

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ
ПЛАНУВАННЯ ПОКУПОК ПРОДУКТІВ ДЛЯ ХАРЧУВАННЯ СІМ'Ї
DEVELOPMENT OF SOFTWARE FOR PLANNING
FOOD PURCHASES FOR A FAMILY

Виконав(ла): студент(ка) 4 курсу, групи КНТ-132

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

КОЛЯДА М.І.

(ПРИЗВИЩЕ та ініціали)

Керівник ПОЛЯКОВ М.О.

(ПРИЗВИЩЕ та ініціали)

Рецензент БАБЕНКО Н.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Факультет КНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
 (КОД І НАЙМЕНУВАННЯ)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2026 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

КОЛЯДИ Микити Івановича

(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розроблення програмного забезпечення для
планування покупок продуктів для харчування сім'ї.

Development of Software for Planning Food Purchases for a Family

керівник проєкту (роботи) д.т.н, професор, ПОЛЯКОВ Михайло Олексійович ,

(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 07 ” квітня 2026 року № 139

2. Строк подання студентом проєкту (роботи) 01 червня 2026 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне
завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Огляд та аналіз технологій розробки програмного продукту.
2. Вибір технологій та проектування програмного забезпечення. 3. Розробка та
тестування програмного забезпечення.

5. Перелік графічного матеріалу (з з точним зазначенням обов'язкових креслень,
 кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ПОЛЯКОВ М.О., професор		
Нормоконтроль	БСЛОВА А.В., асистент		

7. Дата видачі завдання « 20 » квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)


(підпис)

Микита КОЛЯДА
(Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

(підпис)

Михайло ПОЛЯКОВ
(Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
79 с., 8 табл., 28 рис., 2 дод., 34 джерел.

ВЕБЗАСТОСУНОК, ПЛАНУВАННЯ ПОКУПОК, СПИСОК
ПРОДУКТІВ, АВТОМАТИЗАЦІЯ, REACT.

Об'єкт дослідження – процес планування покупок продуктів харчування сім'ї.

Предмет дослідження – програмне забезпечення для автоматизації планування, ведення та контролю списків сімейних покупок продуктів харчування.

Мета роботи – розробка програмного забезпечення для планування покупок продуктів харчування сім'ї, яке забезпечує зручне створення, редагування, перегляд і контроль списків покупок.

Матеріали, методи та технічні засоби: мова програмування TypeScript, бібліотека для створення користувацьких інтерфейсів React, фреймворк для створення дизайну інтерфейсу Tailwind CSS, технологія localStorage для збереження даних, персональний комп'ютер з процесором AMD Ryzen 5 5600 під управлінням операційної системи Microsoft Windows 11.

Результати. Створено програмне забезпечення для планування покупок продуктів харчування сім'ї, яке реалізує основні функції роботи зі списками покупок, продуктами та категоріями, а також забезпечує зручну взаємодію користувача із системою

Висновки. Розроблено програмний продукт, який може бути використаний сім'ями для організації щоденних або періодичних покупок продуктів харчування

Галузь використання – організація та планування покупок продуктів харчування

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 79 pages, 8 tables, 28 figures, 2 appendix, 34 sources.

WEB APPLICATION, SHOPPING PLANNING, PRODUCT LIST, AUTOMATION, REACT.

Object of study is a process of planning family grocery shopping.

Subject of study is a software for automating the planning, maintenance, and control of family grocery shopping lists.

The purpose of the work the development of software for planning family grocery shopping that provides convenient creation, editing, viewing, and control of shopping lists.

Materials, methods, and technical tools: TypeScript programming language, React user interface library, Tailwind CSS interface design framework, localStorage technology for data storage, personal computer with AMD Ryzen 5 5600 processor running Microsoft Windows 11 operating system.

Results. Software for planning family grocery shopping has been developed, implementing the main functions for working with shopping lists, products, and categories, as well as providing convenient user interaction with the system.

Conclusions. A software product has been developed that can be used by families to organize daily or periodic grocery shopping.

The field of use is a organization and planning of grocery shopping.

ЗМІСТ

	С.
Вступ.....	7
1 Огляд та аналіз технологій розробки програмного продукту	9
1.1 Аналіз предметної області планування покупок продуктів харчування сім'ї	9
1.2 Огляд існуючих програмних рішень для планування покупок.....	12
1.3 Визначення вимог до програмного забезпечення.....	15
1.4 Постановка задачі дипломної роботи	18
2 Вибір технологій та проєктування програмного забезпечення.....	21
2.1 Обґрунтування вибору технологій розробки	21
2.2 Проєктування архітектури програмного забезпечення.....	25
2.3 Проєктування бази даних	29
2.4 Розробка алгоритмів роботи програмного забезпечення.....	33
3 Розробка та тестування програмного забезпечення	38
3.1 Розробка основних модулів програмного забезпечення.....	38
3.2 Розробка інтерфейсу користувача	42
3.3 Тестування програмного забезпечення.....	46
3.4 Аналіз результатів роботи програмного забезпечення	53
Висновки	57
Перелік джерел посилання	60
ДОДАТОК А. Лістинг Коду.....	63
ДОДАТОК Б. Слайди презентації	74

ВСТУП

У сучасних умовах організація повсякденного побуту дедалі частіше пов'язана з використанням цифрових інструментів. Одним із таких напрямів є планування покупок продуктів харчування для сім'ї. На перший погляд це завдання здається простим, однак на практиці воно охоплює багато факторів: склад сім'ї, індивідуальні харчові потреби, бюджет, наявні продукти вдома, частоту закупівель, терміни придатності товарів, а також необхідність уникнення зайвих витрат.

