МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК
СЕКЦІЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ПРОЕКТУВАННЯ

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: «Інформаційна система підтримки діяльності магазину з продажу освітлювальних приладів»

за спеціальністю 122 «Комп'ютерні науки»,
освітньо-професійна програма «Інформаційні технології проектування»

Виконавець роботи: студент групи ІТ.м-91 Тищенко Дмитро Вадимович

Кваліфікаційну роботу захищено на засіданні ЕК з оцінкою _____ «___» грудня 2020 р.

Науковий керівник _____ к.т.н., доц., Ващенко С.М.
(підпис)

Голова комісії _____ Шифрін Д.М.
(підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Суми-2020

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук
Секція інформаційних технологій проектування
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
Зав. секцією ІТП

_____ В. В. Шендрик
«___» _____ 2020 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра студентові

Тищенко Дмитро Вадимович
(прізвище, ім'я, по батькові)

1 Тема проекту *Інформаційна система підтримки діяльності магазину з продажу освітлювальних приладів*

затверджена наказом по університету від « 26 » листопада 2020 р. № 1824-III

2 Термін здачі студентом закінченого проекту « 07 » грудня 2020 р.

3 Вхідні дані до проекту існуючий сайт магазину, технічне завдання на розробку

4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) аналіз предметної області, постановка задачі та методи дослідження, моделювання інформаційної системи, розробка інформаційної системи

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) актуальність роботи, постановка задачі, огляд існуючих аналогів, архітектура інформаційної системи, архітектура бази даних, демонстрація розробленої інформаційної системи

6. Консультанти випускної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

Дата видачі завдання _____.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів випускної проекту	Термін виконання етапів проекту	Примітка
1	Розроблення концепції проекту	27.10.20-29.10.20	
2	Планування	30.10.20-02.11.20	
3	Прототипування	03.11.20-04.11.20	
4	Програмна реалізація продукту проекту	05.11.20-27.11.20	
5	Тестування продукту	30.11.20-02.12.20	
6	Здача в експлуатацію	03.12.20-04.12.20	
7	Оформлення документації	04.12.20-07.12.20	

Магістрант _____ Тищенко Д. В.

Керівник роботи _____ к.т.н., доц. Ващенко С. М.

РЕФЕРАТ

Тема кваліфікаційної роботи магістра- «Інформаційна система підтримки діяльності магазину з продажу освітлювальних приладів».

Пояснювальна записка складається зі вступу, 4 розділів, висновків, списку використаних джерел із 27 найменувань, додатків. Загальний обсяг роботи − 97 сторінки, у тому числі 60 сторінок основного тексту, 3 сторінки списку використаних джерел, 27 сторінок додатків.

Кваліфікаційну роботу магістра присвячено розробці інформаційної системи підтримки діяльності магазину з продажу освітлювальних приладів.

В роботі проведено аналіз предметної області та огляд існуючих аналогів, вибір методів дослідження, засобів реалізації, розробку алгоритму формування плану закупівель товарів.

У роботі виконано структурно-функціональне моделювання інформаційної системи, проектування моделі бази даних, моделювання варіантів використання, а також виконано планування робіт для забезпечення досягнення поставленої мети.

Результатом проведеної роботи є створена інформаційна система, що дозволяє вести онлайн продажі товарів, формувати звіти за продажами, що заощаджує час бухгалтерії, а також формувати список рекомендованих до закупівлі товарів.

Практичне значення роботи полягає в тому, що розроблена інформаційна система дозволить підвищити ефективність праці магазину та автоматизувати процеси бухгалтерського обліку, а також допомагатиме формувати плани закупівель на наступний місяць.

Ключові слова: прикладне програмування, бази даних, інтернет магазин, веб-додаток, панель адміністратора, освітлювальні прилади

ЗМІСТ

ВСТУП

В сучасному світі переважна кількість людей робить покупки в мережі Інтернет. Електронна комерція вважається чудовою альтернативою для приватних осіб та компаній для охоплення нових клієнтів. Якість надання послуг через Інтернет є важливою стратегією успіху, важливішою за ціну та присутність в Інтернеті. Веб-сайт електронної комерції визначено таким, що має значний вплив на ділову діяльність при вирішенні географічної проблеми.[1]

У більшості випадків керувати електронною комерцією складно для не підготовлених підприємців, адже кожного місяця вони повинні за законом подавати звітність, а також вирішувати, які товари їм необхідно замовляти. Закупівля товарів на наступний місяць це одна з тих проблем, з якою стикається кожен підприємець і яка несе за собою найбільші ризики, адже якщо не замовити потрібні товари, то люди, що захочуть у вас їх придбати, будуть змушені чекати їх доставки.

Магазини освітлення це одна з таких сфер. Течії дизайну інтер'єрів змінюються дуже швидко і слідкувати за ними дуже важко, коли вам ще необхідно керувати магазином. Підбір освітлювальних приладів на наступний місяць найчастіше виконується опираючись на продажі у поточному місяці. Але такої статистики дуже мало і краще було б використовувати продажі хоча б за пів року. Одна людина не може обробляти сама стільки інформації і їй необхідна допомога.

Можна знайти компанії, що використовують усі варіанти електронних продажів на додаток до традиційної торгівлі. Робота бухгалтерії в таких компаніях може бути кошмаром, без належного програмного забезпечення, що стосується продажів та бухгалтерського обліку, для автоматичного складання звітів про продажі, списання проданих товарів, реєстрації електронного надходження готівки, нарахування ПДВ та відстеження замовлень, дебіторська та кредиторська заборгованість у деталях та узагальнено. На жаль, не настільки рідкісні випадки, коли програмне забезпечення для бізнесу не є збалансованим, щоб служити як для комерційних, так і для

бухгалтерських, податкових та звітних цілей. Це складніше завдання, якщо використовуються всі можливі програмні платформи для електронної комерції, а бухгалтерія має амбіцію бути фінансовою, управлінською та податковою, використовуючи одну і ту ж базу даних та програмне забезпечення.[2]

Мета даної роботи полягає у створенні інформаційної системи підтримки діяльності магазину освітлювальних приладів, яка буде забезпечувати функціонал онлайн продажів, формувати необхідну звітність за продажами, а також допомагати формувати план закупівель.

Щоб досягти поставленої мети, необхідно виконати наступні задачі проекту:

- провести аналіз предметної області та дослідити програмні аналоги;
- визначити вимоги до розробки;
- спроектувати інформацій систему;
- підібрати необхідні засоби програмного забезпечення для реалізації даного проекту;
- провести розробку необхідних програмних модулів;
- розробити систему допомоги прийняття рішень.

Практична значимість роботи полягає в тому, що розроблена інформаційна система дозволить підвищити ефективність праці магазину та автоматизувати процеси бухгалтерського обліку, а також допомагатиме формувати плани закупівель на наступний місяць.

Розроблену інформаційну систему впроваджено в роботі компанії «Преміум Світло», про що складено відповідний акт (додаток А).

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Значний розвиток інформаційних технологій та їх інтеграція в ділове середовище сформували передумови для розвитку електронної комерції, яка зараз все більше набирає обертів.[3]

Традиційна торгівля, яка історично позиціонувала себе як динамічний вид економічної діяльності і формувалася під час переходу від природного способу виробництва до товарного швидко трансформується в сучасних умовах: врахування специфіки торгівлі, як сфери національної економіки, нові методи, форми і моделі торгівлі, предметів, випробовуються засоби та інструменти торгівлі. В той самий час, сама торгівля не перестає виконувати свою основну соціально-економічну функцію (забезпечується розподіл і перерозподіл споживчих товарів та їх доведення до кінцевого споживача; отримання прибутку від таких бізнес-процесів; задоволення потреб кінцевого споживача тощо).

Сьогодні, активний розвиток Інтернет-технологій призводить до використання глобальної мережі як платформи для торгівлі. Внаслідок чого предмет торгівлі набуває форм сучасних торгових мереж і магазинів з формою самообслуговування. Об'єкт торгівлі поширюється на віртуальні товари та електронний контент, а способи (засоби) оплати товарів стають електронними грошима. Інтернет стає платформою для торгівлі.[4]

Протягом останніх років сектор електронної комерції спостерігав колосальне зростання завдяки збільшенню використання таких пристроїв, як смартфони, планшети, доступ до Інтернету через широкосмуговий доступ, 3G, 4G та довіру до компаній електронної комерції, що призвело до збільшення споживачів в Інтернеті.[5]

Технології стають необхідністю, і тому їх застосування в бізнесі та комерції є не питанням вибору, а питанням примусу. Ось чому глобальна конкуренція може зіткнутися з п’ятьма важливими компонентами, а саме якістю, вартістю, зручністю, комунікацією та часом, якщо ми здійснюємо транзакції в електронному вигляді. Ці

п’ять аспектів також мають першорядне значення, коли нам доводиться стикатися з жорсткою конкуренцією.[6]

Необхідність створення інформаційної підтримки магазину освітлювальних приладів виникла в результаті бажання спрощення і автоматизації процесу діяльності магазину освітлювальних приладів.

Розглянемо процес роботи магазину. Кожен день, були придбання товарів чи ні, продавець магазину зобов’язаний надіслати головному бухгалтерові звітність про куплені товари, а також про залишки в магазині. Головним елементами є артикул освітлювального приладу, його виробник, назва та кількість штук, що залишилися. Наприклад, артикул “1-LED-5412”, виробник “Maxus LED”, назва, яку користувачі бачать на сайті “Лампочка світлодіодна 1-LED-5412 G45 F 4W 4100K 220V E14” і кількість штук.

Після того, як головному бухгалтеру надходять всі ці дані він повинен занести їх до програми “1С” для подачі офіційної звітності у податкову. Процес занесення даних і подача звітності відбувається вручну.

1.1 Робота онлайн магазину

На даний час інтернет магазин створений за допомогою маркетплейсу Prom.ua. Prom.ua — найбільший маркетплейс України, де продаються понад 100 мільйонів товарів від тисяч підприємців з усієї країни. Підприємці на платформі Прому самостійно створюють інтернет-магазини або розміщують свої товари в загальному каталозі.[7]

Розглянемо роботу онлайн магазину. Для покупців, які заходять на ресурс магазин виглядає як представлено на рисунка 1.1 – 1.3.

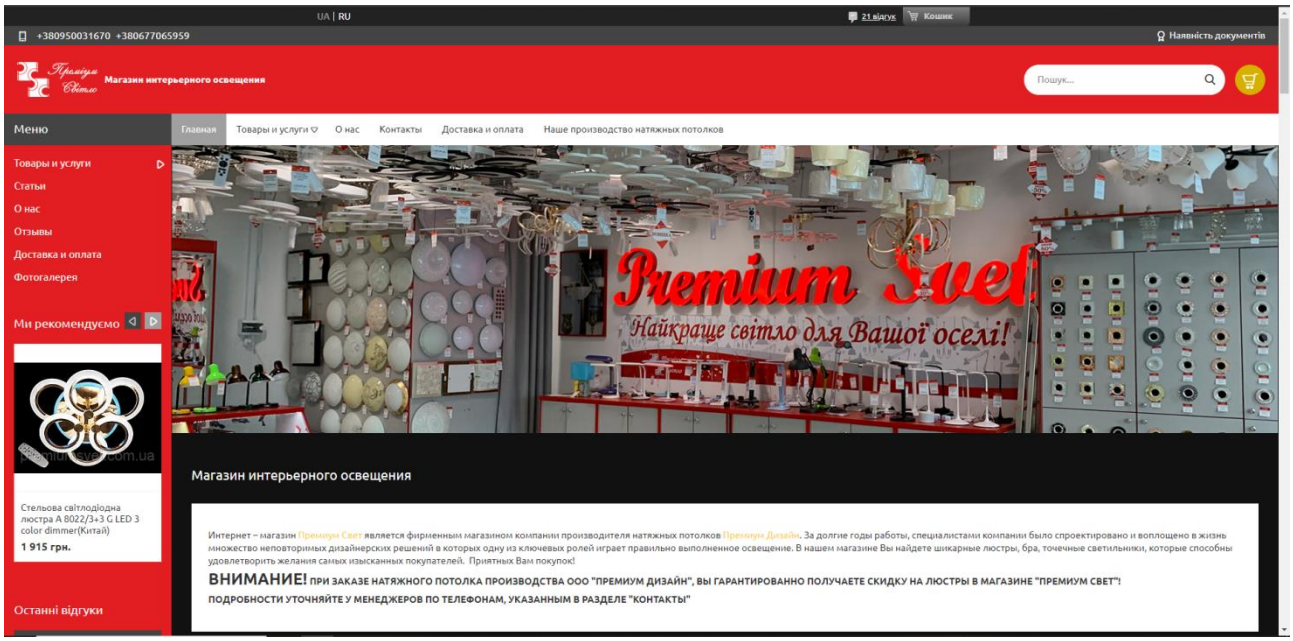


Рисунок 1.1 – Головна сторінка сайту

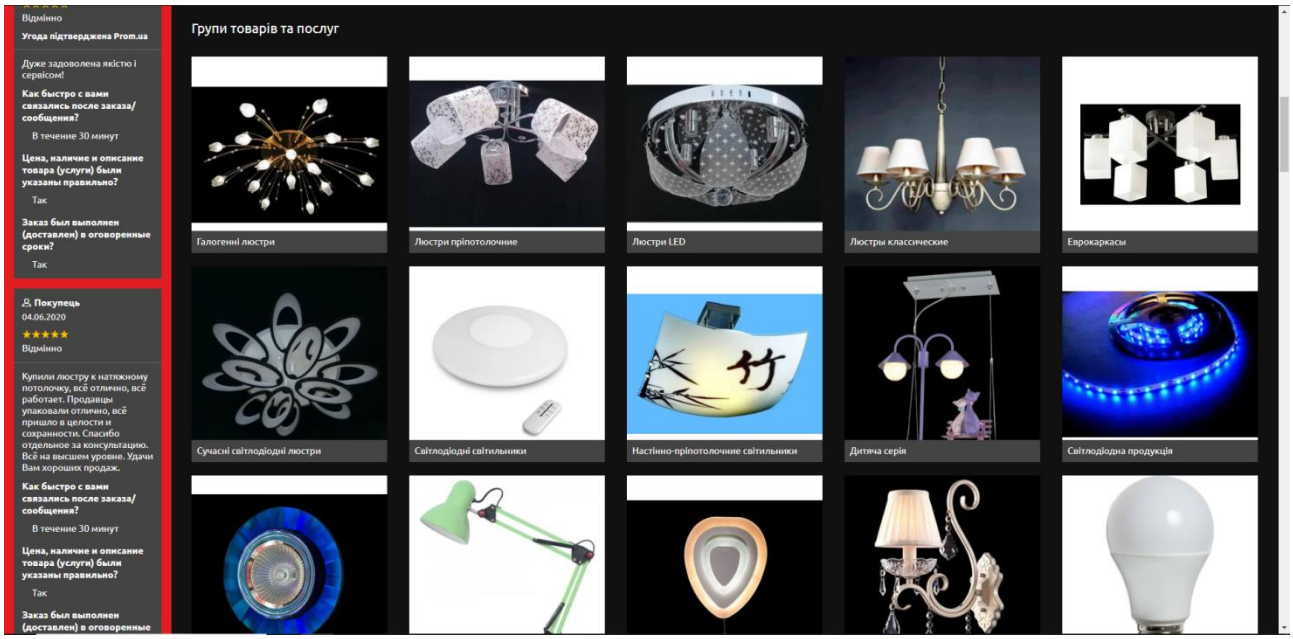


Рисунок 1.2 – Перелік категорій

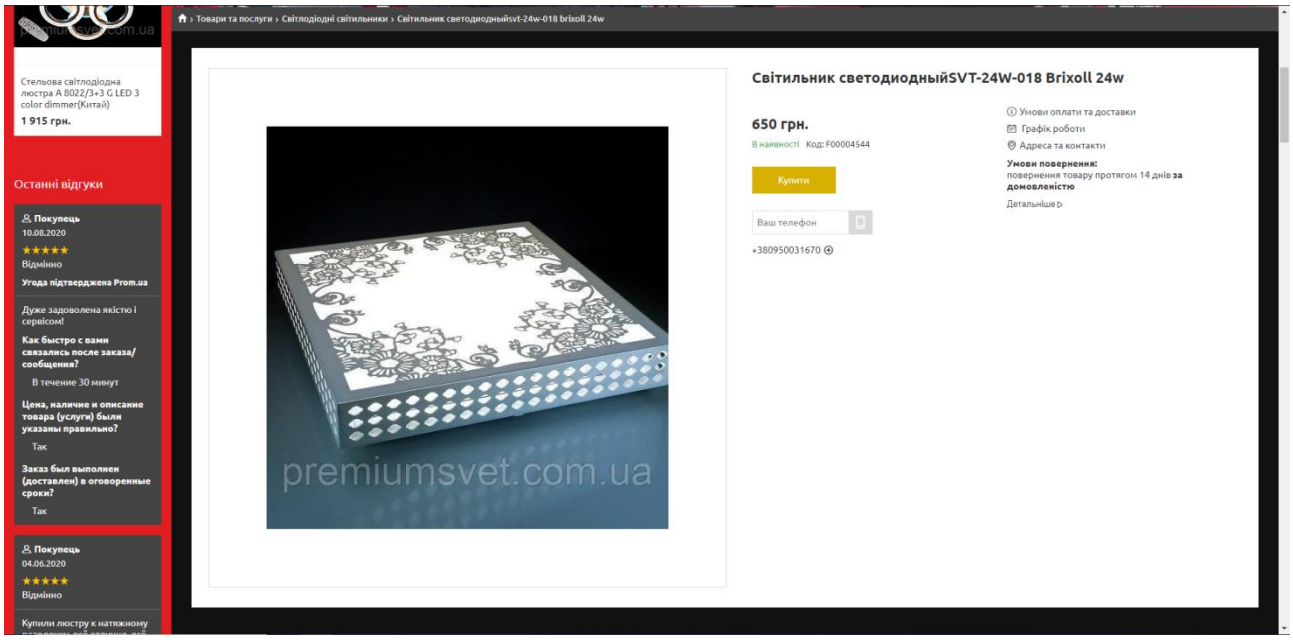


Рисунок 1.3 – Сторінка перегляду товару

На даний час до ресурсу є 2 типи доступу – це покупець та адміністратор. Покупець бачить всі товари які є на сайті, може переглянути інформацію про магазин, контакті дані та місцезнаходження точок продажу в різних містах, способи доставки та оплати товарів, а також регіони доставки. Користувачі сайту можуть також переглядати статті, як представлено на рисунку 1.4. Також можуть залишати свої коментарі про магазин, як показано на рисунку 1.5.

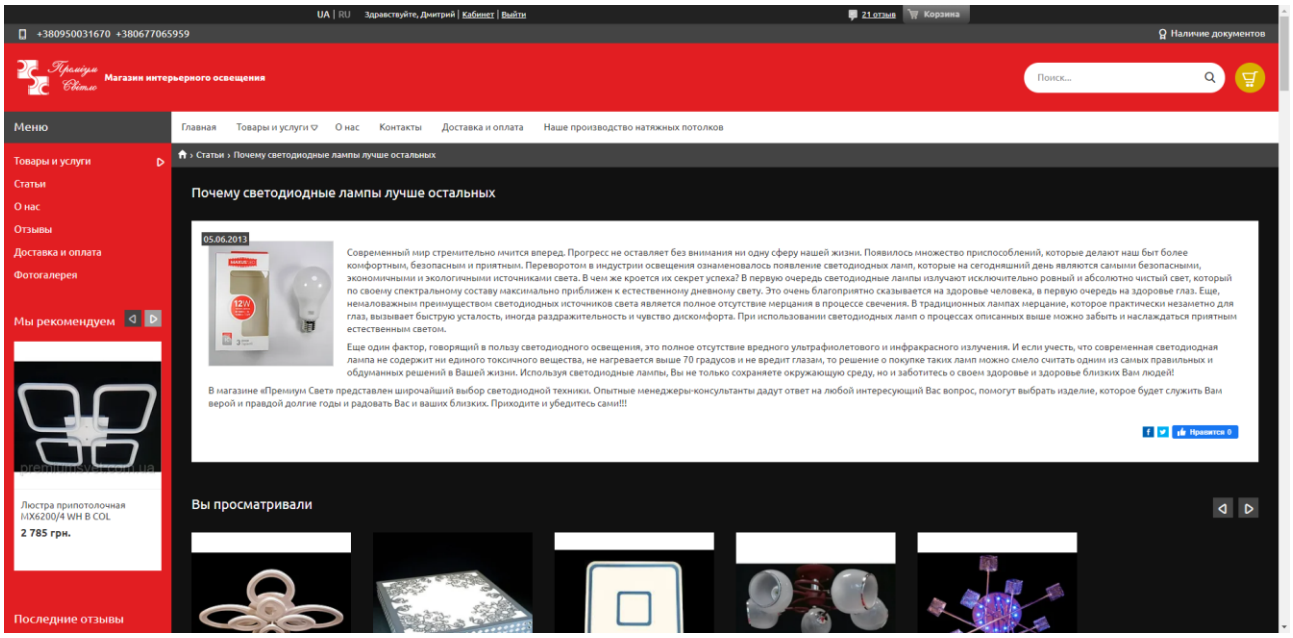


Рисунок 1.4 – Перегляд статей

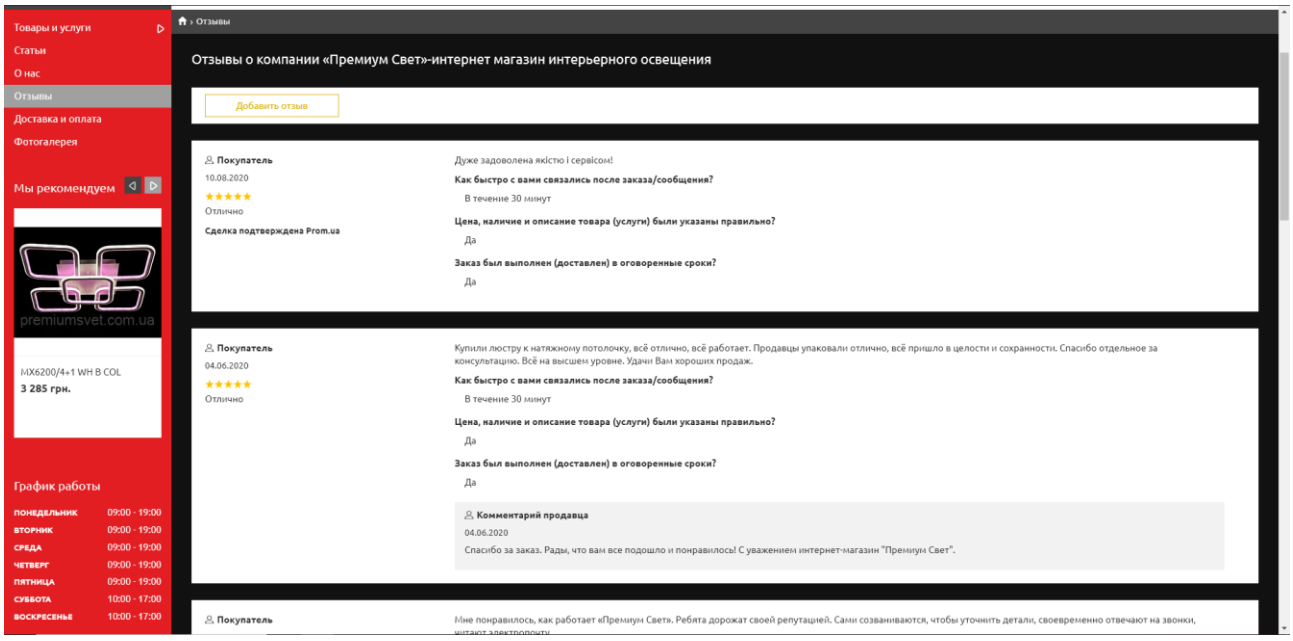


Рисунок 1.5 – Коментарі

Тепер розглянемо можливості адміністратора. Для цього необхідно увійти до платформи Пром як продавець. Після цього адміністратор потрапляє до особистого кабінету керування магазином, як показано на рисунку 1.6.

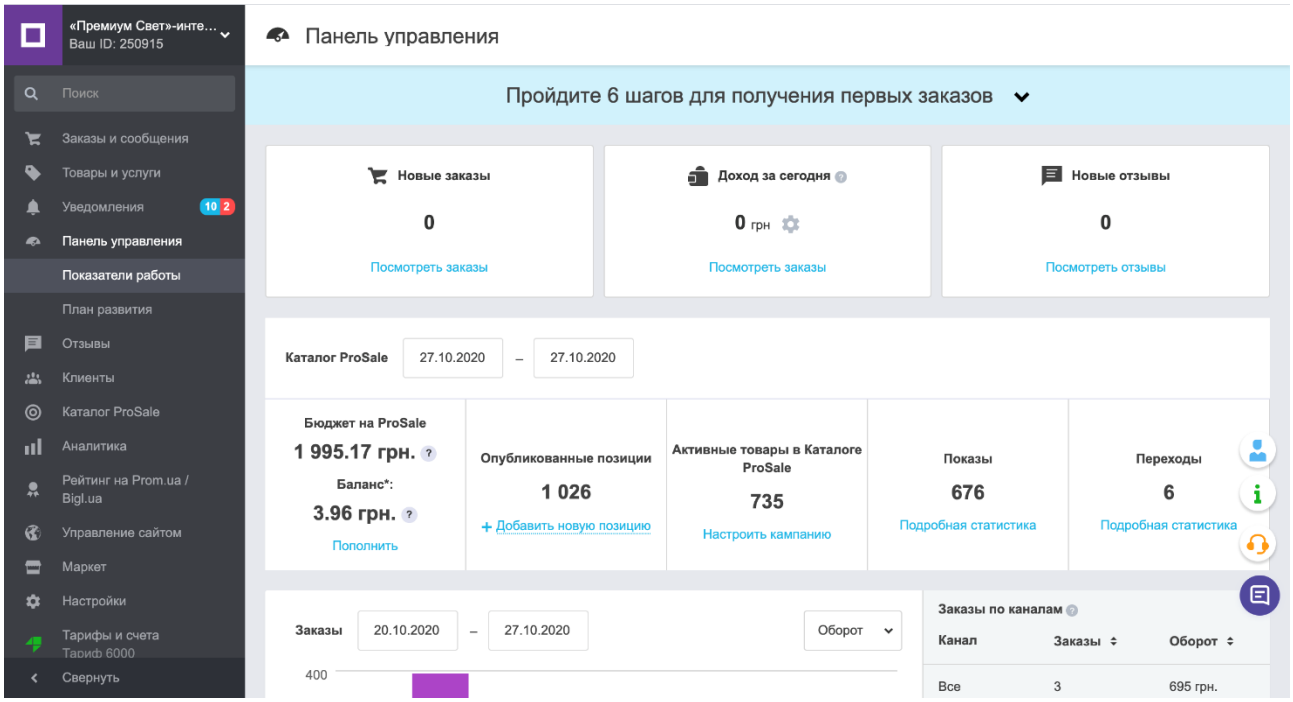


Рисунок 1.6 – Особистий кабінет магазину

В особистому кабінету представлено дуже багато налаштувань, тому спочатку поділимо все на категорії:

- Робота з товарами та замовленнями
- Управління коментарями та статтями
- Налаштування даних та сайту

Розглянемо роботу з товарами та замовленнями. На рисунку 1.7 зображений список замовлень, які надійшли до магазину. До нього входять саме замовлення, загальна вартість замовлення, клієнт, інформація про замовлення (спосіб замовлення, доставки та оплати), а також статус замовлення. Також саме замовлення адміністратор може переглянути натиснувши на його номер. На рисунку 1.8 зображено обране замовлення.

