МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук,
управління та адміністрування
Кафедра інформаційних технологій

Кваліфікаційна робота бакалавра
на тему: Розробка інтернет-магазина зоотоварів

Виконав студент групи К-18
спеціальності 122 «Комп'ютерні науки»
Козлов Микита Сергійович

Керівник к.т.н., доцент
Терещенко Тетяна Михайлівна

Консультант _____

_____

Рецензент к.т.н., доцент _____
Попов В.Л. _____

Одеса 2022

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД – база даних

ОС – операційна система

ПЗ – програмне забезпечення

API – Application Programming Interface

IDE – Integrated Development Environment – інтегроване середовище розробки

SPA – Single-page application

DOM – Document Object Model

ВСТУП

Зростання кількості домашніх тварин В Україні зробило ринок різних послуг від корму до сервісів другим за швидкістю зростання в Європі [1]. Що в свою чергу сприяє утворенню нових виробників.

Такі підприємства, не завжди мають можливість утримувати торгові площі, проте змушені конкурувати з іншими торговими підприємствами. Оскільки торгівля новою продукцією не цікавить діючих магазинів, виробники самі шукають способи реалізації[8].

Найкращий метод із матеріальних і тимчасових витрат за реалізацію своєї продукції і на ринкову торгівлю – це створення та подальша реклама у мережі інтернет-магазину. Даний вид торгівлі не вимагає утримання дорогих торгових площ, великої кількості співробітників, представництва у столиці, що дозволяє досить успішно існувати та розвиватися підприємствам у віддалених регіонах .

Інтернет, і особливо його частина, що називається Світова Павутина (World Wide Web), еволюціонує надзвичайно швидкими темпами. З кожним роком збільшується потреба у Web-додатках, серед яких особливе місце займають програми, які забезпечують засоби ведення бізнесу за допомогою Web. Це один із найбільш складних класів Web-додатків для створення яких необхідні не лише знання про сучасні інструменти, що використовуються для розробки Web-додатків, а також спеціальні знання про електронну комерцію як таку.

Метою кваліфікаційної роботи є розробка інтернет-магазину зоотоварів «Lapki Market». Для досягнення поставленої мети були сформульовані наступні завдання:

1. провести аналіз предметної області зоотоварів;
2. провести порівняльний аналіз існуючих аналогів;
3. обрати програмні засоби розробки та технології;

4. провести проектування системи з використанням мови UML;
5. виконати реалізацію застосунку;
6. підготувати інструкцію користувача;
7. виконати тестування застосунку.

Структура кваліфікаційної роботи бакалавра складається з вступу, 5 розділів, висновків, переліку посилань на 12 найменувань, 1 додатку. Повний обсяг проекту становить 74 сторінки, містить 40 рисунків і 1 таблицю.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ ПРОГРАМНИХ СИСТЕМ

1.1 Аналіз предметної області

Домашні тварини - тварини, які були одомашнені людиною і яких вона містить, надаючи їм дах і їжу. Вони приносять йому користь або як джерело матеріальних благ та послуг, або як тварини-компаньйони, що прикрашають його дозвілля.

Частина свійських тварин (сільськогосподарські тварини) приносить безпосередню матеріальну вигоду людині, наприклад, будучи джерелом їжі (молоко, м'ясо), матеріалів (шерсть, шкіра). Інші тварини (робоча худоба та службові тварини) приносять користь людині, виконуючи робочі функції (перевезення вантажів, охорона тощо).

Друга велика категорія - це тварини-компаньйони, які займають дозвілля, приносять задоволення і з якими можна спілкуватися. Для міських жителів поняття «домашні тварини» найчастіше асоціюється з другою категорією, тобто з «домашніми улюбленцями (вихованцями)». Багато сімей, які тримають вдома якихось тварин, відзначають, що ці тварини створюють затишок, заспокоюють, знімають стрес.

Основні фактори, які сприяли розвитку електронної комерції в сфері товарів для тварин такі ж самі як у всьому ринку електронної комерції, - поява нових зручних інструментів, у тому числі способів та швидкості доставки. До таких інструментів можна віднести регулярну доставку: наприклад, корм для тварин, який ви замовили один раз, але магазин приноситиме його регулярно в заздалегідь обумовлені дати - раз на місяць або інший погоджений період. Далі це закріплення звички купувати онлайн, розуміння, що це зручно, залучення тих, хто раніше не купував онлайн і нарешті зміна поколінь. Також як драйвер зростання варто назвати пандемію. Деякі люди побоюються купувати офлайн, тому для них комфортніше зробити покупку онлайн.

1.2 Аналіз існуючих програмних систем

На даний час існують різноманітні інтернет-магазини зоотоварів, деякі спеціалізуються на товарах для конкретних тварин, інші охоплюють майже всі види. Розглянемо деякі з них.

Інтернет-магазин pethouse (рис.1.1) дозволяє переглянути інформацію про наявні товари, а також зробити замовлення. Є великий асортимент товарів для різних видів домашніх тварин. Зручні категорії.

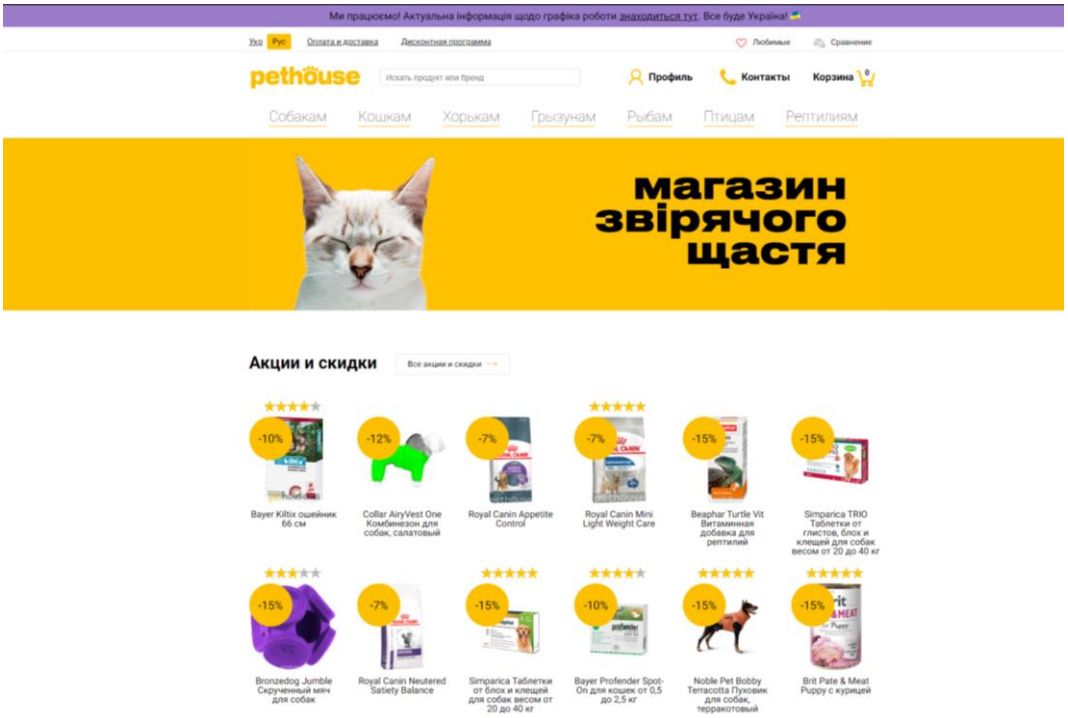


Рисунок 1.1 – Головна сторінка магазина pethouse

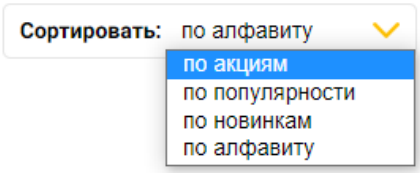


Рисунок 1.2 – Сортування магазина pethouse

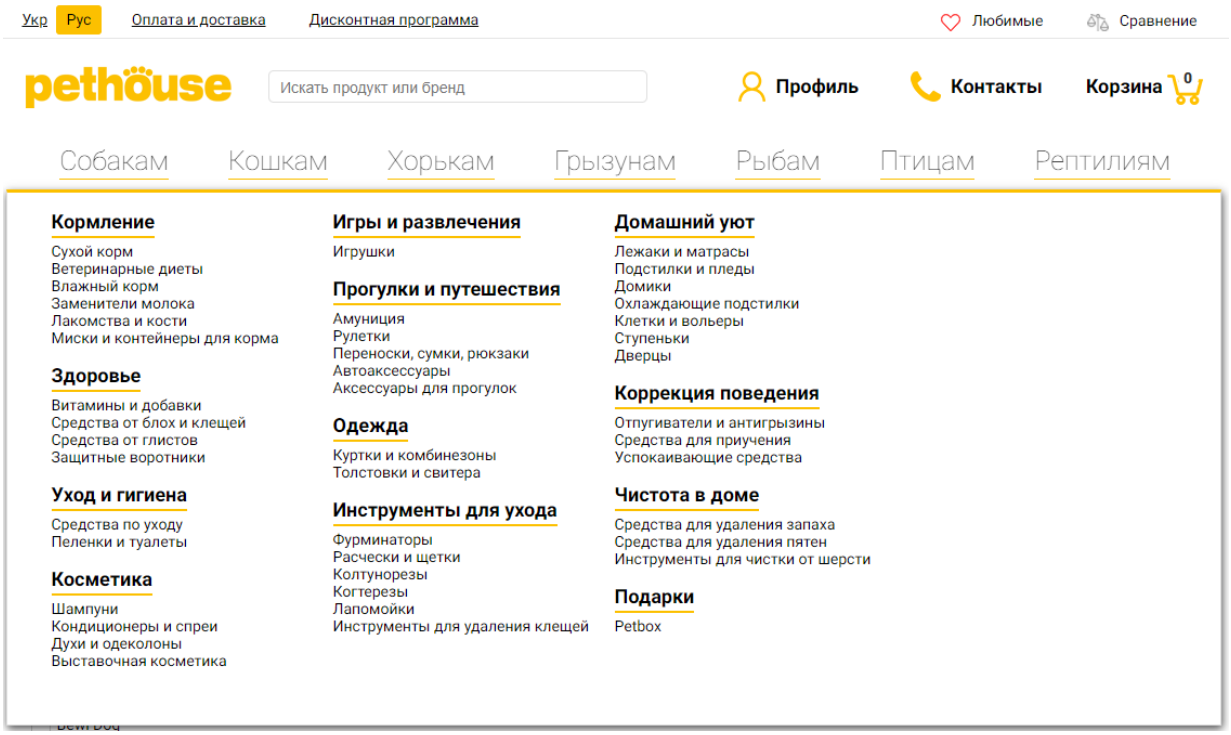


Рисунок 1.3 – Категорії магазина pethouse

Основний недолік магазина pethouse це неможливість сортувати за ціною.

Masterzoo (рис.1.4) – інтернет-магазин товарів для всіх видів домашніх улюбленців: собак, кішок, птахів, рибок, рептилій, екзотичних тварин. На сьогоднішній день функціонує понад 80 точок продажу. Онлайн-каталог включає сертифіковані корми, екологічні наповнювачі, технологічні аксесуари, інші товари для піклування про здоров'я та комфорт домашніх вихованців.

Асортимент зоомагазину MasterZoo: від їжі та ветеринарних препаратів до одягу. Сучасна ветеринарна індустрія дбає про хороше самопочуття та чудовий настрій хвостатих і регулярно представляє новинки. Сьогодні в каталогах можна знайти не тільки їжу, іграшки, повідці та доглядові засоби, але й інструменти для стрижки кігтів та вичісування вовни, зубні пасти та щітки, кігтеточки та лежанки, одяг та аксесуари.

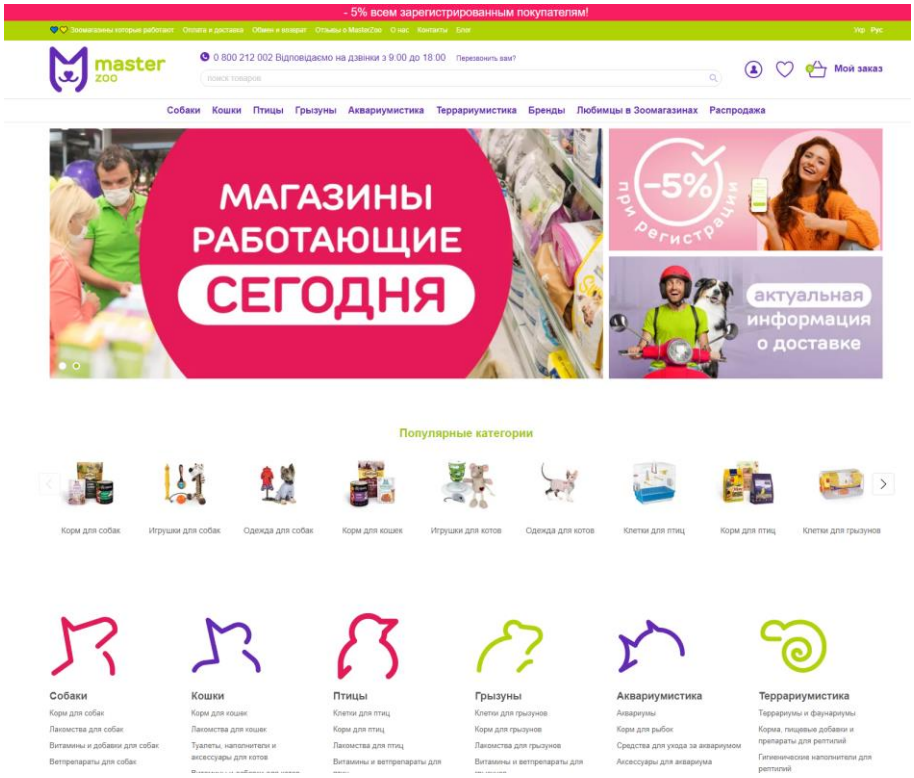


Рисунок 1.4 – Головна сторінка магазина Masterzoo

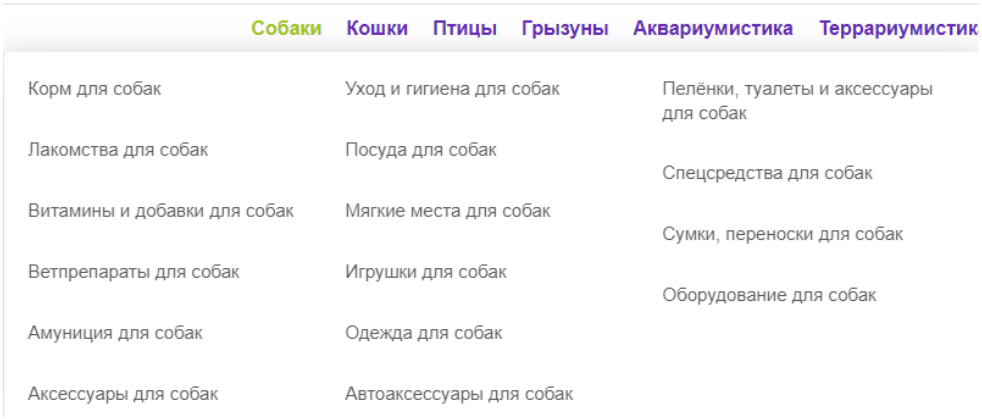


Рисунок 1.5 – Категорії магазина Masterzoo

Інтернет-магазин Zoostore (рис.1.6), пропонує купити товари для домашніх тварин, ветеринарні препарати для тварин, корми для котів, корми для собак, аксесуари та інше.

Основним недоліком є відсутність на сайті всіх товари, які є в наявності.

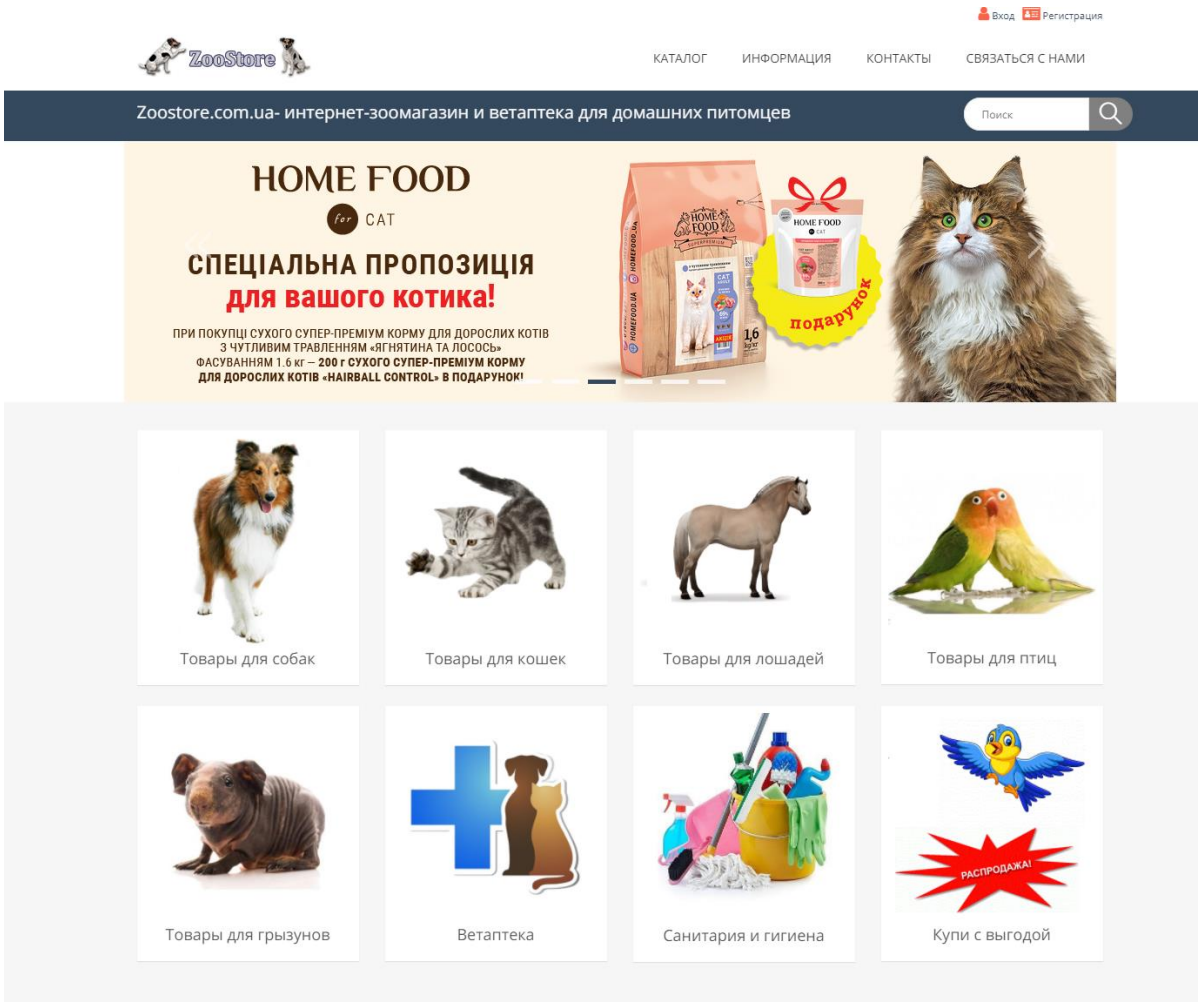


Рисунок 1.6 – Головна сторінка магазина Zoostore

Магазин Zoostore.com.ua має широкий асортимент, товари поділені на багато категорій, кожна з яких має картинку яка візуально відповідає змісту категорії.

