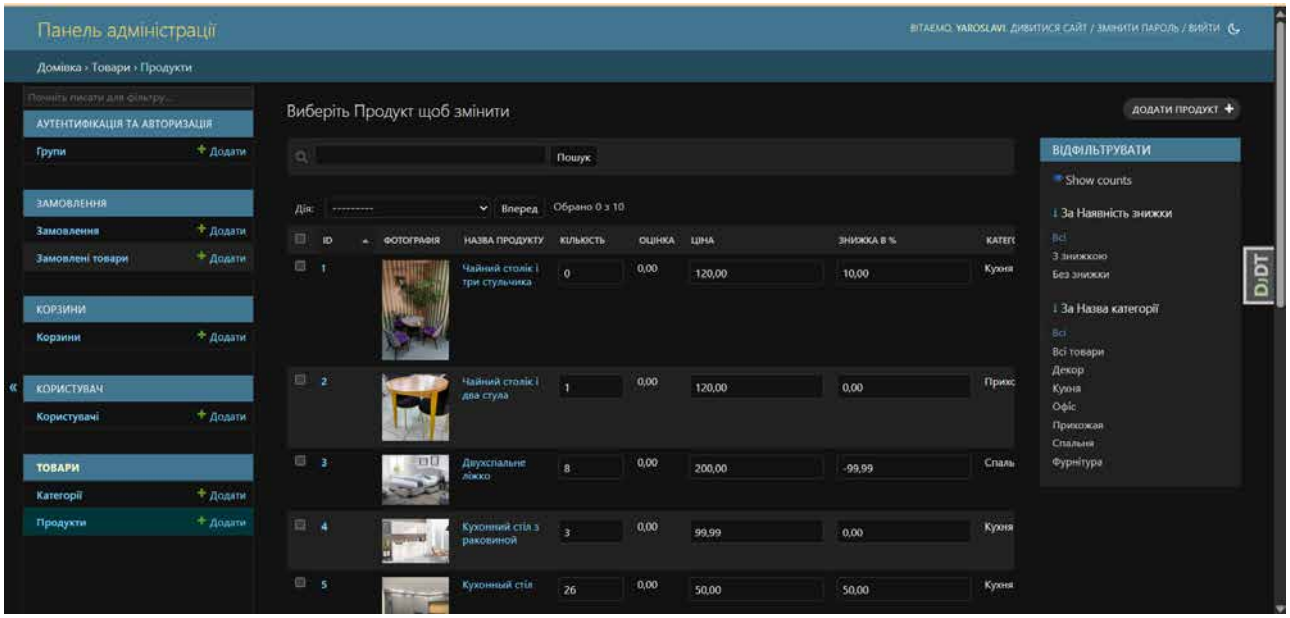


Рис. 37 Список продуктів в адміністративної панелі

На цій сторінці можна відфільтрувати товар, щоби легко знайти потрібний, або натиснути на назву конкретного товару щоби відкрити сторінку з редагуванням даних про неї. Для того щоби додати новий товар до каталогу потрібно натиснути на кнопку додати продукт, тоді відкривається сторінка з формою, яка представлена на Рис. 38, яку потрібно заповнити щоби додати новий товар.

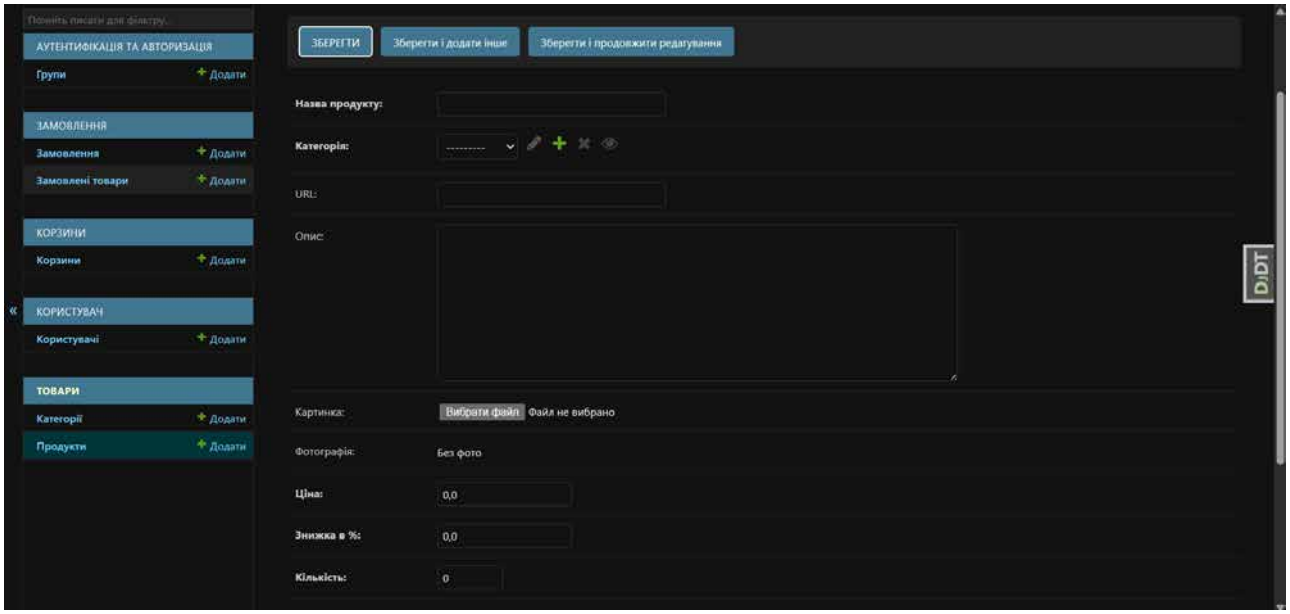


Рис. 38 Форма для додавання нового товару

Після того як була заповнена форма потрібно натиснути на одну з кнопок збереження, щоб після валідації додати до списку каталогу новий товар.

4.5 Висновки до розділу 4

Під час опрацювання даного розділу було проведено тестування системи, визначено інсталяційний пакет і розроблена інструкції для його розгортання і визначено вимоги до апаратного та програмного забезпечення. Також було описано роботу частини програми.

ВИСНОВКИ

Під час роботи з дипломом було розроблено інтернет магазин для продажу меблів і були получено навички роботи з інструментами розробки сайтів як: Django, Python. JavaScript, CSS, PostgreSQL, Reddis та інші.

На сайті булу успішно реалізовані функції: авторизації, реєстрації, додавання товарів, зміна даних товарів, кошик, замовлення, пошук, фільтри, сортування в каталозі, налаштована адміністративна панель з різними ролями.

Було получено практичні навички створення сайтів, також було реалізовано і закріплено на практиці знання які були отриманні під час навчання в університеті.

Було получено практичні навички по роботі з Docker контейнером, як їх створювати та розгортати.

Було засвоєно і поглиблено знання мови уніфікованої язика моделювання-UML. Також в ході роботи з системою були створені декілька діаграм UML.

ПЕРЕЛІК ЛІТЕРАТУРИ

1. Моралес Дж. What is A UML Component Diagram: Symbols and Tutorial. MindOnMap | Free Mind Mapping Tool to Draw Ideas Easily Online. URL: https://www.mindonmap.com/uk/blog/uml-component-diagram/ (date of access: 28.04.2025).
2. Awati R. What is object-relational mapping (ORM)? – TechTarget Definition. TheServerSide.com. URL: https://www.theserverside.com/definition/object-relational-mapping-ORM#:~:text=Object-relational%20mapping%20(ORM)%20is%20a%20way%20to%20align,language%20and%20a%20relational%20database (date of access: 28.04.2025).
3. Django introduction - Learn web development | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/Django/Introduction (date of access: 28.04.2025).
4. Lewis S. What is a Collaboration Diagram?. Search Software Quality. URL: https://www.techtarget.com/searchsoftwarequality/definition/collaboration-diagram (date of access: 28.04.2025).
5. Perveez S. H. What is Git: Features, Commands, Branch and Workflow in Git. Simplilearn.com. URL: https://www.simplilearn.com/tutorials/git-tutorial/what-is-git (date of access: 28.04.2025).
6. UML Class Diagram Tutorial. Lucidchart. URL: https://www.lucidchart.com/pages/uml-class-diagram (date of access: 28.04.2025).
7. UML Practical Guide - All you need to know about UML modeling. Ideal Modeling & Diagramming Tool for Agile Team Collaboration. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-practical-guide/ (date of access: 28.04.2025).
8. What is an Entity Relationship Diagram (ERD)?. Lucidchart. URL: https://www.lucidchart.com/pages/er-diagrams (date of access: 28.04.2025).

9. What is Docker?. Docker Documentation. URL: https://docs.docker.com/get-started/docker-overview/ (date of access: 28.04.2025).
10. What is JavaScript? - Learn web development | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Scripting/What_is_JavaScript (date of access: 28.04.2025).
11. What is SQL? - Structured Query Language (SQL) Explained - AWS. Amazon Web Services, Inc. URL: https://aws.amazon.com/what-is/sql/?nc1=h_ls (date of access: 28.04.2025).