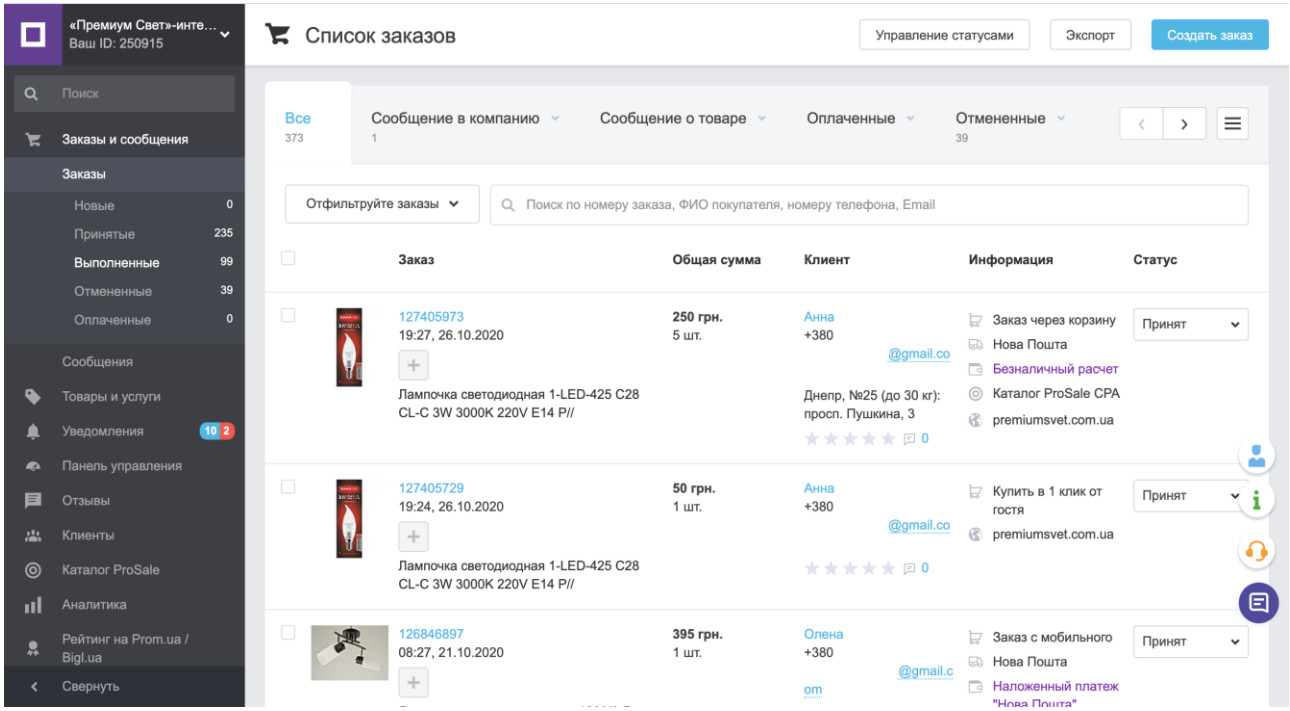


Рисунок 1.7 – Список замовлень

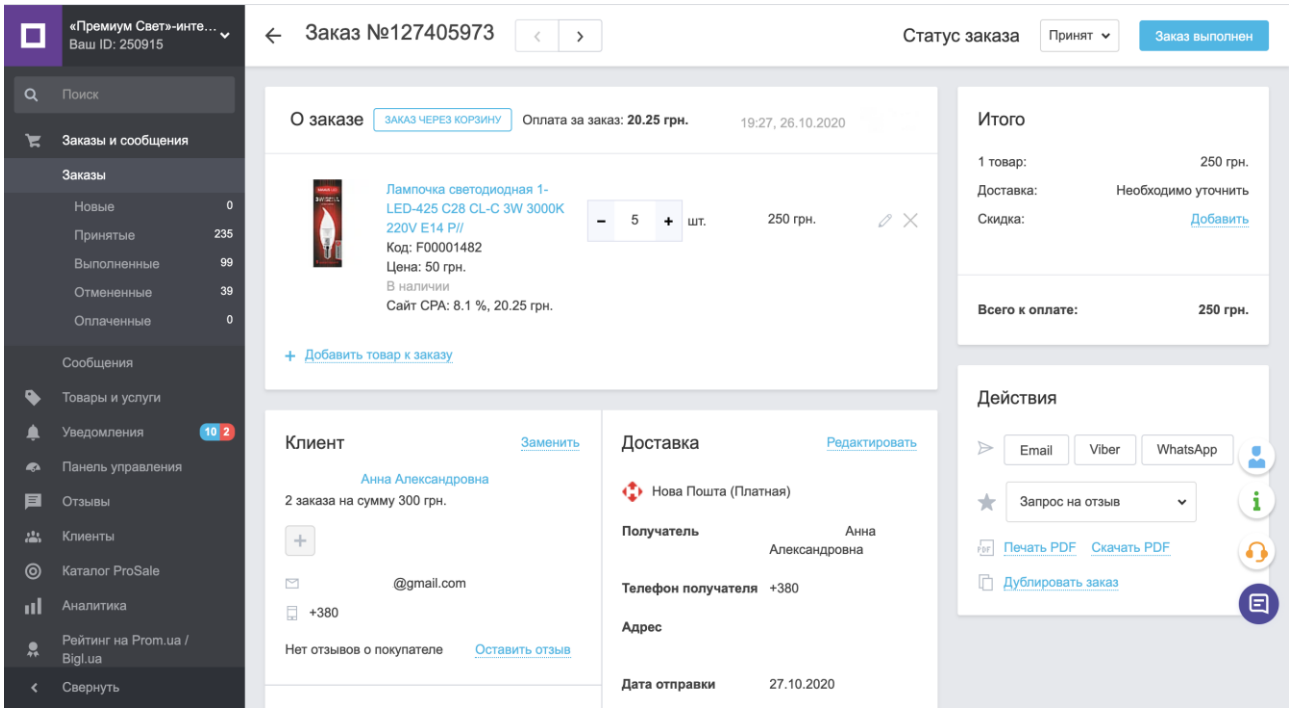


Рисунок 1.8 – Обране замовлення

Для роботи з позиціями, які показуються в магазині необхідно перейти до вкладки «Товари та послуги» і обрати пункт «Позиції». Тут адміністратор може додати нову позицію, або переглянути створені. У списку позицій відображаються назва, дата додавання, код, статус відображення, ціна, кількість замовлень, а також кнопка вибору дій, таких як видалення, а також редагування. На рисунку 1.9 представлений список позицій.

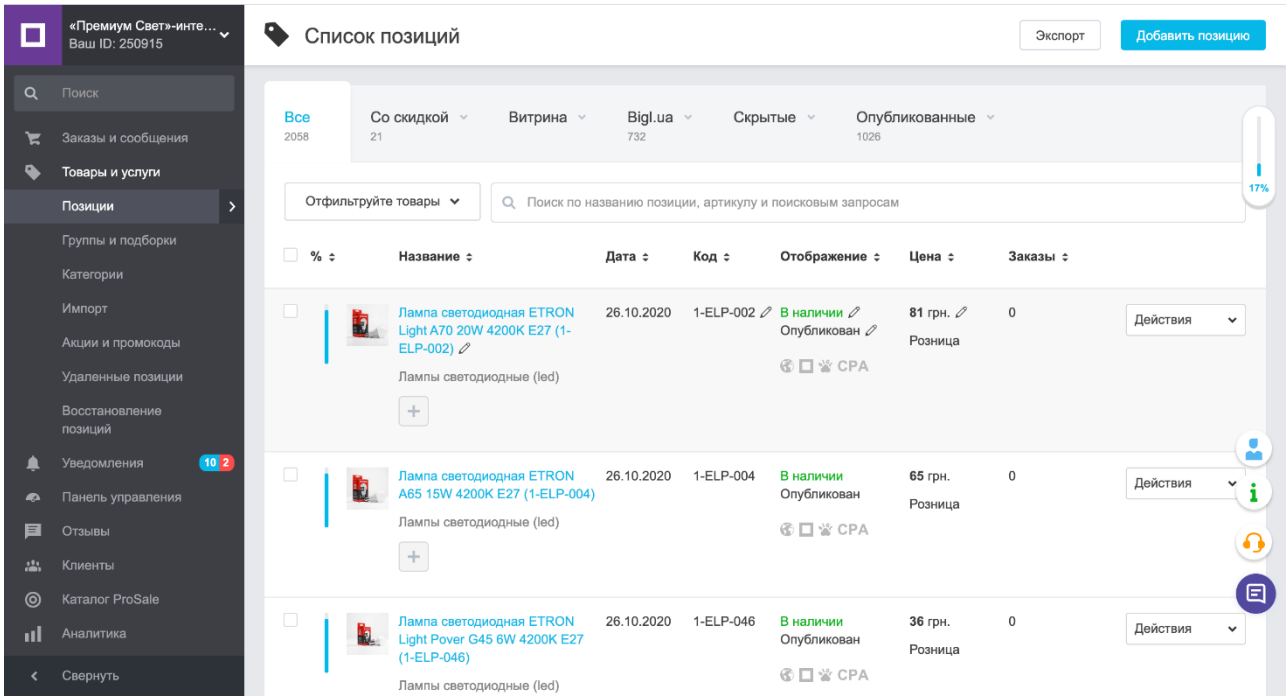


Рисунок 1.9 - Список позицій

Для сортування позицій передбачені групі та добірки товарів. Вони необхідні для спрощення пошуку та підбору необхідного освітлювального прибору покупцю. На рисунку 1.10 представлена сторінка управління групами та добірками товарів.

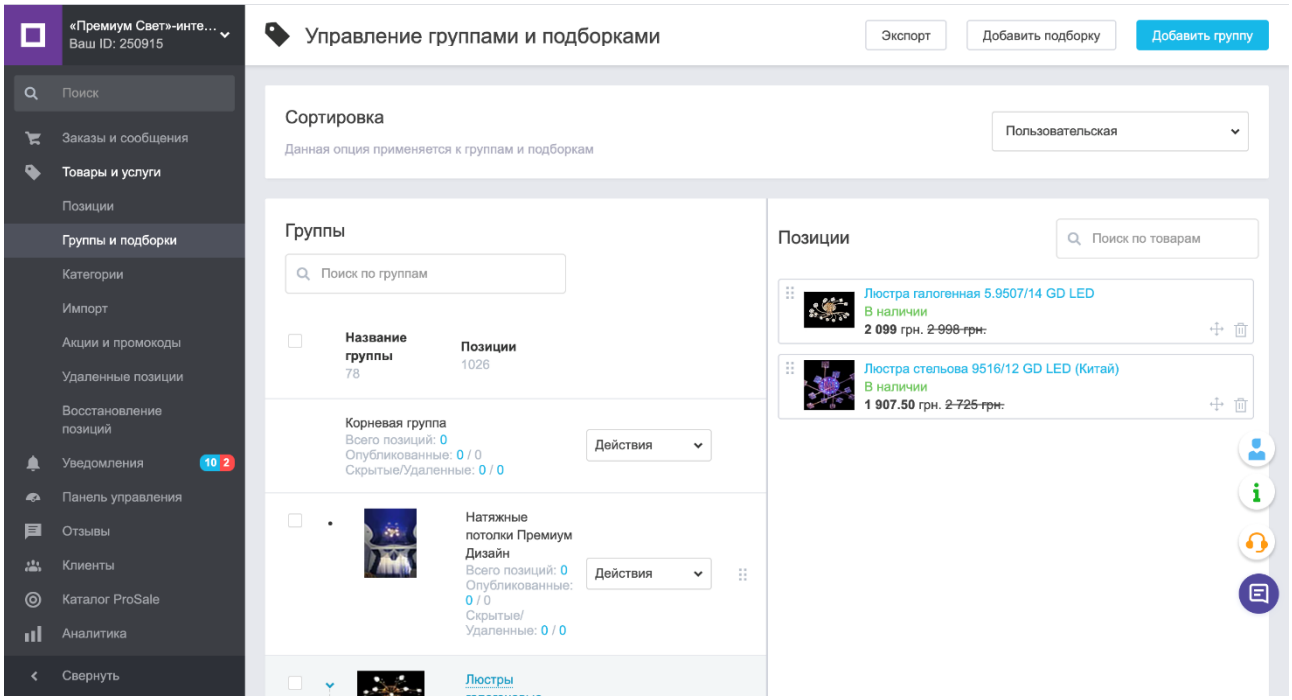


Рисунок 1.10 – Управління групами та добірками

Розглянемо управління відгуками магазину. На рисунку 1.11 зображений список коментарів сайту. На цьому списку є дата додання коментарю, коментар, автор, статус, а також кнопка дії за допомогою якої можна опублікувати новий коментар, або видалити новий, або вже існуючий.

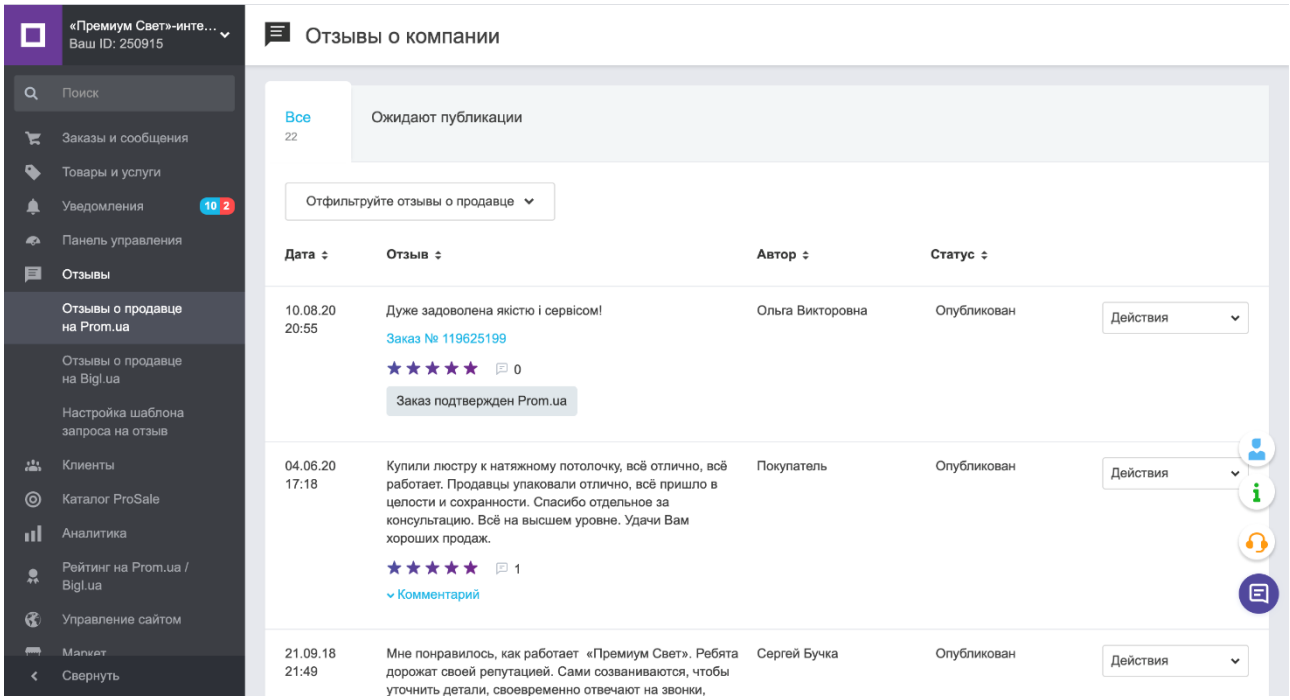


Рисунок 1.11 – Коментарі

Управління статтями сайту дуже схожу на управління коментарями. На рисунку 1.12 представлена сторінка управління статтями. На цій сторінці ви можете переглянути зображення статті, її назву, видимість, папку (якщо ви створювали їх), дату додавання або зміни, а також дві кнопки переходу до редагування статті і її видалення. На рисунку на 1.13 представлено редагування статті.

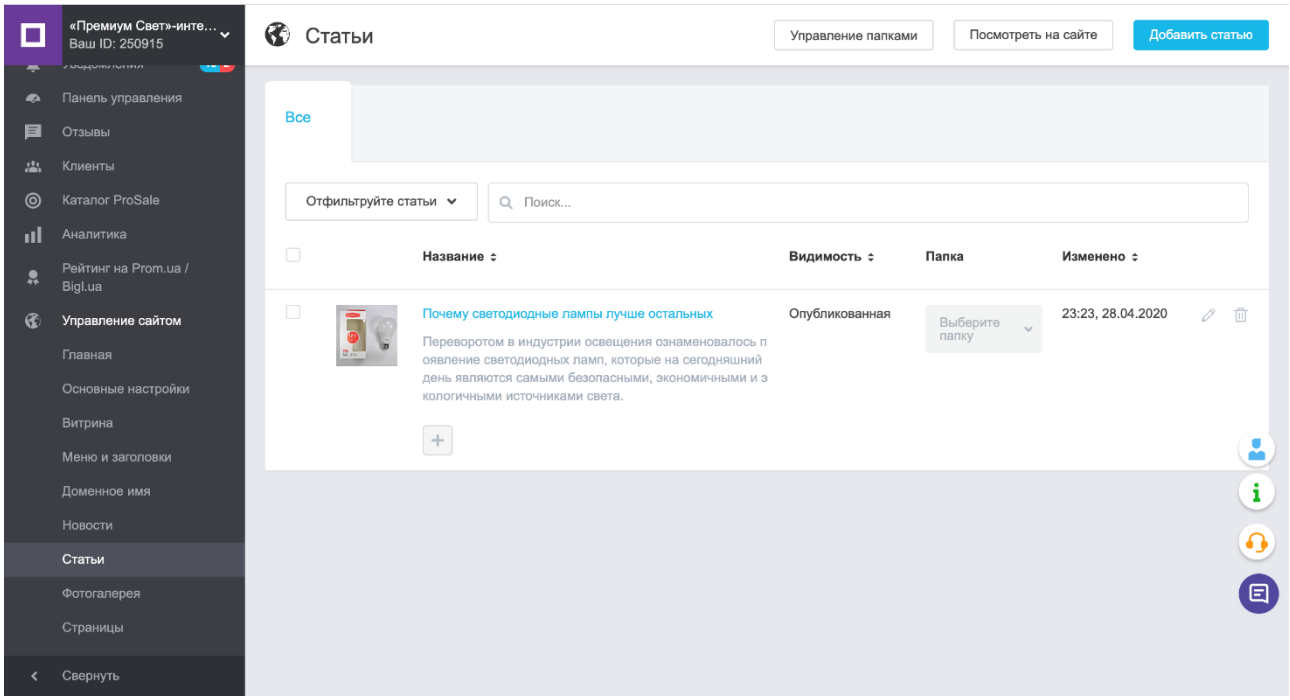


Рисунок 1.12 – Список статей

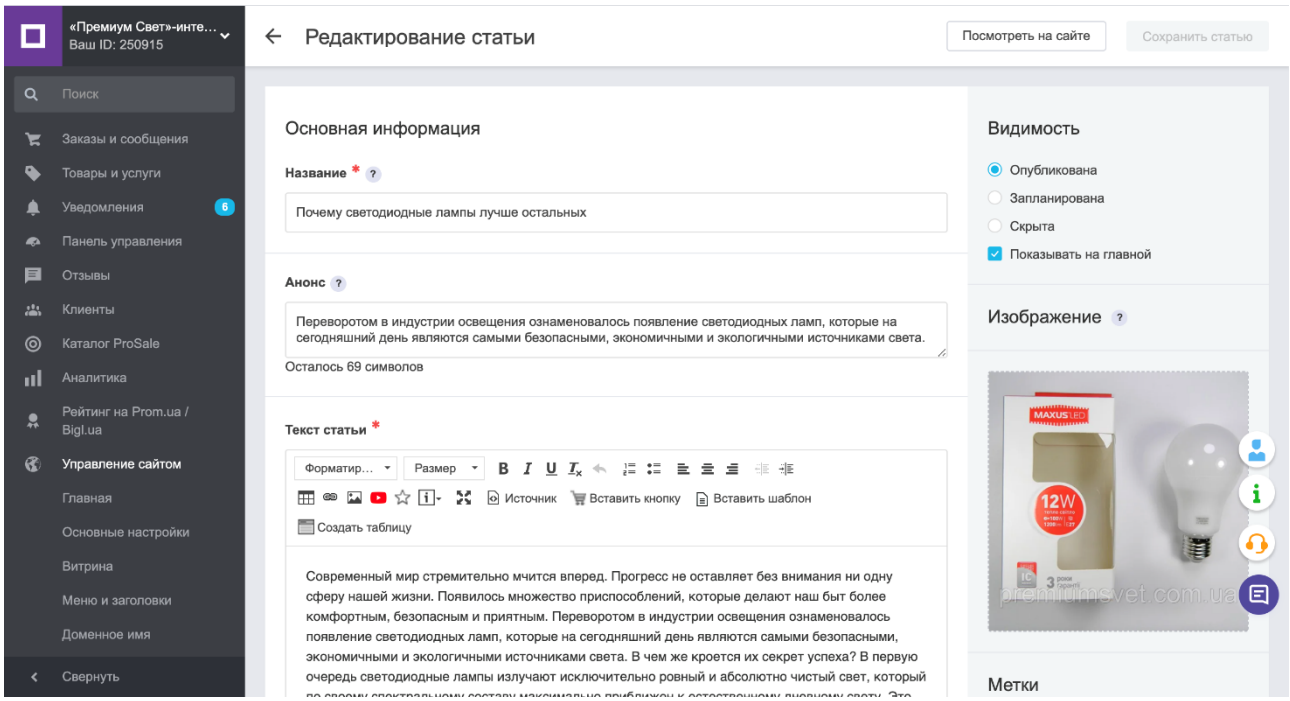


Рисунок 1.13 – Редагування статті

Так як Пром – це конструктор сайтів, тому в ньому присутнє налаштування даних та сайту. Після створення сайту необхідно придбати шаблон для сайту і заповнити всі необхідні дані. Приклади деяких налаштувань зображені на рисунках 1.14-1.16.

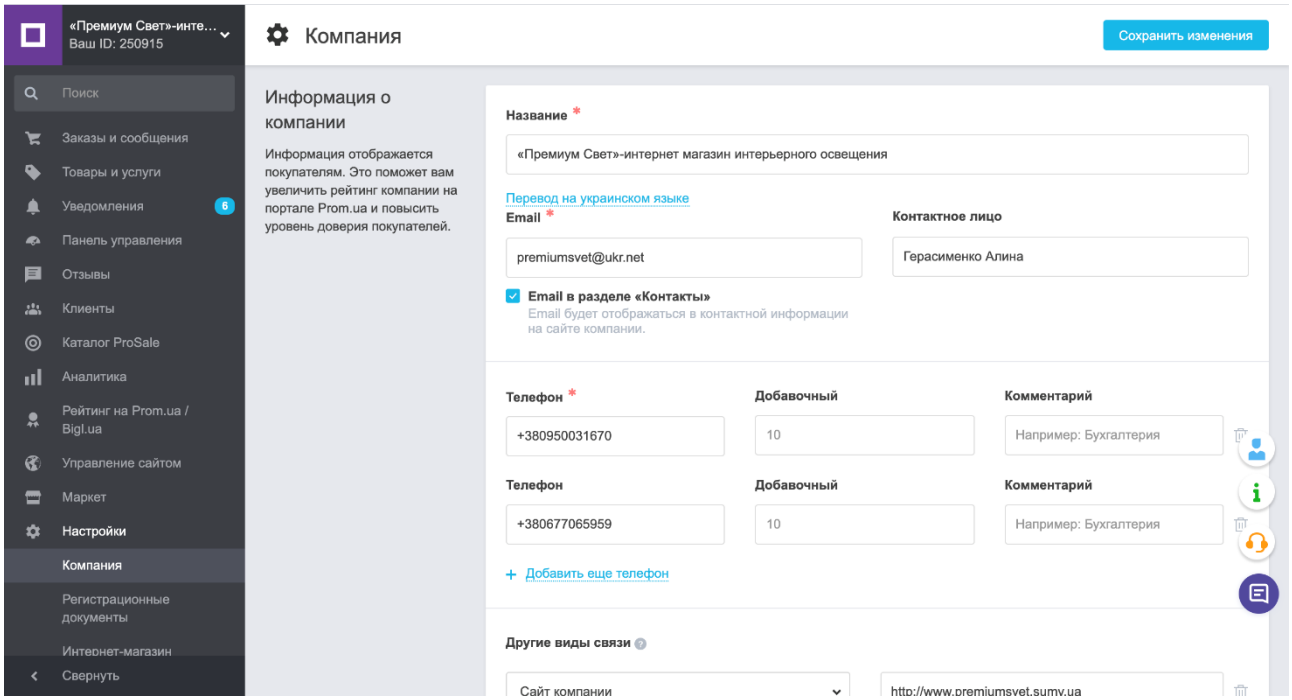


Рисунок 1.14 – Основні налаштування

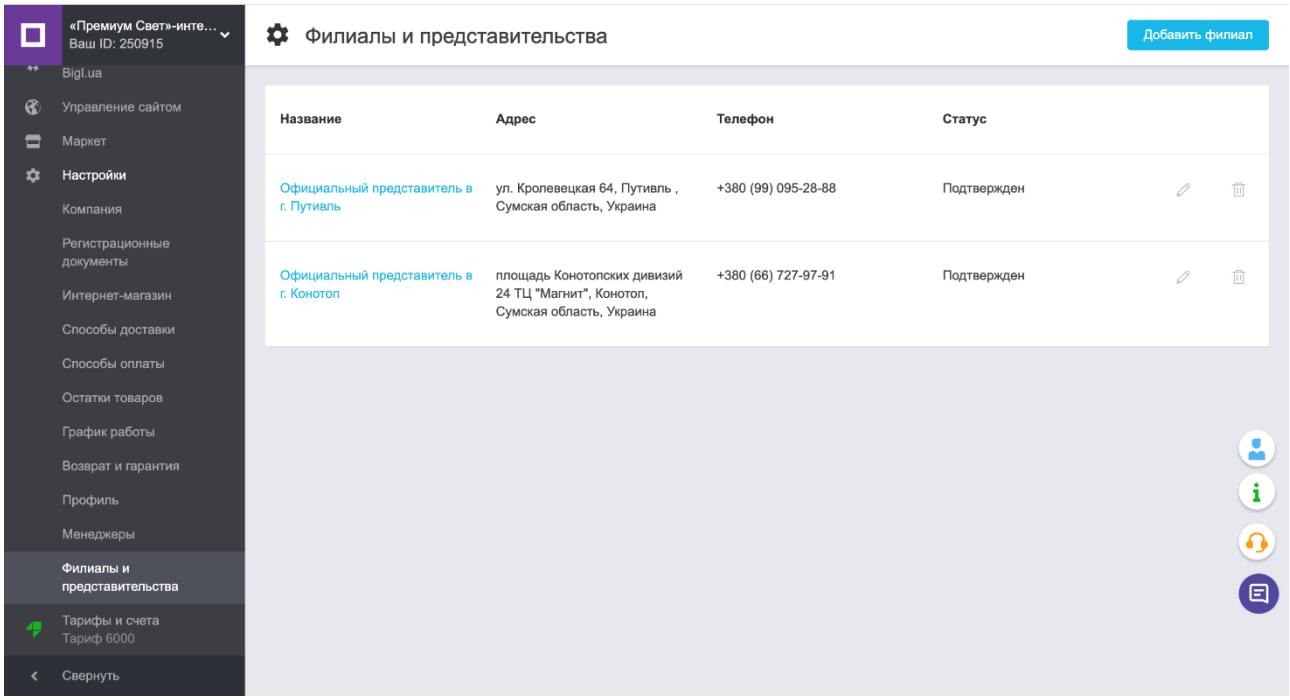


Рисунок 1.15 – Налаштування філіалів

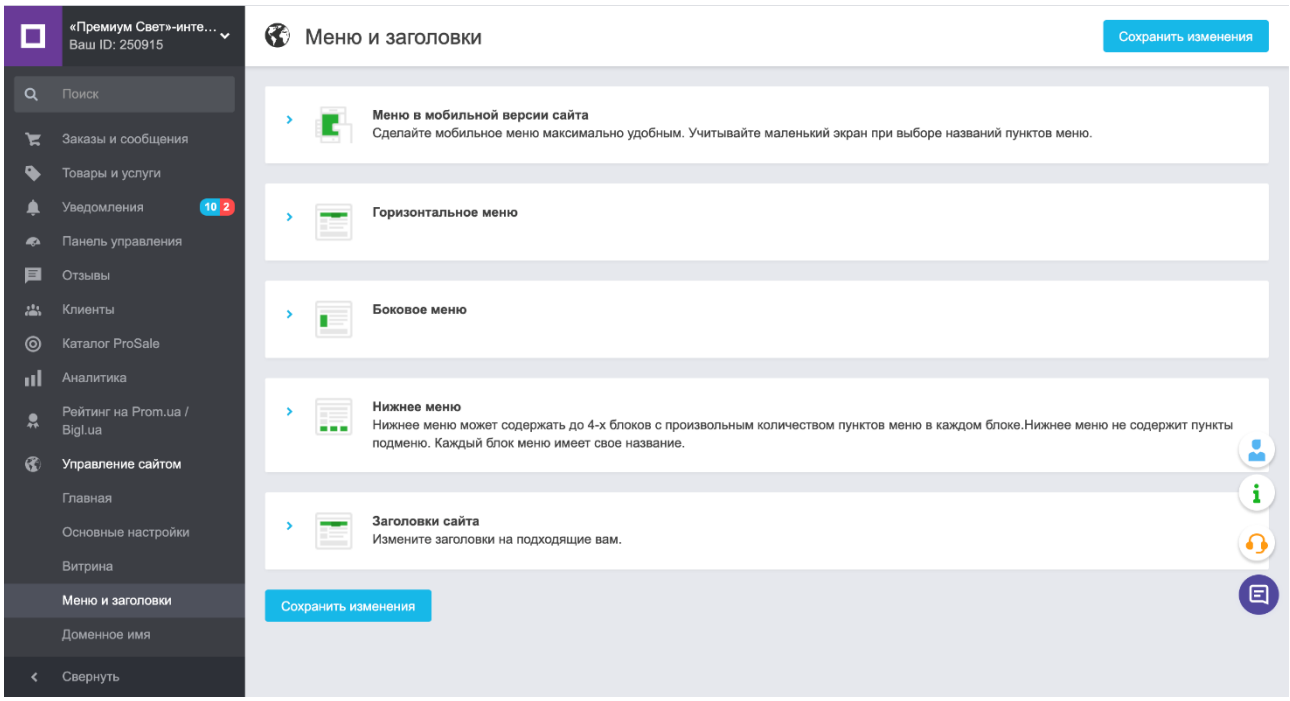


Рисунок 1.16 – Налаштування меню

Після проведеного огляду бачимо, що платформа Пром має дуже багатий функціонал, яким можна вирішити більшість потреб звичайних інтернет магазинів. Зручно виконана робота з позиціями, статтями та коментарями. Також ви можете змінювати шаблон, якщо вас не влаштовує дизайн. Мінусами можна назвати відсутність фільтрації товарів в обраній категорії, відсутність додавання власного функціоналу, наприклад, прив’язку платіжних систем, а також відсутність додавання коментарів до товару.