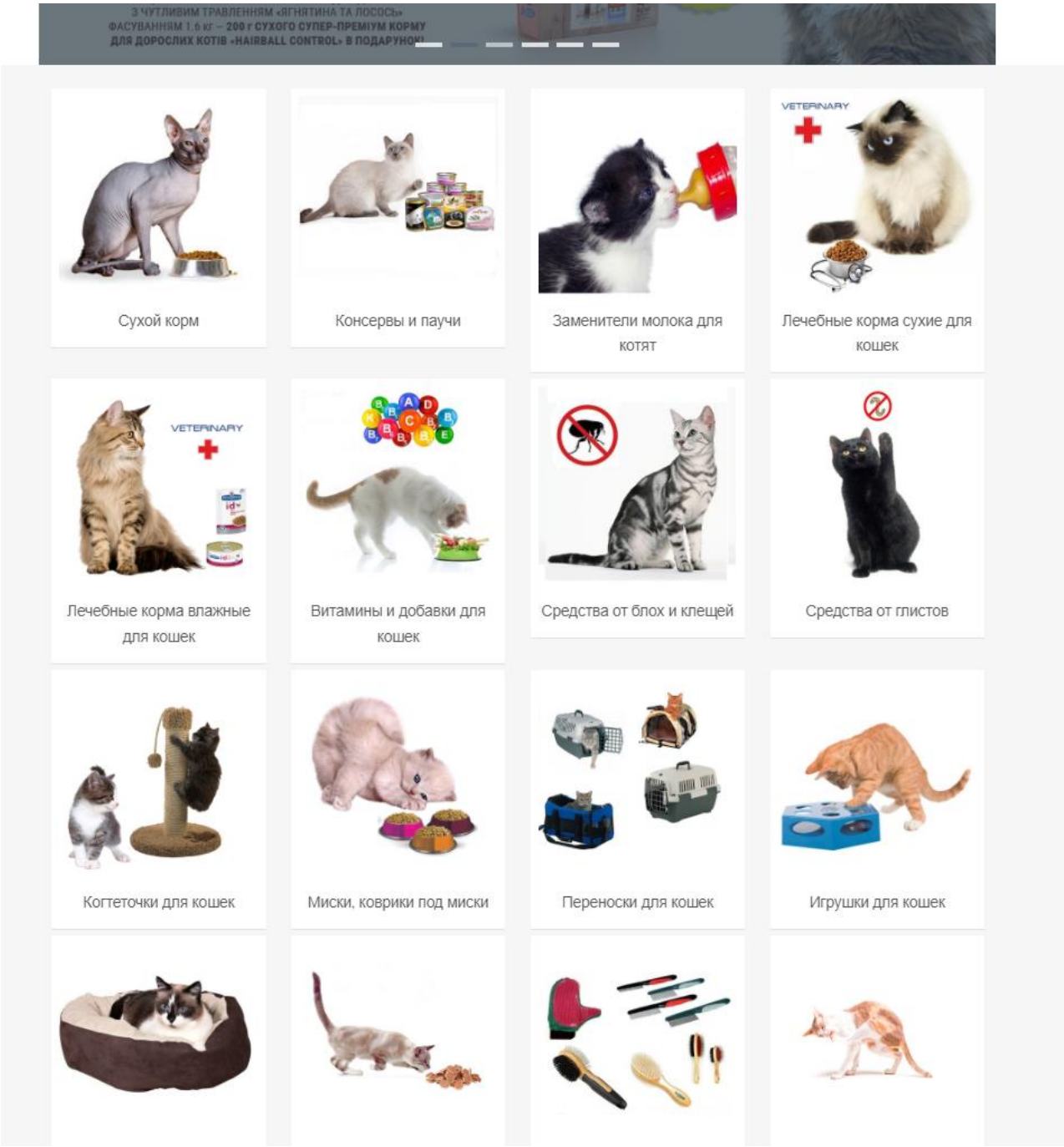


Рисунок 1.7 – Категорії магазина

Таким чином, під час виконання першого розділу бакалаврської роботи була обрана предметна область, в рамках якої розглянуті програмні аналоги і їх особливості. Переваги та недоліки розглянутих аналогів були взяті до уваги при розробці програмного продукту.

2 ВИБІР ТА ОБГРУНТУВАННЯ ЗАСОБІВ РОЗРОБКИ

2.1 Функціональні та нефункціональні вимоги

Перед початком розробки застосунку були визначені функціональні та нефункціональні вимоги до нього.

До функціональних вимог належать:

- наявність БД товарів і клієнтів;
- відображення списку товарів та інформації про обраний;
- можливість реєстрації.
- можливість здійснити замовлення і оплату.

До нефункціональних вимог належать:

- браузер;
- підключення до Інтернету.

2.2 Обґрунтування вибору ОС для сервера web – додатку

Web – додаток має багато переваг у порівнянні зі звичайним додатком, а саме:

- Не потребує установки. Для використання необхідно мати прилад з встановленим браузер і доступом в Інтернет.
- У разі оновлення додатка, його роблять лише на сервері, після чого усі користувачі мають доступ до оновленої версії.
- Web – додаток дуже універсальний та практичний для користувача. Достатньо лише встановити додаток на сервер, з будь-якою ОС і клієнти зможуть користуватися з любого комп’ютера чи мобільного приладу, з різними ОС. А також з будь якого браузера: Mozilla Firefox, Opera, Google Chrome, Internet Explorer або Safari.
- Спрощується доступ до даних, завдяки тому що вони зберігаються в однієї БД.

Основними недоліками є:

- Обмежений доступ до функцій платформи, але у випадку інтернет магазина в них нема потреби.
- Базова продуктивність, але її достатньо для онлайн магазину.

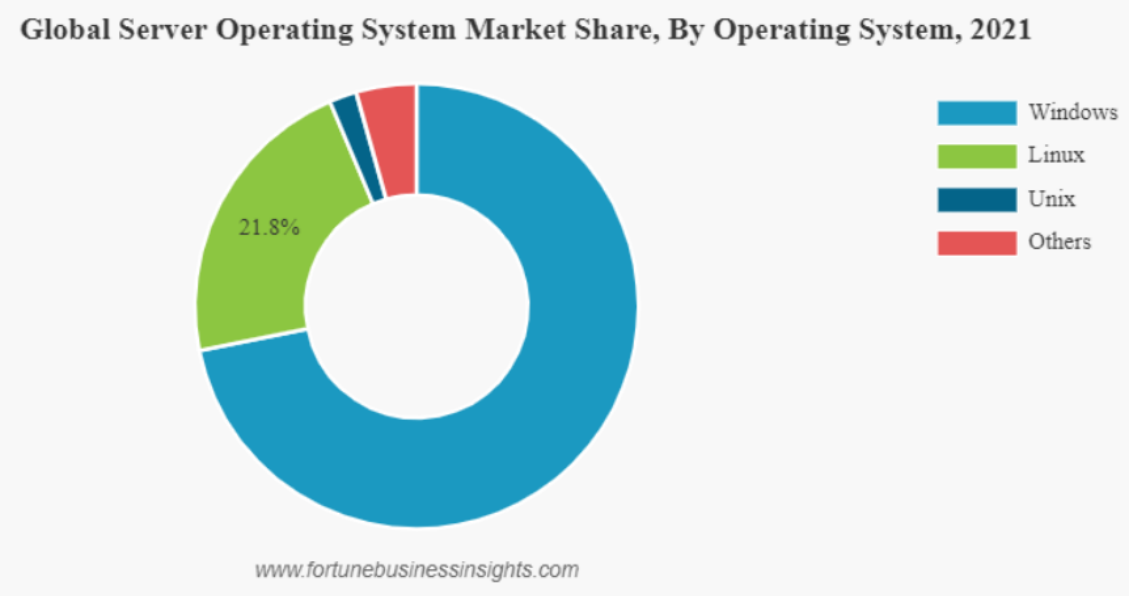


Рисунок 2.1 – Діаграма співвідношення серверних ОС

Windows Server

На питання яку ОС вибрати для сервера найчастіше звучить відповідь: «Windows Server». Перевагами цієї операційної системи вважають її практичність, продуктивність та наявність широких можливостей. За рахунок своєї надійності Windows Server ідеально підходить для терміналів та файлових серверів.

Але ця ОС має і недоліки:

- По-перше, не всі версії 32-розрядну архітектуру.
- По-друге, Windows потребує набагато більших ресурсів, ніж будь-який інший аналог.

- По-третє, ця операційна система є ліцензованим програмним забезпеченням.

Linux Server

Популярність серверних ОС Linux постійно зростає. Серед найбільш популярних ОС, що представляють це сімейство, можна виділити CentOS, Debian і Ubuntu.

CentOS – це безкоштовна операційна система для сервера зі швидким консольним менеджером Yellowdog Updater, Modified та більшістю популярних хостингових панелей. Серед головних переваг CentOS можна відзначити велику кількість форумів і спільнот, що забезпечують всебічну підтримку користувачів цієї ОС. До недоліків варто віднести відсутність програм для роботи з репозиторіями.

Debian - це повністю універсальна оболонка, яка може використовуватися для роботи не тільки сервера, але і звичайного ПК. Debian вважається потужною та добре захищеною ОС. Головним її недоліком є відсутність регулярних оновлень.

Ubuntu відрізняється легким встановленням та налаштуванням, з якими впораються навіть новачки. Автоматичний інтерфейс Ubuntu значно полегшує роботу з цією системою. Вибір Ubuntu буде актуальним за наявності скромного бюджету, а також відносно невеликого навантаження на сервер.

Підсумовуючи, варто сказати, що універсальної ОС для серверів немає. Кожна має свої переваги та недоліки, які зазвичай пізнаються практично. Тому вибір залежить від конкретного завдання. Отже враховуючи особливості системи що створюється, для сервера інтернет-магазину обрана Ubuntu.

2.3 Вибір середовища розробки

Інтегровані середовища розробки (IDE) та редактори коду (CE) – це програми, що використовуються для написання та редагування коду. Технічно можливо писати код у текстовому редакторі, але IDE та CE пропонують багато додаткових функцій, які призначені для спрощення процесу кодування.

IDE зазвичай мають більше функцій, ніж редактори коду, але у деяких редакторах коду є можливість налаштування, подібно до IDE. Редактори коду часто мають такі функції, як виділення синтаксису, автозаповнення та зіставлення дужок. В свою чергу IDE об'єднують кілька інструментів розробника в єдиному графічному інтерфейсі користувача. Як мінімум, ці інструменти зазвичай складаються з редактора коду, компілятора чи інтерпретатора та відладчика.

У кожного програміста є індивідуальні переваги щодо IDE та редакторів коду. Ви можете спробувати кілька програм, перш ніж знайдете те, що вам підходить.

Найважливішими критеріями, які слід враховувати при оцінці ваших варіантів є:

- Вартість: є безкоштовні, і платні програми. В залежності від бюджету спробуйте декілька різних додатків і оберіть підходяще. Якщо ви відкриті для платних варіантів, багато програм пропонують безкоштовний пробний період.
- Час навчання: потрібен час, щоб адаптуватися до нового інтерфейсу та запам'ятати поєднання клавіш. Хоча криві навчання є унікальними для кожного користувача, відомо, що деякі програми вимагають більше навчання для початківців. Якщо ви виберете одну з цих програм, дайте собі час адаптуватися та розробити ефективний робочий процес.
- Функціональність і можливість налаштувань: програми мають різні функції, тому варто знати, які вам необхідні. Деякі популярні функції – це багатомовна підтримка, авто заповнення та інтеграція з Git та GitHub. Якщо

IDE або редактор коду не пропонує певних функцій, перевірте чи можливо їх отримати за допомогою плагінів або розширень.

- Швидкість: потрібно враховувати як репутацію програми щодо швидкості, так і потужність вашого комп'ютера або пристрою для розміщення програми.
- Вимоги до пристрою: характеристики пристрою можуть вплинути на продуктивність програми. Ви зіткнетеся з уповільненням в роботі програми, якщо перевантажте обчислювальну потужність та пам'ять вашого пристрою. Редактор коду зазвичай є більш легким додатком, ніж IDE, але є винятки.
- Сумісність з ОС: деякі IDE та редактори коду не завжди пропонують функціональність для усіх платформ. Під час перегляду варіантів пам'ятайте про сумісність з вашою ОС.
- Підтримка користувачів: програма відомої компанії-розробника програмного забезпечення може мати більш надійну підтримку користувачів, ніж менш популярна програма. Якщо ви новачок, у вас попереду навчання, тому можливо, вам варто пошукати програми з підтримкою клієнтів та активними спільнотами користувачів.

Worldwide, Nov 2020 compared to a year ago:

Rank	Change	IDE	Share	Trend
1	↑	Visual Studio	25.31 %	+3.6 %
2	↑	Eclipse	16.31 %	-1.1 %
3	↓↓	Android Studio	11.39 %	-11.3 %
4	↑	Visual Studio Code	8.74 %	+3.4 %
5	↑	pyCharm	7.73 %	+2.5 %
6	↑	IntelliJ	6.13 %	+1.3 %
7	↓↓↓	NetBeans	5.3 %	-0.2 %
8		Xcode	4.32 %	+0.1 %
9		Sublime Text	3.91 %	+0.1 %
10		Atom	3.86 %	+0.6 %
11		Code::Blocks	2.07 %	+0.4 %
12		Vim	0.93 %	+0.0 %
13	↑	PhpStorm	0.67 %	+0.0 %
14	↓	Xamarin	0.63 %	-0.1 %
15		Komodo	0.49 %	+0.1 %

Рисунок 2.2 – Найпопулярніші IDE для язика JavaScript

Розглянемо деякі з них, для подальшого порівняння особливостей:

1) Visual Studio Code

Платформи: Windows, Linux, macOS

Ціна: безкоштовно

Visual Studio Code [11] — це безкоштовна HTML IDE від Microsoft. Цей редактор підтримує Typescript, C++, Java, Javascript, PHP, Python та інші. VSCode за дуже короткий час створив велику базу шанувальників.

Visual Studio Code доступний не тільки для Windows, а також для macOS і навіть для Linux. Це дозволяє працювати на різних платформах з однаковим комфортом і функціональністю.

VSCode можна легко функціонально розширити за допомогою розширень. Ви можете внести візуальні зміни за допомогою тем, яких також існує велика різноманітність. VS Code працює безпосередньо з Github і пропонує надзвичайно гарне підсвічування синтаксису для різних мов і розширений варіант завершення коду.

VS Code підлягає чотиритижневому циклу оновлення. Журнали змін складаються з місяця в місяць. Продукт масово розробляється, при цьому серйозно враховуються побажання спільноти користувачів. На додаток до оптичних і функціональних параметрів розширення, цей редактор Javascript не нехтує підтримкою користувачів. Доступна повна документація та навчальні відео для початківців.

Плюси:

- Це відкритий код.
- Має багато функцій, таких як провідник рішень, контроль джерел, налагоджувач, область розширень.
- Підтримує термінал всередині вікна.
- Добре для розробки ядра .Net.

Мінуси:

- Треба звикнути до мінімалістичного інтерфейсу.
- Функціональність може бути недостатня для великих проектів.

2) Eclipse

Платформи: Windows / Linux / macOS / Solaris

Ціна: безкоштовно

Це безкоштовне середовище розробки з відкритим кодом, яке добре підходить як для початківців, так і для досвідчених розробників. Eclipse IDE працює з Java, C, C ++, Fortran, Javascript, PHP, Python, Ruby та іншими.

Окрім інструментів налагодження та підтримки систем Git/CVS, Eclipse поставляється з Java та інструментом плагінів. Спочатку Eclipse використовувався тільки для Java, але зараз його функції значно розширилися завдяки плагінам. В наслідок цього ця платформа здобула свою популярність серед розробників. Функціональність Eclipse не така велика, як IntelliJ IDEA, але це середовище розробки поширюється з відкритим вихідним кодом.

Плюси:

- Можливість програмувати багатьма мовами.
- Значна гнучкість середовища завдяки модульності.
- Можливість інтеграції JUnit.
- Віддалене налагодження (при використанні JVM).
- Доповнення коду пропонується з мовного сервера.
- Ви можете перейти до оголошення змінних, класів і методів.
- Проекти, створені за допомогою MS test і xUnit, можуть запускатися безпосередньо в IDE.

Мінуси:

- Новачкам може бути важко зрозуміти різноманітність можливостей.
- Плагін керується спільнотою, тому немає гарантії, що він завжди працюватиме добре.