ДОДАТОК А

База даних

Сторінок 9

Київ-2025

Створення ролі в БД

```
CREATE ROLE meneger WITH
    LOGIN
    SUPERUSER
    CREATEDB
    CREATEROLE
    INHERIT
    NOREPLICATION
    CONNECTION LIMIT -1
    PASSWORD '123456789';
```

Створення бази даних для системи

```
CREATE DATABASE shop_furniture
    WITH
    OWNER = meneger
    ENCODING = 'UTF8'
    LOCALE_PROVIDER = 'libc'
    CONNECTION LIMIT = -1
    IS_TEMPLATE = FALSE;
```

Виправлення помилки неправильного типу в БД напрямую

```
ALTER TABLE "order" ALTER "is_paid" TYPE varchar(100)
ALTER TABLE "order" ALTER "is_paid" TYPE boolean
USING CASE "is_paid" WHEN "" THEN true ELSE false END;
```

Моделі для БД:

Таблиця Категорія

```
class Categoris(models.Model):
    name= models.CharField(max_length=100,unique=True,verbose_name='Назва
категорії')
    slug=
models.SlugField(max_length=200,unique=True,blank=True,null=True,verbose_name='URL'
)
    class Meta:
        db_table='category'
        verbose_name='Категорію'
        verbose_name_plural='Категорії'

    def __str__(self):
        return f"{self.name}"
```

Таблиця Товари

```
class Products(models.Model):
    name= models.CharField(max_length=100,unique=True,verbose_name='Назва
продукту')
    slug=
models.SlugField(max_length=200,unique=True,blank=True,null=True,verbose_name='URL'
)
    description=models.TextField(blank=True,null=True,verbose_name='Опис')
    image=models.ImageField(upload_to='goods_images',blank=True,null=True,
verbose_name='Картинка')

price=models.DecimalField(default=0.00,max_digits=8,decimal_places=2,verbose_name='
Ціна')

discount=models.DecimalField(default=0.00,max_digits=4,decimal_places=2,verbose_nam
e='Знижка в %')
    quantity=models.PositiveIntegerField(default=0,verbose_name='Кількість')

rate=models.DecimalField(default=0.00,max_digits=2,decimal_places=2,editable=False,
verbose_name='Оцінка')
    created = models.DateTimeField(editable=False,verbose_name='Дата створення')
    modified = models.DateTimeField(editable=False,verbose_name='Дата зміни')

category=models.ForeignKey(to=Categoris,on_delete=models.PROTECT,verbose_name='Кате
горія')

    class Meta:
        db_table='product'
        verbose_name='Продукт'
```

```
        verbose_name_plural='Продукти'
        ordering=("id",)

    def save(self, *args, **kwargs):
        if not self.id:
            self.created = timezone.now()
        self.modified = timezone.now()
        return super(Products, self).save(*args, **kwargs)

    def __str__(self):
        return f"{self.display_id()} {self.name}  {self.category}  "



    def delete(self):
        URL_root=str(MEDIA_ROOT)
        d_root=URL_root.split('\\')
        new_URL=''
        for i in d_root:
            new_URL=new_URL+f"{i}/"
        os.remove(f'{str(new_URL)}{str(self.image)}')
        return super(Products, self).delete()



    def display_id(self):
        return f"{self.id:03}"

    def get_absolute_url(self):
        return reverse('goods:product', kwargs={'product_slug': self.slug})

    def d_price(self):
        if self.discount:
            return round(self.price-self.price*self.discount/100,2)
        return self.price

class Rating(models.Model):
    rating=models.PositiveIntegerField(max_length=2,verbose_name='Оцінка
товара',default=0,max_digits=10)

users=models.ForeignKey(to=User,on_delete=models.PROTECT,verbose_name='Користувач')

product=models.ForeignKey(to=Products,on_delete=models.CASCADE,verbose_name='Продук
т')
    class Meta:
        db_table='rating'
        verbose_name='Оцінка'
        verbose_name_plural='Оцінки'

class RatingQueryset(models.QuerySet):
```

```
    def total_rating(self):
        if self:
            total= sum(rating.prating for rating in self)
            counts=count(rating.prating for rating in self)
            rating=round(float(total)/float(counts),2)
            return rating

        return 0
```

Таблиця Користувачів:

```
class AbstractUser(AbstractBaseUser, PermissionsMixin):
    """
    An abstract base class implementing a fully featured User model with
    admin-compliant permissions.

    Username and password are required. Other fields are optional.
    """

    username_validator = UnicodeUsernameValidator()

    username = models.CharField(
        _("username"),
        max_length=150,
        unique=True,
        help_text=_(
            "Required. 150 characters or fewer. Letters, digits and @/./+/-/_
only."
        ),
        validators=[username_validator],
        error_messages={
            "unique": _("A user with that username already exists."),
        },
    )
    first_name = models.CharField(_("first name"), max_length=150, blank=True)
    last_name = models.CharField(_("last name"), max_length=150, blank=True)
    email = models.EmailField(_("email address"), blank=True)
    is_staff = models.BooleanField(
        _("staff status"),
        default=False,
        help_text=_("Designates whether the user can log into this admin site."),
    )
    is_active = models.BooleanField(
        _("active"),
        default=True,
        help_text=_(
```

```
            "Designates whether this user should be treated as active. "
            "Unselect this instead of deleting accounts."
        ),
    )
    date_joined = models.DateTimeField(_("date joined"), default=timezone.now)

    objects = UserManager()

    EMAIL_FIELD = "email"
    USERNAME_FIELD = "username"
    REQUIRED_FIELDS = ["email"]

    class Meta:
        verbose_name = _("user")
        verbose_name_plural = _("users")
        abstract = True

    def clean(self):
        super().clean()
        self.email = self.__class__.objects.normalize_email(self.email)

    def get_full_name(self):
        """
        Return the first_name plus the last_name, with a space in between.
        """
        full_name = "%s %s" % (self.first_name, self.last_name)
        return full_name.strip()

    def get_short_name(self):
        """Return the short name for the user."""
        return self.first_name

    def email_user(self, subject, message, from_email=None, **kwargs):
        """Send an email to this user."""
        send_mail(subject, message, from_email, [self.email], **kwargs)
```

```
class User(AbstractUser):
    image=models.ImageField(upload_to='users_images',blank=True,null=True,
verbose_name='Аватар')
    phone=models.CharField(max_length=10,blank=True,null=True,verbose_name='Номер
телефона')

    class Meta:
        db_table='user'
        verbose_name='Користувач'
        verbose_name_plural='Користувачі'
```

```
        def __str__(self):
         return self.username
```

Таблиця Корзини:

```
class CartQueryset(models.QuerySet):

    def total_quantity(self):
        if self:
            return sum(cart.quantity for cart in self)
        return 0

    def total_price(self):
        return sum(cart.products_total_price() for cart in self)



class Cart(models.Model):

    user = models.ForeignKey(to=User,on_delete=models.CASCADE, blank=True,
null=True, verbose_name='Користувач')
    product = models.ForeignKey(to=Products,on_delete=models.CASCADE,
verbose_name='Продукт')
    quantity = models.PositiveSmallIntegerField(default=0,
verbose_name='Кількість')
    created_timestamp=models.DateField(auto_now_add=True, verbose_name='Дата
додавання')
    session_key=models.CharField(max_length=32,null=True,blank=True)

    class Meta:
        verbose_name="Корзину"
        verbose_name_plural="Корзини"
        db_table='cart'

    objects=CartQueryset().as_manager()
    def __str__(self):
        if self.user:
            return f'Корзина {self.user.username} | Товар {self.product.name} |
Кількість {self.quantity}'
        else:
            return f'Анонимна корзина | Товар {self.product.name} | Кількість
{self.quantity}| Сесіоний ключ {self.session_key}'


    def products_total_price(self):
        return round(self.product.d_price() * self.quantity,2)
```

Таблиця Замовлень:

```
class ENUM:
    OrderStatus=(
        ('В обробці' , "В обробці"),
        ('Відправлено' , "Відправлено"),
        ('Доставлено' , "Доставлено"),
        ('Получено' , "Получено")
   )

class Order(models.Model):

user=models.ForeignKey(to=User,on_delete=models.SET_DEFAULT,null=False,blank=True,
verbose_name="Користувач",default=None)
    created_timestamp = models.DateTimeField(auto_now_add=True,verbose_name="Дата
замовлення")
    phone_number=models.CharField(max_length=20,verbose_name="Номер телефона")
    requires_delivery=models.BooleanField(default=False,verbose_name="Потрібна
доставка")
    delivery_adress=models.TextField(null=True,blank=True,verbose_name="Адрес
доставкі")
    payment_on_get=models.BooleanField(default=False,verbose_name="Оплата при
доставці")
    is_paid=models.BooleanField(default=False,verbose_name="Заплачено")
    status=models.CharField(max_length=50, choices=ENUM.OrderStatus,
default=ENUM.OrderStatus[0], verbose_name="Статус замовлення")


    class Meta:
        db_table="order"
        verbose_name="Замовлення"
        verbose_name_plural="Замовлення"

    def __str__(self):
        return f"Замовлення номер {self.pk} | Покупець {self.user.first_name}
{self.user.last_name}"



class OrderitemQueryset(models.QuerySet):

    def total_quantity(self):
        if self:
            return sum(cart.quantity for cart in self)
        return 0

    def total_price(self):
        return sum(cart.products_total_price() for cart in self)

class OrderItem(models.Model):
```

```
    order = models.ForeignKey(to=Order,
on_delete=models.CASCADE,verbose_name="Замовлення")
    product=models.ForeignKey(to=Products, on_delete=models.SET_DEFAULT,blank=True,
null=True,verbose_name="Продукт",default=None)
    name=models.CharField(max_length=150,verbose_name="Назва")
    price=models.DecimalField(max_digits=10,decimal_places=2,verbose_name="Ціна")
    quantity=models.PositiveIntegerField(default=0,verbose_name="Кількість")
    created_timstamp=models.DateTimeField(auto_now_add=True,verbose_name="Дата
продажі")

    class Meta:
        db_table="order_item"
        verbose_name="Замовлений товар"
        verbose_name_plural ="Замовлені товари"
    objects=OrderitemQueryset.as_manager()
    def product_price(self):
        return round(self.price* self.quantity, 2)

    def __str__(self):
        return f"Товар {self.name} | Замовлення номер {self.order.pk}"
```