1.2 Бухгалтерський облік магазину

Згідно зі статистичними даними, у світі існує 100 мільйонів малих та середніх підприємств, що становить 99% від загальної кількості підприємств.[8]

Згідно зі статистикою по Україні малі та середні підприємства є вирішальною силою для забезпечення національного економічного та соціального розвитку та важливою основою для підтримання стабільного та швидкого розвитку національної економіки.[9] Застосування електронної комерції на малих та середніх підприємствах швидко розвивається, і обсяг транзакцій електронної комерції постійно створює нові записи. Електронна комерція стає новим полем боротьби серед підприємств. Однак рівень електронної комерції на малих та середніх підприємствах у нашій країні все ще перебуває на початковій стадії, обмежуючи їх розвиток та розширення.[10]

Бухгалтерія компанії «Преміум Світло» використовує для обліку програму 1С. На рисунку 1.17 зображено приклад всіх даних згрупованих по типу товару. А на рисунку 1.18 зображено одна з таких папок в якій знаходяться люстри по одному виробнику.

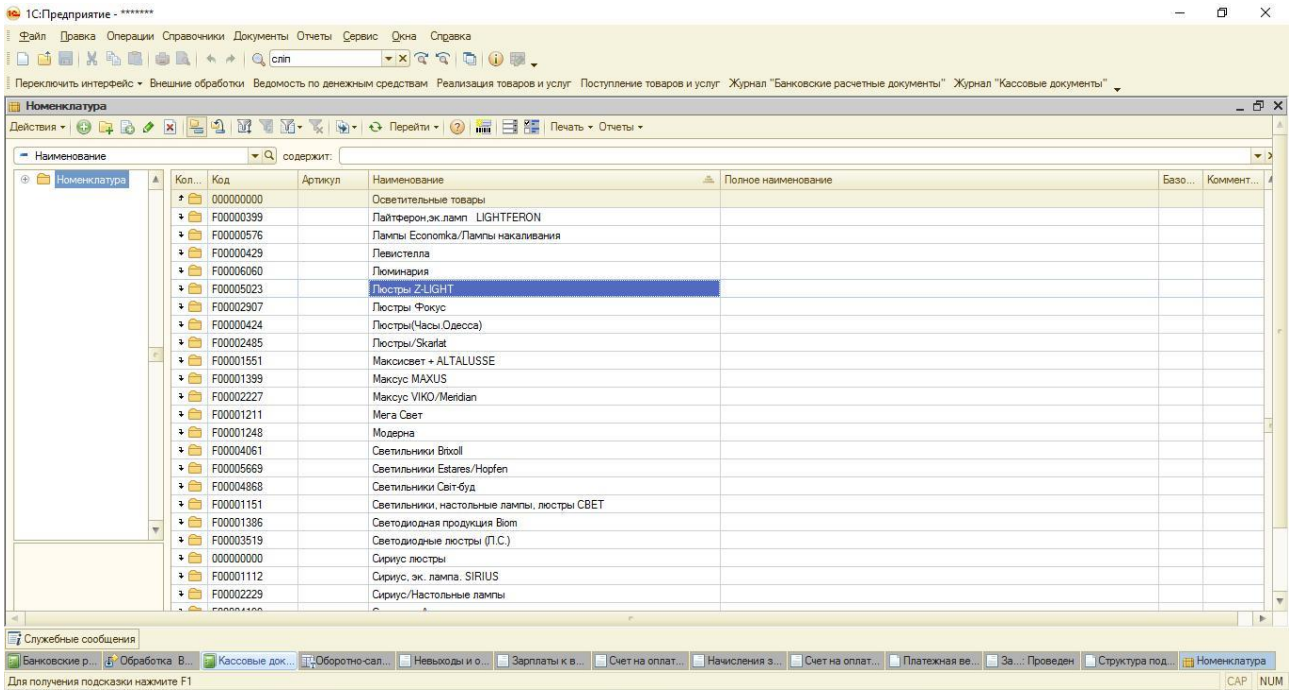


Рисунок 1.17 – Групи товарів

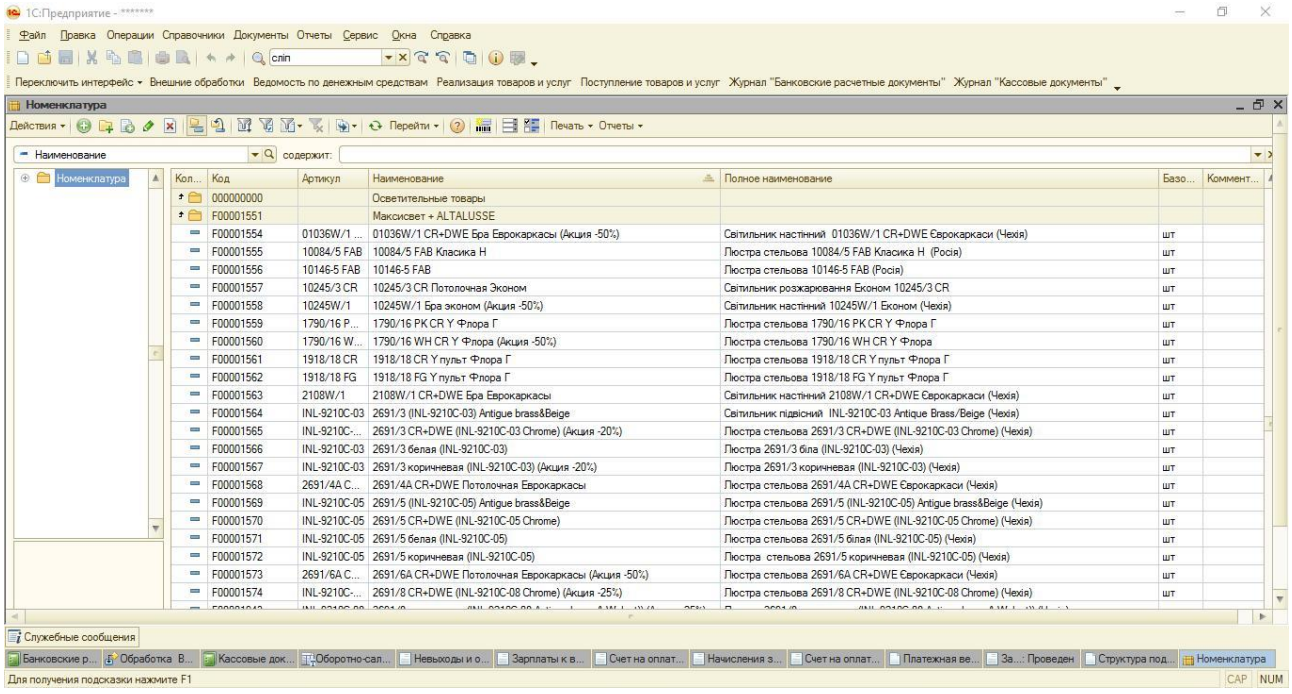


Рисунок 1.18 – Перелік товарів з обраної групи

Завантаження даних проводиться за допомогою модуля для програми 1С від платформи Пром. На рисунку 1.19 зображений інтерфейс цього модулю, де необхідно

обрати потрібні налаштування, а на рисунку 1.20 зображено що саме потрібно завантажити на сайт.

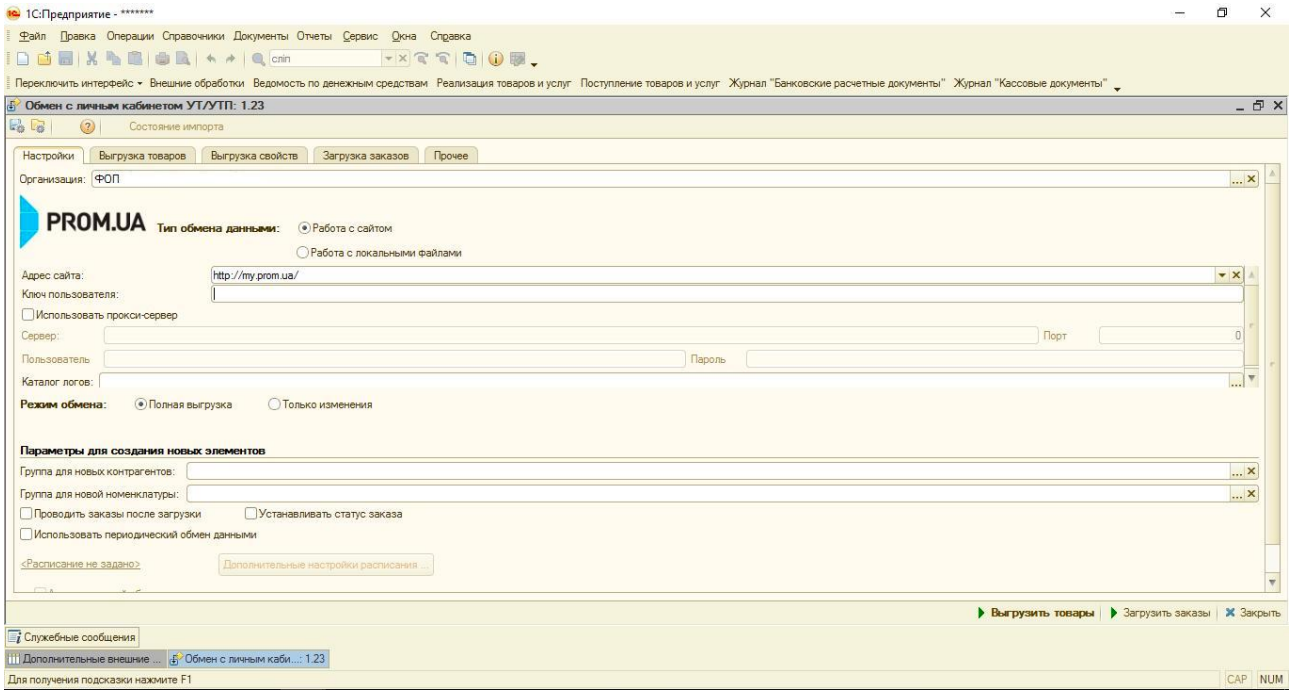


Рисунок 1.19 – Інтерфейс модулю від Пром

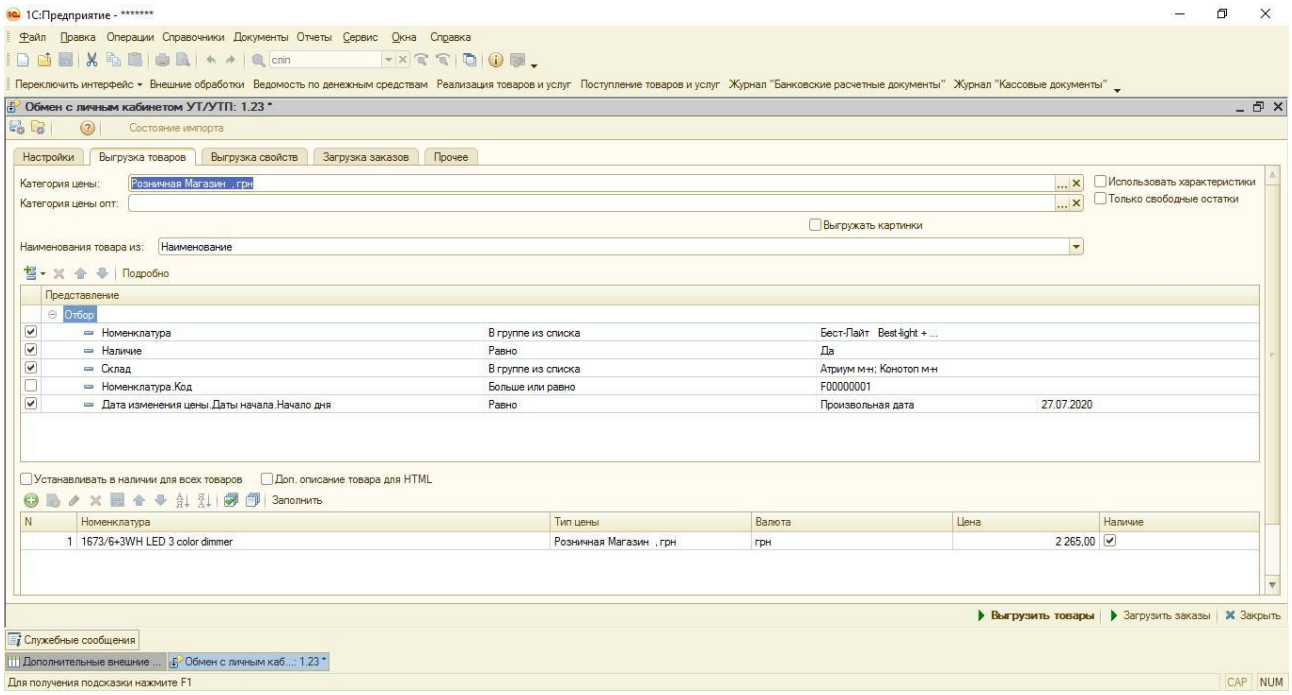


Рисунок 1.20 – Вибір даних для завантаження

Модуль від Прому допомагає дуже швидко завантажити дані на сайт, але більше в ньому не має ніякого функціоналу. Тобто бухгалтеру необхідно вручну в кінці кожного дня проводити списання товарів і формувати звітність.

1.3 Огляд існуючих аналогів

Незабаром електронна комерція становитиме 15% усіх роздрібних продажів у Північній Америці (у Китаї електронна комерція становить 23% від усіх роздрібних продажів), а цифрова впливає майже на 60% всіх роздрібних продажів. Підживлюючись мобільними пристроями, за рік продажі зросли на 55%, електронна комерція в Північній Америці зросла на 16% у 2018 році до понад 500 млрд доларів.[11]

Для огляду існуючих аналогів проведемо пошук аналогічних сайтів, а потім складемо порівняльну таблицю, для опису їх переваг на недоліків.

Перший сайт – це конкурент даної компанії, а саме Lampa.ua. Це інтернет магазин освітлення, головний офіс якого знаходиться у Києві, але доставку товарів він робить по всій Україні.[12] На рисунку 1.21 представлена головна сторінка сайту Lampa.ua.

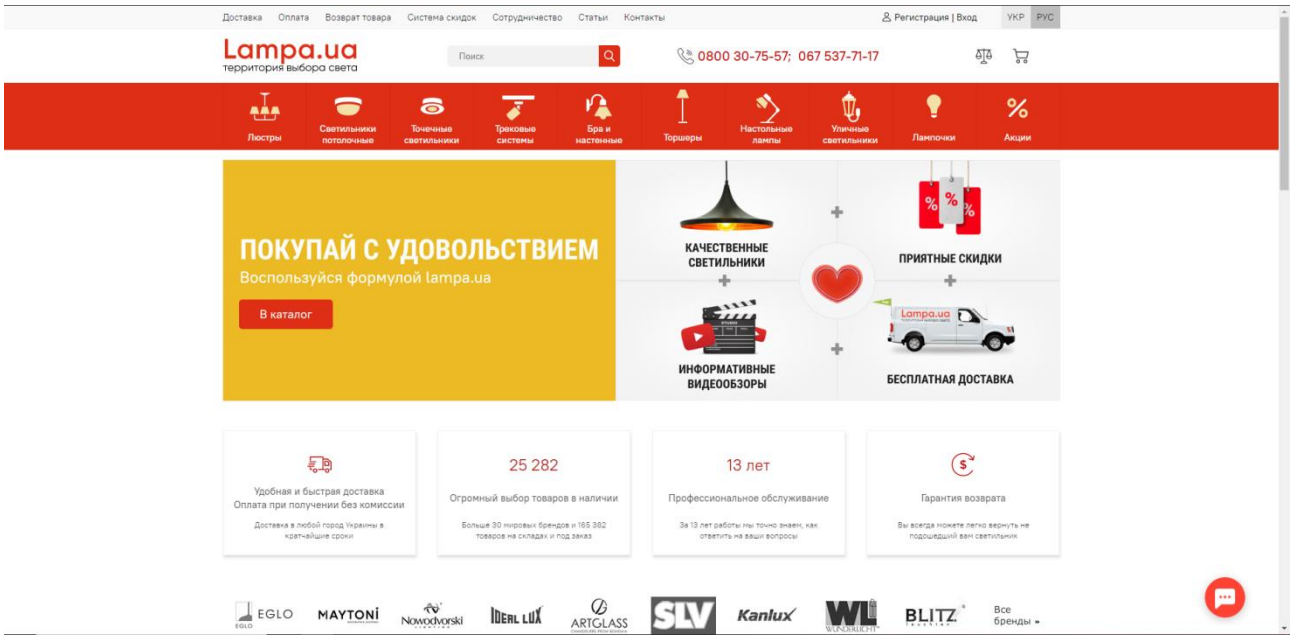


Рисунок 1.21 – Сайт Lampa.ua

Наступний сайт це terra-svet.com.[13] Це інтернет магазин освітлення, що знаходиться у місті Харків. Сайт має лаконічний дизайн, а також свій блог за статтями про освітлення. На рисунку 1.21 зображена головка сторінка сайту terra-svet.

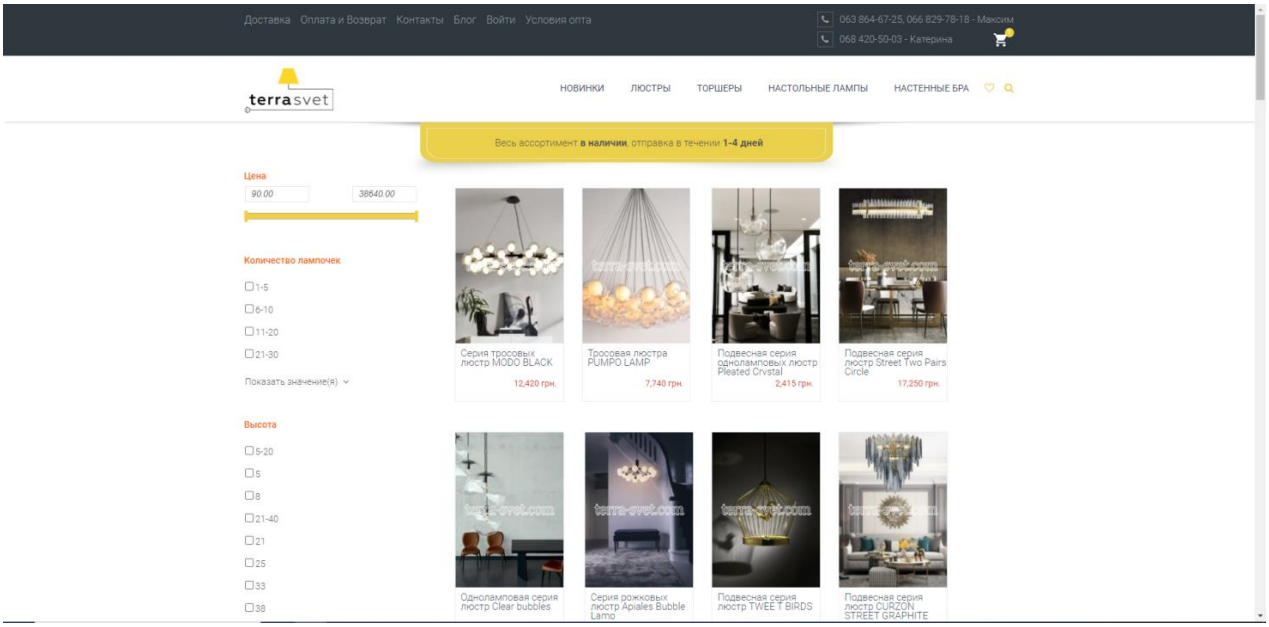


Рисунок 1.22 – Сайт terra-svet.com

Ще один аналог це інтернет магазин «Свет». Це інтернет магазин освітлення в місті Харків, що є офіційним представником найбільшого світлотехнічного підприємства SVET.[14] На рисунку 1.23 зображений сайт магазину «Свет».

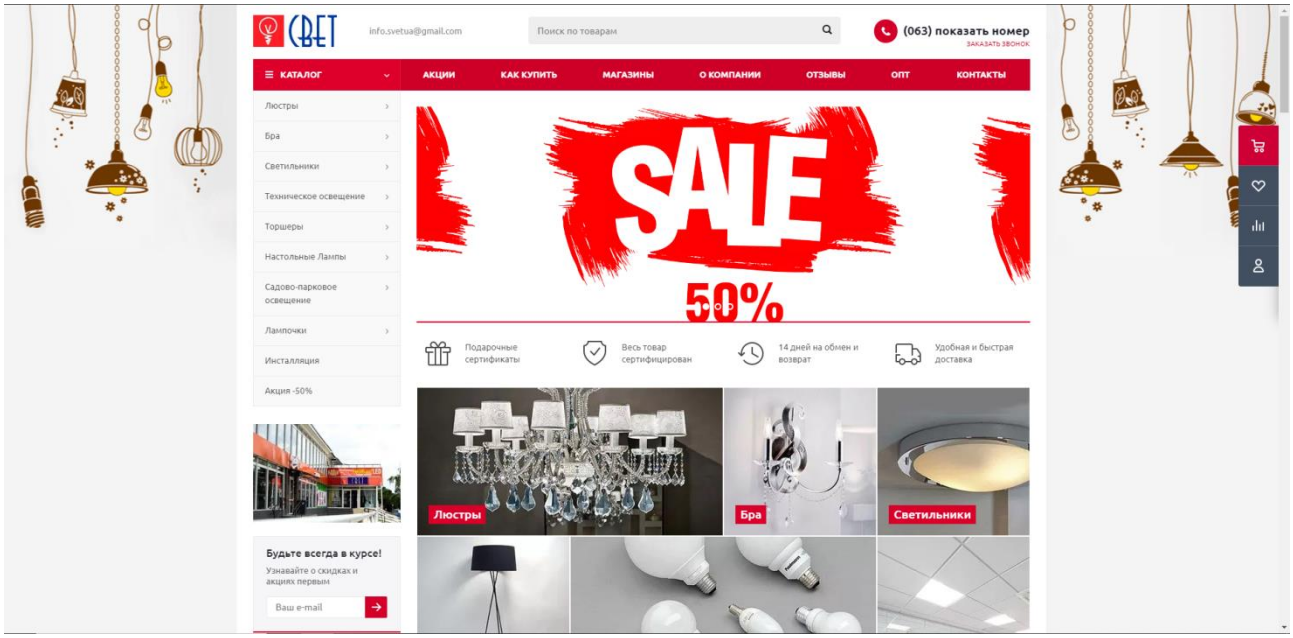


Рисунок 1.23 – Сайт svetua.com.ua

Для проведення аналізу аналогів, що були описані вище, складемо порівняльну таблицю з урахуванням їх переваг та недоліків (табл. 1.1).

Таблиця 1.1 – Огляд існуючих аналогів

Сайт	**Переваги**	**Недоліки**
Lampa.ua	Гарний дизайн Наявність фільтру товарів	Відсутність зворотного дзвінка Відсутність друку звітності

Продовження табл. 1.1 – Огляд існуючих аналогів

Сайт	**Переваги**	**Недоліки**
terra-svet.com	Гарний дизайн Наявність фільтру товарів	Відсутність зворотного дзвінка Відсутність друку звітності
svetua.com.ua	Наявність зворотного дзвінка Наявність фільтру товарів	Застарілий дизайн Відсутність друку звітності

Розглянуті інтернет магазини досить вдало реалізовані і конкурентоспроможні. Кожен з проектів має свій унікальний стиль в дизайні та форму представлення товарів, а також функціонал. Враховуючи описані переваги та недоліки необхідно буде для створюваної інформаційної системи створити гарний дизайн, функціонал зворотного дзвінка, функціонал фільтрування товарів.

2 ПОСТАНОВКА ЗАДАЧІ ТА МЕТОДИ ДОСЛІДЖЕННЯ

2.1 Мета та задачі дослідження

Мета проекту – це створення інформаційної системи підтримки магазину освітлювальних приладів.

Реалізація надасть можливість вести онлайн продажі освітлювальної техніки, формувати звітність за замовленнями, а також проводити аналіз і допомагати формувати плани закупок.

Для досягнення поставленої мети, необхідно дотримуватися виконання таких задач:

- визначити вимоги до розробки;
- обрати необхідні технології та мову програмування для розробки інформаційної системи та її модулів;
- провести планування робіт проекту, скласти календарний план, провести структурно-функціональне моделювання процесів;
- розробити структуру майбутньої системи, визначити які модулі мають забезпечувати її функціонал;
- виконати програмну реалізацію визначених модулів;
- для допомоги формування закупівель на наступний місяць розробити модуль підтримки прийняття рішень;
- виконати реалізацію інформаційної системи для цього: з’єднати між собою всі розроблені програмні модулі та виконати фінальну збірку проекту.

Сайт системи повинен мати лаконічний дизайн для використання на різних типах пристроїв.

Повинно бути два рівня доступу – це покупець та адміністратор. Покупець повинен мати можливість переглядати товари, замовляти їх, виконувати їх фільтрацію та сортування, а також мати особистий кабінет, де він матиме можливість зберігати товари, що сподобалися, зберігати свою адресу для подальших замовлень, а також переглядати свої замовлення. Також покупець може замовити зворотній дзвінок від магазину вказавши своє ім’я та номер телефону.

На сайті має бути передбачене додавання статей та новин, які користувачі зможуть переглядати.

Адміністратор має мати доступ до додавання, редагування, видалення товарів, статей та новин, формуванню необхідної звітності про продані позиції та переглядати залишки у магазині. А також по обраній категорії формувати список рекомендованих до придбання товарів

Підтримка тих, хто приймає рішення, вимагає чіткого розуміння різних елементів, які впливають на результати рішення. Системи підтримки прийняття рішень надавали особам, що приймають рішення, такі уявлення протягом усієї історії використання з різним ступенем успіху. Наявність джерел даних було основним обмеженням того, що можуть робити системи підтримки прийняття рішень.[15] Для допомоги формування закупівель необхідно використати систему підтримки прийняття рішень, яка буде використовувати дані з продажів товарів по обраній категорії.

Для більш детального описання задач створимо мапу сайту - файл або сторінка на сайті, який служить свого роду реєстром контенту (у вигляді посилань на сторінки або медійний контент) і допомагають пошуковим системам краще його індексувати.[16]

Для створення мапи сайту дуже зручним інструментом є FlowMapp (повна стекова платформа UX [17]), де вже міститься готовий шаблон для електронної комерції, який необхідно трішки змінити під конкретні потреби.

В результаті ми отримуємо мапу сайту, яка представлена на рисунку 1.24. А на рисунку 1.25 зображено у вигляді дерева.

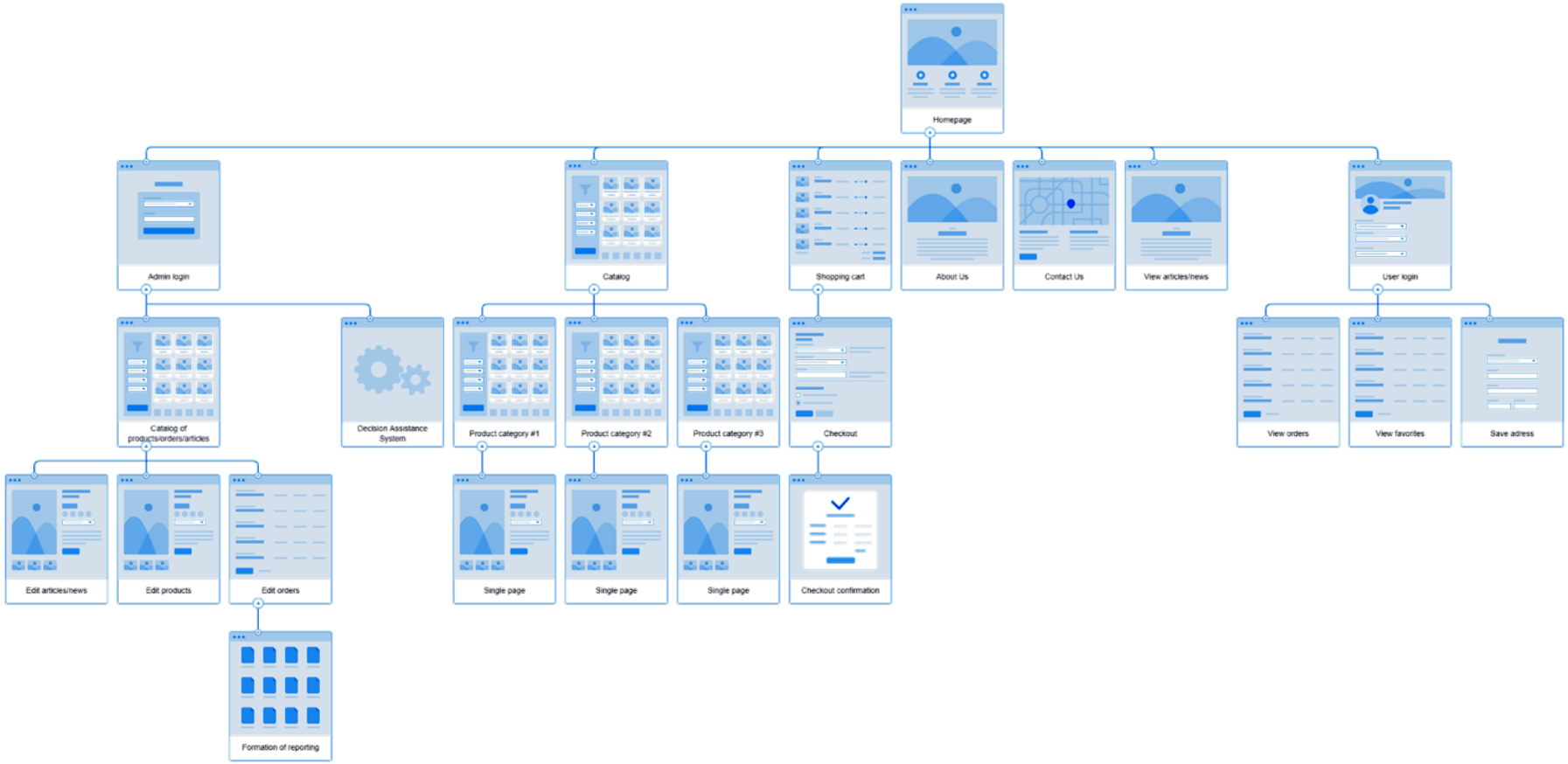


Рисунок 1.24 – Мапа сайту

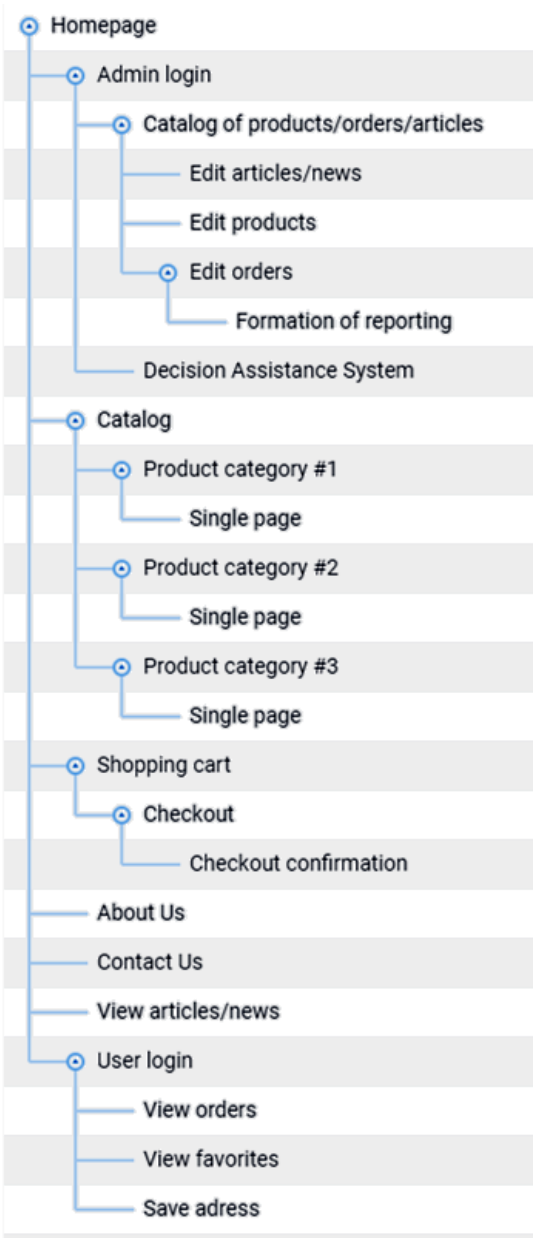


Рисунок 1.25 – Дерево мапи сайту

2.2 Методи дослідження та засоби реалізації

2.2.1 Вибір засобів реалізації

Вибір системи керування базою даних. На даний час існують два основних типи баз даних. Це реляційні бази даних, та не реляційні або документ орієнтовані бази даних.