3) Visual Studio

Платформи: Windows, Linux, macOS

Ціна: від $45/місяць

Microsoft Visual Studio — це IDE від Microsoft. Він використовується для розробки комп’ютерних програм для Microsoft Windows і веб-сайтів, веб-

додатків і веб-сервісів. Visual Studio використовує платформи розробки програмного забезпечення Microsoft, такі як Windows API, Windows Forms, Windows Presentation Foundation, Windows Store і Microsoft Silverlight. Visual Studio містить редактор коду, який підтримує IntelliSense, а також рефакторинг коду. Інтегрований налагоджувач працює як налагоджувач на рівні джерела та налагоджувач на рівні машини. Інші вбудовані інструменти включають конструктор форм для створення додатків із графічним інтерфейсом користувача, веб-дизайнер, конструктор класів і конструктор схем бази даних. Він приймає плагіни, які розширюють функціональність майже на кожному рівні, включаючи додавання підтримки систем контролю версій (наприклад, Subversion) і додавання нових редакторів інструментарію та візуальних дизайнерів для мов, що стосуються домену, або наборів інструментів для інших.

Visual Studio підтримує різні мови програмування і дозволяє редактору коду та налагоджувачу підтримувати практично будь-яку мову програмування, якщо існує спеціальна мовна служба. Вбудовані мови включають XML / XSLT, HTML / XHTML, JavaScript і CSS, C, C ++ і C ++ / CLI, VB.NET, C # і F #. Підтримка інших мов, таких як M, Python, Ruby та інших, доступна через мовні служби, встановлені окремо.

Плюси:

- Багато інструментів у середовищі дуже добре працюють у JS та C#.
- Існує безкоштовна версія під назвою Community Edition.
- У спільноті є все, що потрібно незалежному розробнику.
- Найкраще програмне забезпечення для розробки на будь-якій платформі, не кажучи вже про .Net і C #.
- Є хмарне сховище.

Мінуси:

- Ресурсомісткість.

- Якщо ви перейдете на платну версію, налаштування та корпоративний сервер можуть вийти з ладу.
- Програму важко освоїти самостійно через велику кількість функцій і меню.

4) WebStorm

Платформи: Windows, Linux, macOS

Ціна: $0-129/рік

WebStorm виділяється тим, що це повнофункціональна IDE JavaScript. Розробником цієї IDE є JetBrains.

Цей редактор JS чудово підходить для різних платформ, таких як React, Angular, Vue.js тощо. Його можна використовувати для налагодження скриптів вузлів і виконання тестів на вбудованому сервері. Ви також можете запускати та налагоджувати скрипти npm (за допомогою деревоподібного інтерфейсу). І для всього цього вам не потрібні ніякі плагіни.

Однак доступні плагіни для деяких більш специфічних функцій, не вбудованих у саму IDE. Але в більшості випадків все, що вам може знадобитися, вже є в IDE. Найкраще в цьому полягає в тому, що ви можете відкрити для себе нові функції, про які раніше не знали, і оцінити, наскільки вони чудові.

За замовчуванням WebStorm налаштовано на автоматичне збереження файлів під час роботи з ними. Коли ви переходите на інший редактор Javascript без такої функціональності, ви починаєте відчувати, наскільки йому не вистачає після цієї IDE. Однак це не є унікальною особливістю WebStorm. Просто реалізація тут трохи приємніша.

Деякі люди не завжди довіряють ctrl-z для скасування, але WebStorm має вбудовану систему контролю версій, яка фіксує кожен раз, коли файл зберігається. Це чисто внутрішній, повністю відокремлений від ваших комітів Git. Отже, якщо ви не робили коміту в Git протягом якогось часу, і вам потрібно

повернутися назад і побачити попередній стан після останнього коміту, ви можете зробити це без проблем.

Незважаючи на деякі недоліки, проект постійно розвивається та вдосконалюється.

Плюси:

- Зручне автозаповнення для HTML, CSS і JavaScript.
- Перевірка помилок і просте налагодження коду забезпечується завдяки інтеграції з низкою систем відстеження помилок.
- Вбудована інтеграція з системами контролю джерел, такими як GitHub, Git, а також Subversion, Perforce і Mercurial.
- Гнучкість налаштувань.
- Досить велика кількість плагінів.
- Багато функцій.
- Немає необхідності шукати пакети та налаштовувати їх.
- Чудовий відступ, поради щодо спрощення коду та базова перевірка коду на наявність помилок.
- Чудовий інструмент злиття.
- Розумне завершення коду, перевірка помилок на льоту, швидка навігація по коду та рефакторинг для JavaScript.

Мінуси:

- Завдяки широкій функціональності WebStorm є важким і ресурсомістким;
- Іноді при роботі над дуже великими проектами це з’їдає пам’ять;
- Притаманна всім IDE, вона повільна та ресурсомістка.
- Відносно складні налаштування.
- Платна IDE, що розповсюджується за підпискою.

- Повільна робота з великою кількістю проектів.

Для написання проекту обрана IDE Visual Studio Code.

3 ВИКОРИСТОВАНІ ТЕХНОЛОГІЇ

Веб-додаток — це клієнт-серверна програма, за допомогою якої клієнт взаємодіє з веб-сервером в браузері. Логіка веб-додатка розподілена між клієнтом і сервером, сервер відповідає за зберігання даних, а обмін інформацією відбувається по мережі.

Однією з переваг такого підходу є факт того, що клієнти не залежать від операційної системи на їх пристрою. Замість того щоб писати програму під кожну ОС, вона створюється один раз на обраній платформі.

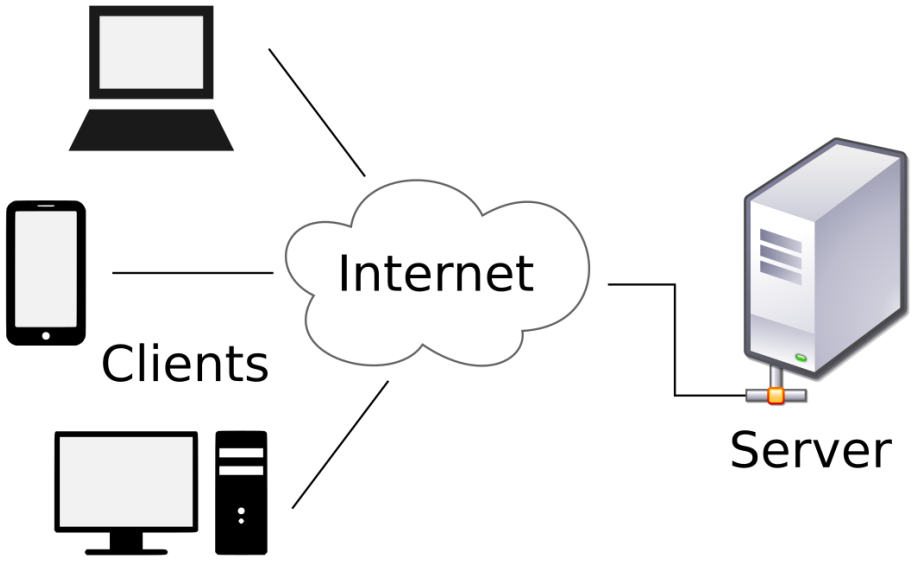


Рисунок 3.1 – Архітектура клієнт-серверного додатку

3.1 Реалізація GUI

Для реалізації клієнтської частини додатку було вирішено використовувати фреймворк React.

React [4] — це безкоштовна бібліотека JavaScript із відкритим вихідним кодом для створення інтерфейсів на основі компонентів. Його підтримують Meta і спільнота окремих розробників і компаній. React можна використовувати як базу для розробки односторінкових (SPA), мобільних або серверних додатків

із такими фреймворками, як Next.js. Однак React займається керуванням станом і відтворенням цього стану в DOM.

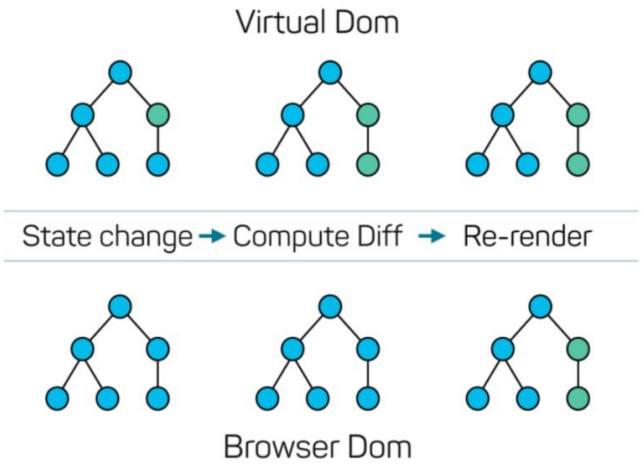


Рисунок 3.2 – Віртуальне дерево компонентів

Код React складається з сутностей, які називаються компонентами. Ці компоненти можна використовувати багаторазово і вони повинні бути сформовані в папці SRC, дотримуючись PascalCase при іменуванні. Компоненти можна відтворити до певного елемента в DOM за допомогою бібліотеки React DOM. Під час візуалізації компонента можна передавати значення між компонентами за допомогою “props”.

Іншою важливою особливістю є використання віртуальної об'єктної моделі документа або віртуальної DOM. React створює кеш структури даних у пам’яті, обчислює відмінності, а потім ефективно оновлює DOM, що відображається у браузері. Цей процес називається узгодження. Завдяки цьому програмісту залишається писати код так, ніби вся сторінка відображається заново під час кожної зміни, тоді як React відображає лише компоненти, в яких фактично відбулися зміни. Таке вибіркове відтворення забезпечує значне підвищення продуктивності. Це заощаджує зусилля, пов’язані з перерахунком стилю CSS, макетом сторінки та відображенням для всієї сторінки.

Односторінковий додаток (SPA) — це веб-додаток або веб-сайт, який взаємодіє з користувачем шляхом динамічного переписування поточної веб-сторінки з новими даними з веб-сервера, замість стандартного методу веб-браузера, який завантажує цілі нові сторінки. Метою є швидші переходи, завдяки чому веб-сайт буде більше схожий на встановлену програму.

У SPA оновлення сторінки ніколи не відбувається; натомість весь необхідний код HTML, JavaScript та CSS або витягується браузером із завантаженням однієї сторінки, або відповідні ресурси динамічно завантажуються та додаються на сторінку за потреби, як правило, у відповідь на дії користувача.

3.2 Реалізація API

API спрощує процес програмування при створенні програм, абстрагуючи базову реалізацію та надаючи лише об'єкти або дії, необхідні розробнику. Наприклад API для вводу/виводу файлів може дати розробнику функцію, яка копіює файл з одного місця в інше, не вимагаючи від розробника розуміння операцій файлової системи, що відбуваються.

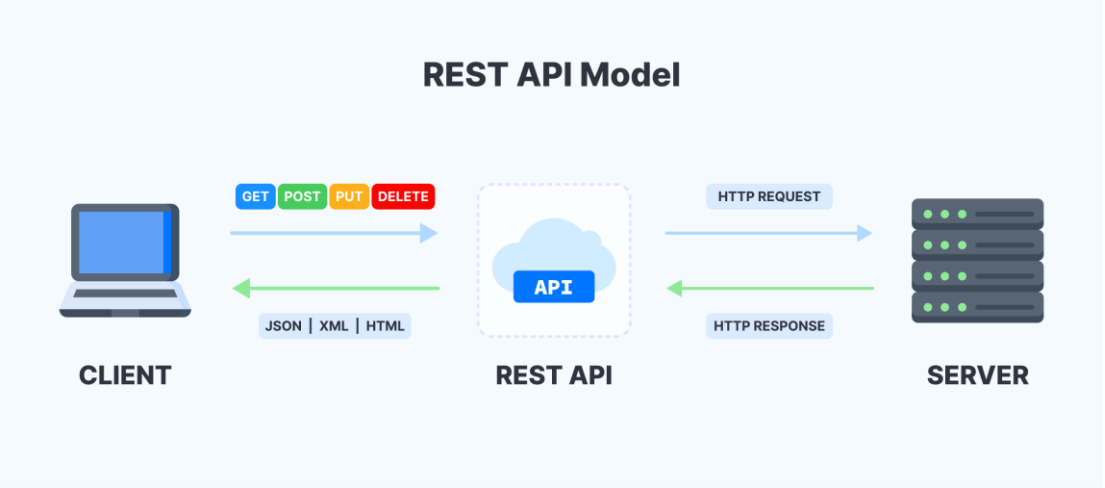


Рисунок 3.2 – Модель REST API

REST API (RESTful API) — це інтерфейс програмування прикладних програм (API або веб-API), який відповідає обмеженням архітектурного стилю REST. REST (REpresentational State Transfer) розшифровується як репрезентаційна передача стану, створений вченим Роєм Філдінгом.

Для того, щоб API вважався RESTful, він повинен відповідати цим критеріям:

1. Клієнт-серверна архітектура керуються за допомогою HTTP.
2. Зв’язок клієнта з сервером не зберігає інформація про клієнта між запитами на отримання, кожен запит є окремим.
3. Можливість кешування даних, для спрощення взаємодії.
4. Єдиний інтерфейс між компонентами, щоб інформація передавалася у стандартній формі.
5. Багатошарова система, яка організовує кожен тип серверів (відповідальних за безпеку, балансування навантаження тощо), передбачала отримання запитуваної інформації в ієрархії, невидимі для клієнта.

3.3 Реалізація сервера

NodeJS [2]

Node.js – це кросплатформене середовище виконання JavaScript з відкритим вихідним кодом, яке виконує код JavaScript за межами браузера. Node.js дозволяє розробникам використовувати JavaScript для написання інструментів командного рядка та для сценаріїв на стороні сервера — запуск сценаріїв на стороні сервера для створення динамічного вмісту веб-сторінки, перш ніж сторінка буде відправлена у веб-браузер користувача.

Цикл подій — це те, що дозволяє Node.js виконувати неблокуючі операції введення-виводу, незважаючи на те, що JavaScript є однопотоковим, шляхом розвантаження операцій до ядра системи, коли це можливо.

На наступній схемі показано спрощений огляд порядку операцій циклу подій:

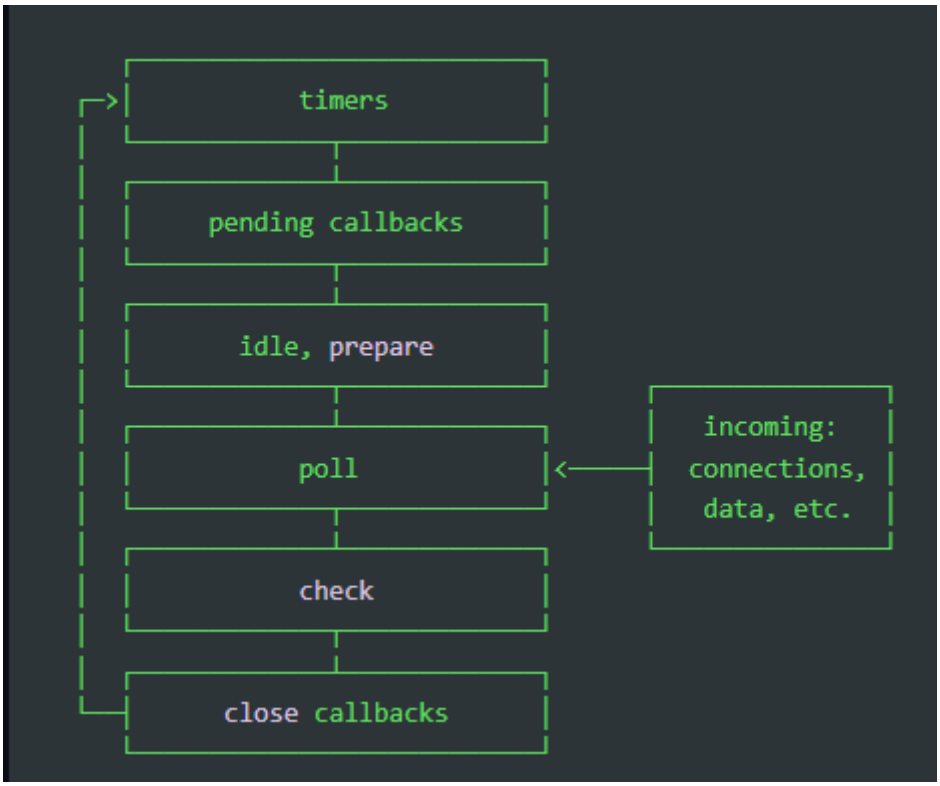


Рисунок 3.3 – Етапи циклу подій

Огляд фаз

- таймери: виконуються зворотні виклики, setTimeout() і setInterval().
- очікувані зворотні виклики: виконує виклики введення-виводу, відкладені до наступної ітерації циклу.
- простою, підготовки: використовується тільки внутрішньо NodeJS.
- опитування: отримати нові події введення-виведення; виконувати зворотні виклики, пов’язані з введенням-виводом (майже всі, за винятком закритих зворотних викликів, запланованих таймерами та setImmediate()); вузол блокуватиме тут, коли це необхідно.
- перевірка: викликаються зворотні виклики setImmediate().
- Закриті зворотні виклики: деякі закриті зворотні виклики, напр. socket.on('закрити', ...).

Для більшою зручності розробки сервера також використовується ExpressJS.

База даних

База даних - це будь-яка колекція даних або інформації, яка спеціально організована для швидкого пошуку та пошуку за допомогою комп’ютера. Бази даних структуровані так, щоб полегшити зберігання, пошук, модифікацію та видалення даних у поєднанні з різними операціями обробки даних. Система управління базами даних (СУБД) витягує інформацію з бази даних у відповідь на запити.

В якості базі даних обрана PostgreSQL

PostgreSQL [3] — це потужна об'єктно-реляційна система баз даних з відкритим вихідним кодом з більш ніж 30-річною активною розробкою, завдяки якій вона заслужила міцну репутацію за надійність, стійкість функцій і продуктивність.