ДОДАТОК Б

Код Програми

Сторінок 26

Київ-2025

Папка app(головна)

settings.py

```
from pathlib import Path

from django.conf.global_settings import AUTH_USER_MODEL, LOGIN_REDIRECT_URL,
MEDIA_ROOT, MEDIA_URL

import os
from dotenv import load_dotenv

load_dotenv()


# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/5.1/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = os.getenv("SECRET_KEY")

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG =  os.getenv("DEBUG")

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.postgres',

    "debug_toolbar",

    'general',
    'goods',
    'users',
    'carts',
    'orders',
]
```

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',

    "debug_toolbar.middleware.DebugToolbarMiddleware",

]

ROOT_URLCONF = 'app.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR/'templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'app.wsgi.application'


# Database
# https://docs.djangoproject.com/en/5.1/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': os.getenv("POSTGRESQL_ENGINE"),
        'NAME': os.getenv("POSTGRESQL_NAME"),
        'USER': os.getenv("POSTGRESQL_USER"),
        'PASSWORD':os.getenv("POSTGRESQL_PASSWORD"),
        'HOST': os.getenv("POSTGRESQL_HOST"),
        'PORT':os.getenv("POSTGRESQL_PORT"),
    }
}

CACHES = {
```

```
    "default": {
        "BACKEND": os.getenv("CACHES_BACKEND"),
        "LOCATION": os.getenv("CACHES_LOCATION"),
    }
}
# Password validation
# https://docs.djangoproject.com/en/5.1/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/5.1/topics/i18n/

LANGUAGE_CODE = 'uk'
TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/5.1/howto/static-files/

STATIC_URL = 'static/'

STATICFILES_DIRS=[
    BASE_DIR / 'static'
    ]

MEDIA_URL='media/'

MEDIA_ROOT= BASE_DIR/'media'

INTERNAL_IPS = [
```

```
    # ...
    "127.0.0.1",
    # ...
]
# Default primary key field type
# https://docs.djangoproject.com/en/5.1/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

AUTH_USER_MODEL='users.User'
LOGIN_URL='/user/login/'
LOGIN_REDIRECT_URL='/'
```

.env

```
POSTGRESQL_ENGINE='django.db.backends.postgresql'
POSTGRESQL_NAME='shop_furniture'
POSTGRESQL_USER='meneger'
POSTGRESQL_PASSWORD='123456789'
POSTGRESQL_HOST='localhost'
POSTGRESQL_PORT='19112'
POSTGRESQL_IMAGE='17.0-alpine'
POSTGRESQL_CONT_NAME='db_diplom'


REDIS_HOST='localhost'
REDIS_PORT='19113'
REDIS_USERNAME='shop'
REDIS_USERNAME_PASSWORD='shoppass'
REDIS_PASSWORD='root'
REDIS_IMAGE='latest'
REDIS_CONT_NAME='redis_container'

SECRET_KEY = 'django-insecure-6hv-i1=qh&my0@%_!x24a&m6xqwuk2sp*oxs4i!sc@c8z2uxs2'
CACHES_BACKEND='django.core.cache.backends.redis.RedisCache'
CACHES_LOCATION='redis://shop:shoppass@localhost:19113'
DEBUG='True'
```

urls.py

```
from django.contrib import admin
from django.urls import include, path
from debug_toolbar.toolbar import debug_toolbar_urls

from django.conf import settings
from django.conf.urls.static import static
```

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('general.urls', namespace='general')),
    path('catalog/', include('goods.urls', namespace='goods')),
    path('user/', include('users.urls', namespace='user')),
    path('cart/', include('carts.urls', namespace='cart')),
    path('orders/',include('orders.urls',namespace='orders')),

]

if settings.DEBUG:
    urlpatterns+= debug_toolbar_urls()
    urlpatterns+= static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)


admin.site.site_header="Панель адміністрації"
```

Папка carts

admin.py

```
from itertools import product
from django.contrib import admin

# Register your models here.
from carts.models import Cart

class CartTabAdmin(admin.TabularInline):
    model =Cart
    fields="product","quantity","created_timestamp"
    search_fields="product","quantity","created_timestamp"
    readonly_fields=("created_timestamp",)
    extra=1

@admin.register(Cart)
class CartAdmin(admin.ModelAdmin):
    list_display=["user","product_display","quantity","created_timestamp"]
    list_filter=["user","product__name","created_timestamp"]

readonly_fields=("created_timestamp","quantity","product","user","session_key",)

    def user_display(self,obj):
        if obj.user:
            return str(obj.user)
        return "Аноноимний користувач"
```

```
    def product_display(self,obj):
        return str(obj.product.name)
```

mixins.py

```
class CartMixin:
    def get_cart(self,request, product=None, cart_id=None):
        if request.user.is_authenticated:
            query_kwargs={"user": request.user}

        else:
            query_kwargs={"session_key": request.session.session_key}

        if product:
            query_kwargs["product"]= product

        if cart_id:
            query_kwargs["id"]=cart_id

        return Cart.objects.filter(**query_kwargs).first()

    def render_cart(self, request):
        user_cart=get_user_carts(request)
        context ={"carts": user_cart}
        referer = request.META.get('HTTP_REFERER')
        if reverse('orders:create_order') in referer:
            context["order"]=True
        return
render_to_string("carts/includes/included_cart.html",context,request=request)
```

urls.py

```
from django.urls import path
from carts import views

app_name='carts'

urlpatterns = [
    path('cart_add/',views.CartAddWiew.as_view(), name='cart_add'),
    path('cart_change/',views.CartChengeView.as_view(), name='cart_change'),
    path('cart_remove/',views.CartRemoveView.as_view(), name='cart_remove'),

]
```

utils.py

```
from carts.models import Cart


def get_user_carts(request):
    if request.user.is_authenticated:
```

```
        return
Cart.objects.filter(user=request.user).select_related('product').order_by('id')
    if not request.session.session_key:
        request.session.create()
    return
Cart.objects.filter(session_key=request.session.session_key).select_related('produc
t').order_by('id')
```

views.py

```
from urllib import response
from django.http import JsonResponse
from django.shortcuts import redirect, render
from django.template.loader import render_to_string
from django.views import View

from carts.mixins import CartMixin
from carts.utils import get_user_carts
from carts.models import Cart
from goods.models import Products

class CartAddWiew(CartMixin,View):
    def post(self,request):
        product_id =request.POST.get("product_id")
        product = Products.objects.get(id=product_id)

        cart= self.get_cart(request,product=product)

        if request.user.is_authenticated:
            carts=Cart.objects.filter(user=request.user, product=product)

            if carts.exists():
                cart=carts.first()

                if cart:
                    cart.quantity += 1
                    cart.save()
            else:
                Cart.objects.create(user=request.user, product=product, quantity=1)
        else:

            carts =Cart.objects.filter(session_key=request.session.session_key,
product=product)
            if carts.exists():
                cart=carts.first()
                if cart:
                    cart.quantity +=1
                    cart.save()
```

```
            else:
                Cart.objects.create(session_key=request.session.session_key,
product=product,quantity=1)

        response_data={
            "message": "Товар додано до корзини",
            "cart_items_html": self.render_cart(request=request),
        }
        return JsonResponse(response_data)

class CartChengeView(CartMixin,View):
    def post(self,request):
        cart_id=request.POST.get("cart_id")

        cart=self.get_cart(request,cart_id=cart_id)
        cart.quantity=request.POST.get("quantity")
        cart.save()
        updated_quantity=cart.quantity
        response_data={
        "message": "Кількість змінена",
        "cart_items_html":self.render_cart(request=request),
        "quantity":updated_quantity,
        }
        return JsonResponse(response_data)

class CartRemoveView(CartMixin,View):
    def post(self,request):
        cart_id=request.POST.get("cart_id")

        cart=self.get_cart(request,cart_id=cart_id)
        updated_quantity=cart.quantity
        cart.delete()
        response_data={
        "message": " Товар видалено",
        "cart_items_html":self.render_cart(request=request),
        "quantity_deleted": updated_quantity,
        }
        return JsonResponse(response_data)
```

Папка general

urls.py

```
from django.urls import path
from general import views
from django.views.decorators.cache import cache_page
app_name='general'
```

```
urlpatterns = [
    path('', views.IndexView.as_view(), name='index'),
    path('about_us/', views.AboutView.as_view(), name='about'),

]
```

views.py

```
from urllib import response
from django.http import JsonResponse
from django.shortcuts import redirect, render
from django.template.loader import render_to_string
from django.views import View

from carts.mixins import CartMixin
from carts.utils import get_user_carts
from carts.models import Cart
from goods.models import Products

class CartAddWiew(CartMixin,View):
    def post(self,request):
        product_id =request.POST.get("product_id")
        product = Products.objects.get(id=product_id)

        cart= self.get_cart(request,product=product)

        if request.user.is_authenticated:
            carts=Cart.objects.filter(user=request.user, product=product)

            if carts.exists():
                cart=carts.first()

                if cart:
                    cart.quantity += 1
                    cart.save()
            else:
                Cart.objects.create(user=request.user, product=product, quantity=1)
        else:

            carts =Cart.objects.filter(session_key=request.session.session_key,
product=product)
            if carts.exists():
                cart=carts.first()
                if cart:
                    cart.quantity +=1
                    cart.save()
            else:
                Cart.objects.create(session_key=request.session.session_key,
product=product,quantity=1)

        response_data={
```

```
            "message": "Товар додано до корзини",
            "cart_items_html": self.render_cart(request=request),
        }
        return JsonResponse(response_data)

class CartChengeView(CartMixin,View):
    def post(self,request):
        cart_id=request.POST.get("cart_id")

        cart=self.get_cart(request,cart_id=cart_id)
        cart.quantity=request.POST.get("quantity")
        cart.save()
        updated_quantity=cart.quantity
        response_data={
        "message": "Кількість змінена",
        "cart_items_html":self.render_cart(request=request),
        "quantity":updated_quantity,
        }
        return JsonResponse(response_data)

class CartRemoveView(CartMixin,View):
    def post(self,request):
        cart_id=request.POST.get("cart_id")

        cart=self.get_cart(request,cart_id=cart_id)
        updated_quantity=cart.quantity
        cart.delete()
        response_data={
        "message": " Товар видалено",
        "cart_items_html":self.render_cart(request=request),
        "quantity_deleted": updated_quantity,
        }
        return JsonResponse(response_data)
```