Реляційні бази даних використовують відношення, які зазвичай називають таблицями, для зберігання даних, а потім зберігають ці дані за допомогою загальних характеристик у наборі даних.

Документ орієнтована база даних базується на документах і відноситься до високопродуктивних нереляційних баз даних, що використовують широкий спектр моделей даних. Ці бази даних широко відомі завдяки простоті у використанні, масштабованій роботі, високій стійкості та широкій доступності.

Однією з найвідоміших систем керування реляційними базами даних є MySQL, а з документ орієнтованих – це MongoDB. Проведемо їх порівняння.

MySQL - це реляційна база даних з відкритим кодом, яка зберігає дані у «таблицях» і використовує структуровану мову запитів (SQL) для доступу до бази даних. У MySQL ми заздалегідь визначаємо схему нашої бази даних, виходячи з наших вимог, і встановлюємо правила для управління взаємозв'язками між полями в наших таблицях.[18]

База даних MongoDB складається з набору баз даних, в яких кожна база даних містить кілька колекцій. Оскільки MongoDB працює з динамічними схемами, кожна колекція може містити різні типи об’єктів. Кожен об'єкт, який також називається документом, представлений у вигляді структури JSON: список пар ключ-значення.[19]

Отже, можна зробити висовок, що використання СКБД MySQL є більш вигідним через наявність структурованої мови запитів, а також через заздалегідь визначеної бази даних з описаними необхідними зв’язками між таблицями.

Вибір засобів розробки серверної частини. Сьогодні для програмування серверної частини веб-додатків найчастіше використовують дві системи – це PHP та Node.js.

PHP та Node.js - це потужні системи для динамічних веб-сайтів. Вони обидві підпадають під одну категорію, проте їх особливості досить чітко виражені. Немає сумнівів - PHP - це найвідоміша та найпоширеніша мова для сценаріїв на стороні сервера. Однак Node.js дав можливість використовувати програмування JavaScript на стороні сервера, коли він був представлений в 2009 році, сприяючи зростанню сайтів із повністю створеними JavaScript стеками як для інтерфейсного, так і для внутрішнього інтерфейсу.[20]

Саме через можливість використання мови JavaScript як однієї основної мови програмування для всієї інформаційної системи засобом реалізації серверної частини буде використано Node.js.

Вибір засобів реалізації клієнтської частини. Так як на серверній частині було обрано використання Node.js, тому доцільно буде використовувати фреймворк, або бібліотеку яка розроблена під JavaScript. На даний час більша частина клієнтських додатків пишеться за допомогою бібліотеки React, або за допомогою фреймворка Angular.

AngularJS – це платформа JavaScript MVC, створена Google для створення належної архітектури та підтримуваного веб-додатку. AngularJS побудований навколо філософії, згідно з якою декларативний код кращий за імперативний код під час створення інтерфейсів та з'єднання різних компонентів веб-програм разом.[21]

ReactJS – це бібліотека інтерфейсу користувача, розроблена у Facebook для спрощення створення інтерактивних компонентів інтерфейсу, що містять статус та багаторазове використання. Він використовується у Facebook у виробництві. ReactJS

найкраще підходить для відтворення складних користувальницьких інтерфейсів з високою продуктивністю.[22]

Через компонентний підхід та високу продуктивність кращим вибором буде React. Також ще одним плюсом є те, що React на відміну від Angular це бібліотека, тому не має чіткої структури та дозволяє використовувати модулі, які забажає розробник. Також Angular потребую використання TypeScript, що ускладнюю розробку.

2.2.2 Вибір методів для формування списку закупівель товарів

Прийняття рішень в умовах невизначеності, як і в умовах ризику, вимагає визначення альтернативних дій, яким відповідають платежі, які залежать від випадкових станів природи.

В даному випадку буде всього два стани природи – це кількість товарів, що залишилися на складі, а також кількість замовлень цих товарів. Альтернативами будуть виступати саме товари. Платежами будуть виступати кількість товару на складі та кількість проданих товарів. Список найкращих альтернатив буде представлений робітнику магазину як рекомендовані до замовлення товари.

Для представленої задачі матрицю платежів можна представити наступним чином:

$$\begin{array}{ccc} & s_1 & s_2 \\ a_1 & v(s_1, a_1) & v(s_2, a_1) \\ a_2 & v(s_1, a_2) & v(s_2, a_2) \\ \dots & \dots & \dots \\ a_m & v(s_1, a_m) & v(s_1, a_m) \end{array} \qquad (2.1)$$

де s_1 – це кількість товару, що залишилася на складі;

s_2 – це кількість замовлень даного товару;

a_m – це m-тий товар;

$v(s_j, a_m)$ - це j-тий платіж по кожному товару.

Для прийняття рішень були обрані декілька критеріїв. Перший і найпростіший це критерій Вальда, або максимінний критерій. Максимінний критерій заснований на консервативному обережному поводженні особи, що приймає рішення і зводиться до вибору найкращої альтернативи з найгірших.[23] Якщо величина v(ai, sj) являє одержуваний прибуток, то відповідно до максимінного критерія вибирається рішення, що забезпечує:

$$\max_{a_i}\left\{\min_{s_j} v(a_i, s_j)\right\} \qquad (2.2)$$

Наступним є критерій Байеса. Значення критерію Байеса - це найбільше значення математичного очікування виграшу.[24] Для обчислення критерію Байеса використовується наступна формула:

$$\max_{a_i}\left\{\sum_{i=1}^{n} p_j v(s_j, a_i)\right\} \qquad (2.3)$$

де m - число стратегій;

n - число станів природи;

$v(s_j, a_i)$ - виграш при і-ої стратегії при j-му стані природи;

p_j - імовірність j-ого стану природи. В даному випадку для кількості товару, що залилися на складі p_j було обрано 0.3, а для кількості замовлень – 0.7.

Останній критерій – це критерій Гурвіца. Критерій Гурвіца є критерієм песимізму - оптимізму.[25] За оптимальну приймається та стратегія, для якої виконується співвідношення:

$$\max_{a_i}\left\{(1-y)\min_{s_j} v(a_i, s_j) + 1\max_{s_j} v(a_i, s_j)\right\} \qquad (2.4)$$

де у – ступінь оптимізму, а всі інші величини взяті з матриці платежів (2.1)

При у = 1 отримаємо критерій Вальді, при у = 0 отримаємо - оптимістичний критерій (максимакс). Критерій Гурвіца враховує можливість як найгіршого, так і найкращого для людини поведінки природи. Як вибирається критерій у? Чим гірше наслідки помилкових рішень, тим більше бажання застрахуватися від помилок, тим у ближче до 1. Для нашої задачі, щоб більше застрахуватися від помилок ступінь оптимізму буде рівна 0.7.

На основі вище описаних критеріїв представлено алгоритм роботи системи формування закупівель за обраною категорією:

- отримати з бази даних всі продукти за обраною категорією та кількість їх продажів, а також залишок на складі;
- сформувати масив вхідних даних з трьох параметрів: id товару, залишок на складі та кількість продажів;
- розрахувати оптимальні для закупівлі товари кожним з методів;
- додати найчастіше обраний по трьом методам товар в результуючий масив;
- якщо ще залишилися товари, які не були обрані, то видалити вже додані до нового масиву товари і почати алгоритм знову, доки не закінчяться товари.

3 МОДЕЛЮВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Структурно-функціональне моделювання

При розробці інформаційної системи першим етапом іде побудова структурно-функціональної моделі основного процесу її діяльності - це процес придбання освітлювальної техніки через онлайн магазин. Моделювання виконується згідно нотації IDEF0. Формат побудови моделі - «to – be».

Придбання освітлювальної техніки в онлайн магазині починається з вимог замовника до необхідного товару. Замовник керуючись правилами роботи сайту та формою оформлення товару може зробити замовлення. В результаті ми отримуємо оброблене замовлення, а також повідомлення адміністратору сайту. Контекстна діаграма структурно-функціональної моделі зображена на рисунку 3.1. Ця діаграма описує модель процесу придбання освітлювальної техніки покупцем в онлайн магазині.

Наступним етапом згідно методології проводиться декомпозиція контекстної діаграми та діаграм нижчих рівнів.

«Придбання освітлювальної техніки через онлайн магазин» декомпозується на чотири процеси - підбір необхідних товарів, додавання товарів у кошик, оформлення замовлення, обробка замовлення. Діаграма декомпозиції першого рівня наведена у додатку В.

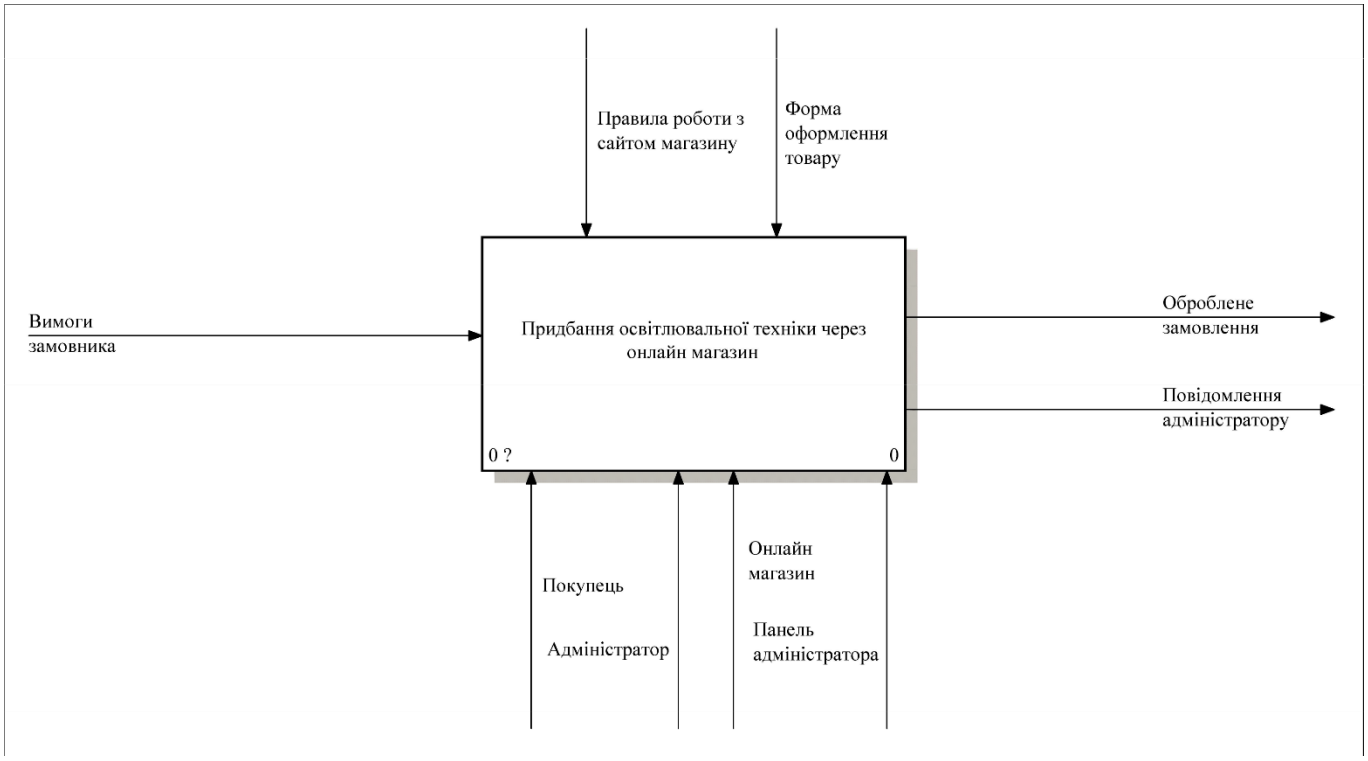


Рисунок 3.1 – Контекстна діаграма

Блок «Підбір необхідних товарів» на вході має вимоги замовника. На виході отримуємо підібрані товари. Управління процесом здійснюється за допомогою правил роботи з сайтом магазину. Механізмами процесу виступають покупець та онлайн магазин. Цей блок декомпозується на чотири процеси – це вибір товару за категорією, перегляд характеристик товару, перегляд фото товару.

Блок «Вибір товару за категорією» на вході має вимоги замовника. На виході отримуємо обраний за категорією товар. Управління процесом здійснюється за допомогою правил роботи з сайтом магазину. Механізмом виступає покупець, а також онлайн магазин.

Блок «Перегляд характеристик товару» на вході отримує обраний за категорією товар. На виході маємо підібрані товари. Управління здійснюється за допомогою правил роботи з сайтом магазину. Механізмом процесу виступають покупець, а також онлайн-магазин.

Блок «Перегляд фото товару» на вході отримує обраний за категорією товар. На виході має підібрані товари. Механізмами виступають покупець, а також онлайн-магазин. Управління здійснюється за допомогою правил роботи з сайтом.

Діаграма декомпозиції другого рівня зображена у додатку В.

Наступним блоком у декомпозиції першого рівня є «Додавання товарів у кошик». На вході даного блоку маємо підібрані товари. Управління здійснюється правилами роботи з сайтом магазину. На виході ми отримуємо сформований список товарів. Механізмом процесу є покупець та онлайн магазин.

Блок «Оформлення замовлення» на вході має список сформованих товарів. На виході сформоване замовлення. Управління процесом здійснюється за допомогою правил роботи з сайтом магазину, а також формою оформлення товару. Механізмом процесу є покупець та онлайн магазин. Процес оформлення замовлення декомпозується на три процеси – це додавання персональних даних, вибір способу оплати, вибір способу доставки.

Блок «Додавання персональних даних» на вході має сформований список товарів. На виході отримуємо персональні дані покупця. Управління процесом здійснюється за допомогою правил роботи з сайтом магазину, а також формою оформлення товару. Механізмами процесу виступають покупець, а також онлайн магазин.

Блок «Вибір способу оплати» на вході має персональні дані. На виході спосіб оплати. Управління процесом здійснюється за допомогою правил роботи з сайтом магазину, а також формою оформлення товару. Механізмом процесу є покупець та онлайн магазин.

Блок «Вибір способу доставки» на вході має спосіб оплати. На виході сформоване замовлення. Управління процесом здійснюється за допомогою правил роботи з сайтом магазину, а також формою оформлення товару. Механізмом процесу є покупець та онлайн магазин.

Останнім блоком на декомпозиції першого рівня є «Обробка замовлення». На вході цей процес має сформоване замовлення, а на виході оброблене замовлення, а також повідомлення адміністратору. Управління здійснюють правила роботи з сайтом магазину. Механізмами процесу виступають адміністратор, а також панель адміністратора.

Схема отриманої функціональної моделі зображена на рисунку 3.2.

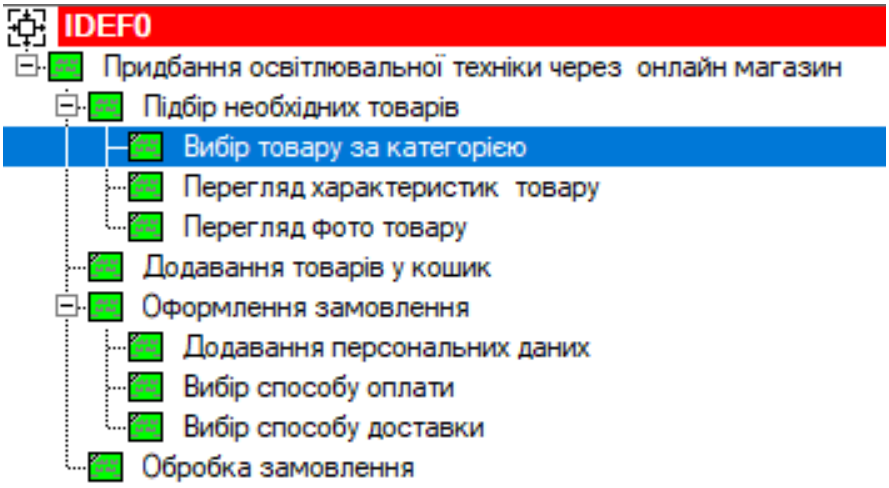


Рисунок 3.2 – Схема функціональної моделі

3.2 Проектування моделі бази даних

Для створення бази даних програмного продукту були визначені наступні сутності:

- «продукт» з характеристиками: назва, опис, код, ціна, акційна ціна, наявність, кількість на складі, зображення та категорія;

- «замовлення» з характеристиками: код замовлення, тип оплати, тип доставки, сума замовлення, коментар до замовлення, статус замовлення;
- «замовник», до якої входять ім’я, прізвище, контактний телефон, електронна пошта, пароль до кабінету (якщо замовник зареєстрований), адреса, місто, регіон, країні, а також опціональні дані, такі як ідентифікаційний код, друга адреса, індекс адреси та підписка на розсилку новин;
- «статті або новини», яка буде мати назву, текст статті, мета описання, аліас, картинку, тип (стаття/новина) та дату публікації;
- «користувач», необхідна для роботи панелі адміністратора. Зберігає дані адміністраторів такі як ім’я, прізвище, контактний телефон, електронна пошта та пароль.

Проведемо нормалізацію бази даних. База даних у першій нормальній формі знаходиться на рисунку 3.3.

Для нормалізації до наступних нормальних форм необхідно буде додати додаткові таблиці для уникнення збитковості даних. Третя нормальна форма зображена на рисунку 3.4.

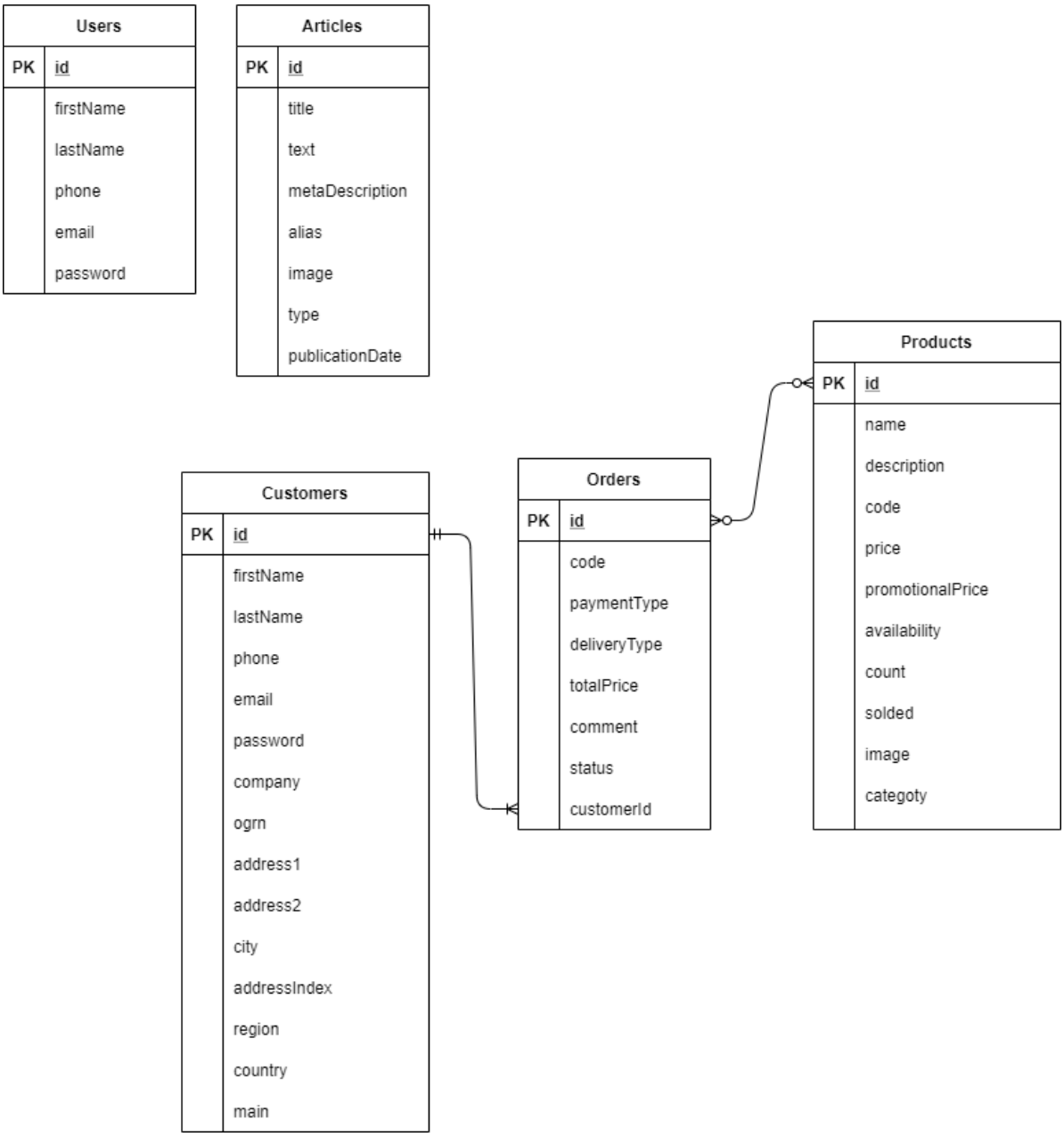


Рисунок 3.3 – Перша нормальна форма

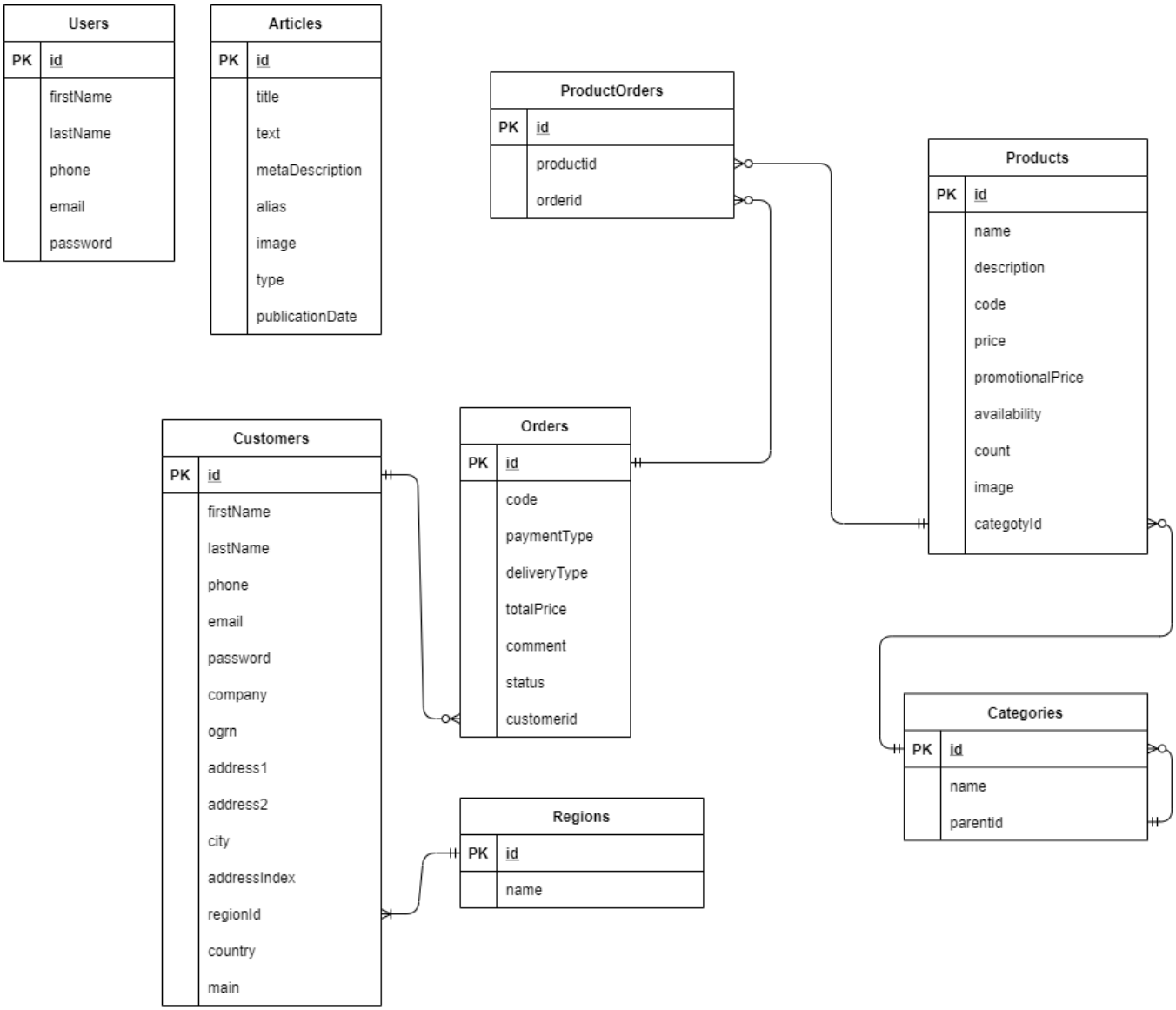


Рисунок 3.4 – Третя нормальна форма

У третій нормальні формі сутностями є customer (замовник), orders (замовлення), regions (регіон), users (користувач панелі адміністратора), articles (стаття), productOrders (товари, що входять до замовлення), products (товари), categories (категорії товарів).

Сутність Customer має наступні атрибути: firstName (ім’я), lastName (прізвище), phone (номер мобільного телефону), email (електронна пошта), password (пароль), company (назва компанії), ogrn (ідентифікаційний код, або інший необхідний код замовлення), adress1 (основна адреса), adress2 (другорядна адреса), city (місто),

addressIndex (індекс міста), regionId (унікальний індекс регіону), country (країна), main (поле, де користувач зазначає чи вказана адреса є основною).

Сутність Regions має один атрибут – це поле name (назва регіону).

Сутність Orders має такі атрибути: code (код замовлення), paymentType (тип оплати), deliveryType (спосіб доставки), totalPrice (сума замовлення), comment (коментар до замовлення), status (статус замовлення), customerId (унікальний індекс замовника).

Сутність ProductOrders у якості атрибутів має productId (унікальний індекс продукту), orderId (унікальний індекс замовлення).

Сутність Products має атрибути name (ім’я продукту), description (характеристики продукту), code (код продукту), price (ціна), promotionalPrice (акційна ціна), availability (наявність), count (кількість товарів на складі), image (шлях до файлу зображення на сервері), categoryId (унікальний індекс категорії).

Сутність Categories має два атрибути – це name (ім’я категорії), parentId (унікальний індекс загальної категорії). Можливість створення підкатегорій реалізується за допомогою parentId. Якщо категорія має вищу категорію, то унікальний індекс вищої категорії записується у parentId, а якщо категорія є основною, то атрибут parentId записується як NULL.

3.3 Діаграма варіантів використання

Для кращого розуміння розроблюваної системи виконаємо моделювання варіантів використання. Схема варіантів використання зображена на рисунку 3.5.

Рисунок 3.5 – Діаграма варіантів використання

Акторами у даній системі є покупець, робітник магазину, а також бухгалтер та база даних. Перегляд товарів – це процес пошуку та підбору необхідних товарів в інтернет-магазині. Замовлення товарів – це процес заповнення замовлення та контактної інформації. Обробка замовлених товарів – це процес перегляду інформації про замовлення та його подальше виконання. Редагування статей – це процес додавання, видалення, або редагування статей і новин в інтернет магазині. Редагування товарів – це процес додавання, видалення або редагування товарів та інформації про них. Прогнозування закупівель – це процес в якому бухгалтер по обраній категорії товарів буде отримувати список рекомендованих до закупівлі товарів. Формування звітів – це процес формування видаткової накладної по виконаному замовленню.