TypeORM [5]

ORM (Object-Relational Mapping, об’єктно-реляційне відображення) — технологія програмування, суть якої полягає в створенні «віртуальної об'єктної бази даних».

Завдяки цій технології розробники можуть використовувати мову програмування, з якою зручно працювати з базою даних, замість опису операторів SQL. Це може значно ускорити розробку програми, особливо на початковому етапі. ORM також дозволяє переключати додаток між різними реляційними базами даних. Наприклад, додаток може бути переключено з MySQL на PostgreSQL з мінімальними змінами коду.

Мова програмування TypeScript [7].

TypeScript — це строго типізована мова програмування, яка базується на JavaScript, що дає вам кращі інструменти в будь-якому масштабі.

4 ПРОЕКТУВАННЯ WEB-ДОДАТКУ

Перед початком розробки бази даних необхідно визначити основні цілі, завдання і правила для вирішуваної проблеми, після чого приступати до проектування. Тому сформулюємо короткий опис завдання. Найменування завдання – розробка інтернет-магазину.

Мета магазину – надання всієї інформації про товари, а також можливість їх покупки.

Функції:

- Пошук товарів за критеріями.
- Повна інформація про конкретний товар.
- Виявлення найпопулярніших груп товарів.
- Можливість замовлення та оплати товарів.
- Облік продажів.

Основні бізнес-правила:

- Товар поділяється на 3 категорії: собаки, кішки та папуги.
- Кожна категорія має кілька видів товарів.
- Клієнт може придбати за один раз кілька товарів.
- Для покупки необхідна реєстрація/логін

Визначимо мінімальний набір сутностей, необхідний проектування інформаційної системи обліку структури магазину. Для визначення первинного набору сутностей буде проведено аналіз технічного завдання та предметної галузі.

- Головним завданням – продаж товарів. Тому можна виділити таку сутність як товар. Ця сутність буде включати опис конкретного товару.
- Купуватимуть товари Клієнти магазину, відомості про які зберігатимуться у відповідній сутності.
- Товар поділяється на категорії, які є окремою сутністю.
- Товари підлягають продажу, тому можна назвати сутність Продаж.

- При цьому за один раз клієнт може купити та оплатити кілька видів товарів, тому необхідно виділити таку сутність як Замовлення, яка об'єднуватиме різні товари від одного покупця та формуватиме загальний рахунок.

Обґрунтування необхідного набору атрибутів для кожної сутності та виділення атрибутів, що ідентифікують

Атрибут - характеристика сутності. Атрибутом сутності є будь-яка деталь, яка служить для уточнення, ідентифікації, класифікації, числової характеристики чи висловлювання стану сутності. Його найменування має бути унікальним для конкретного типу сутності, але може бути однаковим для різного типу сутності. Для кожної виділеної сутності необхідно визначити атрибути:

Таблиця 4.1 – Атрибути сутностей

Сутність	Атрибути
Товар	Назва, Фото, Опис, Ціна, Тип
Клієнт	Пошта, Пароль, Ім’я, Телефон, Кошик, Історія замовлень
Категорії	Назва, Види
Продаж	Назва товару, Ціна
Замовлення	Сума, Список продаж
Вид	Назва, Категорії

4.1 Створення діаграми відношення сутностей

Діаграма відносин сутностей (ERD) - це візуальне уявлення бази даних, яке показує, як пов'язані елементи всередині. Діаграма складається з двох типів об'єктів — сутностей і відносин. Сутність у цьому контексті – це компонент даних із набору даних, що відображається у вигляді фігури на полотні. Відносини між сутностями представлені у вигляді зв'язків, які мають

спеціальні закінчення, які описують, як два елементи бази даних взаємодіють один з одним.

На основі функцій які необхідно реалізувати була створена наступна діаграма бази даних:

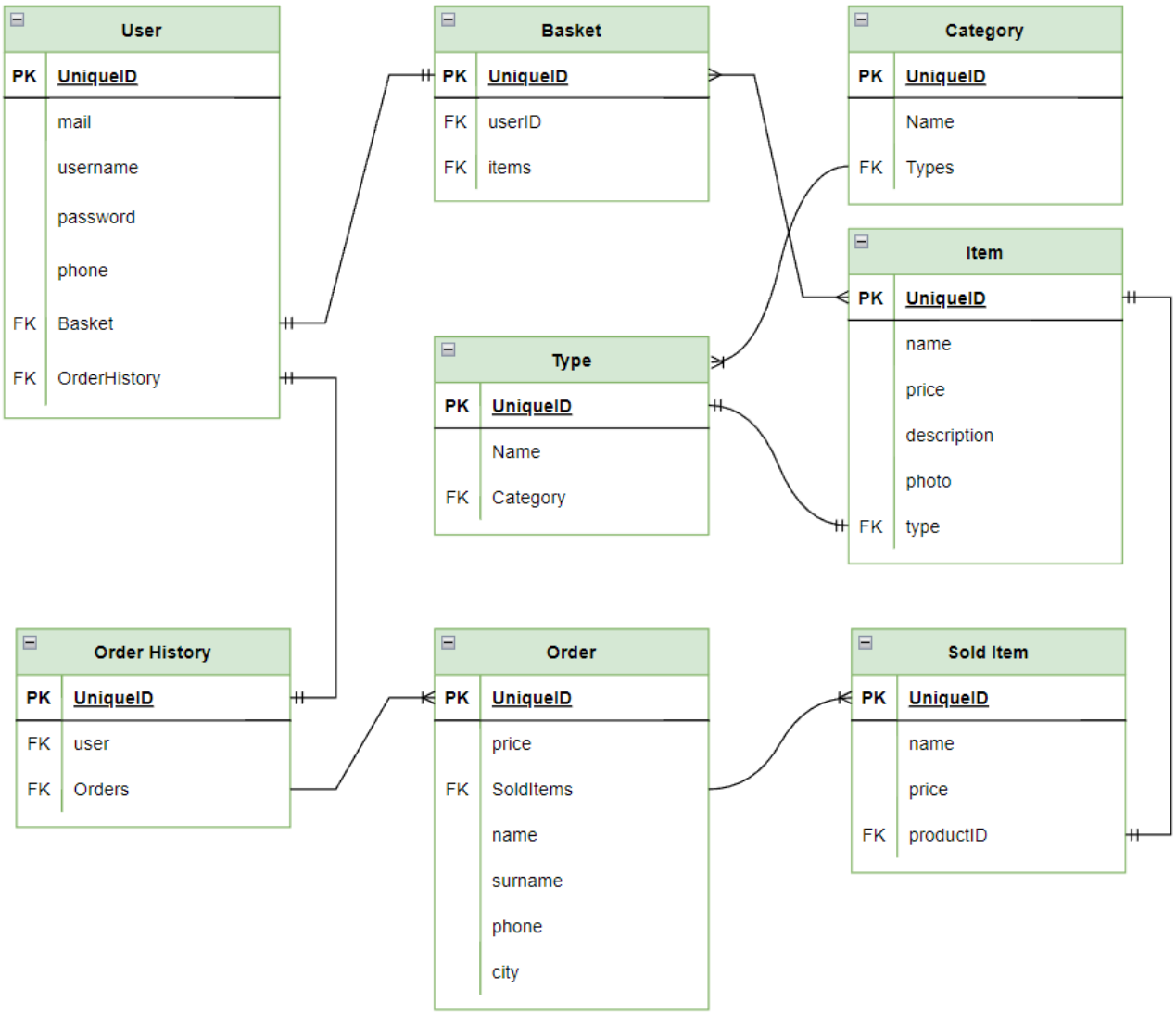


Рисунок 4.1 – Діаграма бази даних додатку Lapki Market

4.2 Створення діаграми варіантів використання системи

Діаграма варіантів використання (англ. use-case diagram) діаграма, що описує, який функціонал програмної системи, що розробляється, доступний кожній групі користувачів.

Тобто користувачі в системі поділяються на деякі групи, в залежності від якої вони отримують доступ до певних функцій системи.

В системі інтернет магазина можна виділити такі групи користувачів:

- Клієнти, будь які нові користувачі які пройшли авторизацію.
- Адміністратори, група користувачів яким доступно редагування категорій, типів і товарів в магазині, а також додавання нових.

Кожна з груп має свій набір функцій взаємодії з додатком:

Клієнти можуть:

- Додати товар к кошик
- Переглянути додані товари
- Створити і оплатити замовлення
- Переглянути попередні замовлення

Адміністратори можуть:

- Редагувати товари
- Додавати товари
- Видаляти товари

Також будь які користувачі можуть:

- Переглянути товари
- Авторизуватись

В результаті виділення необхідних груп користувачів і способів взаємодії для кожної була побудована наступна діаграма варіантів використання.

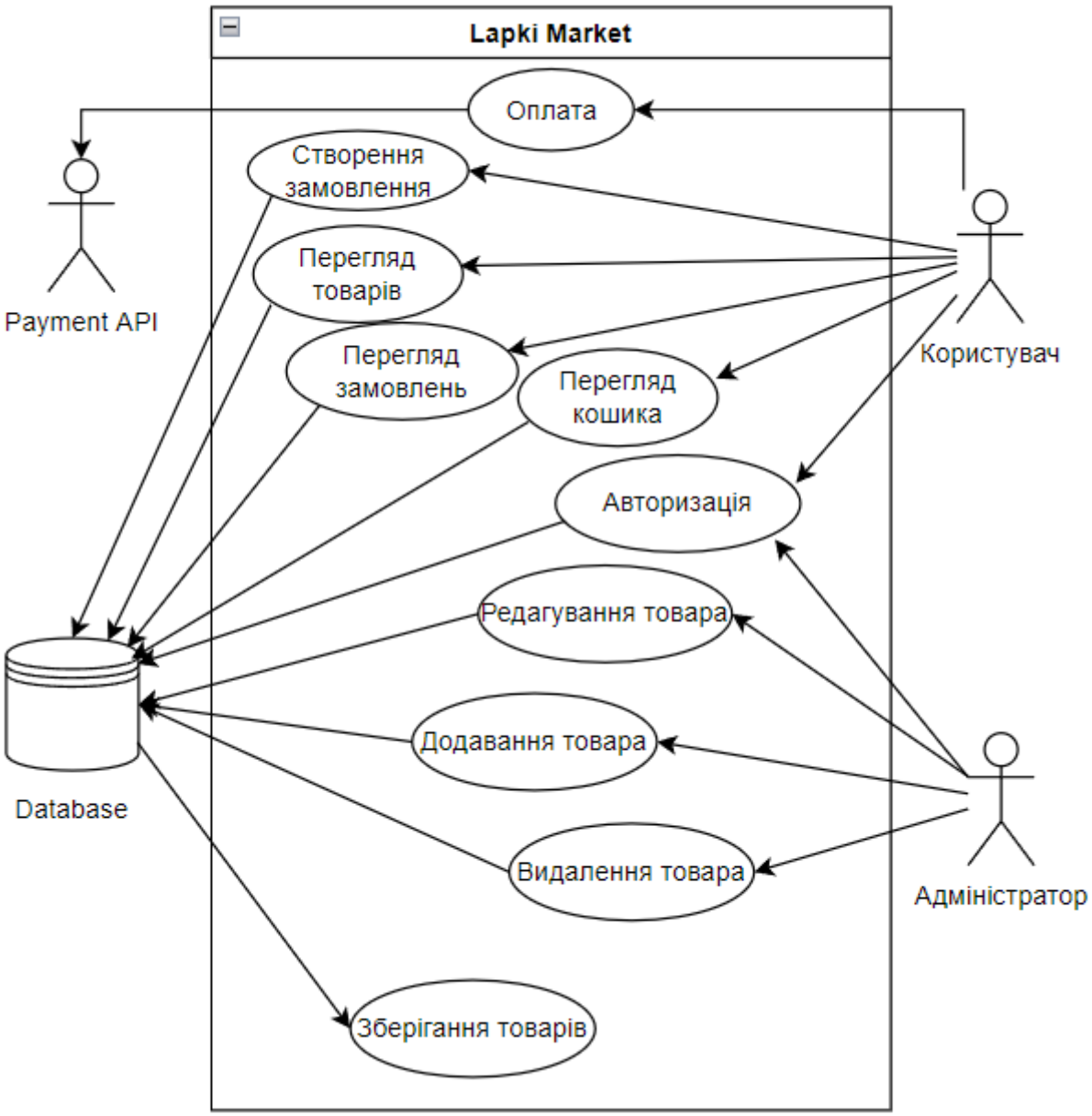


Рисунок 4.2 – Діаграма використання додатку Lapki Market

4.3 Створення діаграми діяльності

Діаграма активностей (видів діяльності) один із доступних видів діаграм. Вона, як і діаграма станів, відображує динамічні аспекти поведінки системи. По суті, ця діаграма є блок-схемою, яка показує, як потік управління переходить від однієї діяльності до іншої, в результаті реакції на дії користувача системи.

Для системи була створена діаграма що відображає діяльність користувача від авторизації до оформлення створення замовлення:

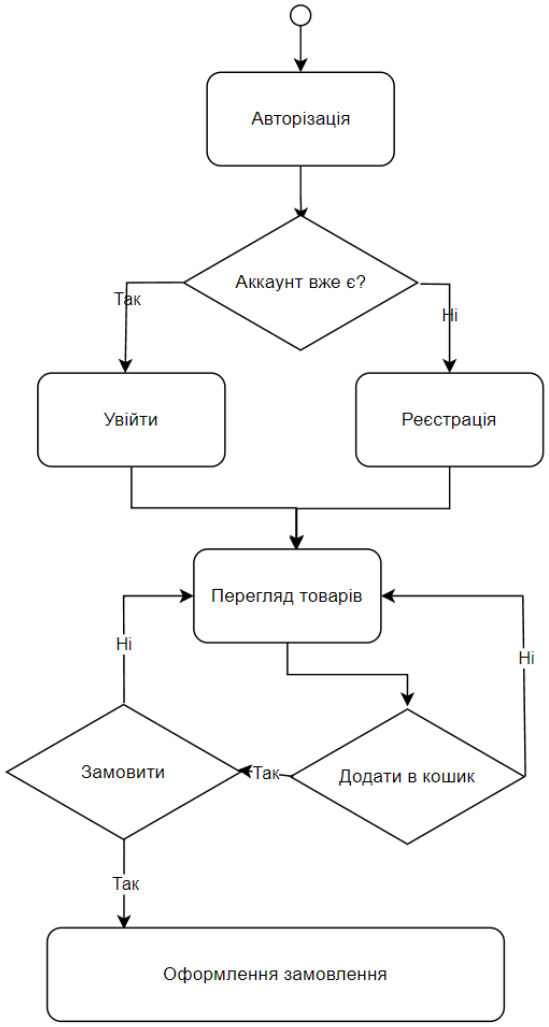


Рисунок 4.3 – Діаграма діяльності користувача

4.4 Створення діаграми послідовності

Діаграма послідовності використовується для демонстрації об’єктів які приймають участь у процесах системи, та послідовність взаємодій між ними.

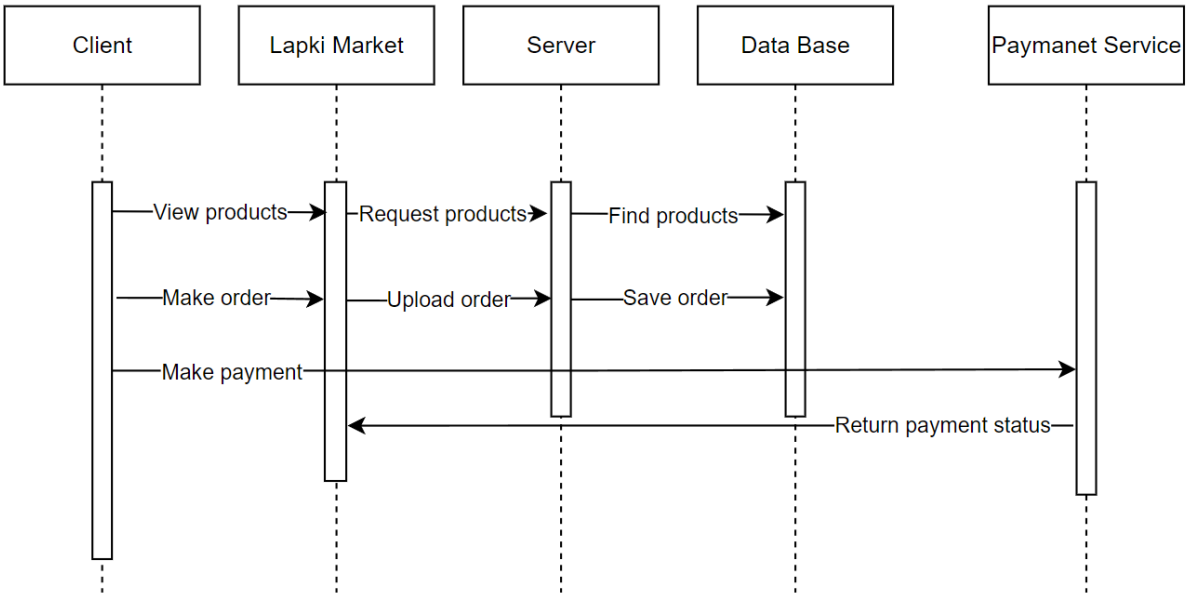


Рисунок 4.4 – Діаграма послідовності

4.5 Створення макетів інтерфейсу

Макет інтерфейсу – це найважливіший крок створення інтерфейсу. Саме на цьому етапі додаток матеріалізується, а ідеї набувають контуру готового продукту.

Макети поділяються на два типи, низької та високої точності:

Макет низької точності – це грубі макети. Що відображують порядок, структуру та розташування елементів на екрані. Таким аспектам дизайну, як сітка, колір, шрифти і т.д. при створенні грубих макетів інтерфейсу увага не

приділяється. Грубі макети інтерфейсу замовник отримує у вигляді динамічного прототипу, який можна використовувати для тестування ергономічності або початку розробки програми.

Макет високої точності – результат розробки дизайну інтерфейсу з кольором, шрифтами, іконками, фоном, сіткою. Замовник отримає такий макет у вигляді графічних файлів у форматі Photoshop, Figma [9] та інші.