Папка goods

admin.py

```
from django.contrib import admin
from django.utils.safestring import mark_safe
from goods.models import Categoris,Products

#admin.site.register(Categoris)
#admin.site.register(Products)

class DiscountFilter(admin.SimpleListFilter):
    title="Наявність знижки"
    parameter_name = 'discount'

    def lookups(self,request,model_admin):
```

```
        return[
            ('discount','З знижкою'),
            ('no_discount','Без знижки'),
        ]
    def queryset(self, request, queryset):
        if self.value()=='discount':
            return queryset.filter(discount__gt=0)
        elif self.value()=='no_discount':
            return queryset.filter(discount__lte=0)



@admin.register(Categoris)
class CategariesAdmin(admin.ModelAdmin):
    prepopulated_fields={'slug':('name',)}
    list_display=('id','name')
    list_display_links=('id','name')



# Register your models here.
@admin.register(Products)
class ProductsAdmin(admin.ModelAdmin):
    prepopulated_fields={'slug':('name',)}

fields=['name','category','slug','description','image','post_image','price','discou
nt','quantity','rate','created','modified']

list_display=('id','post_image','name','quantity','rate','price','discount','catego
ry')
    list_display_links=('id','name')
    list_editable=('price','discount','quantity',)
    list_per_page=10
    search_fields=('name',)
    list_filter=[DiscountFilter,'category__name']
    readonly_fields=['post_image','rate','created','modified',]
    save_on_top=True

    def delete_queryset(self, request, queryset):
        for prod in queryset:
            prod.delete()
        return super().delete_queryset(request, queryset)

    @admin.display(description="Фотографія",ordering='content')
    def post_image(self, product: Products):
        if product.image:
            return mark_safe(f"<img src='{product.image.url}' width=80>")
        else: return "Без фото"
```

urls.py

```
from django.urls import path
from goods import views

app_name='goods'

urlpatterns = [
    path('search/', views.CatalogView.as_view(), name='search'),
    path('<slug:category_slug>/', views.CatalogView.as_view(), name='catalog'),
    path('product/<slug:product_slug>/', views.ProductView.as_view(),
name='product'),
]
```

utils.py

```
from django.db.models import Q, Value
from django.contrib.postgres.search import SearchVector, SearchQuery,
SearchRank,SearchHeadline
from goods.models import Products


def q_search(query):
    if query.isdigit() and len(query)<=3:
        return Products.objects.filter(id=int(query))
    vector = SearchVector("name","description")
    query=SearchQuery(query)
    result=
Products.objects.annotate(rank=SearchRank(vector,query,normalization=Value(2).bitor
(Value(4)))).filter(rank__gt=0).order_by("-rank")
    result=result.annotate(headline=SearchHeadline("name",query,start_sel='<span
style="background-color: blue; color: white ">',stop_sel="</span>"))

result=result.annotate(bodyline=SearchHeadline("description",query,start_sel='<span
style="background-color: blue; color: white">',stop_sel="</span>"))
    return result
```

views.py

```
from django.core.paginator import Paginator
from django.shortcuts import get_list_or_404, render
from django.views.generic import DetailView, ListView
from goods.utils import q_search
from goods.models import Products


class ProductView(DetailView):
    template_name ="goods/product.html"
    context_object_name ="product"
    slug_url_kwarg="product_slug"
```

```
    def get_object(self, queryset = None):
        product = Products.objects.get(slug=self.kwargs.get(self.slug_url_kwarg))
        return product

    def get_context_data(self, **kwargs):
        context =super().get_context_data(**kwargs)
        context["title"]=self.object.name
        return  context

class CatalogView(ListView):
    model=Products
    template_name="goods/catalog.html"
    context_object_name="goods"
    paginate_by=6

    def get_queryset(self):
        category_slug=self.kwargs.get("category_slug")
        on_sale=self.request.GET.get('on_sale')
        order_by=self.request.GET.get('order_by')
        query=self.request.GET.get('q')


        if category_slug=='all':
            goods=Products.objects.all()
        elif query:
            goods=q_search(query)

        else:

goods=get_list_or_404(Products.objects.filter(category__slug=category_slug))

        if on_sale:
            goods=goods.filter(discount__gt=0)

        if order_by and order_by !='default':
            goods=goods.order_by(order_by)

        return goods
    def get_context_data(self, **kwargs):
        context =super().get_context_data(**kwargs)
        context["title"]="Каталог"
        context["slug_url"]=self.kwargs.get("category_slug")
        return  context
```

Папка orders

admin.py

```
from re import search
from django.contrib import admin
```

```
from orders.models import Order, OrderItem

class OrderItemTabulareAdmin(admin.TabularInline):
    model=OrderItem
    fields="product","name","price","quantity"
    search_fields =("product","name",)
    extra = 0


class OrderTabularAdmin(admin.TabularInline):
    model =Order
    fields
=("requires_delivery","status","payment_on_get","is_paid","created_timestamp",)
    search_fields =("id","requires_delivery","status",)
    list_filter=("requires_delivery","status","is_paid","created_timestamp",)
    readonly_fields=("created_timestamp",)

    extra = 0


@admin.register(OrderItem)
class OrderItemAdmin(admin.ModelAdmin):
    list_display ="order","product","name","price","quantity"
    search_fields=("order","product","name",)
    list_display_links=("order","product",)
    list_per_page=10
    save_on_top=True



@admin.register(Order)
class OrderAdmin(admin.ModelAdmin):

list_display=("id","user","requires_delivery","status","delivery_adress","payment_o
n_get","is_paid","created_timestamp",)
    search_fields
=("id","requires_delivery","payment_on_get","is_paid","created_timestamp",)
    readonly_fields=("created_timestamp",)

list_filter=("requires_delivery","status","payment_on_get","is_paid","created_times
tamp",)
    inlines=(OrderItemTabulareAdmin,)
    list_display_links=("id","user",)
    list_editable=('status','is_paid',)
    list_per_page=10
    save_on_top=True
```

forms.py

```
from ast import pattern
import re
from django import forms

class CreateOrderForm(forms.Form):


    first_name = forms.CharField()
    last_name = forms.CharField()
    phone_number = forms.CharField()
    requires_delivery = forms.ChoiceField(choices=[
        ("0",False),
        ("1",True),
    ],)
    delivery_adress = forms.CharField(required=False)
    payment_on_get = forms.ChoiceField(choices=[
        ("0",'False'),
        ("1",'True'),
    ],)

    def clean_phone_number(self):
        data = self.cleaned_data['phone_number']
        if not data.isdigit():
            raise forms.ValidationError("Номер телефона повинен мати лише числа")

        pattern = re.compile(r'^\d{10}$')
        if not pattern.match(data):
            raise forms.ValidationError("Неправільний формат номера телефона")

        return data
```

urls.py

```
from django.urls import path
from orders import views

app_name='orders'

urlpatterns = [
    path('create-order/',views.CreateOrderView.as_view(), name='create_order'),
]
```

utils.py

```
rom django.db import transaction
from django.forms import ValidationError

from orders.forms import CreateOrderForm
from orders.models import Order, OrderItem
from carts.models import Cart
```

```
def create_order(user,form,statusdef):
    with transaction.atomic():
        cart_items = Cart.objects.filter(user=user)

        if cart_items.exists():
            order = Order.objects.create(
                user=user,
                phone_number=form.cleaned_data["phone_number"],
                requires_delivery=form.cleaned_data["requires_delivery"],
                delivery_adress=form.cleaned_data["delivery_adress"],
                payment_on_get=form.cleaned_data["payment_on_get"],
                status=statusdef[1],
            )
            for cart_item in cart_items:
                product = cart_item.product
                name = cart_item.product.name
                price = cart_item.product.d_price()
                quantity = cart_item.quantity

                if product.quantity < quantity:
                    raise ValidationError(
                        f"Недостатня кількість товара {name} на складі. На даний
момент є {product.quantity} штук даного товара"
                    )
                OrderItem.objects.create(
                    order=order,
                    product=product,
                    name=name,
                    price=price,
                    quantity=quantity,
                )
                product.quantity -= quantity
                product.save()
            cart_items.delete()
    return 200
```

views.py

```
from django.contrib import messages
from django.db import transaction
from django.forms import ValidationError
from django.shortcuts import redirect, render
from django.contrib.auth.decorators import login_required
from django.urls import reverse_lazy
from django.views.generic import FormView
from orders.utils import create_order
from orders.forms import CreateOrderForm
from orders.models import ENUM, Order, OrderItem
from carts.models import Cart
from django.contrib.auth.mixins import LoginRequiredMixin
```

```
class CreateOrderView(LoginRequiredMixin, FormView):
    template_name = "orders/create_order.html"
    form_class = CreateOrderForm
    success_url=reverse_lazy("user:profile")

    def get_initial(self):
        initial =super().get_initial()
        initial["first_name"]=self.request.user.first_name
        initial["last_name"]=self.request.user.last_name
        initial["email"]=self.request.user.email
        return initial

    def form_valid(self, form):
        statusdef = ENUM.OrderStatus[0]

        try:
            create_order(user=self.request.user,form=form,statusdef=statusdef)
            messages.success(self.request, "Замовлення офорилене дякую за покупку")
            return redirect("user:profile")
        except ValidationError as e:
            messages.error(self.request, str(e))
            return redirect("orders:create_order")

    def form_invalid(self, form):
        messages.warning(self.request, "Неправільні заповнені данні")
        return redirect('orders:create_order')


    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context["order"] = True
        context['title']="замовлення"

        return context
```