4 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

4.1 Реалізація фізичної моделі бази даних

Для розробки бази даних було обрано локальний сервер Ampps. Засобом для адміністрування сервер був обраний PHPMyAdmin. На рисунку 4.1 зображений інтерфейс PHPMyAdmin.

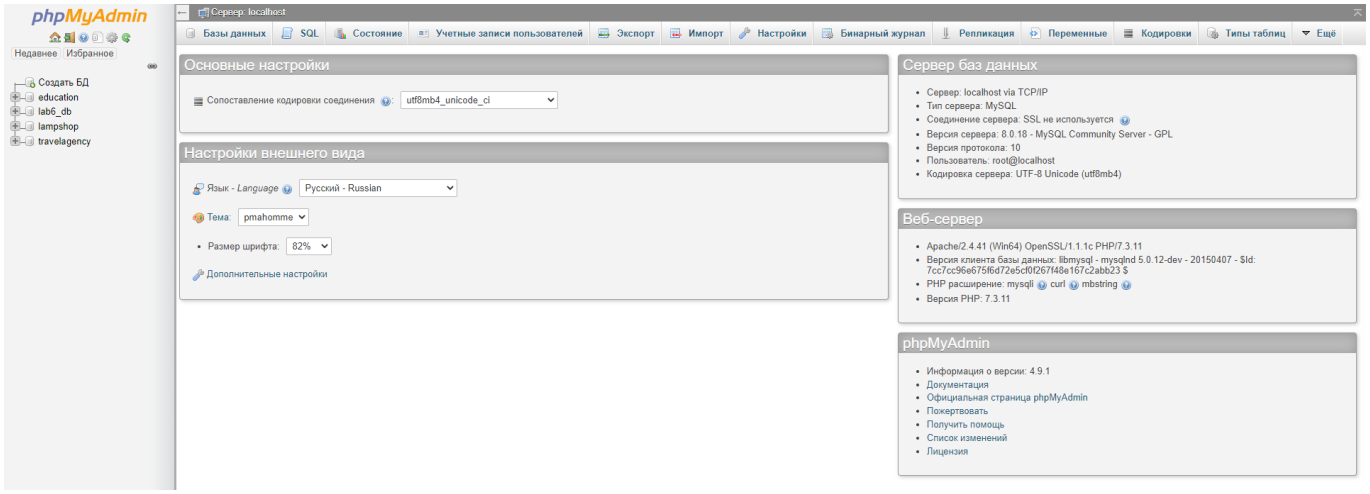


Рисунок 4.1 – Інтерфейс PHPMyAdmin

Для початку роботи була створена база даних під назвою lampshop. Після створення бази даних до неї були додані таблиці: articles, categories, customers, orders, productorders, products, regions, users.

Розглянемо процес створення таблиці users. Була створена нова таблиця із шістьма рядками. Перший рядок це первинний ключ нашої таблиця. Для створення первинного ключа обираємо йому тип INT, у полі індекс обираємо PRIMARY та ставимо галочку на A_I тобто auto increment – ця опція дозволяє створювати

первинний ключ автоматично при додаванні нових даних до таблиці. Після цього проводимо заповнення усіх інших полів таблиці – це firstName з типом даних varchar(30), lastName з типом даних varchar(30), phone з типом даних varchar(25), email з типом даних varchar(50), password з типом даних varchar(20). Тип таблиці вказуємо InnoDB. Скріншот процесу створення таблиці зображений на рисунку 4.2, а на рисунку 4.2 зображена структура бази даних lampshop.

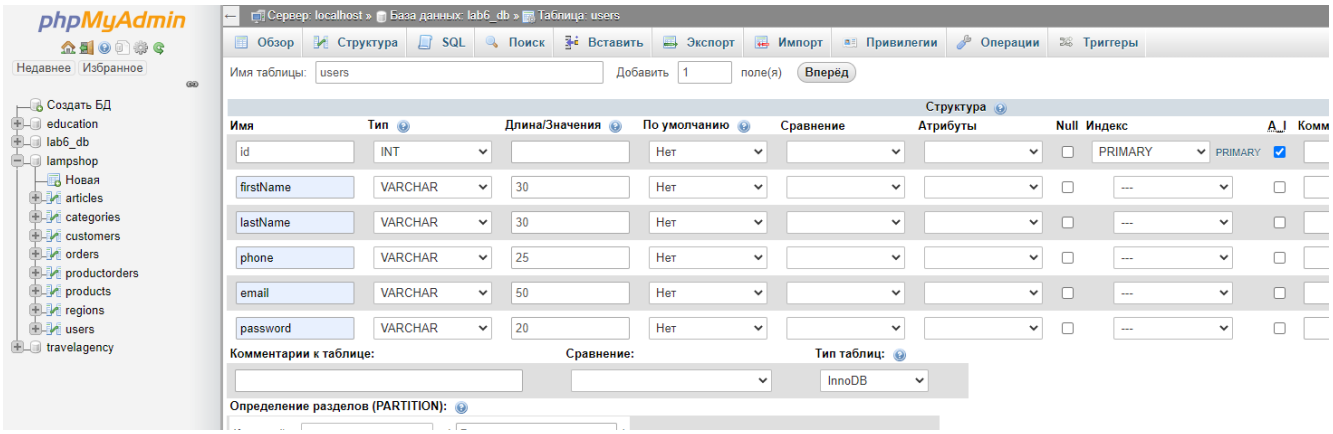


Рисунок 4.2 – Створення таблиці Users

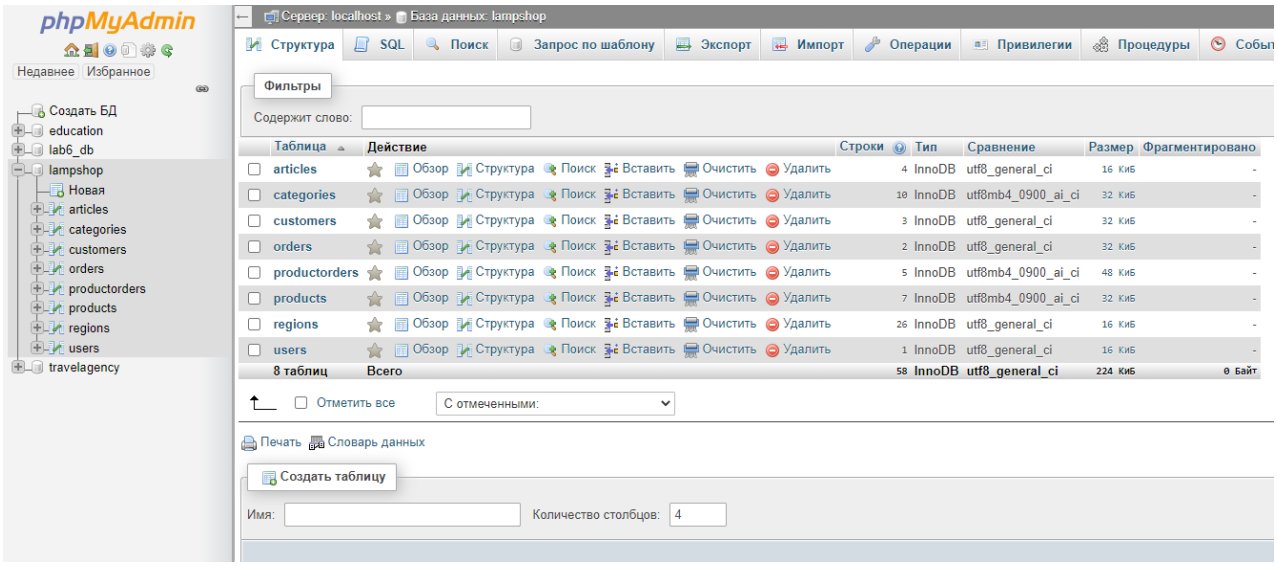


Рисунок 4.3 – Структура бази даних

Всі інші таблиці створюються таким же чином. Наступним кроком є створення зв’язків між таблицями по вторинним ключам. Для цього на поля по типу categoryId або productId додаємо унікальні індекси. Процес зв’язування ключів між собою показаний на рисунку 4.4.

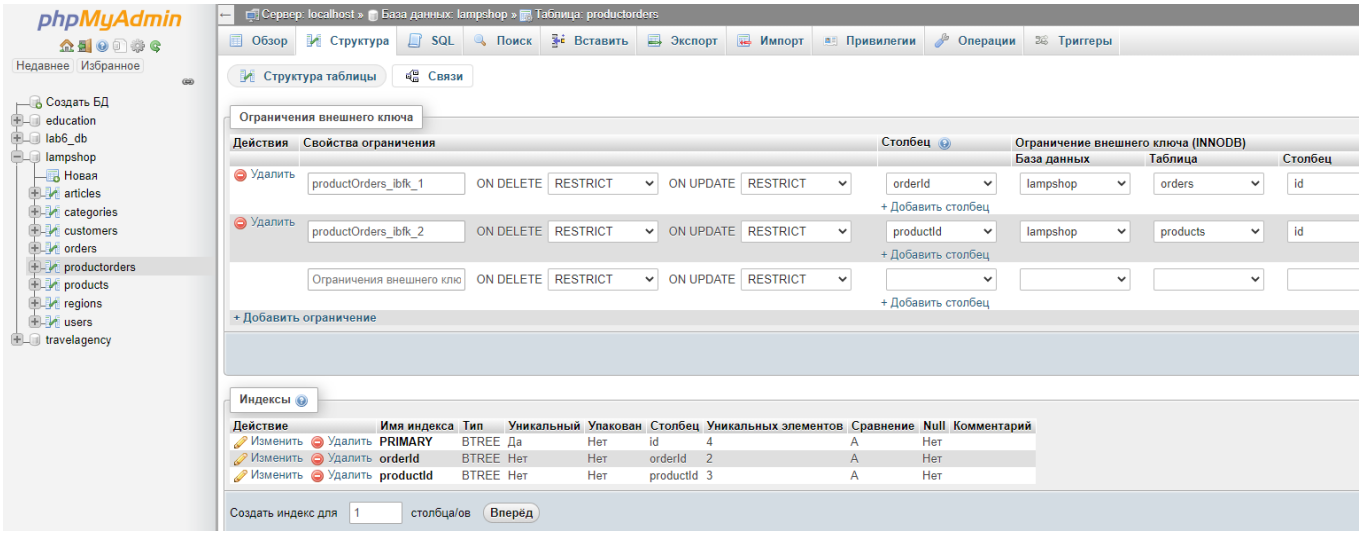


Рисунок 4.4 – Процес пов’язання вторинних ключів

Після виконання всіх дій описаних вище, ми отримуємо базу даних, що відповідає розробленим моделям у пункті 3.2. На рисунку 4.5 зображено графічне представлення бази даних.

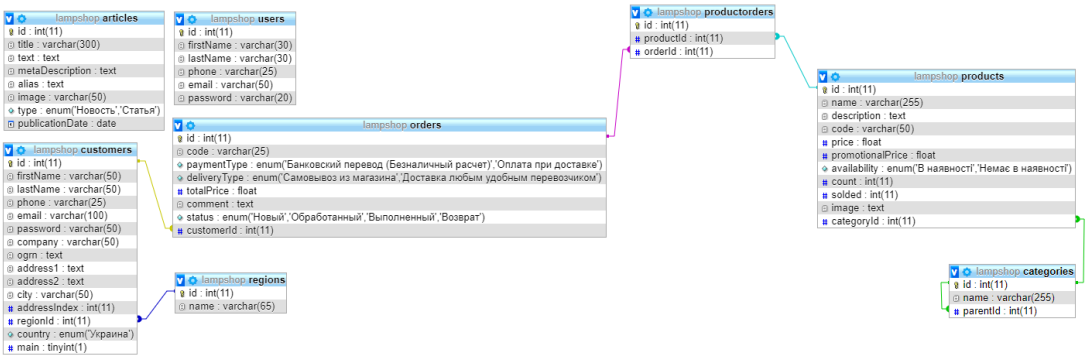


Рисунок 4.5 – Схема бази даних

4.2 Реалізація серверної частини

Для реалізації серверної частини була обрана NodeJS. NodeJS – це JavaScript–оточення побудоване на JavaScript–рушієві Chrome V8.[26] Тому для розробки з використанням мови JavaScript була обрана IDE PhpStorm 2020. Ця IDE має вбудовану підтримку JavaScript, а також NodeJS та ReactJS. На рисунку 4.6 зображений процес роботи в IDE PhpStorm.

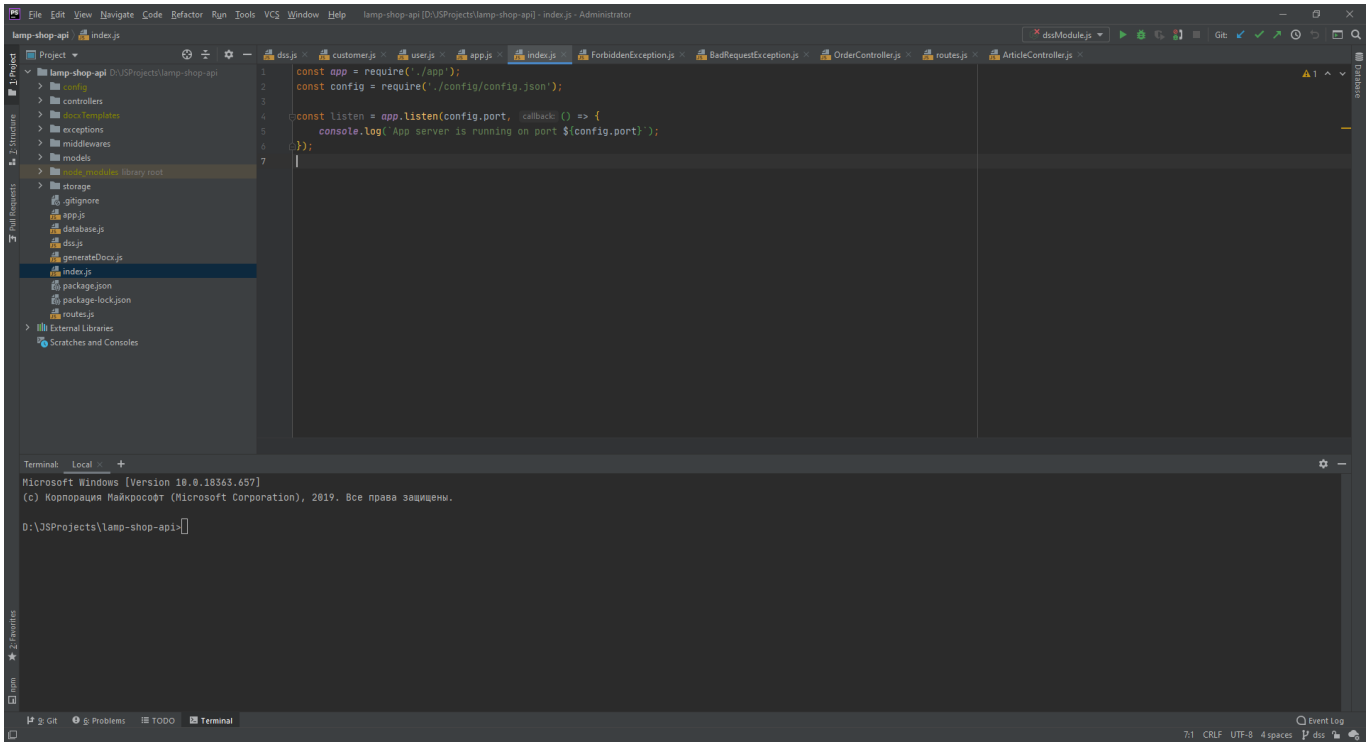


Рисунок 4.6 – Процес розробки в PhpStorm

Процес розробки серверної частини з використанням тільки вбудованих функцій NodeJS можливий, але дуже складний і трудомісткий. Тому було обрано мінімалістичний фреймворк для розробки API, а саме ExpressJS.

На даний час цей фреймворк найпопулярніший з всіх аналогів. Він надає велику кількість функцій маючи в своєму розпорядженні безліч службових методів HTTP і проміжних оброблювачів. За допомогою пакетного менеджеру NodeJS встановлення фреймворку, а також додаткових пакетів є дуже простим.

Встановити фреймворк можна за допомогою команди в терміналі: npm install express –save. Після цього менеджер пакетів автоматично завантажить необхідні дані, та все встановить. На рисунку 4.7 зображено встановлення фреймворку ExpressJS.

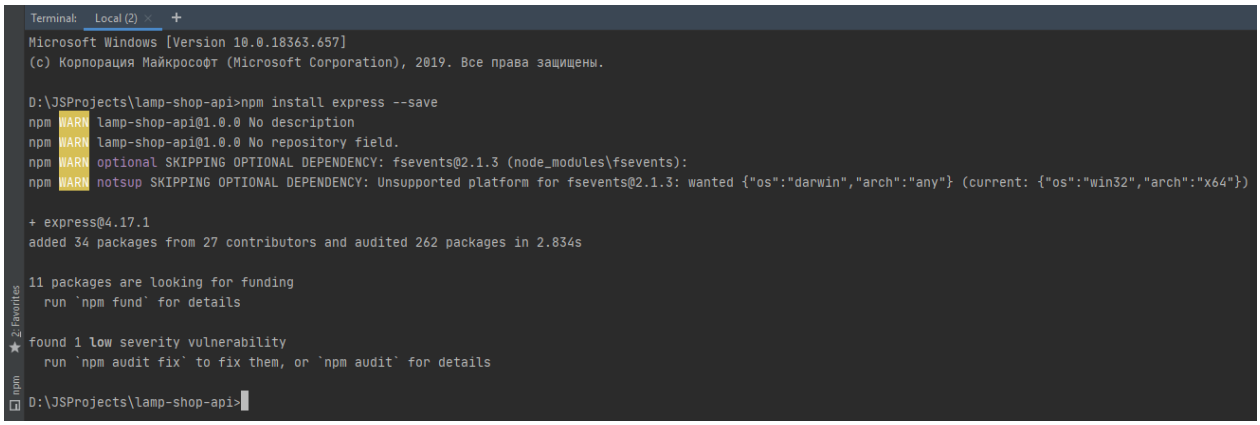


Рисунок 4.7 – Встановлення фреймворку NodeJS

Встановлення всіх інших пакетів є аналогічним до цього. Тепер перейдемо до налаштування API. Після встановлення ExpressJS в проекті не додаються ніякі файли, тому потрібно створювати скелет додатку самостійно.

Головним файлом API є App.js. В ньому створюємо наш сервер за допомогою Express. Робиться за допомогою декількох рядків коду. Для початку отримуємо функцію-конструктор фреймворку const express = require('express'), потім створюємо змінну API, за допомогою якої будемо оброблювати запроси, а також завантажувати сервер і прослуховувати необхідний порт. На рисунку 4.8 зображений код головного модуля App.js.

Рисунок 4.8 – App.js з підключеними модулями

В інформаційній системі використовується клієнт-серверна архітектура. Передачу даних між клієнтом та сервером будемо здійснювати за допомогою HTTP-запитів. Плюс використання таких запитів у тому, що Express має гарну підтримку на роботу з ними, а також вони є дуже розповсюдженими та використовуються майже всюди.

API, що розробляється, буде працювати за архітектурним стилем REST. Для цього необхідно створити роутер, який буде брати HTTP запит, оброблювати його та відправляти відповідь. Наприклад, router.get('/categories', CategoryController.list), де '/categories' – це url-адреса, через яку запитуємо в браузері, а CategoryController.list – це функція-обробник відповідного запиту.

Перейдемо до розробки функцій-обробників HTTP-запитів. В додатку буде лише два типи функцій обробників – це функції, що роблять запити до бази даних і функції, що роблять маніпуляції з даними самостійно.

Спочатку розглянемо функцію, що не робить запитів до бази даних, а саме – це функція формування видаткових накладних. Логіка формування звіту така, що отримуємо у тілі HTTP-запиту дані, які нам потім потрібно буде записати у вже заздалегідь сформований шаблон Word-документу і відправити користувачу цей

документ. Для формування документу використовується пакет docxtemplater. Спочатку створюється об’єкт пакету docxtemplater, а саме doc = new Docxtemplater(). Всі необхідні дані записується у вигляді JavaScript об’єкту і потім передаються у функцію doc.setData(dataForDocx). Після цього на сервері створюється вихідний файл, який далі передається користувачу за допомогою методу res.sendFile(fileName, options). У налаштуваннях (options) вказуємо місцезнаходження файлу (root), а також необхідні для запиту заголовки (headers). І після цього в браузері у користувача завантажується видаткова накладна по обраному замовленню. Скріншот шаблону Word-документу представлений на рисунку 4.9, а на рисунку 4.10 представлена готова видаткова накладна по одному з товарів із використанням тестових даних.

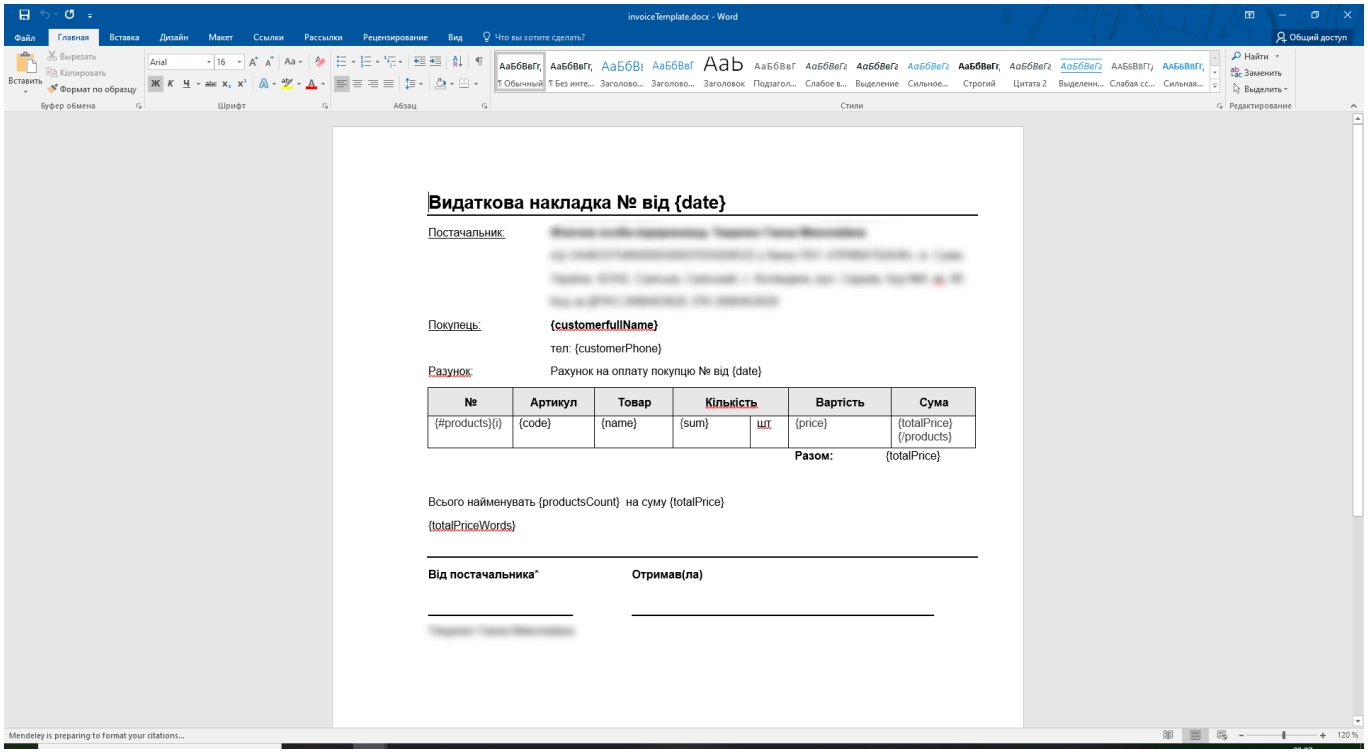
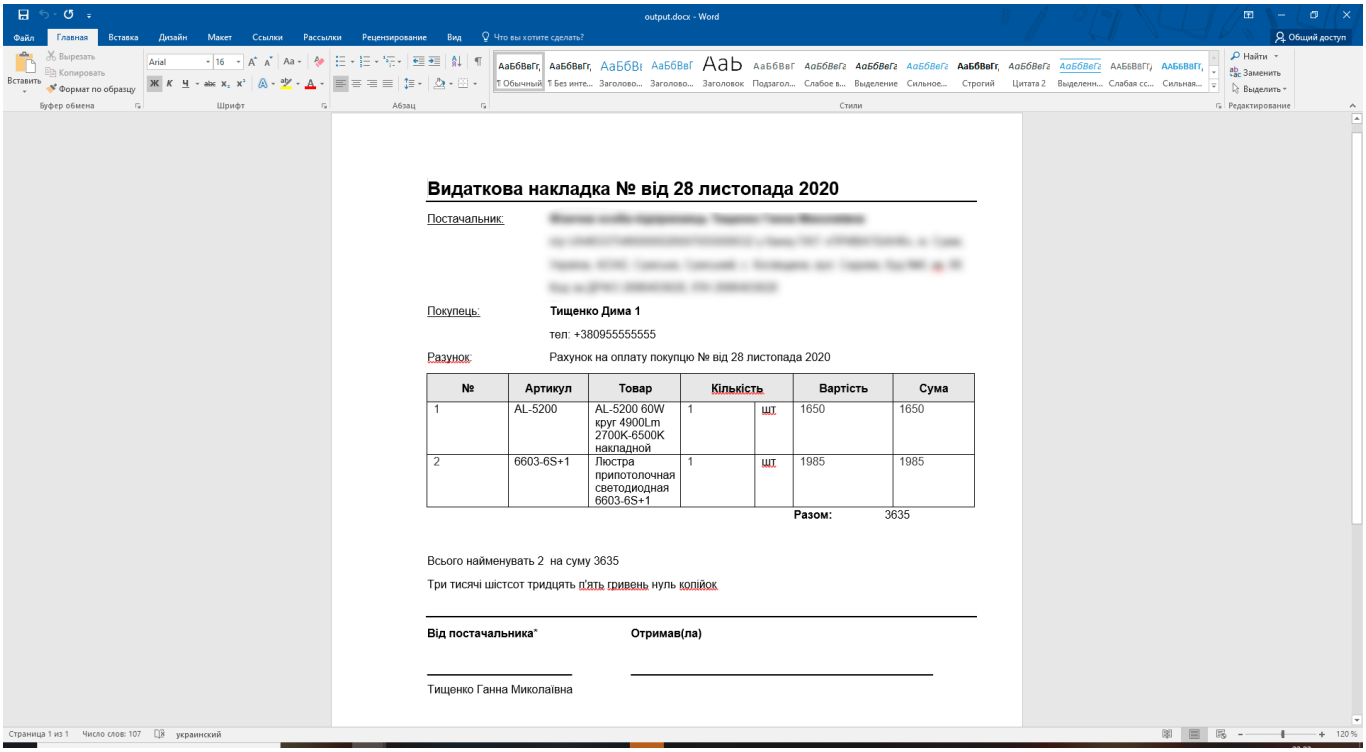


Рисунок 4.9 – Шаблон видаткової накладної



Видаткова накладка № від 28 листопада 2020

Постачальник:

Покупець: **Тищенко Дима 1**

тел: +380955555555

Разунок: Рахунок на оплату покупцю № від 28 листопада 2020

№	Артикул	Товар	Кількість		Вартість	Сума
1	AL-5200	AL-5200 60W круг 4900Lm 2700K-6500K накладной	1	шт	1650	1650
2	6603-6S+1	Люстра припотолочная светодиодная 6603-6S+1	1	шт	1985	1985
					Разом:	3635

Всього найменувать 2 на суму 3635

Три тисячі шістсот тридцять п'ять гривень нуль копійок

Від постачальника" **Отримав(ла)**

Тищенко Ганна Миколаївна

Рисунок 4.10 – Приклад отриманої видаткової накладної

Наступний етап роботи – організація функцій-обробників, що працюють з базою даних. Для отримання, редагування, видалення або змінення даних у базі даних можемо використовувати звичайні запити мовою SQL. Але їх написання і подальша зміна або тестування займає певний час. Тому для роботи з базою даних використаємо об’єктно-реляційне відображення даних (ORM), а саме Sequelize. ORM – це технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних»[27].

Для використання Sequelize потрібно спочатку його налаштувати, а саме для налаштування підключення до бази даних необхідно створити у файлі-конфігурації описи імені бази даних, імені користувача бази даних та його паролю та налаштування такі, як host (місце де знаходиться наша база даних) та dialect (mysql). Після цього створюємо каталог models, в якому описуємо всі сутності, що є в базі даних, з їх атрибутами. Для прикладу візьмемо найменшу сутність - це «категорія». При описанні атрибутів вказуємо його ім’я, як в таблиці бази даних, тип даних, чи

може бути null, а також валідацію, яка допоможе у випадку, якщо в запиті щось пропустили і намагаємося додати до бази даних пустий рядок. У Sequelize не вказуємо первинні та вторинні ключі, вони будуть потрібні пізніше для налаштування зв’язків між моделями. На рисунку 4.11 зображений приклад коду для описання моделі категорії.

```
module.exports = (sequelize, DataTypes) => {
    return sequelize.define('category', {
        name: {
            type: DataTypes.STRING,
            allowNull: false,
            validate: {
                notNull: true,
                notEmpty: true,
            },
        },
    })
};
```

Рисунок 4.11 – Описання моделі категорії

В розробленій базі даних є два типи зв’язків – це «один до багатьох» та «багато до багатьох». На прикладі тієї ж таблиці категорій розглянемо зв’язок «один до багатьох». Для формування підкатегорій в таблиці категорій є атрибут parentId, якщо він null, то категорія не є підкатегорією, а якщо навпаки, то вона має батьківську категорію. Для реалізації даного зв’язку після описання всіх категорій користуємося даним фрагментом коду models.Category.hasMany(models.Category, { as: 'child', foreignKey: 'parentId' }). Функція hasMany вказує на зв’язок «один до багатьох», першим параметром функції йде об’єкт таблиці до якої встановлюємо зв’язок, наступним ідуть опції такі, як коротка назва, необхідна потім для запиту до бази даних з використанням цього зв’язку, та вторинний ключ.