В результаті проектування магазину Lapki Market були створені три основні сторінки: авторизації(рис. 4.5), головної сторінки з товарами(рис. 4.6), сторінки кошика(рис 4.7), а також сторінки особистого кабінету користувача(рис 4.8).

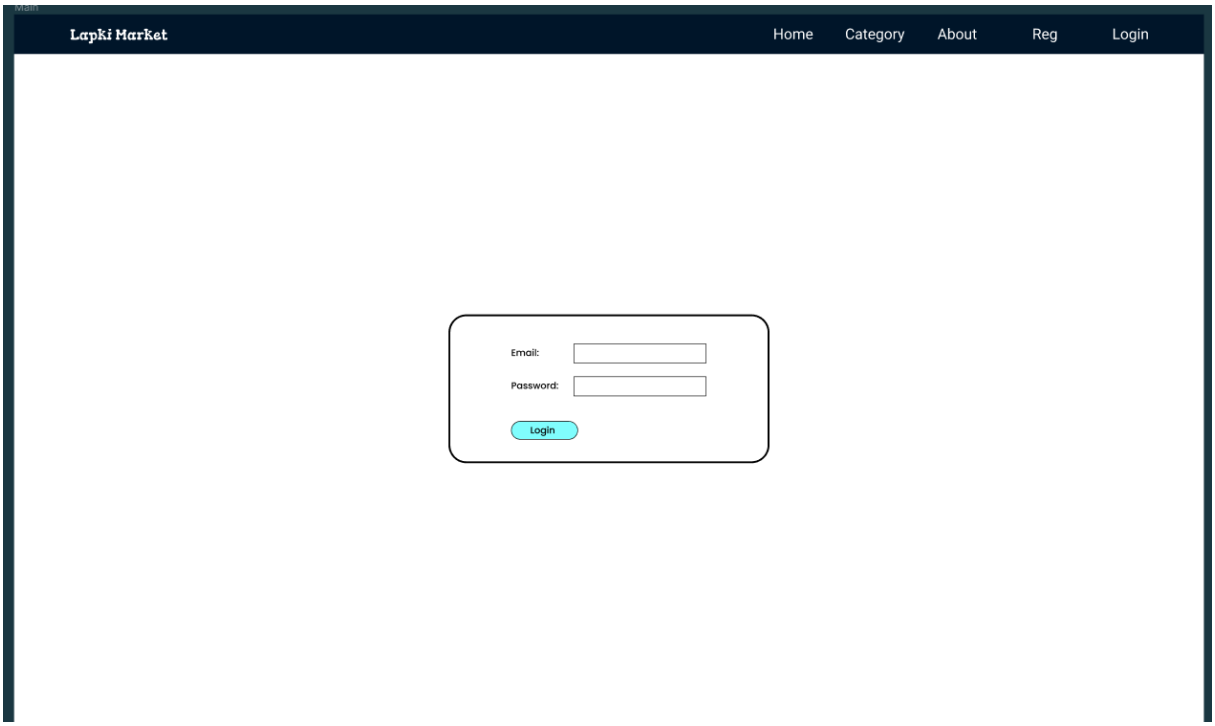


Рисунок 4.5 – Макет сторінки авторизації

На рисунку 4.5 представлена сторінка логіна, тобто для користувачів які вже мають створений акаунт. Сторінка реєстрації виглядає аналогічно, відрізняються лише поля для введення інформації.

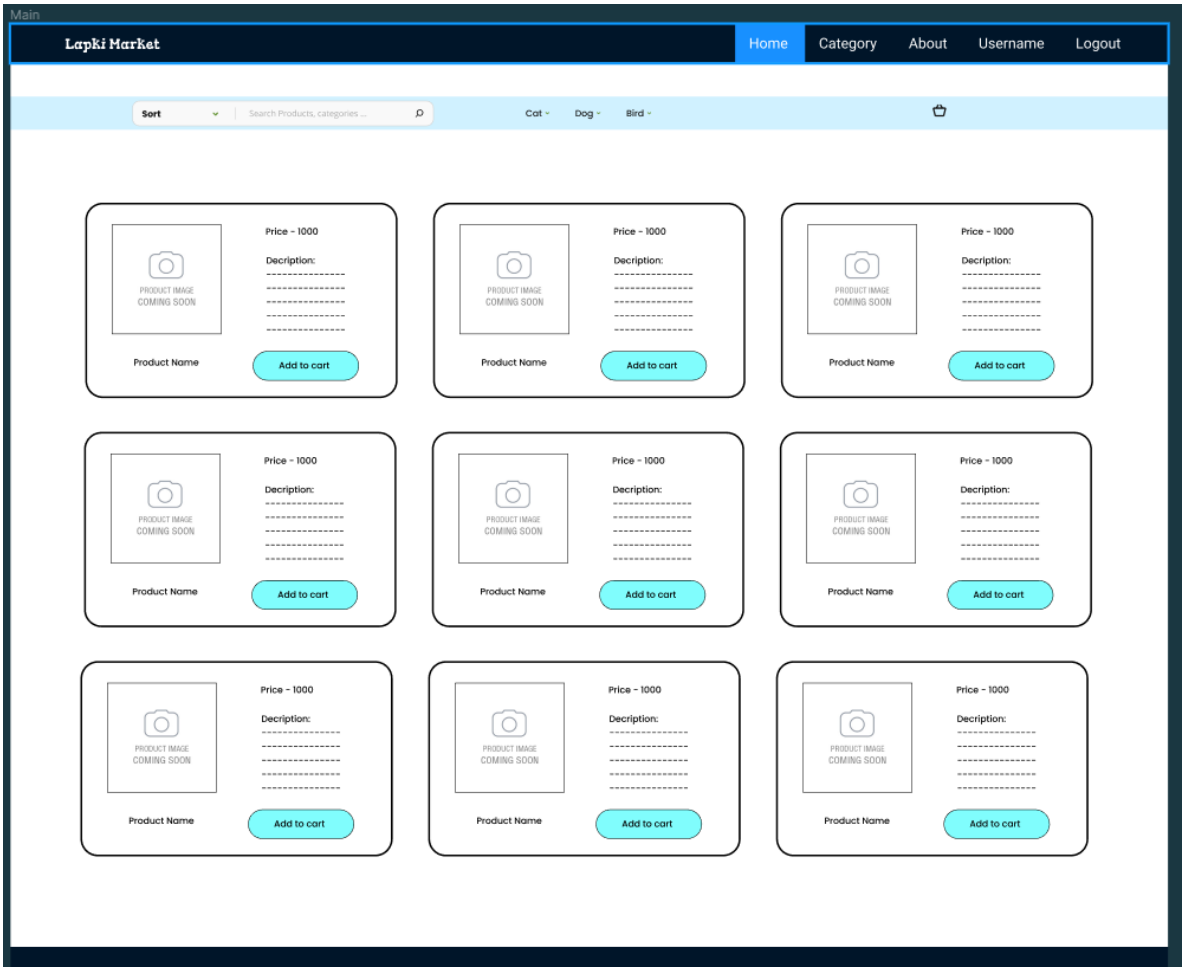


Рисунок 4.6 – Макет головної сторінки магазина

Головна сторінка магазина доступна всім користувачам, навіть не авторизованим. На неї представлені всі товари які є в магазині, а також функції пошуку та сортування для швидкого і зручного пошуку необхідного товару.

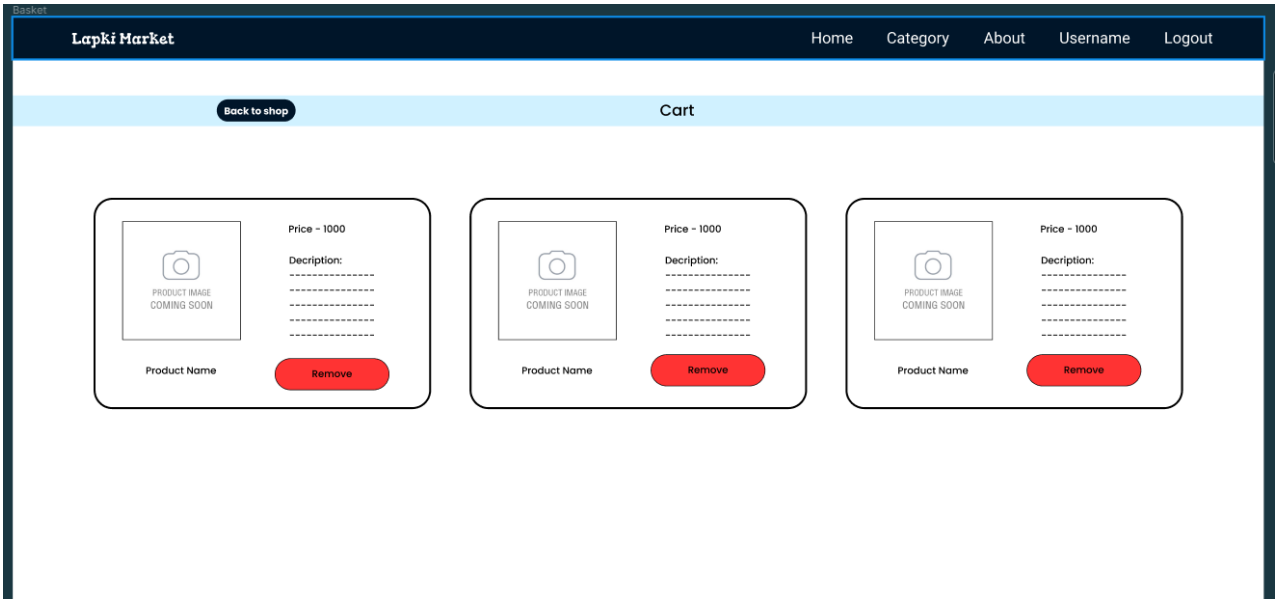


Рисунок 4.7 – Макет сторінки кошика

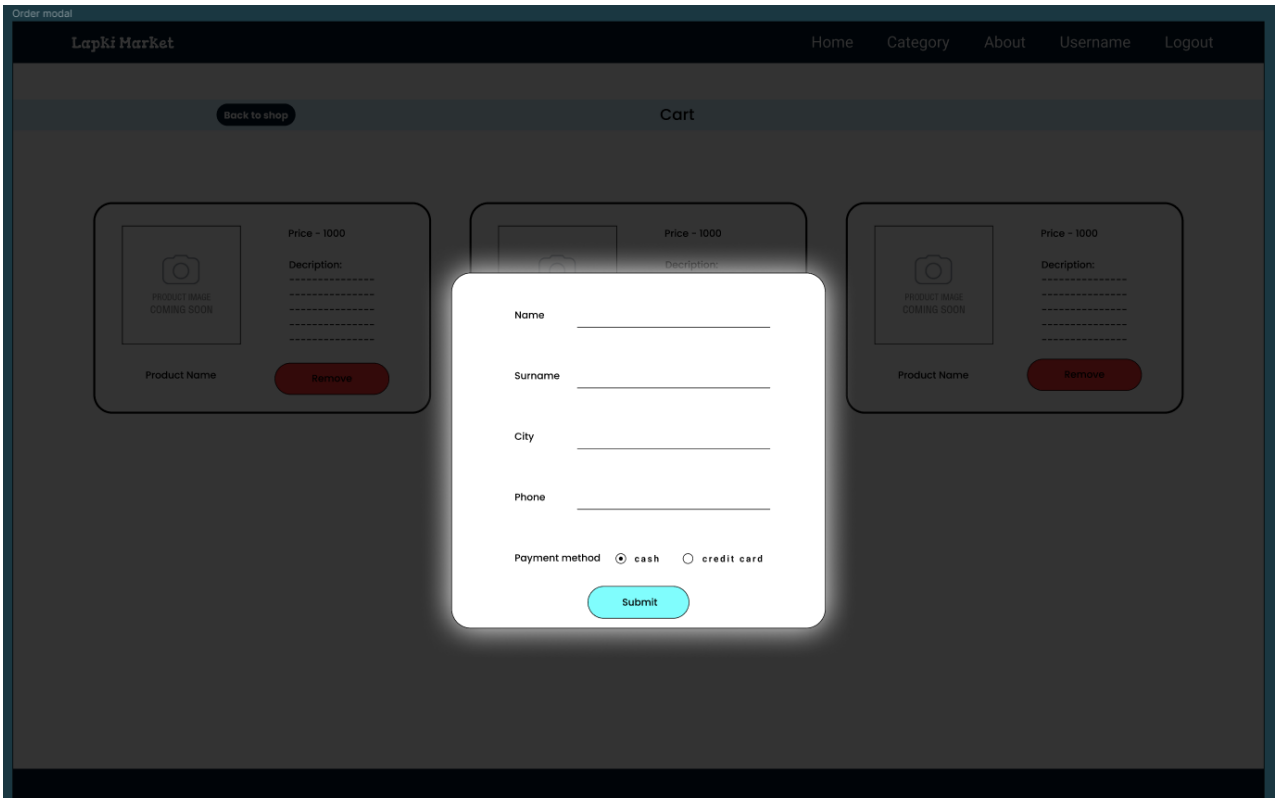


Рисунок 4.8 – Макет форми замовлення

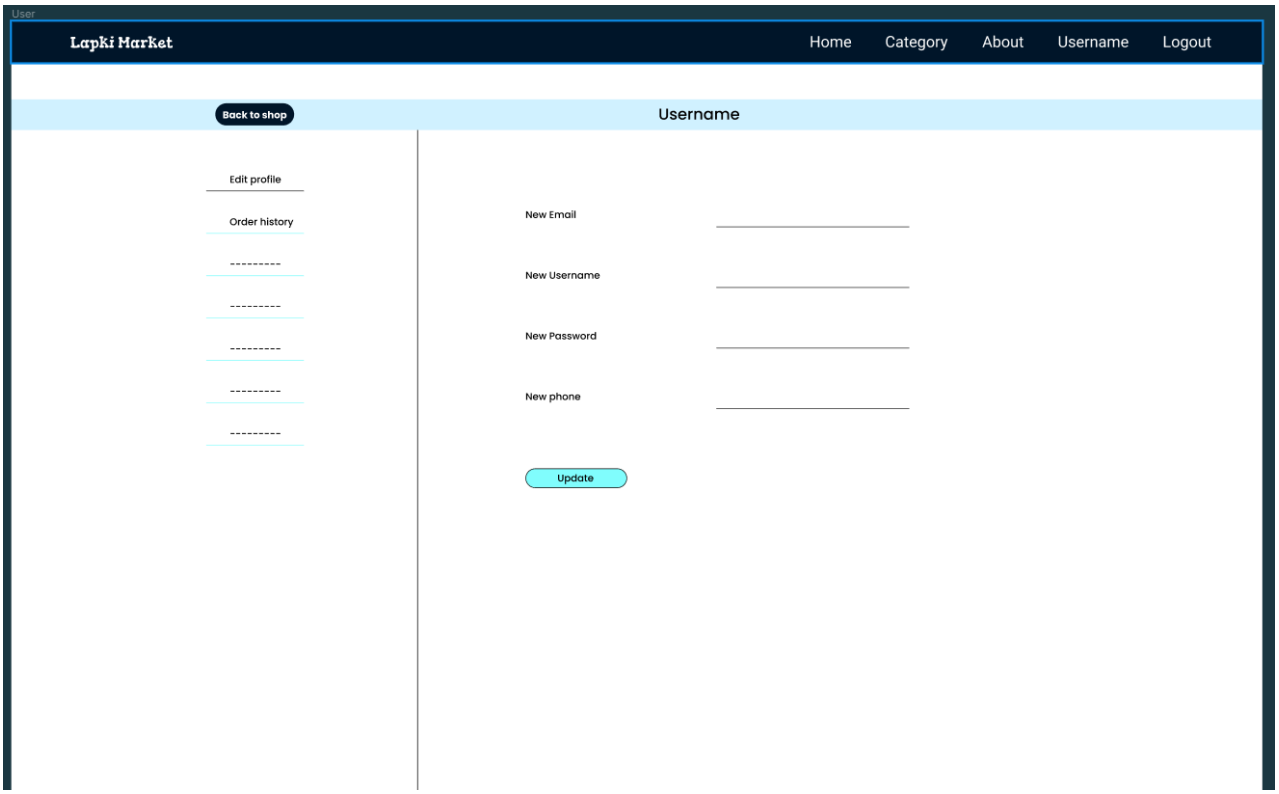


Рисунок 4.9 – Макет сторінки особистого кабінету

Сторінка особистого кабінету призначена для перегляду деякої особистої інформації, та її редагування.

5 РЕАЛІЗАЦІЯ WEB ДОДАТКУ «LAPKI MARKET»

Почнемо розробку з уставки усього необхідного ПО яке буде використовуватись для створення магазину:

IDE VSCode, середовище розробки. А також деякі корисні розширення до нього.

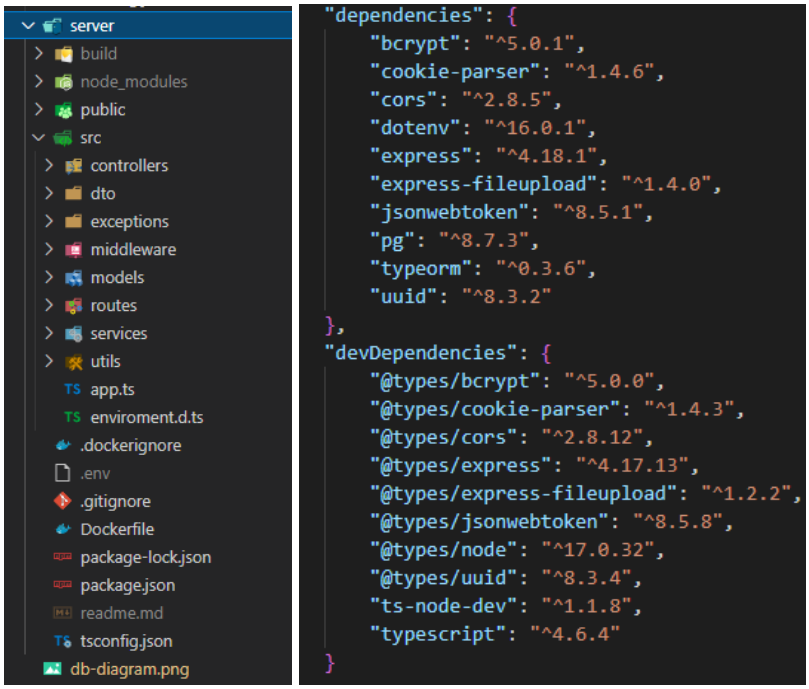
Платформа NodeJS і ExpressJS [6] для створення сервера.

pgAdmin для керування базою даних PostgreSQL

Postman [10] для створення запитів на сервер

5.1 Розробка серверної частини

Спочатку створюємо файлову структуру сервера, а також встановлюємо всі необхідні пакети.