Папка users

admin.py

```
from re import search
from typing import Literal
from django.contrib import admin

from orders.admin import OrderItemTabulareAdmin, OrderTabularAdmin
from carts.admin import CartTabAdmin
from users.models import User
```

```
@admin.register(User)
class UserAdmin(admin.ModelAdmin):
    list_display=["username","first_name","last_name","email",]
    search_display=["username","first_name","last_name","email",]
    readonly_fields=("email","password","date_joined","last_login")

    inlines=[CartTabAdmin, OrderTabularAdmin,]
    save_on_top=True
```

forms.py

```
from django import forms
from django.contrib.auth.forms import
AuthenticationForm,UserCreationForm,UserChangeForm
from django.template.defaultfilters import first

from users.models import User

class UserLoginForm(AuthenticationForm):

    class Meta:
      model=User
      fields =['username','password']

    username = forms.CharField()
    password = forms.CharField()

class ProfileForm(UserChangeForm):
  class Meta:

    model=User
    fields=(
      "image",
      "first_name",
      "last_name",
      "username",
      "email",
    )
    image = forms.ImageField(required=False)
    first_name=forms.CharField()
    last_name=forms.CharField()
    username=forms.CharField()
    email=forms.CharField()

class UserRegistrationForm(UserCreationForm):

  class Meta:
```

```
    model= User
    fields=(
      "first_name",
      "last_name",
      "username",
      "email",
      "password1",
      "password2",
    )
    first_name=forms.CharField()
    last_name=forms.CharField()
    username=forms.CharField()
    email=forms.CharField()
    password1=forms.CharField()
    password2=forms.CharField()
```

urls.py

```
from django.urls import path
from users import views

app_name='users'

urlpatterns = [
    path('login/', views.UserLoginView.as_view(), name='login'),
    path('registration/', views.UserRegistrationView.as_view(),
name='registration'),
    path('profile/', views.UserProfileView.as_view(), name='profile'),
    path('logout/', views.logout, name='logout'),
    path('users-cart/', views.UserCartView.as_view(), name='users_cart'),


]
```

views.py

```
from string import Template
from django.contrib import auth, messages
from django.contrib.auth.decorators import login_required
from django.core.cache import cache
from django.db.models import Prefetch
from django.contrib.auth.views import LoginView
from django.contrib.auth.mixins import LoginRequiredMixin
from django.http import HttpResponseRedirect
from django.shortcuts import redirect, render
from django.urls import reverse, reverse_lazy
from django.views.generic import CreateView, TemplateView, UpdateView
from common.cache import CacheMixin
from orders.models import Order, OrderItem
```

```
from carts.models import Cart
from users.forms import ProfileForm, UserLoginForm, UserRegistrationForm

class UserLoginView(LoginView):
    template_name='users/login.html'
    form_class =UserLoginForm
    #success_url =reverse_lazy('general:index')
    def get_success_url(self):
        redirect_page=self.request.POST.get('next',None)
        if redirect_page and redirect_page != reverse('user:logout'):
            return redirect_page
        return reverse_lazy('general:index')
    def form_valid(self, form):
        session_key=self.request.session.session_key

        user = form.get_user()
        cart=Cart.objects.filter(session_key=session_key)
        if user:
            auth.login(self.request, user)
            if session_key and cart:
                forgot_carts=Cart.objects.filter(user=user)
                if forgot_carts.exists():
                    forgot_carts.delete()
                Cart.objects.filter(session_key=session_key).update(user=user)

            messages.success(self.request, f"Вітаю {user.username}")
            return HttpResponseRedirect(self.get_success_url())



    def get_context_data(self, **kwargs):
        context=super().get_context_data(**kwargs)
        context['title']="Авторізація"
        return context

class UserRegistrationView(CreateView):
    template_name = 'users/registration.html'
    form_class=UserRegistrationForm
    success_url=reverse_lazy('user:login')

    def form_valid(self, form):
        session_key=self.request.session.session_key
        user=form.instance
        if user:
            form.save()

            if session_key:
                Cart.objects.filter(session_key=session_key).update(user=user)
            messages.success(self.request, f"{user.username}, Ви успішно
зарегестрірувались войдіть в свій акаунт")
```

```
        return HttpResponseRedirect(reverse('user:login'))


    def get_context_data(self, **kwargs):
        context=super().get_context_data(**kwargs)
        context['title']="Реєстрація"
        return context

class UserProfileView(LoginRequiredMixin,CacheMixin, UpdateView):
    template_name='users/profile.html'
    form_class=ProfileForm
    success_url=reverse_lazy('users:profile')

    def get_object(self, queryset = None):
        return self.request.user

    def form_valid(self, form):
        messages.success(self.request, f"Профайл успішно оновлений")
        return super().form_valid(form)

    def form_invalid(self, form):
        messages.warning(self.request, f"Виникла ошибка")

        return super().form_invalid(form)
    def get_context_data(self, **kwargs):
        context=super().get_context_data(**kwargs)
        context['title']="Профіль користувача"


orders=Order.objects.filter(user=self.request.user).prefetch_related(Prefetch("orde
ritem_set",queryset=OrderItem.objects.select_related("product"),)).order_by("-id")

context['orders']=self.set_get_cache(query=orders,cache_name=f"user_{self.request.u
ser.username}_orders",cache_time= 60*3 )
        return context

class UserCartView(TemplateView):
    template_name='users/users_cart.html'

    def get_context_data(self, **kwargs):
        context=super().get_context_data(**kwargs)
        context['title']="Корзина користувача"
        return context

@login_required
def logout(request):
    messages.success(request, f"{request.user.username}, Ви успішно вийшли з
акаунта")
```

```
    auth.logout(request)
    return redirect(reverse('general:index'))
```

Інші

cache.py

```
from collections import deque
from django.contrib import messages
from django.core.cache import cache
from django.shortcuts import redirect
import redis
import asyncio
from django.http import HttpResponse
import redis.exceptions



class CacheMixin:
   def set_get_cache(self, query, cache_name, cache_time):

        if check_cache():
            data = cache.get(cache_name)
            if not data:
                data = query
                cache.set(cache_name, data, cache_time)
            return data

        else:
            data=query
            return data


def  check_cache():
    r =  redis.Redis(
        host="localhost", port=19113, db=0, username="shop",
password="shoppass",socket_connect_timeout=1
    )
    try:
        response = r.ping()
        if response:
            return True
        else:
            return False


    except redis.exceptions.ConnectionError as e:
        print(f"Помилка: {e}")
        return False
```

jquery-ajax.js

```
$(document).ready(function () {


    var successMessage = $("#jq-notification");


    $(document).on("click", ".add-to-cart", function (e) {
         e.preventDefault();

        var goodsInCartCount = $("#goods-in-cart-count");
        var cartCount = parseInt(goodsInCartCount.text() || 0);

        var product_id = $(this).data("product-id");

        var add_to_cart_url = $(this).attr("href");

         $.ajax({
            type: "POST",
            url: add_to_cart_url,
            data: {
                product_id: product_id,
                csrfmiddlewaretoken: $("[name=csrfmiddlewaretoken]").val(),
             },
         success: function (data) {
               successMessage.html(data.message);
               successMessage.fadeIn(400);
                setTimeout(function () {
                    successMessage.fadeOut(400);
               }, 1000);

               cartCount++;
                 goodsInCartCount.text(cartCount);

               var cartItemsContainer = $("#cart-items-container");
                cartItemsContainer.html(data.cart_items_html);

           },

            error: function (data) {
                console.log("Ошибка при добавлении товара в корзину");
             },
         });
    });
```

```
$(document).on("click", ".remove-from-cart", function (e) {
    e.preventDefault();

    var goodsInCartCount = $("#goods-in-cart-count");
    var cartCount = parseInt(goodsInCartCount.text() || 0);

    var cart_id = $(this).data("cart-id");
    var remove_from_cart = $(this).attr("href");

    $.ajax({

        type: "POST",
        url: remove_from_cart,
        data: {
            cart_id: cart_id,
            csrfmiddlewaretoken: $("[name=csrfmiddlewaretoken]").val(),
        },
        success: function (data) {
            successMessage.html(data.message);
            successMessage.fadeIn(400);
            setTimeout(function () {
                successMessage.fadeOut(400);
            }, 1000);

            cartCount -= data.quantity_deleted;
            goodsInCartCount.text(cartCount);

            var cartItemsContainer = $("#cart-items-container");
            cartItemsContainer.html(data.cart_items_html);

        },

        error: function (data) {
            console.log("Ошибка при добавлении товара в корзину");
        },
    });
});




$(document).on("click", ".decrement", function () {
    var url = $(this).data("cart-change-url");
    var cartID = $(this).data("cart-id");
    var $input = $(this).closest('.input-group').find('.number');
    var currentValue = parseInt($input.val());
    if (currentValue > 1) {
        $input.val(currentValue - 1);
```

```
        updateCart(cartID, currentValue - 1, -1, url);
    }
});

$(document).on("click", ".increment", function () {
    var url = $(this).data("cart-change-url");
    var cartID = $(this).data("cart-id");
    var $input = $(this).closest('.input-group').find('.number');
    var currentValue = parseInt($input.val());

    $input.val(currentValue + 1);


    updateCart(cartID, currentValue + 1, 1, url);
});

function updateCart(cartID, quantity, change, url) {
    $.ajax({
        type: "POST",
        url: url,
        data: {
            cart_id: cartID,
            quantity: quantity,
            csrfmiddlewaretoken: $("[name=csrfmiddlewaretoken]").val(),
        },

        success: function (data) {
            successMessage.html(data.message);
            successMessage.fadeIn(400);
            setTimeout(function () {
                 successMessage.fadeOut(400);
            }, 1000);

            var goodsInCartCount = $("#goods-in-cart-count");
            var cartCount = parseInt(goodsInCartCount.text() || 0);
            cartCount += change;
            goodsInCartCount.text(cartCount);

            var cartItemsContainer = $("#cart-items-container");
            cartItemsContainer.html(data.cart_items_html);

        },
        error: function (data) {
            console.log("Помилка при додавнні товара в корзину");
        },
    });
}


var notification = $('#notification');
```