Зв’язок «багато до багатьох» налаштовується подібним способом, але вже використовується функція belongsToMany(…), в яку ми передаємо ті ж самі дані, що і при зв’язку «один до багатьох», а також таблицю, через яку цей зв’язок повинен

працювати. На рисунку 4.12 показаний фрагмент коду зв’язку «багато до багатьох» між таблицями замовлень та продуктів через таблицю ProductOrder.

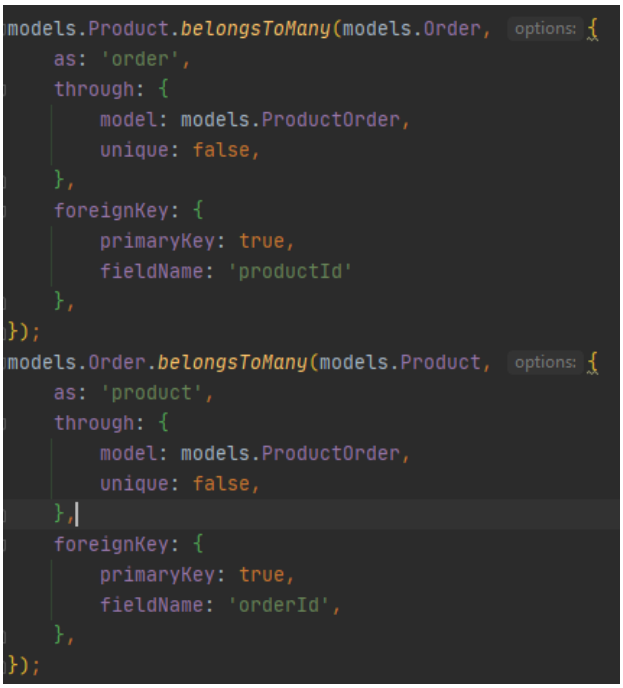


Рисунок 4.12 – Зв’язок багато до багатьох

Перейдемо до розгляду однієї з функцій-обробників запитів, а саме до функції виводу підкатегорій. Для отримання даних з бази даних у Sequelize передбачено декілька функцій. Одна з таких findAll. Ця функція допомагає замінити SQL запит з select. Для її роботи необхідно передати до неї певні параметри, а саме так як потрібні саме підкатегорії, тобто ті, що мають якийсь параметр у атрибуті parentId, тому в налаштуваннях вказуємо where: {parentId: {[Op.ne]: null }}. Після цього зберігаємо отримані дані до змінної data та відправляємо їх HTPP-відповіддю разом зі статусом за допомогою функції res.status(200).send({ data, }).

Для завантаження або збереження файлів з серверу використовується пакет multer. Для збереження фото статей, а також продуктів, була створена директорія storage, де вони всі зберігаються. Для збереження файлу перед запитом нам потрібно встановити додаткову функцію, так звану middleware – це функція яка перериває

виконання запиту, виконує свої дії і потім виконання запиту продовжується. Для збереження файлу у роутах вказуємо перед функцією-обробником upload.single('image'), тому виглядати це буди приблизно так: router.post('/products', upload.single('image'), auth, ProductController.create);

Також для розмежування доступу до бази даних потрібно використати ще одне middleware авторизації. Воно буде перевіряти, чи є в заголовках HTTP-запиту так званий токен доступу – зашифрований набір цифр і букв, по якому за допомогою пакету jwt будемо його зашифровувати або розшифровувати. У розшифрованих даних буде зберігатися електронна пошта користувача та його пароль. Якщо даний користувач є в базі даних, то йому буде даватися можливість редагувати, створювати або видаляти дані, а якщо ні, то повертається помилка. Копія коду middleware авторизації користувача показаний на рисунку 4.13.

```
module.exports = async (req, res, next) => {
    try {
        const payload = jwt.verify(req.header('access_token'), auth.secret);
        if(payload){
            const user = await User.findByPk(payload.id);

            if(user){
                req.user = user;
            }
        }
        next();
    } catch (error){
        return next(new UnauthorizedException(error));
    }
};
```

Рисунок 4.13 – Middleware авторизації

Також для обробки помилок було розроблено відповідні функції, що повертають статус помилки та повідомлення з її описом. Одна з таких помилок це «404 – ресурс не знайдений». Вона виникає у випадку, коли до API надходить запит на дані, що не існують. Код такої помилки зображений на малюнку 4.14.

```
module.exports = function NotFoundException(error) {
    Error.call(this, error) ;

    this.name = 'NotFoundException';
    this.message = 'Oops! Resource not found.';
    this.status = 404;
    this.originalError = error;

    if (Error.captureStackTrace) {
        Error.captureStackTrace(this, NotFoundException);
    } else {
        this.stack = (new Error()).stack;
    }
};
```

Рисунок 4.14 – Помилка «Ресурс не знайдено»

Останнім етапом виконання роботи стало реалізація модуля допомоги формування списку закупівель на наступний місяць. Сам метод описано у контролері продуктів. У HTTP-запиті в параметрах передається id обраної категорії, за якім потім буде виконуватися пошук. Після цього виконується запит до бази даних, з якого отримуємо товари за обраною категорією та кількість замовлень по кожному товару. Далі формується об’єкт, в якому ключем буде id продукту, а значенням – масив з двох параметрів (це кількість товарів на складі та кількість замовлень товару). Після цього знаходимо оптимальний до закупівлі товар по кожному з критеріїв. На рисунку 4.15 зображено розрахунки за критеріями Вальда, Байеса, а також за критерієм Гурвіца.

```
let vald = { value: 0 };

for (let key in soldedProducts) {
    if(vald.value < Math.min(soldedProducts[key][0], soldedProducts[key][1])) {
        vald.id = key;
        vald.value = Math.min(soldedProducts[key][0], soldedProducts[key][1]);
    }
}

let bayes = { value: -1 };

for (let key in soldedProducts) {
    if(bayes.value < soldedProducts[key][0] * 0.3 + soldedProducts[key][1]*0.7) {
        bayes.id = key;
        bayes.value = soldedProducts[key][0] * 0.3 + soldedProducts[key][1]*0.7;
    }
}

let hurviz = { s: 0 };

for (let key in soldedProducts) {
    const minA = Math.min(soldedProducts[key][0], soldedProducts[key][1]);
    const maxA = Math.max(soldedProducts[key][0], soldedProducts[key][1]);
    if (hurviz.s < 0.3 * minA + 0.7 * maxA) {
        hurviz.id = key;
        hurviz.s = 0.3 * minA + 0.7 * maxA;
    }
}
```

Рисунок 4.15 – Розрахунки за критеріями Вальда, Байеса та Гурвіца

Після розрахунків обирається той товар, який частіше всього зустрічається в них, як показано на рисунку 4.16.

```
const criteriesIds = [vald.id, bayes.id, hurviz.id];

let mf = 1;
let m = 0;
let item;
for (let i=0; i<criteriesIds.length; i++)
{
    for (let j=i; j<criteriesIds.length; j++)
    {
        if (criteriesIds[i] == criteriesIds[j])
            m++;
        if (mf<m)
        {
            mf=m;
            item = criteriesIds[i];
        }
    }
    m=0;
}

productIdsToPurchase.push(item);
delete soldedProducts[item];
```

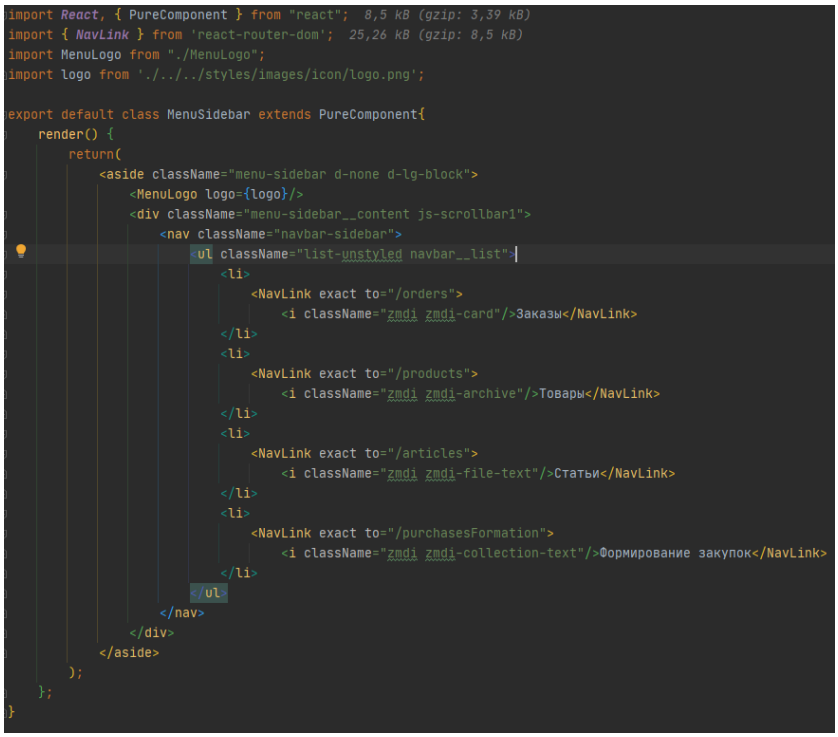
Рисунок 4.16 – Пошук оптимального товару по трьом критеріям

Далі додається id обраного товару до результуючого масиву, видаляється з початкового, а далі проводиться пошук знову. Ці дії повторюються, доки не буде опрацьовано останній товар. Після цього формується список товарів з необхідними даними та відправляється користувачу.

4.3 Реалізація клієнтської частини

Для реалізації клієнтської частини була обрана бібліотека ReactJS. Клієнтська частина поділена на два додатки – це онлайн магазин та панель адміністратора. Онлайн магазин складається з простого показу даних, переходів між сторінками та простими формами для відправки даних, а панель адміністратора має більший функціонал. Тому для розгляду реалізації клієнтської частини приклади будуть взяті з панелі адміністратора.

Після створення додатку не маємо чіткої структури проекту. Тому повинні створити її самі. ReactJS реалізує компонентний підхід, тому всі частини інтерфейсу – це невеликі компоненти. Об’єднують ці компоненти сторінки, між якими відбуваються переходи. При зміні сторінки деякі компоненти змінюються, а компоненти, що не потребують зміни (такі як бокове меню навігації в панелі адміністратора, або шапка сайту онлайн-магазину), залишаються незмінними. Саме в цьому полягає швидкість роботи ReactJS. На рисунку 4.17 зображений скріншот коду компоненту бокового меню панелі адміністратора.



```
import React, { PureComponent } from "react";  8,5 kB (gzip: 3,39 kB)
import { NavLink } from 'react-router-dom';  25,26 kB (gzip: 8,5 kB)
import MenuLogo from "./MenuLogo";
import logo from '../../../styles/images/icon/logo.png';

export default class MenuSidebar extends PureComponent{
    render() {
        return(
            <aside className="menu-sidebar d-none d-lg-block">
                <MenuLogo logo={logo}/>
                <div className="menu-sidebar__content js-scrollbar1">
                    <nav className="navbar-sidebar">
                        <ul className="list-unstyled navbar__list">
                            <li>
                                <NavLink exact to="/orders">
                                    <i className="zmdi zmdi-card"/>Заказы</NavLink>
                            </li>
                            <li>
                                <NavLink exact to="/products">
                                    <i className="zmdi zmdi-archive"/>Товары</NavLink>
                            </li>
                            <li>
                                <NavLink exact to="/articles">
                                    <i className="zmdi zmdi-file-text"/>Статьи</NavLink>
                            </li>
                            <li>
                                <NavLink exact to="/purchasesFormation">
                                    <i className="zmdi zmdi-collection-text"/>Формирование закупок</NavLink>
                            </li>
                        </ul>
                    </nav>
                </div>
            </aside>
        );
    };
}
```

Рисунок 4.17 – Компонент бокового меню

Перехід між сторінками додатку реалізований за допомогою пакету react-router-dom. У головному файлі програми створюємо за допомогою модулю Switch розтер, і при зміні url-адреси відбувається зміна компонентів, які потрібно рендерити. На рисунку 4.18 зображений приклад роутеру панелі адміністратора.

```
<Switch>
    <Route exact path="/" component={ Orders }/>
    <Route exact path="/orders" component={ Orders }/>
    <Route exact path="/products" component={ Products }/>
    <Route exact path="/products/:id" component={ ProductItem }/>
    <Route exact path="/articles" component={ Articles }/>
    <Route exact path="/articles/:id" component={ ArticlesItem }/>
    <Route exact path="/login" component={ Login }/>
    <Route exact path="/purchasesFormation" component={ PurchasesFormation }/>
</Switch>
```

Рисунок 4.18 – Роутер панелі адміністратора

З нього бачимо, що початковою сторінкою буде сторінка з замовленнями. Також при зміні адреси на одну з заданих буде рендеритися потрібний компонент.

Наступним кроком є перевірка прав доступу. Її потрібно постійно проводити, адже доступ до панелі адміністратора повинен мати тільки адміністратор. Логіка роботи перевірки така: при завантаженні компоненту або його оновленні, запускаємо функцію checkAuth(), яка бере дані зі сховища і перевіряє зареєстрований користувач чи ні. Якщо користувач не зареєстрований, то відбувається перенаправлення на сторінку авторизації. Приклад ідентифікації користувача зображений на рисунку 4.19.

```
checkAuth = () => {
    const { isLoggedIn, location: { pathname }, history } = this.props;

    if (!isLoggedIn && pathname !== '/login') {
        history.push('/login');
    } else if (isLoggedIn && pathname === '/login') {
        history.push('/');
    }
}
```

Рисунок 4.19 – Перевірка доступу користувача

Для виконання запиту до API, збереженню даних, які необхідні у додатку, будемо використовувати бібліотеку Redux. У своїй структурі Redux має чотири головні компоненти – це actions, reducers, store, view. Взаємодія цих компонентів зображена на рисунку 4.20

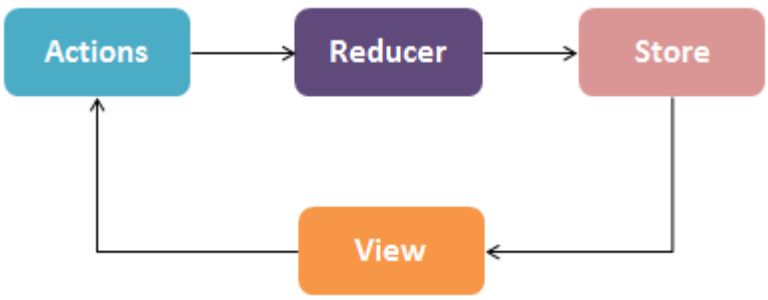


Рисунок 4.20 – Архітектура Redux

В даному випадку елементом View є react-компонент. Action – це подія, яку викликаємо для того, щоб щось змінити. Наприклад, форму заповнено і потрібно відправити дані. В цьому випадку після натискання на кнопку додавання даних з форми викликається подія, у яку передаємо сформовані дані. Reducer – це обробник подій. За допомогою нього вказуємо, які елементи у сховищі нам потрібно змінити. Store – це сховище, де зберігаємо необхідні нам дані.

Наступний крок – налаштування сховища. Створення сховища відбувається за допомогою функції createStore(), у яку повинні передаватися ред'юсер, а також початковий стан елементів сховища. Оскільки ред'юсер використовується не один, то потрібно використати функцію для об'єднання всіх необхідних ред'юсерів. Ця функція є стандартною і називається combineReducers(). До неї передаємо об'єкт, у який записуємо ключем назву ред'юсера, а параметром - імпортований об'єкт ред'юсера. На рисунку 4.21 зображений приклад комбінування ред'юсерів.

```
import { combineReducers } from 'redux';  3,62 kB (gzip: 1,49 kB)

import Auth from './auth';
import Article from "./Article";
import Product from "./Product";
import Category from "./Category";
import Order from "./Order";
import File from "./File";
import Purchase from "./Purchase";

export default () => combineReducers( reducers: {
    auth: Auth,
    article: Article,
    product: Product,
    category: Category,
    order: Order,
    file: File,
    purchase: Purchase,
});
```

Рисунок 4.21 – Комбінування ред'юсерів

Наступним етапом роботи з Redux є підключення компонента до сховища для отримання доступу до даних. Для виконання підключення до сховища потрібно визначити функцію-декоратор connect, у яку передаємо два параметри. У першому аргументі буде стан, а у другому - відправлення (dispatch).

Розглянемо процес роботи Redux. Наприклад, сторінку з переліком замовлень ми повинні підключити до сховища. У сховищі беремо два атрибути – це loading (завантаження) та orders (замовлення). Змінна завантаження необхідна для того, щоб контролювати, коли йде запит на отримання даних. Під час оброблення запиту змінна loading дорівнює true, а коли він виповнився, змінюємо її на false та передаємо дані. Зміна значення loading та orders виконується у ред'юсері. На рисунку 4.22 представлений код ред'юсера Order (замовлення), який обробляє два типи подій – це завантаження замовлень та оновлення замовлень при зміні статусу.

```
export default function Order (state = INITIAL_STATE, action) {
    switch (action.type) {
        case LOAD_ORDERS_REQUEST:
        case UPDATE_ORDER_REQUEST:
            return Object.assign( target: {}, state, source2: {
                loading: true,
            });

        case LOAD_ORDERS_SUCCESS:
            return Object.assign( target: {}, state, source2: {
                list: action.data,
                loading: false,
            });

        case UPDATE_ORDER_SUCCESS:
            return Object.assign( target: {}, state, source2: {
                loading: false,
            });

        case LOAD_ORDERS_ERROR:
        case UPDATE_ORDER_ERROR:
            return Object.assign( target: {}, state, source2: {
                loading: false,
            });

        default:
            return state;
    }
}
```

Рисунок 4.22 – Ред'юсер замовлень

Розглянемо розробку actions (подій) для Redux. Наприклад, для завантаження файлу з видатковою накладною необхідно передати у подію дані про замовлення. Після чого за допомогою пакету axios робимо HTTP-запит до API. Після цього виконуємо завантаження отриманого файлу, а саме створюємо новий елемент-посилання, до якого додаємо атрибути завантаження файлу та його назву. Потім виконуємо натискання на створений елемент, і у користувача автоматично

починається завантаження файлу. На рисунку 4.23 приведений скріншот коду з подією завантаження файлу.

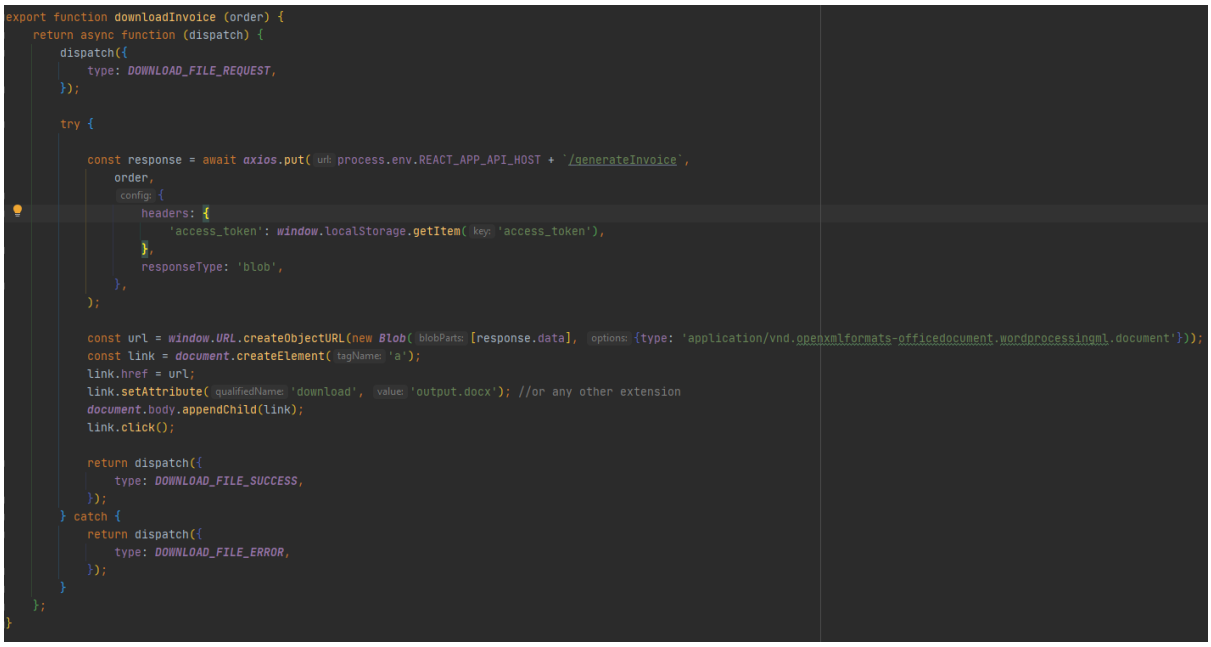


Рисунок 4.23 – Подія завантаження видаткової накладної

Для розробки візуального оформлення інтернет-магазину та панелі адміністратора використовувався синтаксис JSX. JSX – це розширення мови JavaScript для більш легкого верстання сайту з невеликими відмінностями. Для оформлення стилів були використані CSS бібліотеки, такі як Bootstrap та Material Design. В панелі адміністратора додатково була використана бібліотека MaterialUI для формування таблиць та списків товарів.

Загальний вигляд головної сторінки інтернет магазину зображений на рисунку 4.24, а вигляд головної сторінки панелі адміністратора зображений на рисунку 4.25.

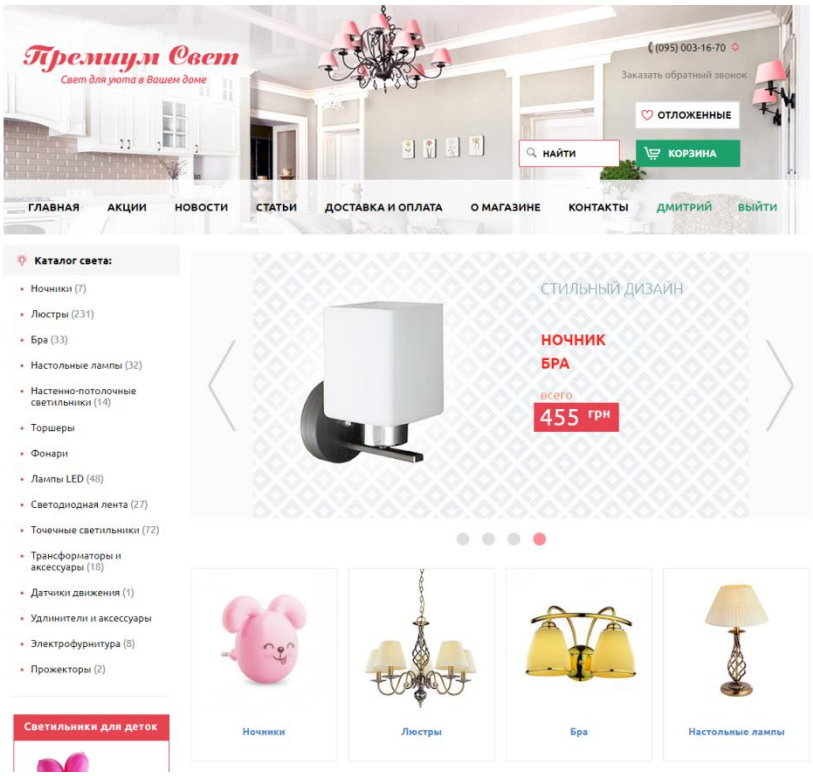


Рисунок 4.24 – Головна сторінка інтернет-магазину

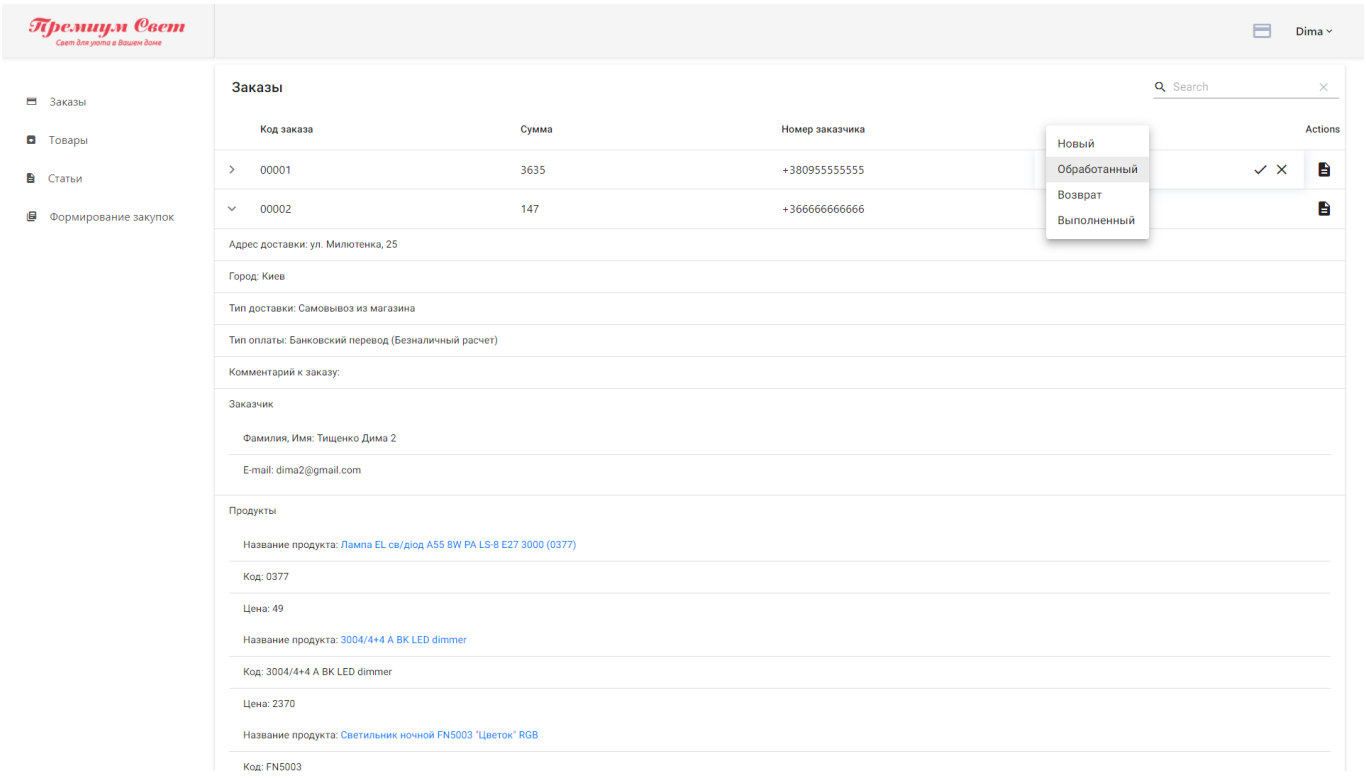


Рисунок 4.25 – Головна сторінка панелі адміністратора

ВИСНОВКИ

В результаті виконання дипломного проекту було розроблено інформаційну систему підтримки діяльності онлайн-магазину.

Під час аналізу предметної області була досліджена робота існуючого онлайн-магазину, процес ведення бухгалтерського обліку в компанії, а також був проведений огляд існуючих аналогів. Виконаний огляд програмних аналогів показав, що є потреба у створенні онлайн-магазину з сучасним власним дизайном та широким функціоналом.

Після прийняття рішення про розробку інформаційної системи була сформована мета роботи та поставлені задачі для її виконання. Були обрані методи дослідження даної роботи, а саме було проведено вибір методів реалізації інформаційної системи, а також вибір методів для формування списку закупівель товарів по обраній категорії. По результатам проведених робіт мовою програмування було обрано JavaScript, бібліотекою для розробки клієнтської частини системи було обрано ReactJS, а для серверної частини – NodeJS. Для формування списку закупівель товарів було розроблено алгоритм, що включає в себе методи допомоги прийняття рішень в умовах невизначеності такі як критерій Вальда, Байеса і Гурвіца.

Виконано планування робіт для забезпечення досягнення поставленої мети, о визначено календарний графік робіт та проведено аналіз можливих ризиків. Процес роботи інформаційної системи був описаний за допомогою функціональних діаграм нотації IDEF0, а також моделі варіантів використання. Проведено моделювання бази даних.

За результатами моделювання було виконано фізичну реалізацію бази даних за допомогою MySQL, серверної частини за допомогою NodeJS з використання

фреймворку ExpressJS. Клієнтська частина написана з використанням бібліотек React та Redux.

Всі поставлені задачі виконано. Створена інформаційна система дозволяє вести онлайн продажі товарів, формувати звіти за продажами, що заощаджує час бухгалтерії, а також формувати список рекомендованих до закупівлі товарів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Sharma, G., Lijuan, W. The effects of online service quality of e-commerce Websites on user satisfaction. Electronic Library. 2015. Vol. 33, No. 3.