а) б)

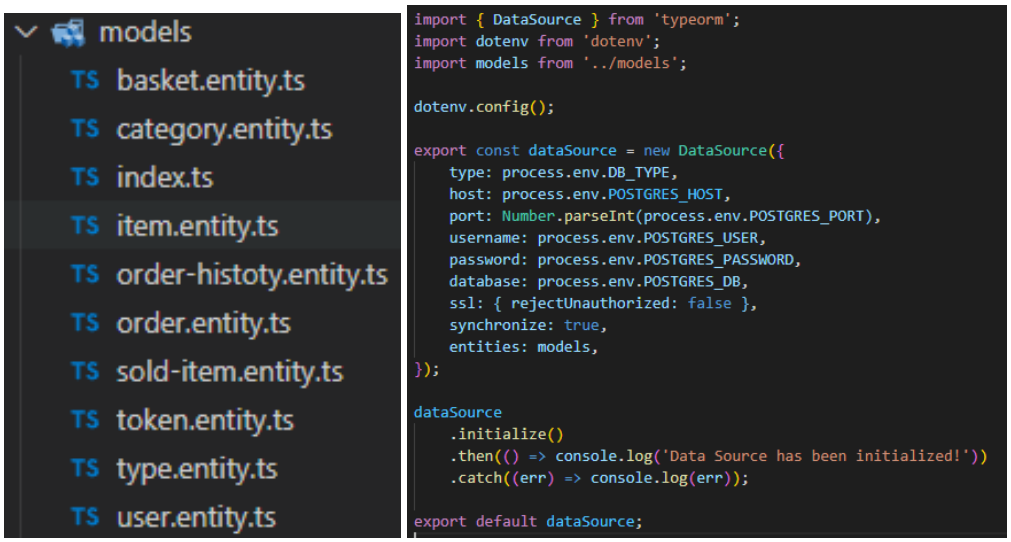
Рисунок 5.1 – Файлова структура сервера

– а) Файлова структура – б) Список пакетів

В кожній директорії буде зберігатись певні частини сервера. Наприклад models зберігає сутності які відповідають сутностям в базі даних, а services зберігає усі операції з базою даних.

Першим етапом написання сервера є створення базу даних і підключення до неї.

Використовуючи діаграму бази даних, створену на етапі проектування додатку, описуємо всі сутності, їх атрибути та зв’язки між ними. А потім підключаємося до бази даних.



а) б)

Рисунок 5.2 – Створення бази даних

– а) Моделі сутностей БД – б) Підключення і синхронізація з БД

Далі створюємо навігацію, використовуючи ці маршрути клієнтська частина додатку зможе взаємодіяти з сервером, загружати нові дані, або отримати якусь інформацію.

```ts
const router: Application = Router();

router.use('/user', userRouter);
router.use('/basket', basketRouter);
router.use('/category', categoryRouter);
router.use('/type', typeRouter);
router.use('/item', itemRouter);
router.use('/order-history', orderHistoryRouter);
router.use('/order', orderRouter);
router.use('/sold-item', soldItemRouter);

export default router;
```

а)

```ts
const router: Application = Router();

router.post('/reg', userController.register);
router.post('/login', userController.login);
router.get('/logout', userController.logout);
router.get('/refresh', userController.refresh);
router.get('/', authMiddleware, userController.check);
router.put('/', authMiddleware, userController.editUser);

export default router;
```

б)

Рисунок 5.3 – Маршрути сервера

– а) Загальне розподілення– б) Маршрути для /user

На кожный маршрут викликається певний контролер, який відповідає за отримання даних які були відправленні разом з запитом, якщо вони є, передачу в певні сервіси, та відправки відповіді.

Тому наступним етапом необхідно створити контролери.

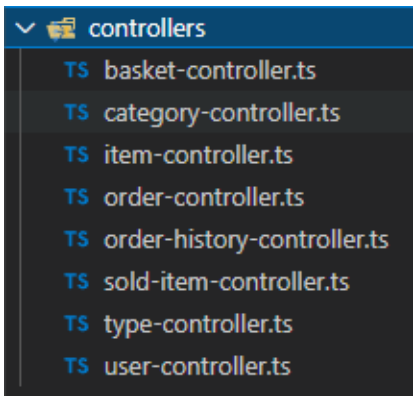


Рисунок 5.4 – Контролери сервера

Контролери використовують сервіси для обробки даних. Задача сервісів є взаємодія з базою даних, створення запитів і повернення отриманих даних.

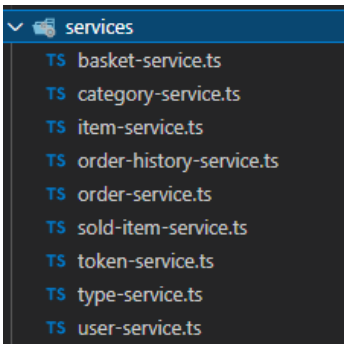


Рисунок 5.5 – Сервіси сервера

Після завершення розробки сервера робимо сборку, далі створюємо Docker контейнер і загружаєм на хостінг.

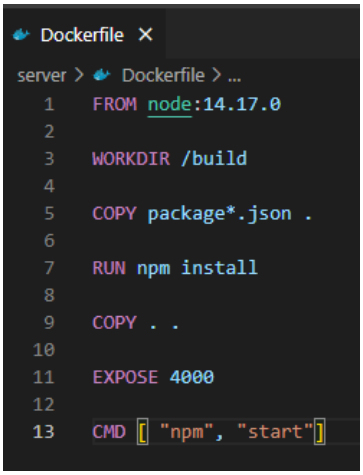
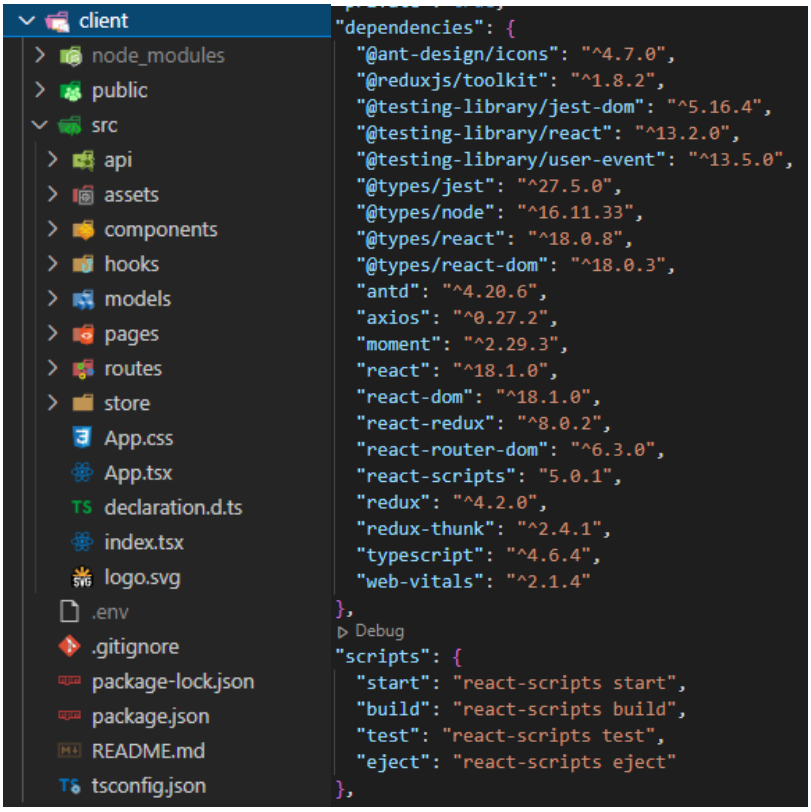


Рисунок 5.6 – Файл конфігурації контейнера

5.2 Розробка клієнтської частини

Створення клієнтської частини додатку також починаємо з файлової структури і установки необхідних пакетів. В якості бібліотеки React елементів обрана Ant Design [12].



а) б)

Рисунок 5.7 – Файлова структура React додатку – а) Файлова структура – б) Список пакетів

Структура клієнтської частини також як і серверна має свої директорії для зберігання певних фрагментів коду:

- api – запити на сервер
- components – всі реакт компоненти з яких створений додаток.
- pages – набір сторінок, кожна з яких використовує деякі компоненти
- routes – список маршрутів додатку
- store – директорія зберігає глобальний стан реалізований за допомогою Redux
- models – інтерфейси

Почнемо зі створення маршрутів і навігації:

```
export enum RouteNames {
    REGISTRATION = '/reg',
    LOGIN = '/login',
    USER = '/user',
    ITEM = '/item',
    BASKET = '/basket',
    ORDER_HISTORY = '/history',
}

export const publicRoutes: IRoute[] = [
    {
        key: RouteNames.REGISTRATION,
        path: RouteNames.REGISTRATION,
        element: Reg,
    },
    {
        key: RouteNames.LOGIN,
        path: RouteNames.LOGIN,
        element: Login,
    },
    {
        key: RouteNames.ITEM,
        path: RouteNames.ITEM,
        element: Item,
    },
];
```

```
export const privateRoutes: IRoute[] = [
    {
        key: RouteNames.BASKET,
        path: RouteNames.BASKET,
        element: Basket,
    },
    {
        key: RouteNames.USER,
        path: RouteNames.USER,
        element: User,
    },
    {
        key: RouteNames.ORDER_HISTORY,
        path: RouteNames.ORDER_HISTORY,
        element: OrderHistory,
    },
    {
        key: RouteNames.ITEM,
        path: RouteNames.ITEM,
        element: Item,
    },
];
```

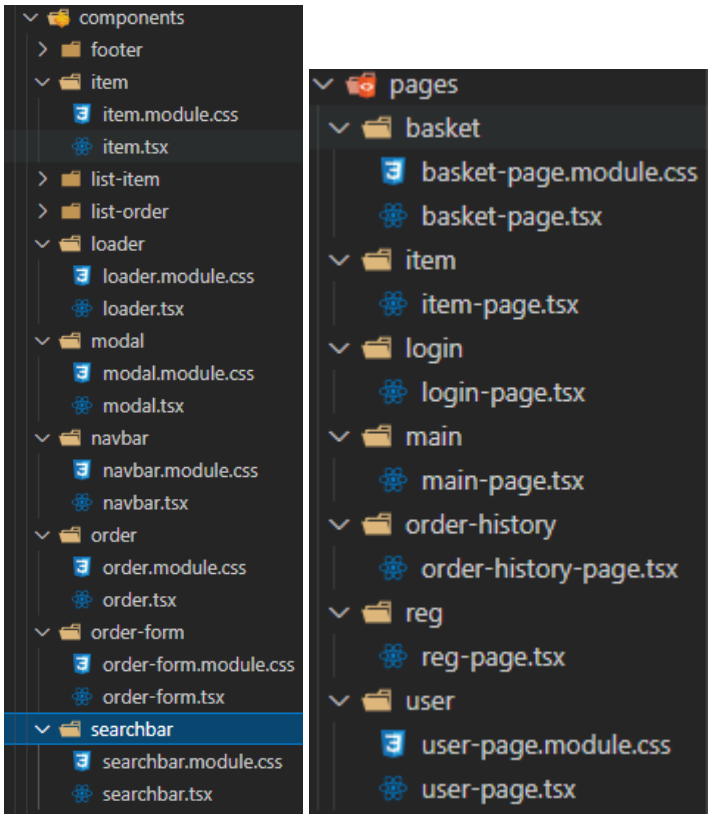
Рисунок 5.8 – Список маршрутів для навігації

В залежності від того авторизований користувач чи ні, йому доступні певні маршрути. Тому створюємо компонент роутер в якому це реалізуємо.

```
const AppRouter: FC = () => {
    const { isAuth } = useAppSelector((state) => state.authReducer);

    return (
        <Routes>
            {isAuth &&
                privateRoutes.map(({ element: Element, ...props }) => (
                    <Route element={<Element />} {...props} />
                ))}

            {!isAuth &&
                publicRoutes.map(({ element: Element, ...props }) => (
                    <Route element={<Element />} {...props} />
                ))}
            <Route path="/" element={<Main />} />
            <Route path="*" element={<Navigate to="/" replace />} />
        </Routes>
    );
};

export default AppRouter;
```

Рисунок 5.9 – Компонент навігації

Кожним маршруту відповідає певна сторінка, яка в свою чергу уявляє деякий набір компонентів, з яких вона створена. Тому далі використовуючи макети створюємо компоненти та стилізацію для них.



а) б)

Рисунок 5.10 – Компоненти і сторінки
– а) Структура компонентів – б) Сторінки

В процесі розробки деякий стан може знадобиться в різних частинах додатку, тому його краще роботи глобальним. Наприклад інформація про користувача, стан авторизації.

```
interface AuthState {
    isAuth: boolean;
    user: IUser;
    isLoading: boolean;
    error: string;
}

const initialState: AuthState = {
    isAuth: false,
    user: {} as IUser,
    isLoading: false,
    error: '',
};

export const authSilce = createSlice({
    name: 'auth',
    initialState,
    reducers: {
        setAuth(state, action: PayloadAction<boolean>) {
            state.isAuth = action.payload;
            state.isLoading = false;
        },
        setUser(state, action: PayloadAction<IUser>) {
            state.user = action.payload;
            state.isLoading = false;
        },
        setIsLoading(state, action: PayloadAction<boolean>) {
            state.isLoading = action.payload;
        },
        setError(state, action: PayloadAction<string>) {
            state.error = action.payload;
            state.isLoading = false;
        },
    },
});

export default authSilce.reducer;
```

Рисунок 5.11 – Приклад створення глобального стану

Використовуючи інструменти розробника в Google Chrome перевіряємо правильність роботи додатку. Також для більшої зручності варто використовувати розширення React Dev Tools та Redux Dev Tools. Вони дозволяють переглянути компонентну структуру додатку і значення глобального стану в Redux відповідно.

Командою **npm run build** робимо фінальну зборку додатку, після чого можна загрузити його на хостінг, де він буде доступний для всіх користувачів інтернету.

5.3 Опис і тестування додатку

Коли користувач вперше заходить на сайт він потрапляє на головну сторінку. На неї він може переглянути усі наявні товари, а також виконати пошук або сортування по категоріям.

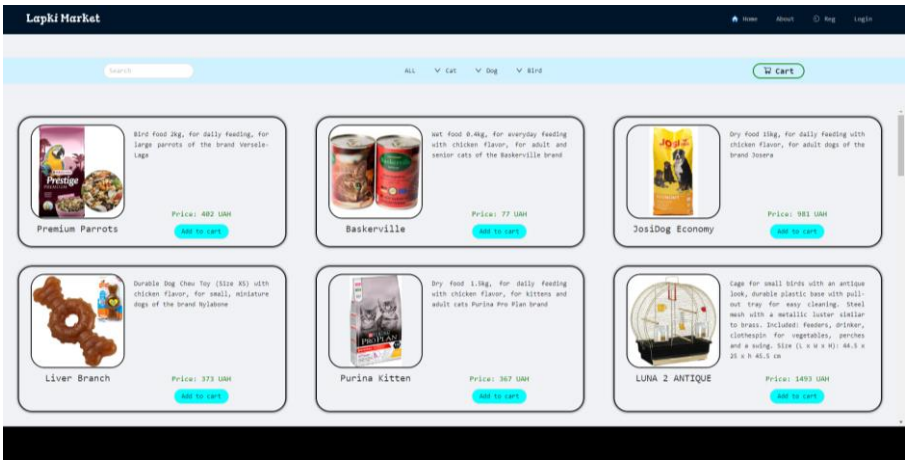


Рисунок 5.12 – Головна сторінка «Lapki Market»

Для створення замовлення необхідно пройти реєстрацію, або авторизуватись, якщо акаунт вже був створений.