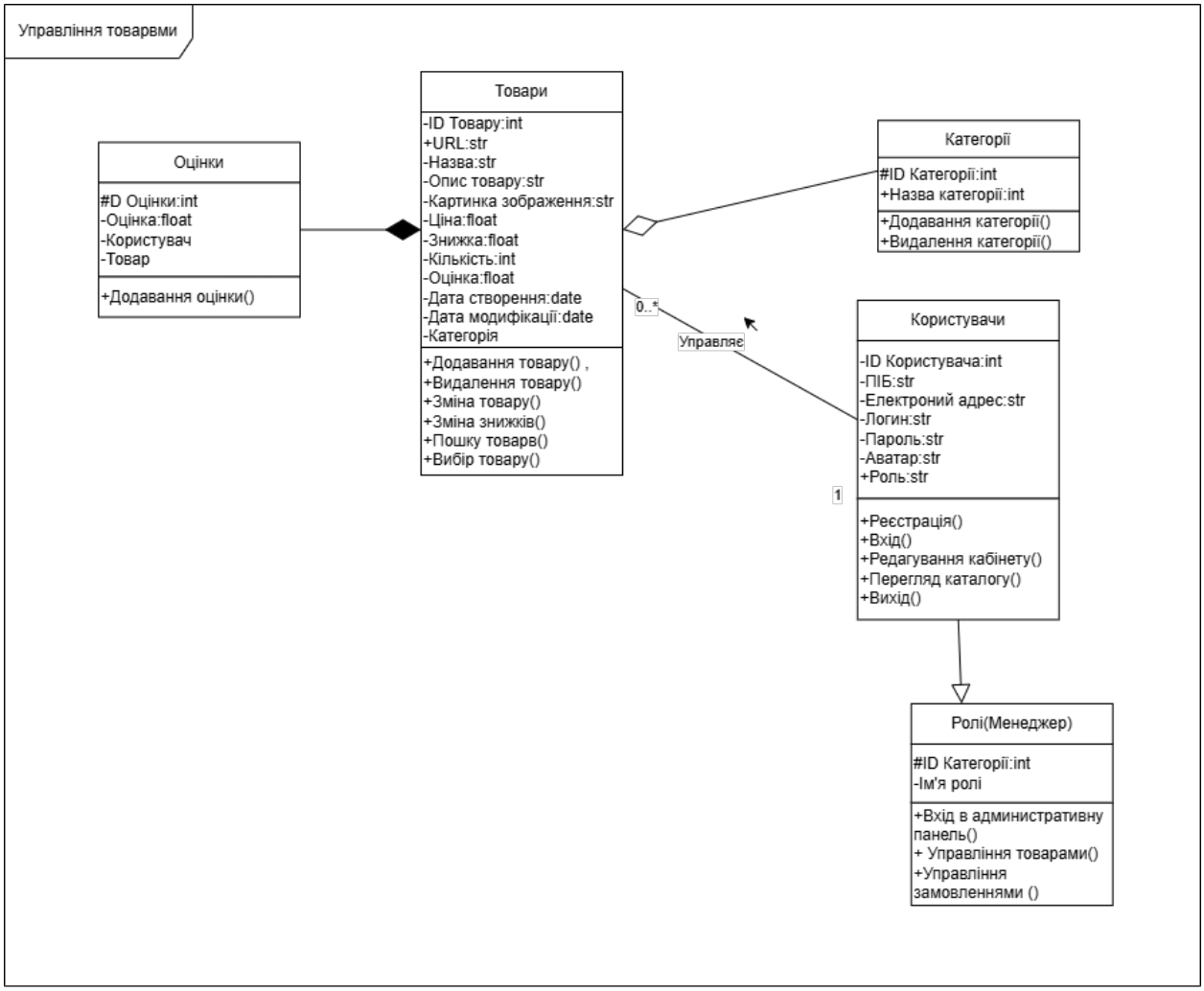
```
    if (notification.length > 0) {
        setTimeout(function () {
            notification.alert('close');
        }, 1000);
    }

    $('#modalButton').click(function () {
        $('#exampleModal').appendTo('body');

        $('#exampleModal').modal('show');
    });

    $('#exampleModal .btn-close').click(function () {
        $('#exampleModal').modal('hide');
    });

    $("input[name='requires_delivery']").change(function() {
        var selectedValue = $(this).val();
        if (selectedValue === "1") {
            $("#deliveryAddressField").show();
        } else {
            $("#deliveryAddressField").hide();
        }
    });

});
```