2. Atanassova, A. Accounting problems in e-commerce. Ekonomiczne Problemy Usług. 2018. Vol. 131. C. 12.

3. Keith, M. Global eCommerce Sales, Trends and Statistics 2015: URL: https://www.remarkety.com/global-ecommerce-sales-trends-and-statistics-2015remarkety(дата звернення: 26.10.20).

4. Zavyalova, M., Skrynko, N., Helevachuk, Z. E-COMMERCE: GLOBAL AND DOMESTIC DEVELOPMENT TRENDS. Scientific Journal of Polonia University. 2019. Vol. 35, No. 4.

5. ENACHE, M. C. E-commerce Trends. Annals of Dunarea de Jos University. Fascicle I : Economics and Applied Informatics. 2018. Vol. 24, No. 2. C. 67–71.

6. Alam Iqbal, B. E-Commerce Vs Mobile Commerece. International Journal of Applied Research. 2013. Vol. 2. C. 1–24.

7. EVOService. Що таке Prom.ua? URL: https://help.prom.ua/hc/uk/articles/360008087177-Що-таке-Prom-ua-(дата звернення: 26.10.20).

8. OECD. SDBS Structural Business Statistics (ISIC Rev. 4) : Number of SMEs and large firms: URL: https://stats.oecd.org/Index.aspx?QueryId=81354(дата звернення: 28.10.20).

9. D. Elagina. Number of businesses in Ukraine 2018, by economic activity: URL: https://www.statista.com/statistics/995502/ukraine-number-business-entities-by-economic-activity/statista, (дата звернення: 28.10.20).

10. Hao, W., Yue, X. The study on the application of E-commerce in small and medium-

sized Enterprises: *CCTAE 2010 - 2010 International Conference on Computer and Communication Technologies in Agriculture Engineering*, 10. С. 142–148.

11. Absolunet. 10 eCommerce Trends for 2019 - Presented by Absolunet: https://10ecommercetrends.com/.

12. Lampla.ua. Как нас найти |: URL: https://www.lampa.ua/ru/contact.html(дата звернення: 27.10.20).

13. ТерриторияСвета. Контактная информация интернет-магазина «Территория света»: URL: https://www.terra-svet.com/contact/(дата звернення: 27.10.20).

14. Свет. О компании - - интернет магазин Свет: URL: https://svetua.com.ua/company/(дата звернення: 27.10.20).

15. Islam, O., Alfakeeh, A., Nadeem, F. A Framework for Effective Big data Analytics for Decision Support Systems. International Journal of Computer Networks And Applications. 2017. Vol. 4, No. 5. С. 129–137.

16. ProPosition. Карта сайта - как составить Sitemap (с примерами): URL: https://proposition.digital/sitemap(дата звернення: 28.10.20).

17. FlowMapp UX tools — Sitemap, User Flow, CJM, Personas: URL: https://www.flowmapp.com/(дата звернення: 28.10.20).

18. Mandeep Kaur. SQL vs NoSQL : MySQL vs MongoDB — The Difference | by Mandeep Kaur | Medium: URL: https://medium.com/@mandeepkaur1/sql-vs-nosql-mysql-vs-mongodb-the-difference-6145e437cd40(дата звернення: 30.11.20).

19. Sai Vittal B. Introduction to MongoDB: URL: https://medium.com/@saivittalb/introduction-to-mongodb-859ed4426994(дата звернення: 30.11.20).

20. SHIVAM VARSHNEY. Node.js vs PHP for BackEnd: URL: https://medium.com/@varshney.shivam786/node-js-vs-php-for-backend-4078a3f65741(дата звернення: 30.11.20).

21. Jain, N., Mangal, P., Mehta, D. AngularJS: A Modern MVC Framework in JavaScript. Journal of Global Research in Computer Science. 2015. Vol. 5, No. 12.

22. Comparative analysis of angularjs and reactjs. International Journal of Latest Trends in Engineering and Technology. 2016. Vol. 7, No. 4.

23. Бродецкий, Г. Л. Системный анализ в логистике. Выбор в условиях неопределенности: Москва: 2010. 336с.

24. Khudov, H., Khizhnyak, I., Zots, F., та ін. The Bayes rule of decision making in joint optimization of search and detection of objects in technical systems. International Journal of Emerging Trends in Engineering Research. 2020. Vol. 8, No. 1.

25. Фрумкин, А. М. О задачах оптимизации показателя Гурвица для семейств многочленов. Ученые записки. Электронный научный журнал Курского государственного университета. 2008. Vol. 1, No. 1. С. 53–61.

26. OpenJS Foundation. Node.js: URL: https://nodejs.org/uk/(дата звернення: 01.12.20).

27. Borys, M., Panczyk, B. Improving data processing performance for web applications using object-relational mapping. Actual Problems of Economics. 2015. Vol. 164, No. 2.

ДОДАТОК А

Акт впровадження

ДОДАТОК Б
Планування робіт

Б.1 Ідентифікація мети IT-проекту

Деталізація мети проекту методом SMART. Мета проекту: створити інформаційну систему підтримки магазину освітлювальних приладів, яка зможе надавати функціонал онлайн продажів, формувати необхідну звітність за продажами та залишками, а також в кінці кожного місяця допомагати формувати план закупівель на наступний місяць. Мета є досяжною, підґрунтям створення програмного продукту є навички отримані на курсі з дисципліни Проектування веб-орієнтованих інформаційних систем, а також з дисципліни Дизайн сайтів. Проект буде виконано вчасно, що підтверджується календарним планом проекту. Результати деталізації методом SMART розміщені у табл. Б.1.

Таблиця Б.1 – Деталізація мети методом SMART

Specific (конкретна)	Створити інформаційну систему для підтримки магазину освітлювальних приладів
Measurable (вимірювана)	Результатом роботи проекту є оцінка замовника.
Achievable (досяжна)	Реалізації системи здійснюється за допомогою середовища розробки PhpStorm, з використанням мови програмування JavaScript
Relevant (реалістична)	У наявності є всі необхідні технічні та програмні засоби. Розробники достатньо кваліфіковані для виконання поставлених задач.
Time-framed (обмежена у часі)	Ціль має часове обмеження. Робота повинна бути виконана у терміни, що були оговорені замовником проекту. Проект повинен бути виконаний згідно з календарним планом.

Б.2 Планування змісту структури робіт

Структура розподілу робіт (WBS) є потужним інструментом управління проектами. Це наріжний камінь ефективного планування, виконання, контролю, формування статусу та складання звітності. Вся робота, яка міститься в WBS, повинна бути визначена, оцінена та запланована.

Побудуємо структуру WBS, у якій детально опишемо роботи, які потрібно виконати на кожному етапі створення проекту. Виконаємо декомпозицію робіт для даного проекту. Діаграма WBS зображена на рис. Б.1.

OBS визначає взаємозв'язок організації чи окремої людини з управлінням проектами, який часто використовується у поєднанні з WBS для визначення змісту роботи та відповідальності кожної організації чи окремої людини.

Діаграма OBS зображена на рис. Б.2. Список виконавців, що функціонують в проекті знаходиться в табл. Б.2.

Таблиця Б.2 – Виконавці проекту

Роль	Ім’я	Проектна роль
Виконавець	Тищенко Д. В.	Виконує розробку функціоналу проекту, інтерфейсу користувача та введення проекту в експлуатацію.
Керівник проекту	Ващенко С. М.	Приймає участь у розробленні мети проекту та її деталізації
Замовник		Формує проблему проекту та приймає здачу проекту в експлуатацію

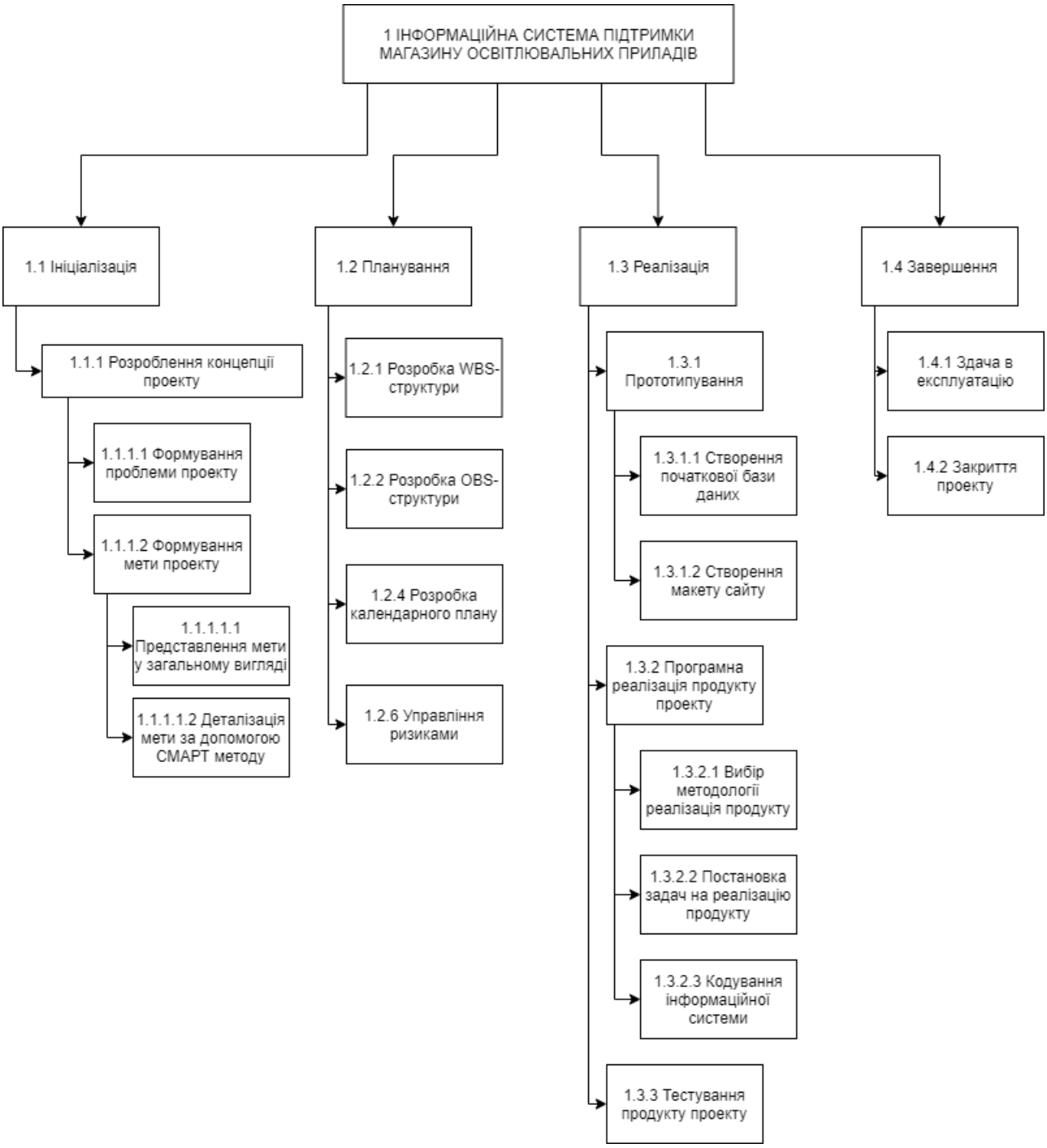


Рисунок Б.1 – WBS. Структура робіт проекту

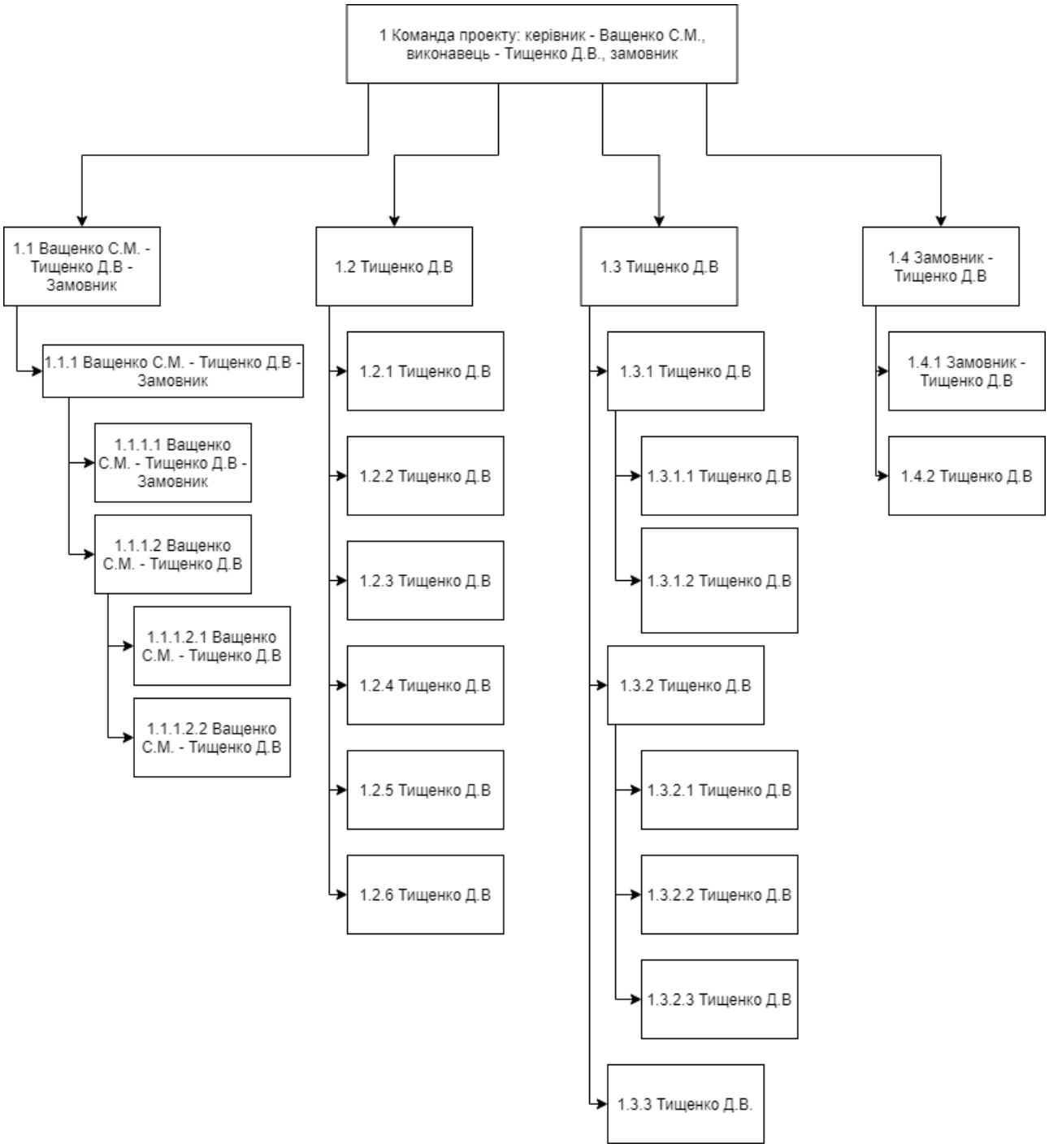


Рисунок Б.2 - Організаційна структура проекту (OBS)

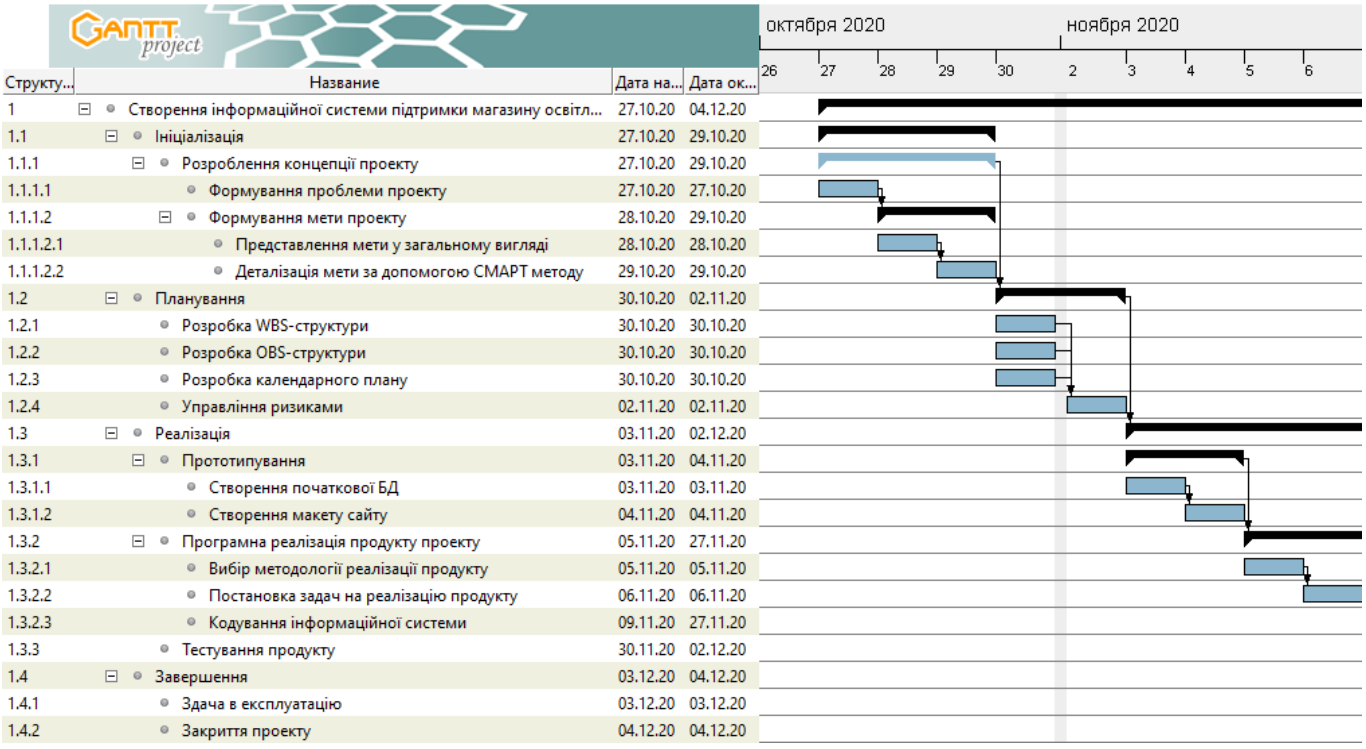
Б.3 Побудова календарного графіку виконання ІТ – проекту

Календарні плани проектів мають важливе значення для коректного та вчасного виконання робіт.

Це простий, проте ефективний інструмент, який використовується для візуалізації часу та ресурсів протягом даного проекту.

Діаграма Ганта - це графічне зображення діяльності, яка повинна виконуватися в ході реалізації проекту. Діаграма Ганта дає графічне представлення діяльності в проекті та надає пряме розуміння стану проекту стосовно часу пропускної здатності, залежностей та прогресу.

Для побудови діаграми Ганта використано програмний засіб GanttProject. На рисунках Б.3 - Б.4 представлено вигляд діаграми Ганта робіт щодо розробки інформаційної системи в рамках дослідження.



Структу...	Название	Дата на...	Дата ок...
1	Створення інформаційної системи підтримки магазину освітл...	27.10.20	04.12.20
1.1	Ініціалізація	27.10.20	29.10.20
1.1.1	Розроблення концепції проекту	27.10.20	29.10.20
1.1.1.1	Формування проблеми проекту	27.10.20	27.10.20
1.1.1.2	Формування мети проекту	28.10.20	29.10.20
1.1.1.2.1	Представлення мети у загальному вигляді	28.10.20	28.10.20
1.1.1.2.2	Деталізація мети за допомогою СМАРТ методу	29.10.20	29.10.20
1.2	Планування	30.10.20	02.11.20
1.2.1	Розробка WBS-структури	30.10.20	30.10.20
1.2.2	Розробка OBS-структури	30.10.20	30.10.20
1.2.3	Розробка календарного плану	30.10.20	30.10.20
1.2.4	Управління ризиками	02.11.20	02.11.20
1.3	Реалізація	03.11.20	02.12.20
1.3.1	Прототипування	03.11.20	04.11.20
1.3.1.1	Створення початкової БД	03.11.20	03.11.20
1.3.1.2	Створення макету сайту	04.11.20	04.11.20
1.3.2	Програмна реалізація продукту проекту	05.11.20	27.11.20
1.3.2.1	Вибір методології реалізації продукту	05.11.20	05.11.20
1.3.2.2	Постановка задач на реалізацію продукту	06.11.20	06.11.20
1.3.2.3	Кодування інформаційної системи	09.11.20	27.11.20
1.3.3	Тестування продукту	30.11.20	02.12.20
1.4	Завершення	03.12.20	04.12.20
1.4.1	Здача в експлуатацію	03.12.20	03.12.20
1.4.2	Закриття проекту	04.12.20	04.12.20

Рисунок Б.3 – Діаграма Ганта

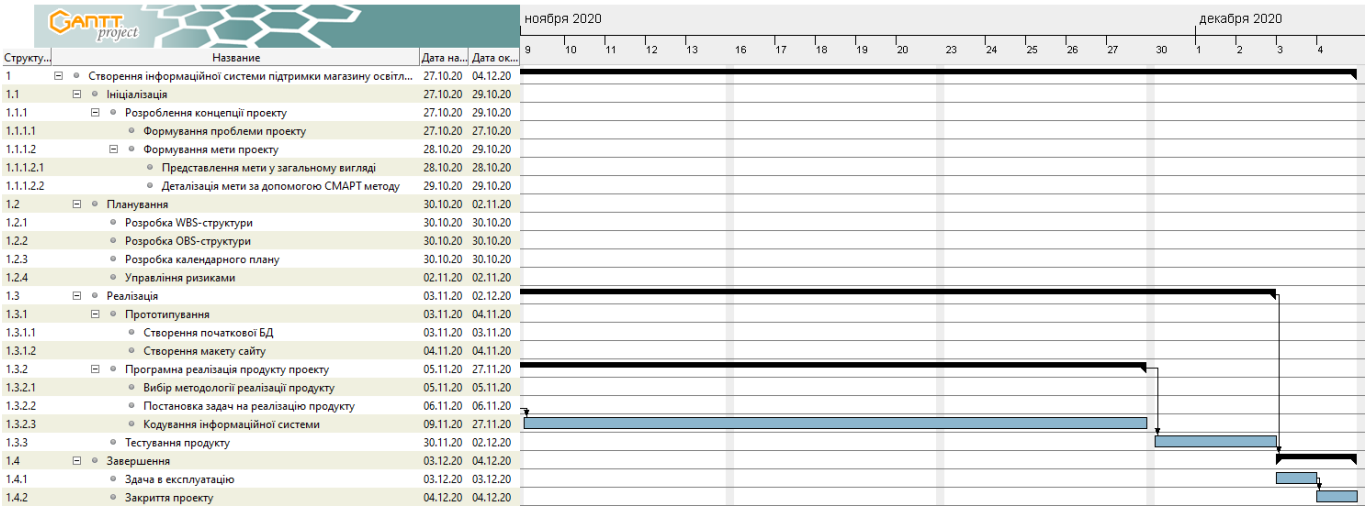


Рисунок Б.4 – Продовження діаграми Ганта

Б.4 Планування ризиків проекту

Розробка та функціонування інформаційних систем має значні ризики та невизначеності. Для мінімізації впливу та зменшення ймовірності таких ризиків застосовується проектний аналіз ризиків та процеси управління ризиками.

Для оцінки ризиків проекту необхідно виконати їх кількісну і якісну оцінку. Для оцінки ризиків будемо використовувати показники, що знаходяться у таблиці Б.3. У табл. Б.4 розміщена класифікація ризиків за показниками ймовірності виникнення ризику та величині втрат. На основі оцінки побудуємо матрицю ймовірності виникнення ризиків та їх впливу, у якій зеленим кольором позначено прийнятні ризики, жовтим – виправдані ризики, а червоним – недопустимі ризики. Дана матриця зображена на рисунку Б.5.

Таблиця Б.3 - Шкала оцінювання ймовірності виникнення та впливу ризику на виконання проекту

Оцінка	Ймовірність виникнення	Вплив ризику
1	Низька	Низький
2	Середня	Середній
3	Висока	Високий

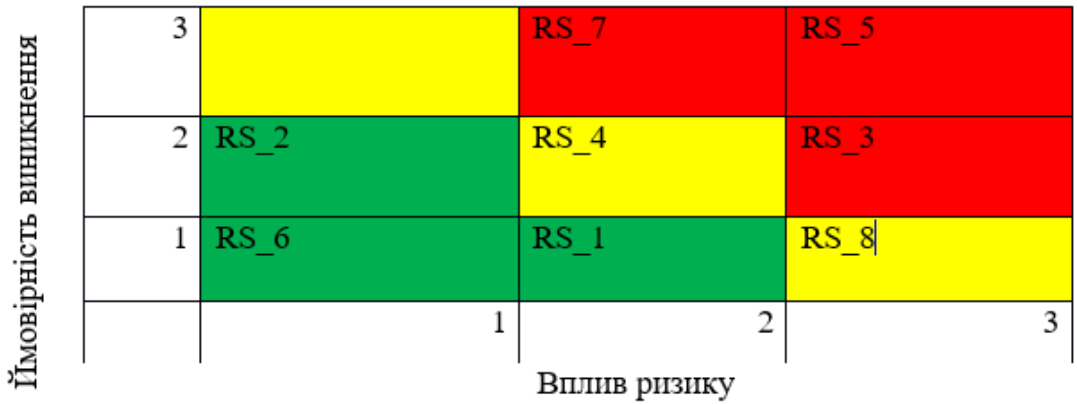


Рисунок Б.5 – Матриця ймовірності виникнення ризиків та впливу ризику

Тепер, створимо шкалу оцінювання за рівнем ризику на підставі отриманого значення індексу, що зображена у таблиці Б.4. Оцінка ймовірності виникнення, впливу і рангу ризику наведена у таблиці Б.5.

Таблиця 1.4 – Шкала оцінювання за рівнем ризику

№	Назва	Межі	Ризики, які входять(номера)
1	Прийнятні	1≤R≤2	1, 2, 6
2	Виправдані	3≤R≤4	4, 8
3	Недопустимі	6≤R≤9	3, 5, 7

Таблиця Б.5 − Оцінка ймовірності виникнення, впливу і рангу ризику.

ID	Статус ризику	Опис ризику	Ймовірність виникнення	Вплив ризику	Ранг ризику	План А	Тип стратегії реагування	План Б
RS_1	Відкритий	Непорозуміння між розробником та замовником	Низька	Середній	2	1. Дотримуватися ділового етикету спілкування. 2. Створити комфортні умови для співпраці	Попередження	При виявленні проблеми непорозуміння потрібно з’ясувати причину. Після знаходження причини її потрібно усунути
RS_2	Відкритий	Поява альтернативного продукту	Середній	Низька	2	Провести аналіз ринку на наявність альтернативних продуктів	Прийняття	
RS_3	Відкритий	Нечітке завдання на розробку	Середня	Високий	6	1. Ясно і однозначно обговорити із замовником усі види вимог і задач програмного продукту.	Попередження	Уважно та чітко все, що було виконано невірно та зробити правки.

Продовження таблиці Б.5

ID	Статус ризику	Опис ризику	Ймовірність виникнення	Вплив ризику	Ранг ризику	План А	Тип стратегії реагування	План Б
RS_4	Відкри-тий	Низька кваліфікація розробників проекту	Середня	Середній	4	1. Підвищити кваліфікацію персоналу. 2. Переглянути онлайн-ресурси для підвищення рівня знань.	Пом’якшення	Врахувати час на підготовку працівників. Видати літературу, переглянути онлайн-уроки.
RS_5	Відкритий	Неоптимальний розподіл часу	Висока	Високий	9	Провести аналіз актуальності найважливіших процесів та робіт, розрахувати час необхідний для їхнього виконання. Створити календарний план з урахуванням резервного строку виправлення помилок і чітко його дотримуватися	Пом’якшення	Змінити порядок пріоритетів робіт. Знайти способи оптимізації роботи із вже існуючою розстановкою. Обговорити варіанти внесення правок до термінів реалізації із замовником.

Продовження таблиці Б.5

ID	Статус ризику	Опис ризику	Ймовірність виникнення	Вплив ризику	Ранг ризику	План А	Тип стратегії реагування	План Б
RS_6	Відкритий	Не вірна оцінка масштабів проекту	Низька	Низька	1	Провести детальний аналіз проекту. Визначити основні етапу проекту, розподілити час на їх виконання. Проаналізувати масштаби проекту на основі додаткових джерел.	Пом’якшення	Переоцінка масштабів проекту. Перебудова стратегії реалізації проекту.
RS_7	Відкритий	Помилки розробки	Високий	Середній	6	На етапі розробки тісно співпрацювати із замовником та на певних етапах демонструвати поточні результати.	Пом’якшення	Здійснювати проміжний контроль результатів в ході виконання проекту.
RS_8	Відкритий	Збої в роботі програмного забезпечення	Низька	Високий	3	Залучити спеціаліста для усунення збоїв.	Попередження	Замінити програмне забезпечення.