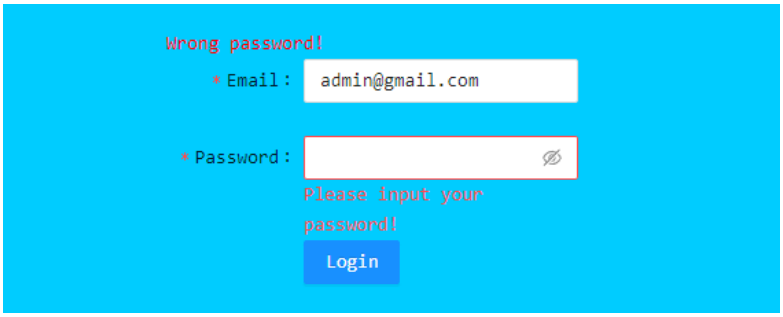


Рисунок 5.13 – Форма авторизації

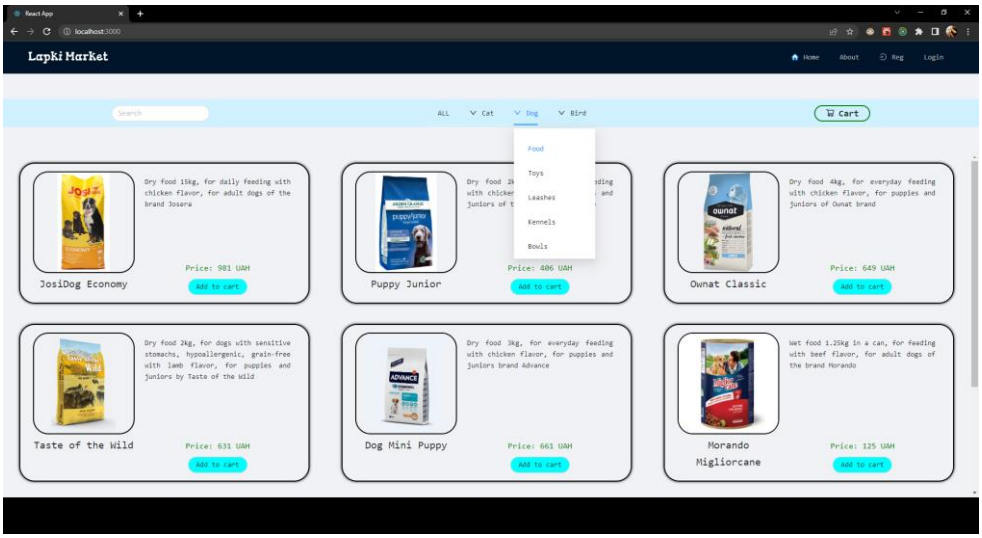


Рисунок 5.14 – Сортування за категоріями магазину «Lapki Market»

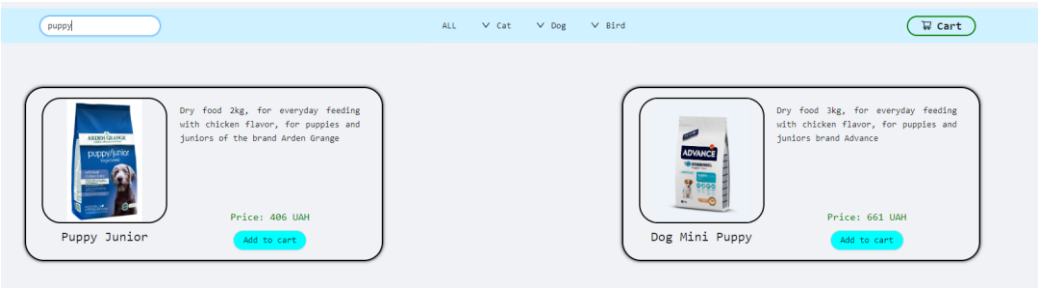


Рисунок 5.15 – Пошук за назвою

Після проходження авторизації користувач може додати необхідні товари до кошика. А далі оформити замовлення.

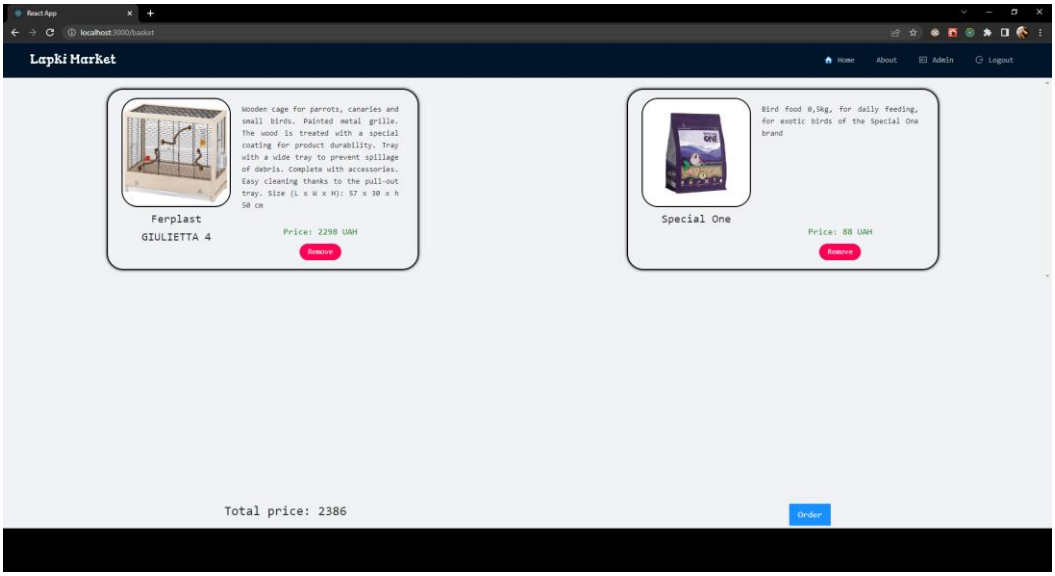


Рисунок 5.16 – Сторінка кошика користувача

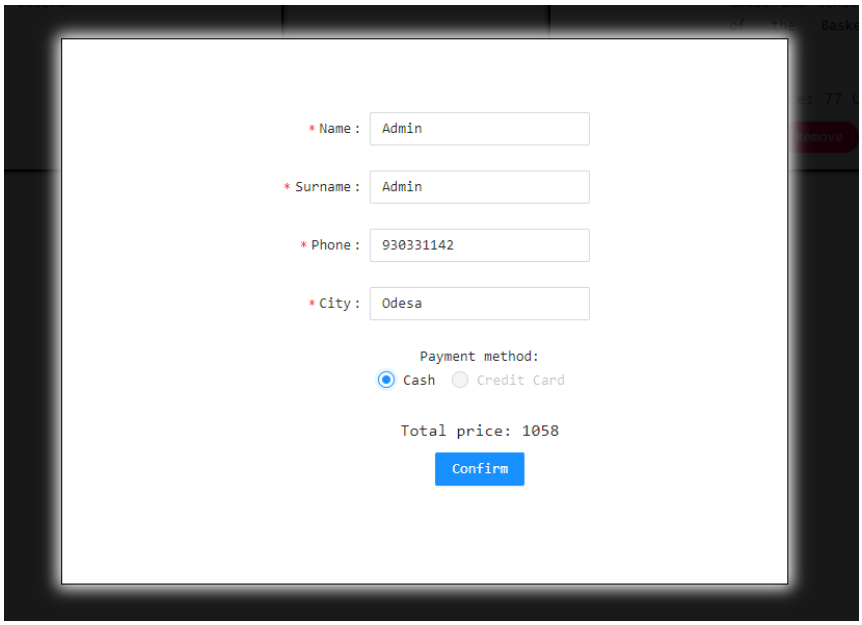


Рисунок 5.17 – Оформлення замовлення

Також користувач має можливість редагувати введену при реєстрації інформацію і переглянути історію попередніх замовлень.

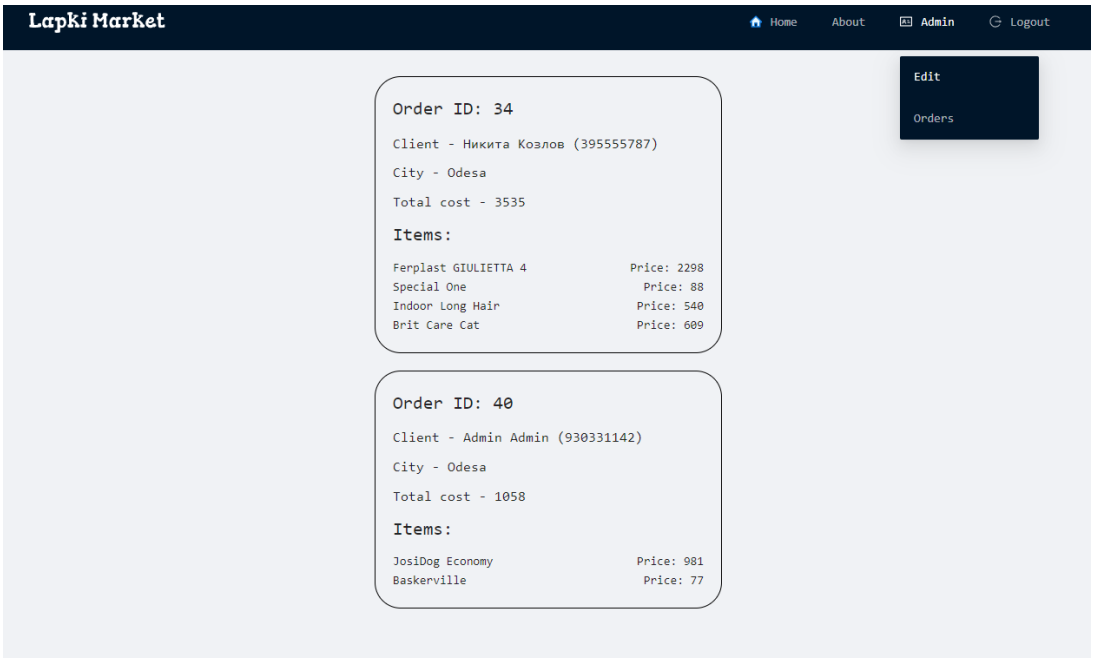


Рисунок 5.18 – Історія замовлень

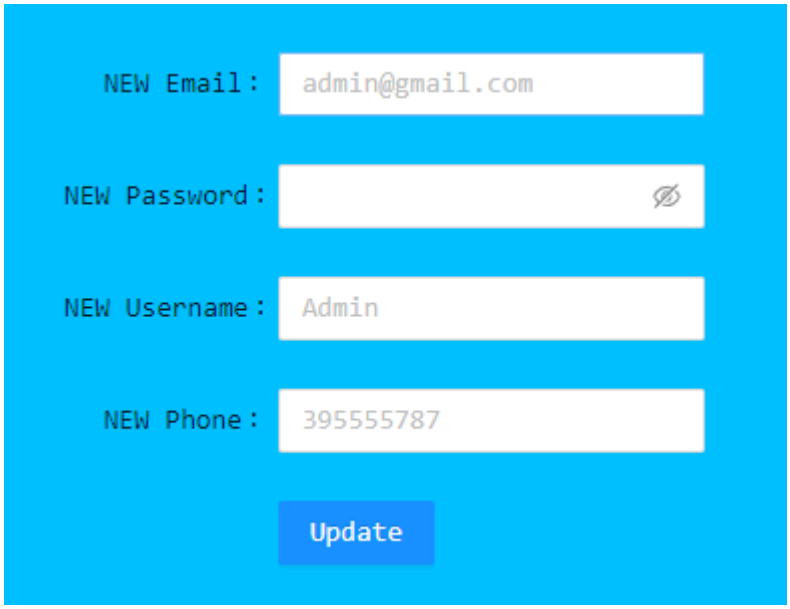


Рисунок 5.19 – Редагування інформації користувача

ВИСНОВКИ

В ході виконання кваліфікаційної роботи бакалавра був спроектований та розроблений web – додаток інтернет магазину товарів для домашніх тварин «Lapki Market». Першим етапом проводився аналіз предметної області та порівняння вже існуючих інтернет магазинів, таких як Pethouse, Zoostore і MasterZoo.

Середовищем для розробки була обрана Visual Studio Code, так як вона є безкоштовною і має відкритий вихідний код, що дає можливість створювати багато розширень які спрощують роботу. А також вона є однією з найпопулярніших IDE, завдяки чому має велику спільноту.

Основними технологіями для реалізації магазину є фреймворк React для клієнтської частини, і платформа NodeJS для серверної частини з використанням фреймворка Express. В якості бази даних використовується PostgreSQL. Також були використані такі технології і бібліотеки як Redux Toolkit, Ant Design, TypeORM, Axios.

На етапі проектування була створена UML-діаграма бази даних, а також діаграми варіантів використання, активності, послідовності. Розроблені макети інтерфейсу сторінок магазину.

Тестування сервера проводилося за допомогою ПЗ Postman, а інтерфейсу за допомогою інструментів розробника в Google Chrome.

В роботі описані можливості додатку і приклад його використання, в базу даних були додані деякі товари в різні категорії для демонстрації роботи магазину, функцій пошуку та сортування.

Отже, розроблена система вже має необхідний набор функцій, якого буде достатньо для повноцінного використовування, але є багато варіантів розширення і покращення магазину в майбутньому, наприклад:

1) Збільшити кількість наявних категорій і типів товарів.
2) Покращити критерії сортування і пошуку товарів.
3) Додати нові способи оплати замовлення.
4) Розширити функціонал особистого кабінету користувача.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Друге місце за швидкістю зростання в Європі [Електронний ресурс]. Режим доступу: https://biz.nv.ua/ukr/markets/domashni-tvarini-yak-roste-ukrajinskiy-rinok-poslug-i-tovariv-dlya-sobak-i-kishok-50163726.html.
2. Офіційний веб-сайт NodeJS [Електронний ресурс]. Режим доступу: https://nodejs.org/en/about/.
3. Офіційний веб-сайт PostgreSQL [Електронний ресурс]. Режим доступу: https://www.postgresql.org.
4. Офіційний веб-сайт ReactJS [Електронний ресурс]. Режим доступу: https://ru.reactjs.org
5. Офіційний веб-сайт TypeORM [Електронний ресурс]. Режим доступу: https://typeorm.io
6. Офіційний веб-сайт ExpressJS [Електронний ресурс]. Режим доступу: https://expressjs.com
7. Офіційний веб-сайт TypeScript [Електронний ресурс]. Режим доступу: https://www.typescriptlang.org
8. ЧМИР І. О. ЕЛЕКТРОННА КОМЕРЦІЯ Режим доступу: http://eprints.library.odeku.edu.ua/id/eprint/8564/1/Чмырь%20И.А.%20ЭК.%202021.pdf
9. Офіційний веб-сайт Figma Режим доступу: https://www.figma.com/
10. Офіційний веб-сайт Postman Режим доступу: https://www.postman.com/
11. Офіційний веб-сайт VSCode Режим доступу: https://code.visualstudio.com
12. Офіційний веб-сайт Ant Design Режим доступу: https://ant.design