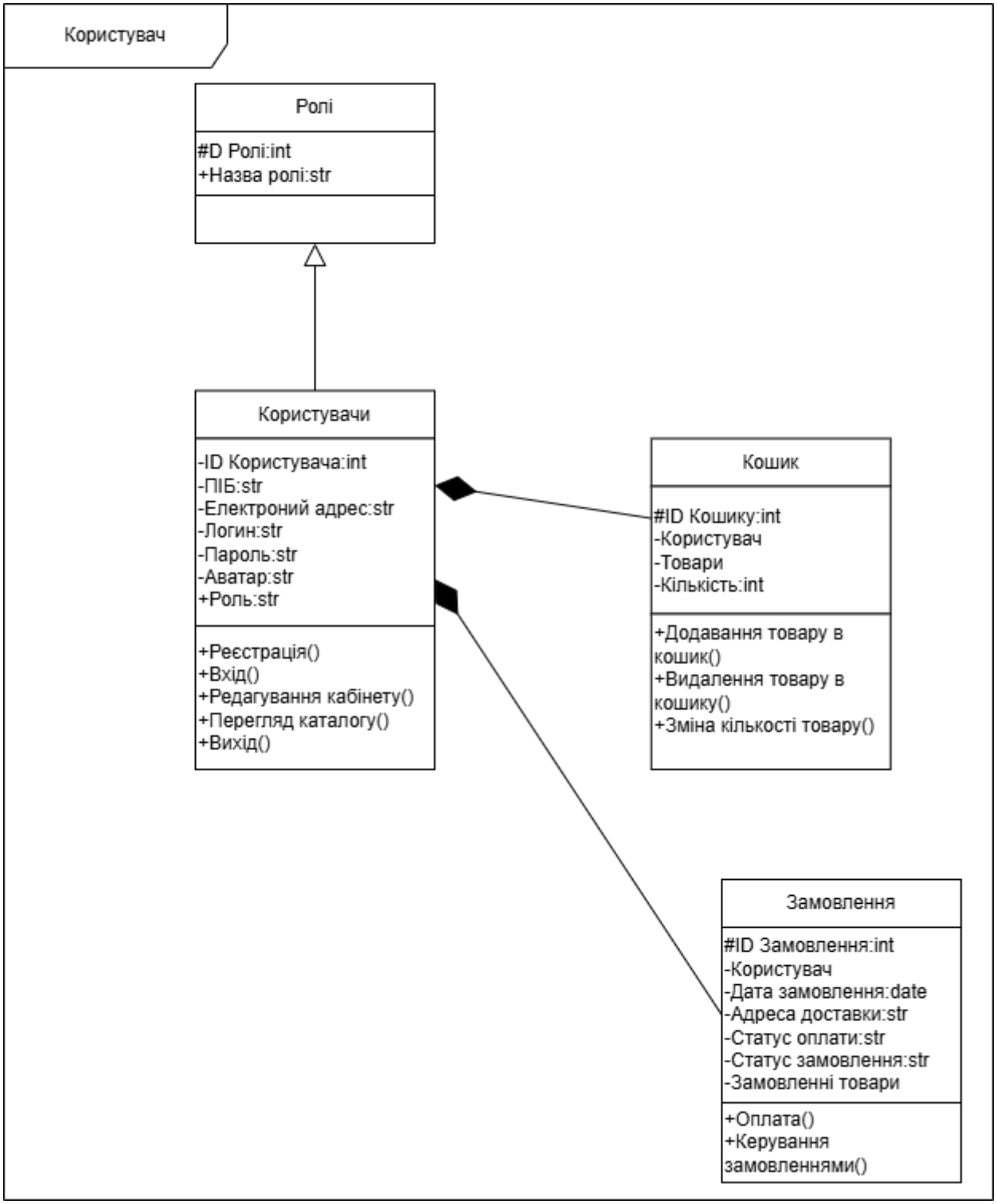
ДОДАТОК В

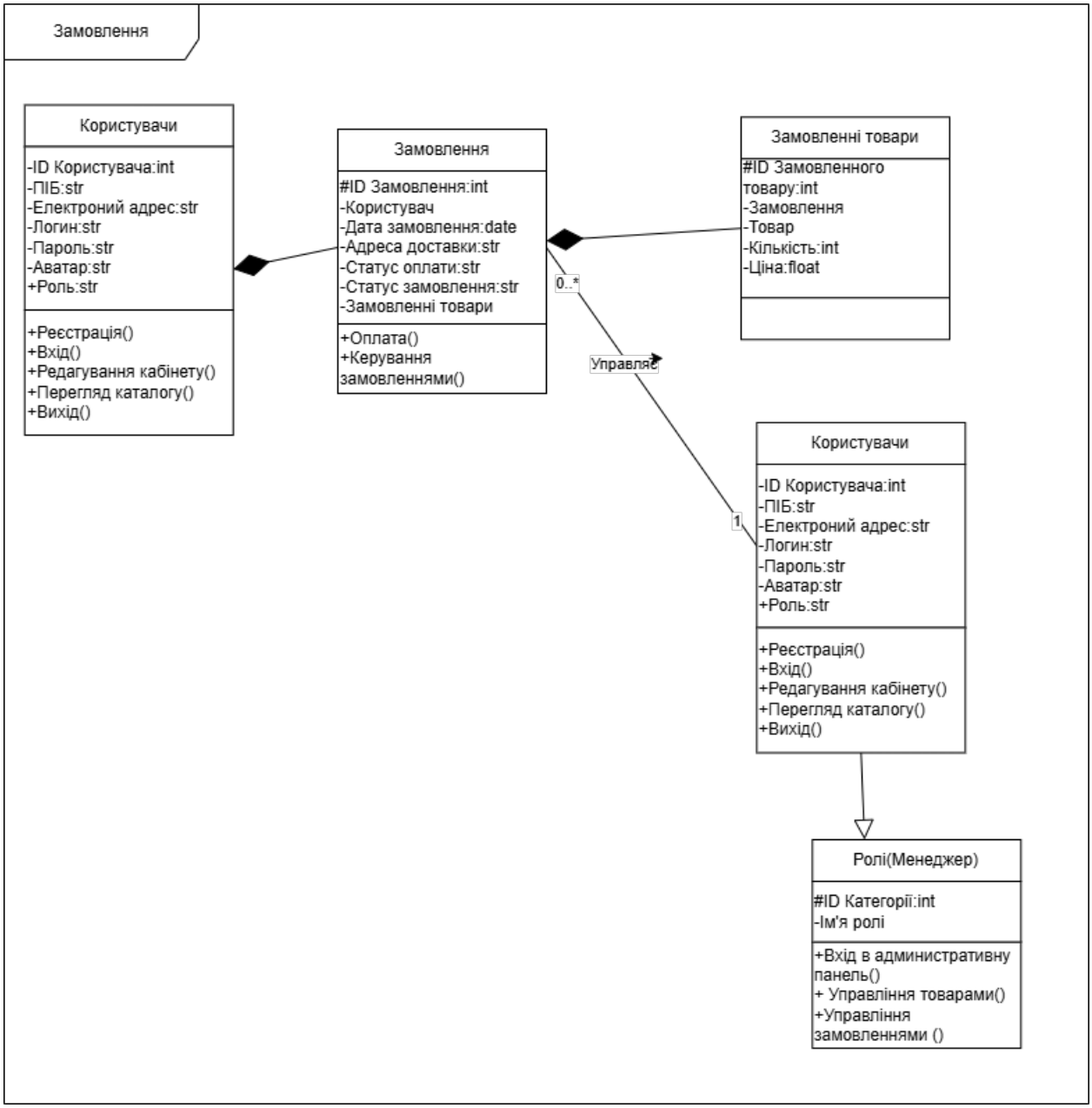
Діаграми кооперацій

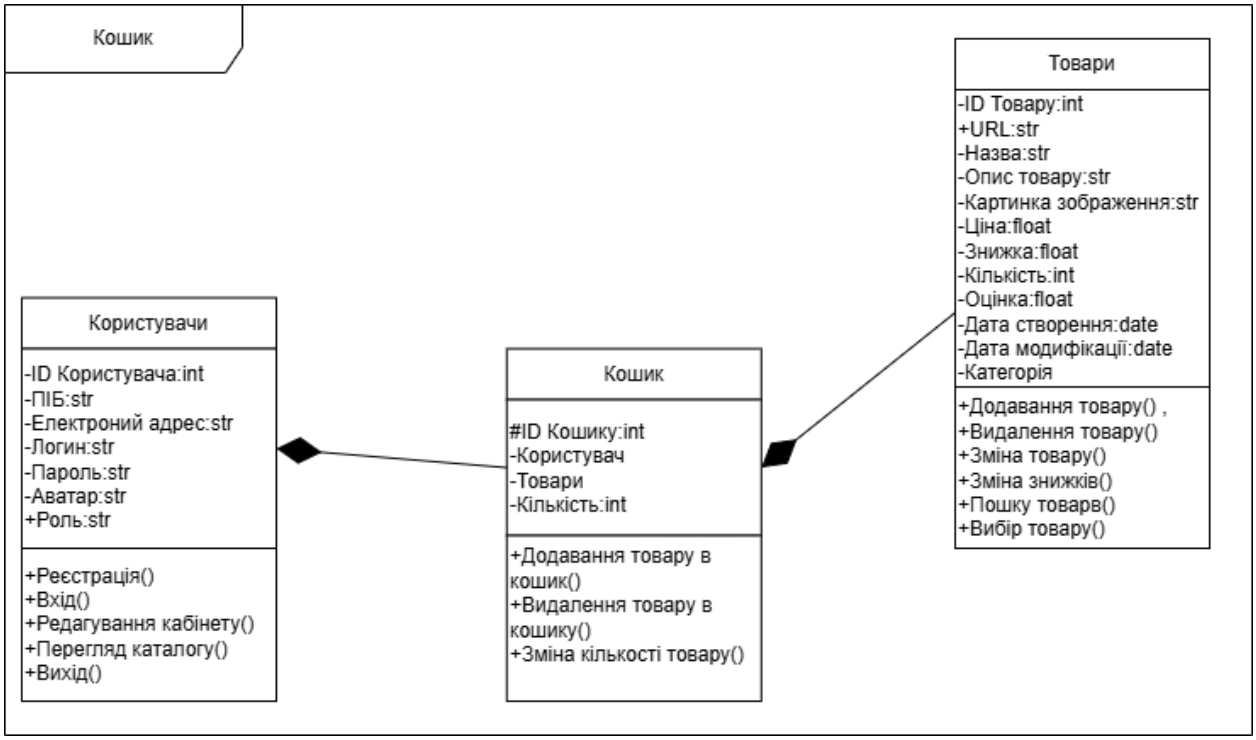
Сторінок 5

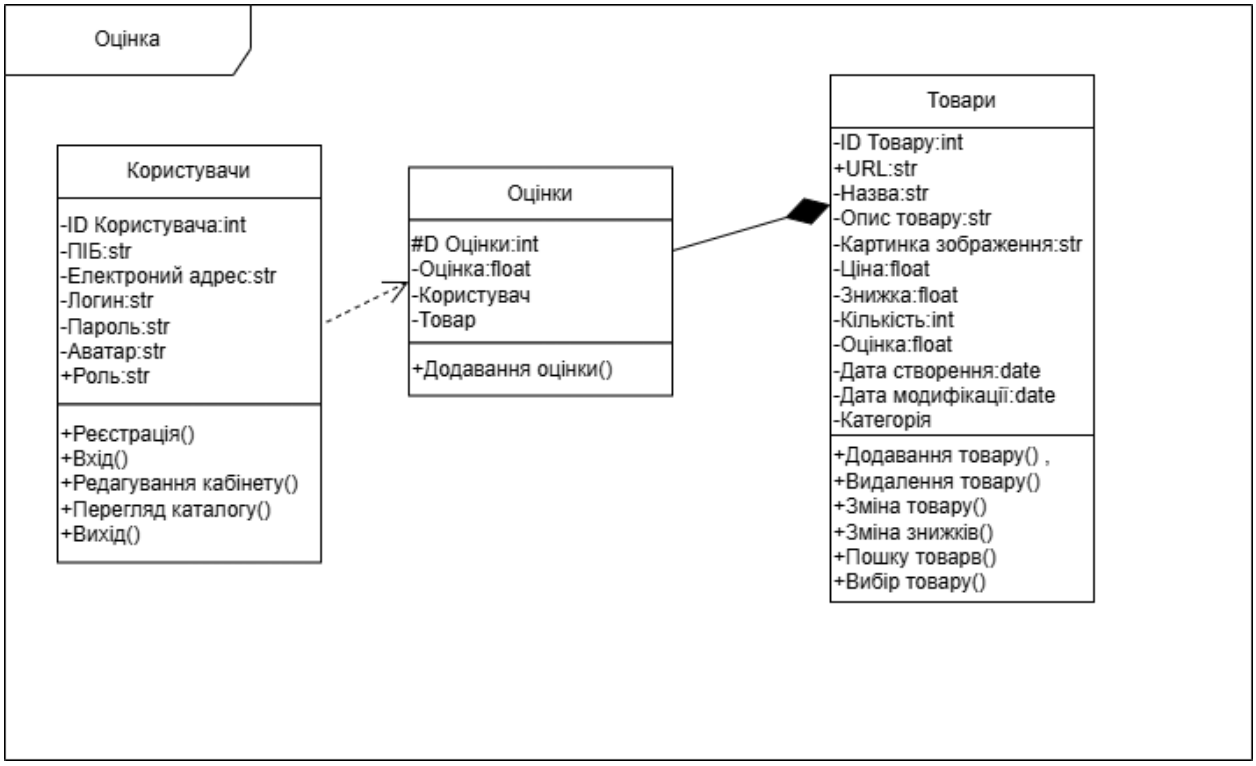
Київ-2025

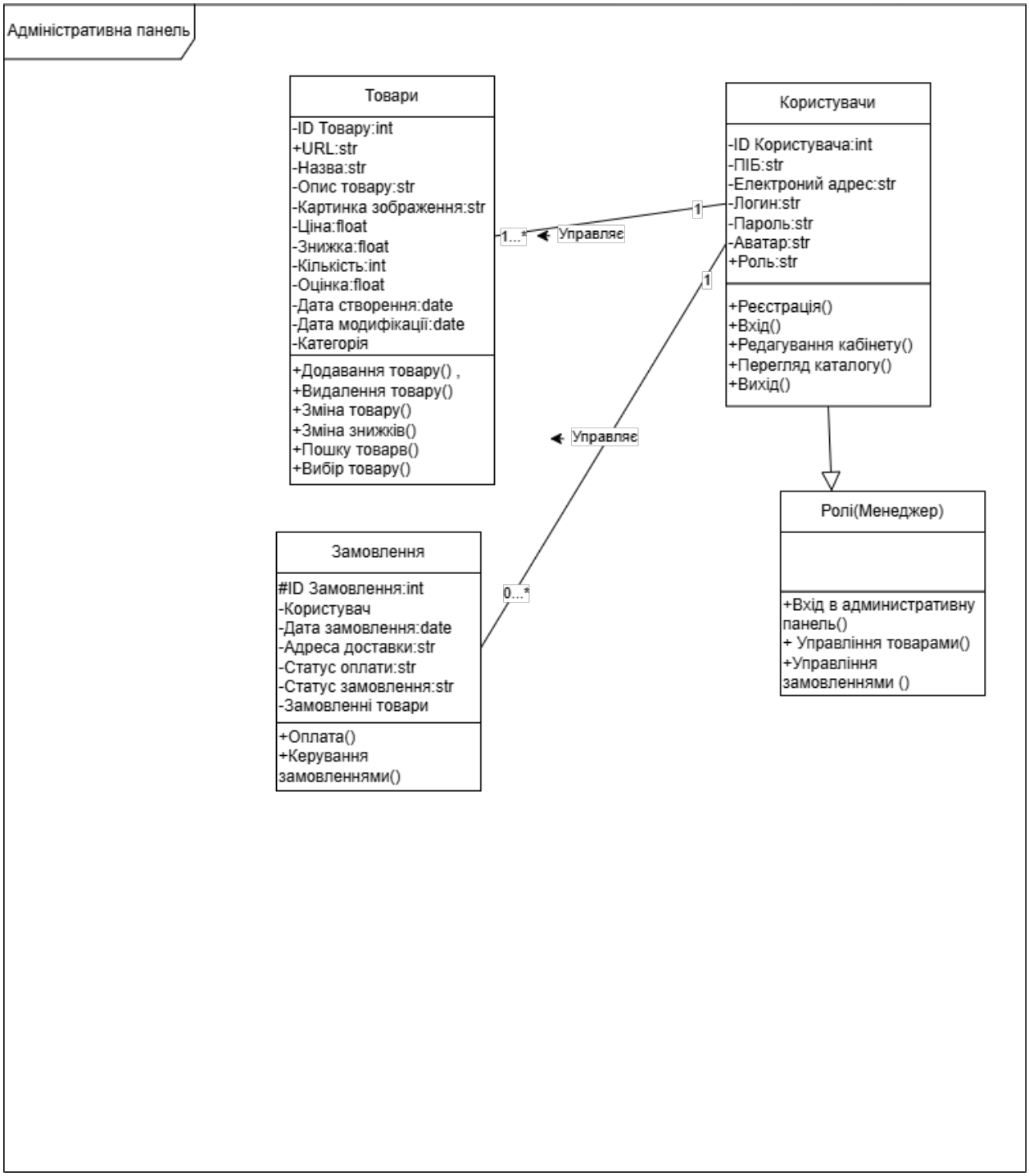

Управління товарвми
Товари
-ID Товару:int
+URL:str
-Назва:str
-Опис товару:str
-Картинка зображення:str
-Ціна:float
-Знижка:float
-Кількість:int
-Оцінка:float
-Дата створення:date
-Дата модифікації:date
-Категорія
+Додавання товару() ,
+Видалення товару()
+Зміна товару()
+Зміна знижків()
+Пошку товарв()
+Вибір товару()
Оцінки
#D Оцінки:int
-Оцінка:float
-Користувач
-Товар
+Додавання оцінки()
Категорії
#ID Категорії:int
+Назва категорії:int
+Додавання категорії()
+Видалення категорії()
0..*
Управляє
Користувачи
-ID Користувача:int
-ПІБ:str
-Електроний адрес:str
-Логин:str
-Пароль:str
-Аватар:str
+Роль:str
1
+Реєстрація()
+Вхід()
+Редагування кабінету()
+Перегляд каталогу()
+Вихід()
Ролі(Менеджер)
#ID Категорії:int
-Ім'я ролі
+Вхід в адміністративну панель()
+ Управління товарами()
+Управління замовленнями ()


Користувач
Ролі
#D Ролі:int
+Назва ролі:str
Користувачи
-ID Користувача:int
-ПІБ:str
-Електроний адрес:str
-Логин:str
-Пароль:str
-Аватар:str
+Роль:str
+Реєстрація()
+Вхід()
+Редагування кабінету()
+Перегляд каталогу()
+Вихід()
Кошик
#ID Кошику:int
-Користувач
-Товари
-Кількість:int
+Додавання товару в кошик()
+Видалення товару в кошику()
+Зміна кількості товару()
Замовлення
#ID Замовлення:int
-Користувач
-Дата замовлення:date
-Адреса доставки:str
-Статус оплати:str
-Статус замовлення:str
-Замовленні товари
+Оплата()
+Керування замовленнями()


Замовлення
Користувачи
-ID Користувача:int
-ПІБ:str
-Електроний адрес:str
-Логин:str
-Пароль:str
-Аватар:str
+Роль:str
+Реєстрація()
+Вхід()
+Редагування кабінету()
+Перегляд каталогу()
+Вихід()
Замовлення
#ID Замовлення:int
-Користувач
-Дата замовлення:date
-Адреса доставки:str
-Статус оплати:str
-Статус замовлення:str
-Замовленні товари
+Оплата()
+Керування замовленнями()
Замовленні товари
#ID Замовленного товару:int
-Замовлення
-Товар
-Кількість:int
-Ціна:float
0..*
Управляє
1
Користувачи
-ID Користувача:int
-ПІБ:str
-Електроний адрес:str
-Логин:str
-Пароль:str
-Аватар:str
+Роль:str