ДОДАТОК В

Структурно – функціональна модель

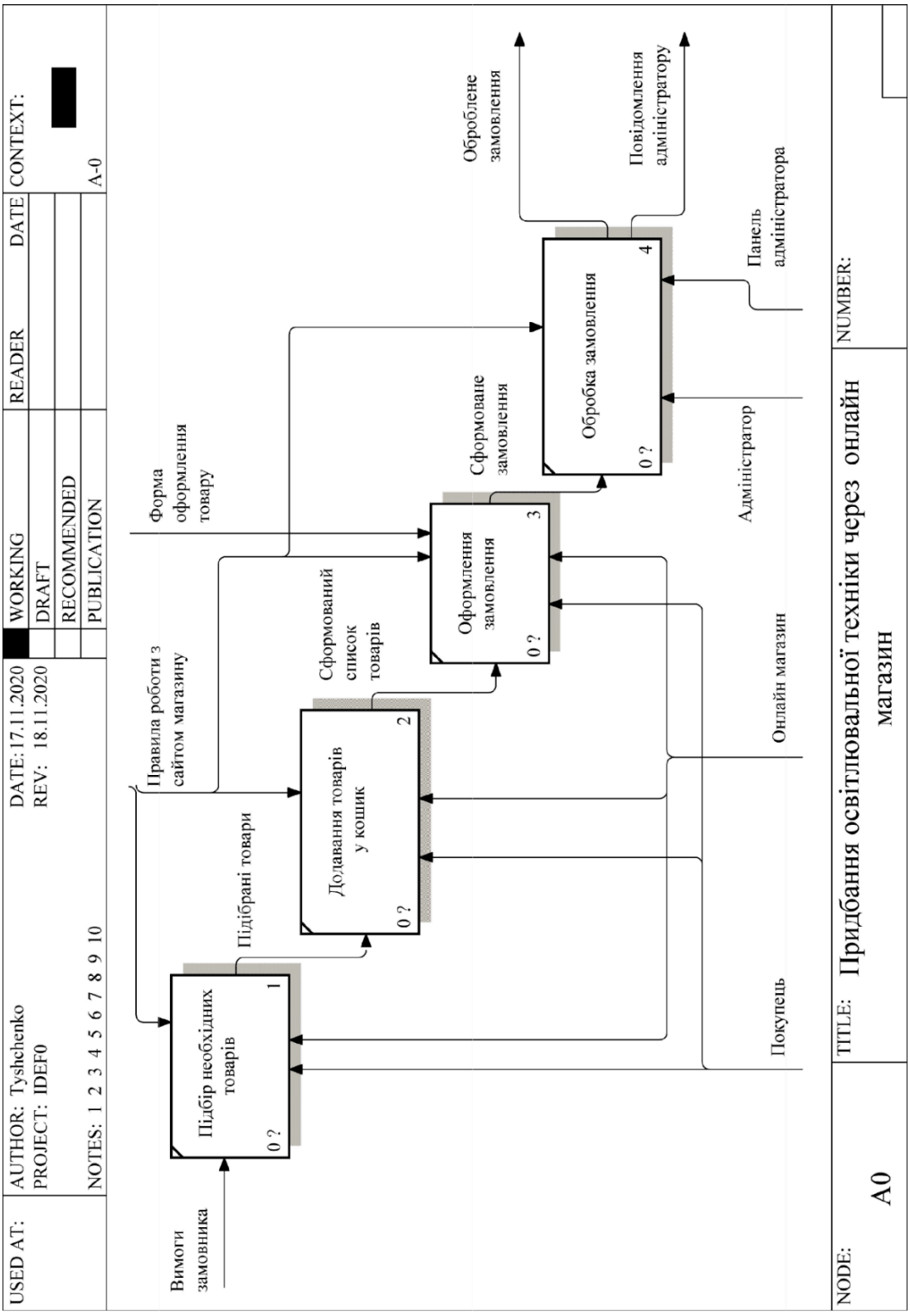


Рисунок В.1 – Діаграма декомпозиції першого рівня

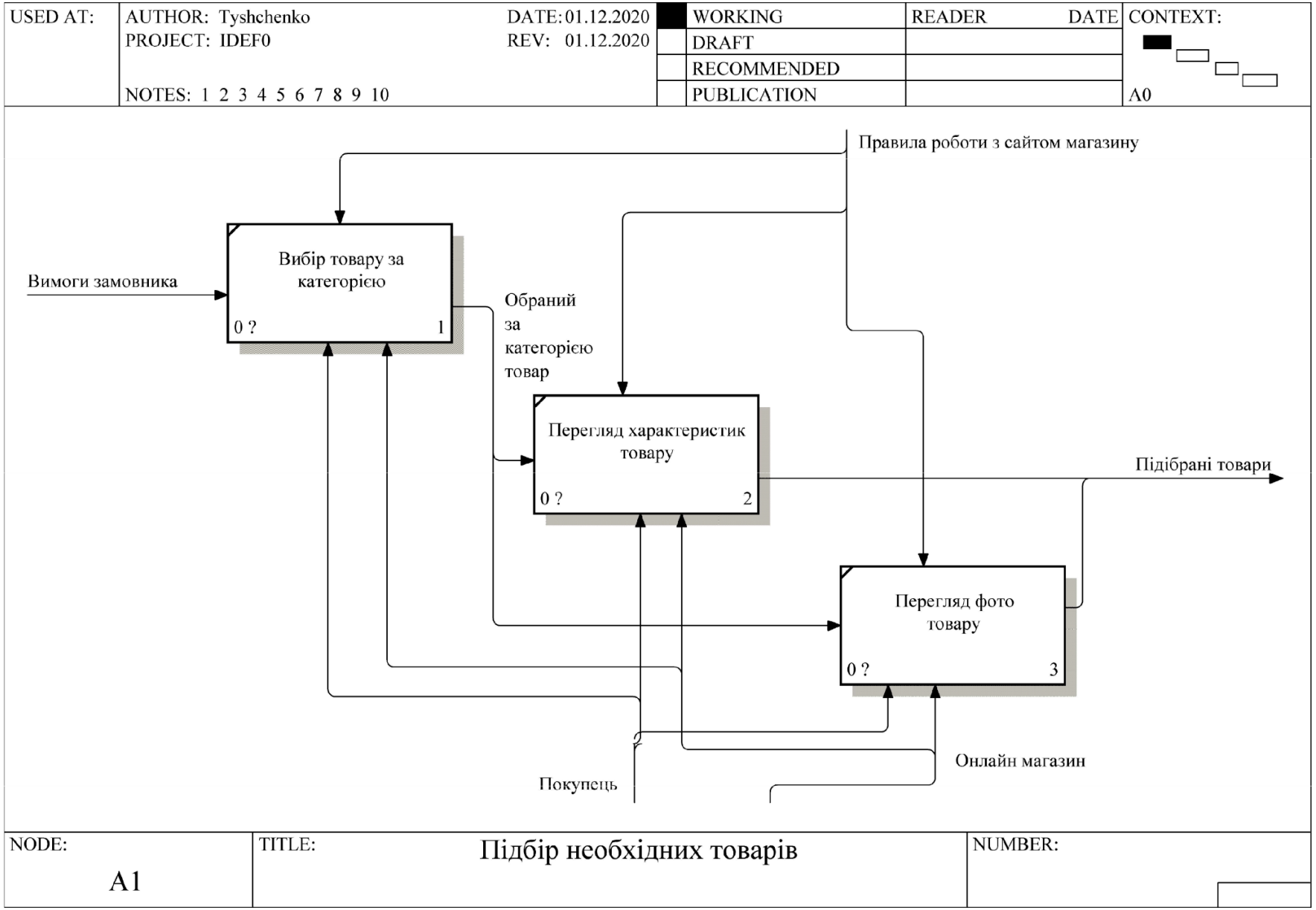


Рисунок В.2 – Діаграма декомпозиції другого рівня

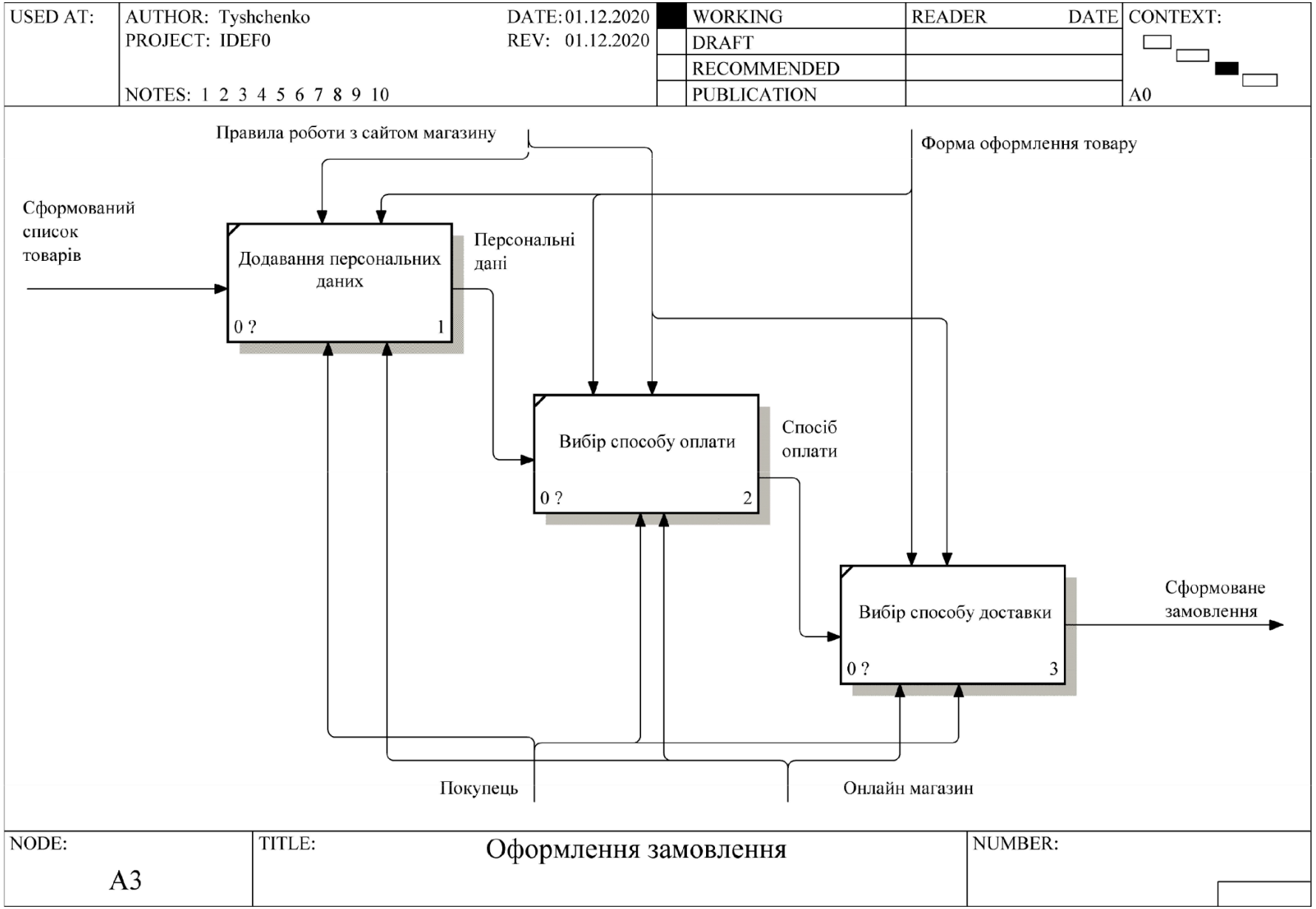


Рисунок В.3 – Діаграма декомпозиції другого рівня

ДОДАТОК Г

Лістинг програмного коду

Код серверної частини
Функція dss (формування списку закупівель)

```
module.exports = (soldedProducts) => {
   if(!soldedProducts) return new Error('Undefined input data!')
   const productCount = Object.keys(soldedProducts).length < 5 ? Object.keys(soldedProducts).length : 5;
   let productIdsToPurchase = [];
   for(let k = 0; k < productCount; k++) {
      let vald = { value: 0 };
      for (let key in soldedProducts) {
         if(vald.value < Math.min(soldedProducts[key][0], soldedProducts[key][1])) {
            vald.id = key;
            vald.value = Math.min(soldedProducts[key][0], soldedProducts[key][1]);
         }
      }
      let bayes = { value: -1 };
      for (let key in soldedProducts) {
         if(bayes.value < soldedProducts[key][0] * 0.3 + soldedProducts[key][1]*0.7) {
            bayes.id = key;
            bayes.value = soldedProducts[key][0] * 0.3 + soldedProducts[key][1]*0.7;
         }
      }
      let hurviz = { s: 0 };
      for (let key in soldedProducts) {
         const minA = Math.min(soldedProducts[key][0], soldedProducts[key][1]);
         const maxA = Math.max(soldedProducts[key][0], soldedProducts[key][1]);
         if (hurviz.s < 0.3 * minA + 0.7 * maxA) {
            hurviz.id = key;
            hurviz.s = 0.3 * minA + 0.7 * maxA;
         }
      }
      const criteriesIds = [vald.id, bayes.id, hurviz.id];
      let mf = 1;
      let m = 0;
      let item;
      for (let i=0; i<criteriesIds.length; i++)
      {
         for (let j=i; j<criteriesIds.length; j++)
         {
            if (criteriesIds[i] == criteriesIds[j])
               m++;
            if (mf<m)
```

```
                {
                    mf=m;
                    item = criteriesIds[i];
                }
            }
            m=0;
        }
        productIdsToPurchase.push(item);
        delete soldedProducts[item];
    }
    return productIdsToPurchase;
}
```

Контроллер формування звітності generateDocx

```
const dayjs = require('dayjs');
const numberToString = require('number-to-cyrillic');
require('dayjs/locale/uk');
const PizZip = require('pizzip');
const Docxtemplater = require('docxtemplater');
const BadRequestException = require('./exceptions/BadRequestException');
const InternalErrorException = require('./exceptions/InternalErrorException');
const fs = require('fs');
const path = require('path');
// The error object contains additional information when logged with JSON.stringify (it contains a properties object containing all suberrors).
function replaceErrors(key, value) {
    if (value instanceof Error) {
        return Object.getOwnPropertyNames(value).reduce(function(error, key) {
            error[key] = value[key];
            return error;
        }, {});
    }
    return value;
}
function errorHandler(error) {
    console.log(JSON.stringify({error: error}, replaceErrors));
    if (error.properties && error.properties.errors instanceof Array) {
        const errorMessages = error.properties.errors.map(function (error) {
            return error.properties.explanation;
        }).join("\n");
        console.log('errorMessages', errorMessages);
        // errorMessages is a humanly readable message looking like this :
        // 'The tag beginning with "foobar" is unopened'
    }
    throw error;
}
async function createDocx(myObj) {
    const target = myObj.product;
```

```
    const getCountIds = target => {
        const result = new Object;
        target.forEach(item => result[item.id] ? result[item.id]++ : result[item.id] = 1)

        return Object.keys(result).map(item => {
            return {
                id: parseInt(item),
                sum: result[item]
            }
        })
    }
    const countIds = getCountIds(target);
    countIds.forEach(item => {
        target.forEach(product => {
            if(item.id === product.id){
                item.name = product.name;
                item.code = product.code;
                item.price = product.price;
            }
        })
    })
    let i = 1;
    countIds.forEach(item => {
        item.totalPrice = item.sum * item.price;
        item.i = i;
        i++;
    });
    const dataForDocx = {};
    dataForDocx.date = dayjs().locale('uk').format('D MMMM YYYY');
    dataForDocx.customerfullName = myObj.customer.lastName + ' ' + myObj.customer.firstName;
    dataForDocx.customerPhone = myObj.customer.phone;
    dataForDocx.totalPrice = myObj.totalPrice;
    const totalPriceWords = numberToString.convert(myObj.totalPrice.toString());
    dataForDocx.totalPriceWords = totalPriceWords.convertedInteger.charAt(0).toUpperCase() +
        totalPriceWords.convertedInteger.slice(1) + ' ' +
        totalPriceWords.integerCurrency + ' ' +
        totalPriceWords.convertedFractional + ' ' +
        totalPriceWords.fractionalCurrency;
    dataForDocx.productsCount = countIds.length;
    dataForDocx.products = countIds;
//Load the docx file as a binary
    const content = fs
        .readFileSync(path.resolve('./docxTemplates/invoiceTemplate.docx'), 'binary');
    const zip = new PizZip(content);
    let doc;
    try {
        doc = new Docxtemplater(zip);
    } catch(error) {
```

```
        // Catch compilation errors (errors caused by the compilation of the template : misplaced tags)
        errorHandler(error);
    }
//set the templateVariables
    doc.setData(dataForDocx);
    try {
        // render the document (replace all occurences of {first_name} by John, {last_name} by Doe, ...)
        doc.render()
    }
    catch (error) {
        // Catch rendering errors (errors relating to the rendering of the template : angularParser throws an
error)
        errorHandler(error);
    }
    var buf = doc.getZip()
        .generate({type: 'nodebuffer'});
// buf is a nodejs buffer, you can either write it to a file or do anything else with it.
    fs.writeFileSync(path.resolve('./docxTemplates/output.docx'), buf);
}
module.exports = {
    generateInvoice: async (req, res, next) => {
        if(!req.body) {
            return next(new BadRequestException());
        }
        await createDocx(req.body);
        const options = {
            root: path.join(__dirname, '/docxTemplates'),
            dotfiles: 'deny',
            headers: {
                'x-timestamp': Date.now(),
                'x-sent': true,
            }
        }
        const fileName = 'output.docx';
        return res.sendFile(fileName, options, function (err) {
            if (err) {
                next(new InternalErrorException());
            }
        });
    },
};
```

Контроллер продуктів

```
const { Product } = require('./../database.js');
const { sequelize } = require('./../database.js');
const NotFoundException = require('./../exceptions/NotFoundException');
const BadRequestException = require('./../exceptions/BadRequestException');
const UnauthorizedException = require('./../exceptions/UnauthorizedException');
```

```
const DSS = require('../dss');

module.exports = {
    list: async (req, res) => {
        const data = await Product.findAll({
            include: [
                'category',
            ],
        });
        return res.status(200).send({
            data,
        });
    },
    item: async (req, res, next) => {
        const { id } = req.params;
        const item = await Product.findByPk(id, {
            include: [
                'comments',
                'category',
            ],
        });
        if (!item) {
            return next(new NotFoundException());
        }
        return res.status(200).send({
            item,
        });
    },
    update: async (req, res, next) => {
        if (!req.user) {
            return next(new UnauthorizedException());
        }
        const { id } = req.params;
        const item = await Product.findByPk(id);
        if (!item) {
            return next(new NotFoundException());
        }
        const data = req.body;
        data.image = req.file ? `${req.file.destination}/${req.file.filename}` : null;
        try {
            await item.update(data);
        }catch (error) {
            return next(new BadRequestException(error));
        }
        return res.status(204).send();
    },
    delete: async (req, res, next) => {
        if (!req.user) {
```

```
            return next(new UnauthorizedException());
        }
        const { id } = req.params;
        const item = await Product.findByPk(id);
        if (!item) {
            return next(new NotFoundException());
        }
        await item.destroy();
        return res.status(204).send();
    },
    create: async (req, res, next) => {
        if (!req.user) {
            return next(new UnauthorizedException());
        }
        try {
            const data = req.body;
            data.image = req.file ? `${req.file.destination}/${req.file.filename}` : null;
            await Product.create(data);
        } catch (error) {
            return next(new BadRequestException(error));
        }
        return res.status(201).send();
    },
    dss: async (req, res, next) => {
        const { id } = req.params;
        const soldedProducts = {};
        sequelize.query(`select p.id, p.name, p.code, p.price, p.count, count(*) as solded from products p join productorders po on po.productId = p.id join orders o on o.id = po.orderId where p.categoryId = ${id} group by p.code order by p.code`, { type: sequelize.QueryTypes.SELECT})
            .then(async function(products) {
                products.map(product => soldedProducts[`${product.id}`] = [
                    product.count,
                    product.solded,
                ]);
                let productIdsToPurchase;
                try {
                    productIdsToPurchase = DSS(soldedProducts);
                } catch (err){
                    next(BadRequestException(err));
                }
                let purchases = [];
                productIdsToPurchase.map(item => {
                    products.map((prod) => {
                        if (item === prod.id.toString()){
                            purchases.push(prod);
                        }
                    })
                });
                return res.status(201).send({
```

```
                item: purchases,
            });
        });
    },
};
```

Модель продукту

```
module.exports = (sequelize, DataTypes) => {
    return sequelize.define('product', {
        name: {
            type: DataTypes.STRING,
            allowNull: false,
            validate: {
                notNull: true,
                notEmpty: true,
                len: [
                    2,
                    255,
                ],
            },
        },
        description: {
            type: DataTypes.TEXT,
            allowNull: false,
            validate: {
                notNull: true,
                notEmpty: true,
            },
        },
        code: {
            type: DataTypes.TEXT,
            allowNull: false,
            validate: {
                notNull: true,
                notEmpty: true,
                len: [
                    3,
                    50,
                ],
            },
        },
        price: {
            type: DataTypes.FLOAT,
            allowNull: false,
            validate: {
                notNull: false,
                notEmpty: true,
            },
```

```
        },
        promotionalPrice: {
            type: DataTypes.FLOAT,
            allowNull: true,
        },
        availability: {
            type: DataTypes.ENUM('В наявності', 'Немає в наявності'),
            allowNull: false,
            validate: {
                notNull: true,
                notEmpty: true,
            },
        },
        count: {
            type: DataTypes.INTEGER,
            allowNull: false,
            validate: {
                notNull: true,
                notEmpty: true,
            },
        },
        image: {
            type: DataTypes.TEXT,
            allowNull: false,
            validate: {
                notNull: true,
                notEmpty: true,
            },
        },
    })
};
```

Підключення до бази даних

```
const Sequelize = require('sequelize');
const config = require('./config/config.json');
const User = require('./models/user');
const Article = require('./models/article');
const Category = require('./models/category');
const Product = require('./models/product');
const Customer = require('./models/customer');
const Order = require('./models/order');
const ProductOrder = require('./models/productOrder');
const Region = require('./models/region');
const sequelize = new Sequelize(
    config.connection.database,
    config.connection.username,
    config.connection.password,
    {
```

```
        host: config.connection.options.host,
        dialect: config.connection.options.dialect,
        logging: config.connection.options.logging,
        define: {
            timestamps: false
        }
    }
);
const models = {
    User: User(sequelize, Sequelize),
    Article: Article(sequelize, Sequelize),
    Category: Category(sequelize, Sequelize),
    Product: Product(sequelize, Sequelize),
    Customer: Customer(sequelize, Sequelize),
    Order: Order(sequelize, Sequelize),
    ProductOrder: ProductOrder(sequelize, Sequelize),
    Region: Region(sequelize, Sequelize),
};
models.Category.hasOne(models.Product, {
    as: 'category',
    foreignKey: 'categoryId',
});
models.Product.belongsTo(models.Category);
models.Category.hasMany(models.Category, {
    as: 'child',
    foreignKey: 'parentId'
});
models.Customer.hasMany(models.Order, {
    as: 'customer',
    foreignKey: 'customerId',
});
models.Order.belongsTo(models.Customer);
models.Region.hasOne(models.Customer, {
    as: 'region',
    foreignKey: 'regionId',
});
models.Customer.belongsTo(models.Region);
models.Product.belongsToMany(models.Order, {
    as: 'order',
    through: {
        model: models.ProductOrder,
        unique: false,
    },
    foreignKey: {
        primaryKey: true,
        fieldName: 'productId'
    },
});
models.Order.belongsToMany(models.Product, {
```

```
    as: 'product',
    through: {
        model: models.ProductOrder,
        unique: false,
    },
    foreignKey: {
        primaryKey: true,
        fieldName: 'orderId',
    },
});
module.exports = {
    ...models,
    sequelize,
};
```

Роутер

```
const express = require('express');
const router = express.Router();
const multer = require('multer');
let storage = multer.diskStorage({
    destination: function (req, file, cb) {
        cb(null, './storage')
    },
    filename: function (req, file, cb) {
        cb(null, file.originalname + '-' + Date.now());
    }
});
const upload = multer({
    storage,
});
const AuthController = require('./controllers/AuthController');
const ArticleController = require('./controllers/ArticleController');
const CategoryController = require('./controllers/CategoryController');
const ProductController = require('./controllers/ProductController');
const OrderController = require('./controllers/OrderController');
const CustomerController = require('./controllers/CustomerController');
const GenerateDocx = require('./generateDocx');
const auth = require('./middlewares/auth');
router.get('/', (req, res) => {
    return res.send({
        code: 200,
        message: 'ok',
    });
});
router.post('/login', AuthController.login);

router.get('/articles', ArticleController.list);
router.post('/articles', upload.single('image'), auth, ArticleController.create);
```

```
router.get('/articles/:id', ArticleController.item);
router.put('/articles/:id', upload.single('image'), auth, ArticleController.update);
router.delete('/articles/:id', auth, ArticleController.delete);

router.get('/categories', CategoryController.list);
router.post('/categories', auth, CategoryController.create);
router.get('/categories/:id', CategoryController.item);
router.put('/categories/:id', auth, CategoryController.update);
router.delete('/categories/:id', auth, CategoryController.delete);

router.get('/products', ProductController.list);
router.post('/products', upload.single('image'), auth, ProductController.create);
router.get('/products/:id', ProductController.item);
router.put('/products/:id', upload.single('image'),auth, ProductController.update);
router.delete('/products/:id', auth, ProductController.delete);

router.get('/customers', CustomerController.list);
router.post('/customers', auth, CustomerController.create);
router.get('/customers/:id', CustomerController.item);
router.put('/customers/:id', auth, CustomerController.update);
router.delete('/customers/:id', auth, CustomerController.delete);

router.get('/orders', OrderController.list);
router.post('/orders', auth, OrderController.create);
router.get('/orders/:id', OrderController.item);
router.put('/orders/:id', auth, OrderController.update);
router.delete('/orders/:id', auth, OrderController.delete);

router.put('/generateInvoice', auth, GenerateDocx.generateInvoice);
router.get('/dss/:id', auth, ProductController.dss);

module.exports = router;
```

Головний файл панелі адміністратора

```
import React, { PureComponent } from 'react';
import { Switch, Route, withRouter } from "react-router-dom";
import { connect } from 'react-redux';

import TopBar from './components/TopBar';
import MenuSidebar from "./components/MenuSidebar";

import Orders from "./pages/Orders";
import Products from "./pages/ProductsList";
import ProductItem from "./pages/ProductsItem";
import Login from "./pages/Login";
import Articles from "./pages/ArticlesList";
import ArticlesItem from "./pages/ArticlesItem";
import PurchasesFormation from "./pages/PurchasesFormation";
```

```
import { logout } from "./actions/login";
import { loadOrders } from "./actions/loadOrders";

class App extends PureComponent {
   checkAuth = () => {
      const { isLoggedIn, location: { pathname }, history } = this.props;

      if (!isLoggedIn && pathname !== '/login') {
         history.push('/login');
      } else if (isLoggedIn && pathname === '/login') {
         history.push('/');
      }
   }

   componentDidMount() {
      this.checkAuth();
      this.props.loadOrders();
   };

   componentDidUpdate(prevProps, prevState, snapshot) {
      this.checkAuth();
   }

   render() {
      const { isLoggedIn, user, logout, orders } = this.props;
      const newOrders = orders.filter((order) => order.status === 'Новый');
      return (
         <>
            {isLoggedIn && <TopBar user={ user } logout={ logout } orders={ newOrders }/>}
            {isLoggedIn && <MenuSidebar/>}
            <Switch>
               <Route exact path="/" component={ Orders }/>
               <Route exact path="/orders" component={ Orders }/>
               <Route exact path="/products" component={ Products }/>
               <Route exact path="/products/:id" component={ ProductItem }/>
               <Route exact path="/articles" component={ Articles }/>
               <Route exact path="/articles/:id" component={ ArticlesItem }/>
               <Route exact path="/login" component={ Login }/>
               <Route exact path="/purchasesFormation" component={ PurchasesFormation }/>
            </Switch>
         </>
      );
   };
}

export default withRouter(connect(
   (store) => ({
      isLoggedIn: store.auth.isLoggedIn,
      user: store.auth.user,
```

```
        orders: store.order.list,
    }),
    (dispatch) => ({
        logout: () => dispatch(logout()),
        loadOrders: () => dispatch(loadOrders()),
    }),
)(App));
```

Компонент навігації інтернет-магазину

```
import React, { PureComponent } from "react";
import { NavLink } from 'react-router-dom';

export default class Menu extends PureComponent {
    componentDidMount() {
        this.props.checkAuth();
    }

    componentDidUpdate() {
        this.props.checkAuth();
    }

    render() {
        const {userIsLoggin, user, logout} = this.props;
        let finalTabs;
        if (userIsLoggin) {
            finalTabs = <>
                <li>
                    <NavLink exact to="/login">
                        <p className="nav-item">Вход</p></NavLink>
                </li>
                <li>
                    <NavLink exact to="/register">
                        <p className="nav-item">Регистрация</p></NavLink>
                </li>
            </>;
        } else {
            finalTabs = <>
                <li>
                    <NavLink exact to="/account">
                        <p className="nav-item">{user.name}</p></NavLink>
                </li>
                <li>
                    <NavLink exact to="/logout">
                        <button type="button" className="nav-item"> onClick={ logout }>
                            Выход</button> </NavLink>
                </li>
            </>;
        }
```

```
    return (
      <nav className="nav d-none d-lg-block">
        <nav className="navbar">
          <ul className="list-unstyled navbar__list">
            <li>
              <NavLink exact to="/">
                <p className="nav-item">Главная</p></NavLink>
            </li>
            <li>
              <NavLink exact to="/shares">
                <p className="nav-item">Акции</p></NavLink>
            </li>
            <li>
              <NavLink exact to="/news">
                <p className="nav-item">Новости</p></NavLink>
            </li>
            <li>
              <NavLink exact to="/articles">
                <p className="nav-item">Статьи</p></NavLink>
            </li>
            <li>
              <NavLink exact to="/delivery">
                <p className="nav-item">Доставка и оплата</p></NavLink>
            </li>
            <li>
              <NavLink exact to="/about">
                <p className="nav-item">О магазине</p></NavLink>
            </li>
            <li>
              <NavLink exact to="/contacts">
                <p className="nav-item">Контакты</p></NavLink>
            </li>
            {
              finalTabs
            }
          </ul>
        </nav>
      </nav>
    );
  };
}
```