ДОДАТОК А
Вихідний код програми

```
import dotenv from 'dotenv';
dotenv.config();
import express, { Application } from 'express';
import fileUpload from 'express-fileupload';
import cors from 'cors';
import cookieParser from 'cookie-parser';
import router from './routes/index';
import errorMiddleware from './middleware/error-middleware';
import path from 'path';

const PORT = process.env.PORT || 4000;
const app: Application = express();

app.use(express.json());
app.use(cookieParser());
app.use(
    cors({
        origin: process.env.CLIENT_URL,
        credentials: true,
    })
);
app.use(fileUpload({}));
app.use(express.static(path.resolve(__dirname, '..', 'public')));
app.use('/api', router);
app.use(errorMiddleware);

const start = async () => {
    try {
        app.listen(PORT, () => {
            console.log(`App is running at http://localhost:${PORT}`);
        });
    } catch (error) {
        console.log(error);
    }
};

start();

const router: Application = Router();

router.use('/user', userRouter);
router.use('/basket', basketRouter);
router.use('/category', categoryRouter);
router.use('/type', typeRouter);
router.use('/item', itemRouter);
router.use('/order-history', orderHistoryRouter);
router.use('/order', orderRouter);
router.use('/sold-item', soldItemRouter);

export default router;
```

```
import Router, { Application, Request, Response } from 'express';
import userController from '../controllers/user-controller';
import authMiddleware from '../middleware/auth-middleware';
const router: Application = Router();

router.post('/reg', userController.register);
router.post('/login', userController.login);
router.get('/logout', userController.logout);
router.get('/refresh', userController.refresh);
router.get('/', authMiddleware, userController.check);
router.put('/', authMiddleware, userController.editUser);

export default router;

import { NextFunction, Request, Response } from 'express';
import { CreateUserDto } from '../dto/user-dto';
import tokenService from '../services/token-service';
import userService from '../services/user-service';

class UserController {
    async getAllUser(req: Request, res: Response, next: NextFunction) {
        try {
            res.json({ message: 'ALL USER GET' });
        } catch (error) {
            next(error);
        }
    }

    async register(
        req: Request<{}, {}, CreateUserDto>,
        res: Response,
        next: NextFunction
    ) {
        try {
            const { email, password, username, phone } = req.body;
            const userData = await userService.registration({
                email,
                password,
                username,
                phone,
            });
            res.cookie('refreshToken', userData.refreshToken, {
                maxAge: 30 * 24 * 60 * 60 * 1000,
                httpOnly: true,
            });

            return res.json(userData);
        } catch (error) {
            next(error);
        }
    }

    async login(req: Request, res: Response, next: NextFunction) {
        try {
            const { email, password } = req.body;
```

```
            const userData = await userService.login({ email, password });
            res.cookie('refreshToken', userData.refreshToken, {
                maxAge: 30 * 24 * 60 * 60 * 1000,
                httpOnly: true,
            });

            return res.json(userData);
        } catch (error) {
            next(error);
        }
    }

    async logout(req: Request, res: Response, next: NextFunction) {
        try {
            const { refreshToken } = req.cookies;
            const token = await userService.logout(refreshToken);
            res.clearCookie('refreshToken');

            return res.json(token);
        } catch (error) {
            next(error);
        }
    }

    async refresh(req: Request, res: Response, next: NextFunction) {
        try {
            const { refreshToken } = req.cookies;
            const userData = await userService.refresh(refreshToken);

            res.cookie('refreshToken', userData.refreshToken, {
                maxAge: 30 * 24 * 60 * 60 * 1000,
                httpOnly: true,
            });

            return res.json(userData);
        } catch (error) {
            next(error);
        }
    }

    async check(req: Request, res: Response, next: NextFunction) {
        try {
            const token = tokenService.generateTokens({
                //@ts-ignore
                id: req.user.id,
                //@ts-ignore
                email: req.user.email,
                //@ts-ignore
                phone: req.user.phone,
                //@ts-ignore
                username: req.user.username,
            });
            res.json({ token });
        } catch (error) {
            next(error);
```

```
        }
    }

    async editUser(req: Request, res: Response, next: NextFunction) {
        try {
            //@ts-ignore
            const userId = req.user.id;
            const { email, password, username, phone } = req.body;
            const userData = await userService.editUser({
                userId,
                email,
                password,
                username,
                phone,
            });

            return res.json(userData);
        } catch (error) {
            next(error);
        }
    }
}

export default new UserController();
```

```
import { NextFunction, Request, Response } from 'express';
import { UserDto } from '../dto/user-dto';
import ApiError from '../exceptions/api-error';
import tokenService from '../services/token-service';

export default function (req: Request, res: Response, next: NextFunction) {
    try {
        const authorizationHeader = req.headers.authorization;
        if (!authorizationHeader) {
            return next(ApiError.UnauthorizedError());
        }

        const accessToken = authorizationHeader.split(' ')[1];
        if (!accessToken) {
            return next(ApiError.UnauthorizedError());
        }

        const userData = tokenService.validateAccessToken(accessToken);
        if (!userData) {
            return next(ApiError.UnauthorizedError());
        }

        //@ts-ignore
        req.user = userData;
        next();
    } catch (error) {
        return next(ApiError.UnauthorizedError());
    }
}
```

```
import { CreateUserDto, UserDto } from '../dto/user-dto';
import { User } from '../models/user.entity';
import ApiError from '../exceptions/api-error';
import dataSource from '../utils/connect-db';
import bcrypt from 'bcrypt';
import uuid from 'uuid';
import tokenService from './token-service';
import { OrderHistory } from '../models/order-histoty.entity';
import basketService from './basket-service';
import orderHistoryService from './order-history-service';

class UserService {
    async registration(createUserDto: CreateUserDto) {
        if (
            !createUserDto.email ||
            !createUserDto.password ||
            !createUserDto.phone ||
            !createUserDto.username
        ) {
            throw ApiError.BadRequest(`Fields can't be empty!`);
        }

        const candidate = await dataSource.manager.findOneBy(User, {
            email: createUserDto.email,
        });
        if (candidate) {
            throw ApiError.BadRequest(
                `User with email ${createUserDto.email} already exist`
            );
        }

        const hashPassword = await bcrypt.hash(createUserDto.password, 3);
        //const activationLink = uuid.v4();
        const user = new User();
        user.email = createUserDto.email;
        user.password = hashPassword;
        user.username = createUserDto.username;
        user.phone = createUserDto.phone;
        user.orderHistory = new OrderHistory();

        await dataSource.manager.save(User, user);
        await basketService.createBasket(user.id);
        await orderHistoryService.createOrderHistory(user.id);
        const tokens = tokenService.generateTokens({
            id: user.id,
            email: user.email,
            phone: user.phone,
            username: user.username,
        });
        await tokenService.saveToken(user.id, tokens.refreshToken);

        return {
            ...tokens,
            user: new UserDto(user),
        };
```

```
}

async login({ email, password }) {
    if (!email || !password) {
        throw ApiError.BadRequest('Empty fields!');
    }

    const user = await dataSource.manager.findOneBy(User, { email });
    if (!user)
        throw ApiError.BadRequest(`User with email ${email} not found!`);

    const isPassEquals = await bcrypt.compare(password, user.password);
    if (!isPassEquals) throw ApiError.BadRequest('Wrong password!');

    const userDto = new UserDto(user);
    const tokens = tokenService.generateTokens({
        id: user.id,
        email,
        phone: user.phone,
        username: user.username,
    });
    await tokenService.saveToken(userDto.id, tokens.refreshToken);

    return {
        ...tokens,
        user: userDto,
    };
}

async logout(refreshToken: string) {
    const token = await tokenService.removeToken(refreshToken);
    return token;
}

async refresh(refreshToken: string) {
    if (!refreshToken) {
        throw ApiError.UnauthorizedError();
    }

    const userData = tokenService.validateRefreshToken(refreshToken);
    const tokenFromDB = await tokenService.findToken(refreshToken);
    if (!userData || !tokenFromDB) {
        throw ApiError.UnauthorizedError();
    }

    const user = await dataSource.manager.findOneBy(User, {
        id: (<UserDto>userData).id,
    });
    const userDto = new UserDto(user);
    const tokens = tokenService.generateTokens({ ...userDto });
    await tokenService.saveToken(userDto.id, tokens.refreshToken);
    return {
        ...tokens,
        user: userDto,
    };
```

```
    }

    async editUser({ userId, email, password, username, phone }) {
        const user = await dataSource.manager.findOneBy(User, { id: userId });
        if (!user) {
            throw ApiError.BadRequest(`User not found!`);
        }
        const hashPassword = await bcrypt.hash(password, 3);
        if (email) user.email = email;
        if (password) user.password = hashPassword;
        if (username) user.username = username;
        if (phone) user.phone = phone;

        const savedUser = dataSource.manager.save(User, user);
        return savedUser;
    }
}



export default new UserService();

import {
    Entity,
    Column,
    BaseEntity,
    PrimaryGeneratedColumn,
    OneToOne,
} from 'typeorm';
import { Basket } from './basket.entity';
import { OrderHistory } from './order-histoty.entity';

@Entity('User')
export class User extends BaseEntity {
    @PrimaryGeneratedColumn()
    id: number;

    @Column('varchar', { unique: true })
    email: string;

    @Column('varchar')
    username: string;

    @Column('varchar')
    password: string;

    @Column('int')
    phone: number;

    @OneToOne(() => Basket)
    basket: Basket;

    @OneToOne(() => OrderHistory)
    orderHistory: OrderHistory;
}
```

```
export default class ApiError extends Error {
    status: number;
    errors: Error[];

    constructor(status: number, message: string, errors: Error[] = []) {
        super(message);
        this.status = status;
        this.errors = errors;
    }

    static UnauthorizedError() {
        console.log(new ApiError(401, 'User is not authorized'));
        return new ApiError(401, 'User is not authorized');
    }

    static BadRequest(message: string, errors: Error[] = []) {
        return new ApiError(400, message, errors);
    }
}
```

```
import { DataSourceOptions } from 'typeorm';

declare global {
    namespace NodeJS {
        interface ProcessEnv {
            DB_TYPE: 'postgres';
            PORT: string;
            POSTGRES_PORT: string;
            POSTGRES_USER: string;
            POSTGRES_PASSWORD: string;
            POSTGRES_DB: string;
            JWT_ACCESS_KEY: string;
            JWT_REFRESH_KEY: string;
            CLIENT_URL: string;
        }
    }
}

export {};
```

```
import { User } from '../models/user.entity';

interface UserReg {
    email: string;
    username: string;
    phone: number;
    password: string;
}

export class CreateUserDto {
```

```
    readonly email: string;
    readonly username: string;
    readonly phone: number;
    readonly password: string;

    constructor({ email, username, phone, password }: UserReg) {
        this.email = email;
        this.username = username;
        this.phone = phone;
        this.password = password;
    }
}

export class UserDto {
    readonly id: number;
    readonly email: string;
    readonly username: string;
    readonly phone: number;

    constructor(model: User) {
        this.email = model.email;
        this.username = model.username;
        this.phone = model.phone;
        this.id = model.id;
    }
}
```

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1" />
        <title>React App</title>
    </head>
    <body>
        <noscript>You need to enable JavaScript to run this app.</noscript>
        <div id="root"></div>
    </body>
</html>
```

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import { Provider } from 'react-redux';
import { BrowserRouter } from 'react-router-dom';
import { setupStore } from './store/store';

const store = setupStore();

const root = ReactDOM.createRoot(
    document.getElementById('root') as HTMLElement
);
root.render(
```

```
    <Provider store={store}>
        <BrowserRouter>
            <App />
        </BrowserRouter>
    </Provider>
);

import { Layout } from 'antd';
import React, { FC, useEffect } from 'react';
import './App.css';
import AppRouter from './components/AppRouter';
import Navbar from './components/navbar/navbar';
import { useAppDispatch, useAppSelector } from './hooks/redux';
import { checkAuth } from './store/reducers/action-creator';

const App: FC = () => {
    const dispatch = useAppDispatch();

    useEffect(() => {
        if (localStorage.getItem('token')) {
            dispatch(checkAuth());
        }
    }, []);

    return (
        <Layout>
            <Navbar />
            <Layout.Content>
                <AppRouter />
            </Layout.Content>
        </Layout>
    );
};

export default App;

import React from 'react';
import Login from '../pages/login/login-page';
import Reg from '../pages/reg/reg-page';
import Basket from '../pages/basket/basket-page';
import Item from '../pages/item/item-page';
import User from '../pages/user/user-page';
import OrderHistory from '../pages/order-history/order-history-page';

export interface IRoute {
    key: string;
    path: string;
    element: React.ComponentType;
}

export enum RouteNames {
    REGISTRATION = '/reg',
    LOGIN = '/login',
    USER = '/user',
    ITEM = '/item',
```

```
    BASKET = '/basket',
    ORDER_HISTORY = '/history',
}

export const publicRoutes: IRoute[] = [
    {
        key: RouteNames.REGISTRATION,
        path: RouteNames.REGISTRATION,
        element: Reg,
    },
    {
        key: RouteNames.LOGIN,
        path: RouteNames.LOGIN,
        element: Login,
    },
    {
        key: RouteNames.ITEM,
        path: RouteNames.ITEM,
        element: Item,
    },
];

export const privateRoutes: IRoute[] = [
    {
        key: RouteNames.BASKET,
        path: RouteNames.BASKET,
        element: Basket,
    },
    {
        key: RouteNames.USER,
        path: RouteNames.USER,
        element: User,
    },
    {
        key: RouteNames.ORDER_HISTORY,
        path: RouteNames.ORDER_HISTORY,
        element: OrderHistory,
    },
    {
        key: RouteNames.ITEM,
        path: RouteNames.ITEM,
        element: Item,
    },
];

import React, { FC } from 'react';
import { Navigate, Route, Routes } from 'react-router-dom';
import { useAppSelector } from '../hooks/redux';
import Main from '../pages/main/main-page';
import { privateRoutes, publicRoutes } from '../routes/routes';
import Loader from './loader/loader';

const AppRouter: FC = () => {
    const { isAuth } = useAppSelector((state) => state.authReducer);
```

```
    return (
        <Routes>
            {isAuth &&
                privateRoutes.map(({ element: Element, ...props }) => (
                    <Route element={<Element />} {...props} />
                ))}

            {!isAuth &&
                publicRoutes.map(({ element: Element, ...props }) => (
                    <Route element={<Element />} {...props} />
                ))}
            <Route path="/" element={<Main />} />
            <Route path="*" element={<Navigate to="/" replace />} />
        </Routes>
    );
};

export default AppRouter;

import axios, { AxiosRequestConfig } from 'axios';
import { IUserResponse } from '../models/IUser';

const host = axios.create({
    baseURL: 'https://lapki-market.herokuapp.com/',
    withCredentials: true,
});

const authHost = axios.create({
    baseURL: 'https://lapki-market.herokuapp.com/',
    withCredentials: true,
});

const authReqInterceptor = (config: AxiosRequestConfig) => {
    config.headers!.authorization = `Bearer ${localStorage.getItem('token')}`;
    return config;
};
authHost.interceptors.request.use(authReqInterceptor);

authHost.interceptors.response.use(
    (config) => {
        return config;
    },
    async (error) => {
        const originalReq = error.config;
        if (
            error.response.status === 401 &&
            error.config &&
            !error.config._isRetry
        ) {
            originalReq._isRetry = true;
            try {
                const response = await host.get<IUserResponse>(
                    '/api/user/refresh'
                );
                localStorage.setItem('token', response.data.accessToken);
```

```
                return authHost.request(originalReq);
            } catch (error) {
                console.log('User is not authorized');
            }
        }
        throw error;
    }
);

export { host, authHost };

import { IOrderHistory } from '../models/IOrderHistory';
import { IUser, IUserResponse } from '../models/IUser';
import { authHost, host } from './index';

export interface RegUser {
    email: string;
    password: string;
    username: string;
    phone: number;
}

export interface LoginUser {
    email: string;
    password: string;
}

class UserService {
    async reg({ email, password, username, phone }: RegUser) {
        const response = await host.post<IUserResponse>('api/user/reg', {
            email,
            password,
            username,
            phone,
        });
        localStorage.setItem('token', response.data.accessToken);

        return response;
    }

    async login({ email, password }: LoginUser) {
        const response = await host.post<IUserResponse>('api/user/login', {
            email,
            password,
        });
        localStorage.setItem('token', response.data.accessToken);

        return response;
    }

    async logout() {
        const response = await authHost.get('api/user/logout');
        localStorage.removeItem('token');

        return response;
```

```
    }

    async check() {
        const response = await host.get<IUserResponse>('api/user/refresh');
        localStorage.setItem('token', response.data.accessToken);

        return response;
    }

    async editProfile({ email, password, username, phone }: RegUser) {
        const response = await authHost.put<IUser>('api/user', {
            email,
            password,
            username,
            phone,
        });

        return response;
    }

    async getOrdersHistory() {
        const response = await authHost.get<IOrderHistory>('api/order-
history');

        return response;
    }
}

export default new UserService();

import { authHost } from '.';
import { IOrder } from '../models/IOrder';

class OrderService {
    async createOrder(order: IOrder) {
        const response = authHost.post('api/order', order);
    }
}

export default new OrderService();

import { authHost } from '.';
import { IItem } from '../models/IItem';

interface Basket {
    id: number;
    items: IItem[];
}

class BasketService {
    async getItems(userId: number) {
        const response = await authHost.get<Basket>(`api/basket/${userId}`);

        return response;
    }
```

```
    async addItem(userId: number, itemId: number) {
        const response = await authHost.post('api/basket/', { userId, itemId
});

        return response;
    }

    async removeItem(userId: number, itemId: number) {
        const response = await authHost.delete('api/basket/', {
            data: { userId, itemId },
        });

        return response;
    }
}

export default new BasketService();

import { Row } from 'antd';
import React, { FC } from 'react';
import { IItem } from '../../models/IItem';
import Item from '../item/item';

interface Props {
    items: IItem[];
    setItems?: (items: IItem) => void;
    isBasket: boolean;
}

const ItemList: FC<Props> = ({ items, setItems, isBasket }) => {
    return (
        <Row
            style={{ overflowY: 'scroll', maxHeight: '70vh' }}
            justify="space-around"
        >
            {items ? (
                items.map((item) => {
                    return (
                        <Item
                            key={item.id}
                            item={item}
                            setItems={setItems}
                            isBasket={isBasket}
                        />
                    );
                })
            ) : (
                <h1>Product list is empty!</h1>
            )}
        </Row>
    );
};

export default ItemList;
```

```
import React, { FC } from 'react';
import { Button } from 'antd';
import userService from '../../api/basket-service';
import { useAppSelector } from '../../hooks/redux';
import { IItem } from '../../models/IItem';
import styles from './item.module.css';

interface Props {
    item: IItem;
    setItems?: (items: IItem) => void;
    isBasket: boolean;
}

const Item: FC<Props> = ({ item, setItems, isBasket }) => {
    const { user } = useAppSelector((state) => state.authReducer);

    const addToBasket = async () => {
        const response = await userService.addItem(user.id, item.id);
        alert('Product "' + item.name + '" added to basket!');
    };

    const removeFromBasket = async () => {
        const response = await userService.removeItem(user.id, item.id);
        setItems!(item);
        alert('Product "' + item.name + '" added to basket!');
    };

    return (
        <div className={styles.card}>
            <div>
                <img
                    className={styles.photo}
                    src={'https://lapki-market.herokuapp.com/' + item.photo}
                    alt="product"
                />
                <h2 className={styles.title}>{item.name}</h2>
            </div>
            <div className={styles.info}>
                <p className={styles.description}>{item.description}</p>
                <div>
                    <h3 className={styles.price}>Price: {item.price} UAH</h3>
                    <Button
                        className={
                            isBasket ? styles.btn_remove : styles.btn_add
                        }
                        onClick={
                            isBasket
                                ? () => removeFromBasket()
                                : () => addToBasket()
                        }
                    >
                        {isBasket ? 'Remove' : 'Add to cart'}
                    </Button>
                </div>
            </div>
```

```
        </div>
    );
};

export default Item;

import React, { FC, useState } from 'react';
import { Layout, Menu, Row } from 'antd';
import {
    HomeTwoTone,
    IdcardOutlined,
    LogoutOutlined,
    LoginOutlined,
} from '@ant-design/icons';
import type { MenuProps } from 'antd';
import logo from './../../assets/logo.svg';
import { useNavigate } from 'react-router-dom';
import { RouteNames } from '../../routes/routes';
import { useAppDispatch, useAppSelector } from '../../hooks/redux';
import { logoutUser } from '../../store/reducers/action-creator';

const Navbar: FC = () => {
    const dispatch = useAppDispatch();
    const navigate = useNavigate();
    const { isAuth, user } = useAppSelector((state) => state.authReducer);

    const [current, setCurrent] = useState('mail');

    const logout = () => {
        dispatch(logoutUser());
    };

    const onClick: MenuProps['onClick'] = (e) => {
        navigate(e.key);
        setCurrent(e.key);
    };

    const nav_items: MenuProps['items'] = [
        {
            label: 'Home',
            key: '',
            icon: <HomeTwoTone />,
        },

        {
            label: (
                <a
                    href="https://t.me/+AZT8ieQyiyozYWFi"
                    target="_blank"
                    rel="noopener noreferrer"
                >
                    About
                </a>
            ),
            key: 'alipay',
```

```
        },
        isAuth
            ? {
                  label: user.username,
                  key: RouteNames.USER,
                  icon: <IdcardOutlined />,
                  children: [
                      {
                          label: 'Edit',
                          key: RouteNames.USER,
                      },
                      {
                          label: 'Orders',
                          key: RouteNames.ORDER_HISTORY,
                      },
                  ],
              }
            : {
                  label: 'Reg',
                  key: 'reg',
                  icon: <LoginOutlined />,
              },
        isAuth
            ? {
                  label: 'Logout',
                  onClick: logout,
                  key: 'logout',
                  icon: <LogoutOutlined />,
              }
            : {
                  label: 'Login',
                  key: 'login',
              },
    ];

    return (
        <Layout.Header>
            <Row justify="space-between">
                <img src={logo} alt="" />
                <Menu
                    selectable={false}
                    onClick={onClick}
                    theme="dark"
                    mode="horizontal"
                    disabledOverflow={true}
                    items={nav_items}
                ></Menu>
            </Row>
        </Layout.Header>
    );
};

export default Navbar;
```