+Реєстрація()
+Вхід()
+Редагування кабінету()
+Перегляд каталогу()
+Вихід()
Ролі(Менеджер)
#ID Категорії:int
-Ім'я ролі
+Вхід в адміністративну панель()
+ Управління товарами()
+Управління замовленнями ()


Кошик
Товари
-ID Товару:int
+URL:str
-Назва:str
-Опис товару:str
-Картинка зображення:str
-Ціна:float
-Знижка:float
-Кількість:int
-Оцінка:float
-Дата створення:date
-Дата модифікації:date
-Категорія
+Додавання товару() ,
+Видалення товару()
+Зміна товару()
+Зміна знижків()
+Пошку товарв()
+Вибір товару()
Користувачи
-ID Користувача:int
-ПІБ:str
-Електроний адрес:str
-Логин:str
-Пароль:str
-Аватар:str
+Роль:str
+Реєстрація()
+Вхід()
+Редагування кабінету()
+Перегляд каталогу()
+Вихід()
Кошик
#ID Кошику:int
-Користувач
-Товари
-Кількість:int
+Додавання товару в кошик()
+Видалення товару в кошику()
+Зміна кількості товару()


Оцінка
Товари
-ID Товару:int
+URL:str
-Назва:str
-Опис товару:str
-Картинка зображення:str
-Ціна:float
-Знижка:float
-Кількість:int
-Оцінка:float
-Дата створення:date
-Дата модифікації:date
-Категорія
+Додавання товару() ,
+Видалення товару()
+Зміна товару()
+Зміна знижків()
+Пошку товарв()
+Вибір товару()
Користувачи
-ID Користувача:int
-ПІБ:str
-Електроний адрес:str
-Логин:str
-Пароль:str
-Аватар:str
+Роль:str
+Реєстрація()
+Вхід()
+Редагування кабінету()
+Перегляд каталогу()
+Вихід()
Оцінки
#D Оцінки:int
-Оцінка:float
-Користувач
-Товар
+Додавання оцінки()


Адміністративна панель
Товари
-ID Товару:int
+URL:str
-Назва:str
-Опис товару:str
-Картинка зображення:str
-Ціна:float
-Знижка:float
-Кількість:int
-Оцінка:float
-Дата створення:date
-Дата модифікації:date
-Категорія
+Додавання товару() ,
+Видалення товару()
+Зміна товару()
+Зміна знижків()
+Пошку товарв()
+Вибір товару()
1...*
Управляє
1
Користувачи
-ID Користувача:int
-ПІБ:str
-Електроний адрес:str
-Логин:str
-Пароль:str
-Аватар:str
+Роль:str
+Реєстрація()
+Вхід()
+Редагування кабінету()
+Перегляд каталогу()
+Вихід()
1
Управляє
Ролі(Менеджер)
+Вхід в адміністративну панель()
+ Управління товарами()
+Управління замовленнями ()
Замовлення
#ID Замовлення:int
-Користувач
-Дата замовлення:date
-Адреса доставки:str
-Статус оплати:str
-Статус замовлення:str
-Замовленні товари
+Оплата()
+Керування замовленнями()
0...*