Традиційний підхід до формування списку покупок часто базується на паперових нотатках, повідомленнях у месенджерах або пам'яті користувача. Такий спосіб є недостатньо зручним, оскільки інформація може дублюватися, втрачатися або швидко втрачати актуальність. Крім того, члени сім'ї не завжди мають спільний доступ до актуального списку, що призводить до неузгоджених покупок, придбання зайвих товарів або, навпаки, пропуску необхідних продуктів.

Актуальність теми дипломної роботи зумовлена потребою у створенні зручного програмного забезпечення, яке дозволить автоматизувати процес планування сімейних закупівель продуктів харчування. Такий застосунок має забезпечувати формування списків покупок, редагування позицій, категоризацію продуктів, контроль виконання покупок та збереження інформації для подальшого використання. Використання подібної системи може підвищити організованість побутових процесів, зменшити витрати часу на планування та зробити закупівлі більш раціональними.

Об'єктом дослідження є процес планування покупок продуктів харчування сім'ї.

Предметом дослідження є програмне забезпечення для автоматизації планування, ведення та контролю списків сімейних покупок продуктів харчування.

Метою роботи є розробка програмного забезпечення для планування покупок продуктів харчування сім'ї, яке забезпечує зручне створення, редагування, перегляд і контроль списків покупок.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати предметну область планування сімейних покупок продуктів харчування;
- розглянути існуючі програмні рішення та визначити їх переваги й недоліки;
- сформулювати функціональні та нефункціональні вимоги до програмного забезпечення;
- обґрунтувати вибір технологій розробки;
- спроектувати архітектуру програмного забезпечення та структуру бази даних;
- реалізувати основні модулі програмного забезпечення;
- розробити користувацький інтерфейс;
- провести тестування програмного забезпечення та проаналізувати результати його роботи.

Практичне значення роботи полягає у створенні програмного продукту, який може бути використаний сім'ями для організації щоденних або періодичних покупок продуктів харчування. Запропоноване рішення дозволяє спростити процес складання списків, покращити взаємодію між користувачами та зменшити ймовірність помилок під час закупівель.

У процесі виконання роботи використовуються методи аналізу предметної області, об'єктно-орієнтованого проєктування, моделювання структури бази даних, програмної реалізації клієнтської та серверної частин, а також функціонального тестування розробленого програмного забезпечення.

Результатом дипломної роботи є програмне забезпечення для планування покупок продуктів харчування сім'ї, яке реалізує основні функції роботи зі списками покупок, продуктами та категоріями, а також забезпечує зручну взаємодію користувача із системою.

1 ОГЛЯД ТА АНАЛІЗ ТЕХНОЛОГІЙ РОЗРОБКИ ПРОГРАМНОГО ПРОДУКТУ

1.1 Аналіз предметної області планування покупок продуктів харчування сім'ї

Планування покупок продуктів харчування є звичайною частиною побуту, але на практиці цей процес не завжди такий простий, як здається. Для однієї людини список покупок може складатися з кількох позицій, які легко запам'ятати. Проте для сім'ї ситуація ускладнюється, оскільки потрібно враховувати потреби кількох людей, наявність продуктів удома, бюджет, частоту покупок і навіть те, у якому магазині зручніше купувати певні товари.

У повсякденному житті список покупок часто формується не за один раз. Наприклад, хтось помічає, що закінчилося молоко, інший член сім'ї просить купити фрукти, а перед самим походом до магазину додаються ще хліб, крупи або побутові товари. Якщо все це записується в різних місцях або тримається в пам'яті, частина продуктів легко забувається. Саме тому для такої задачі корисним є простий цифровий інструмент, який дозволяє швидко фіксувати потрібні товари (рис. 1.1).

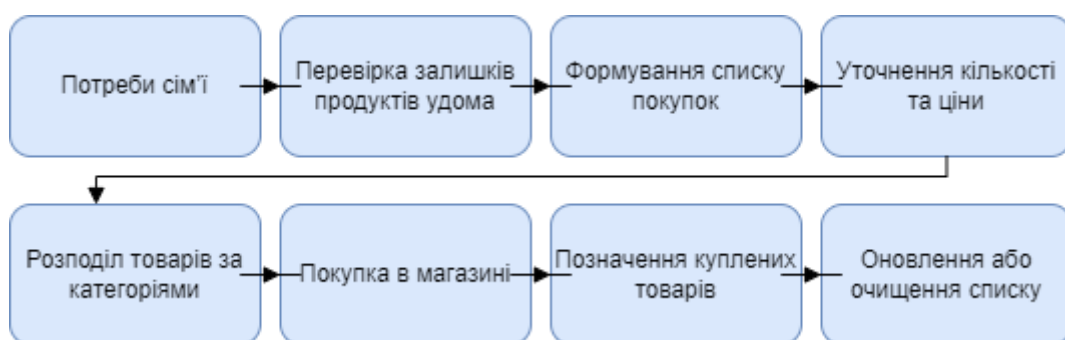


Рисунок 1.1 – Схема процесу планування сімейних покупок

Предметна область планування покупок охоплює кілька послідовних дій. Спочатку користувач визначає, які продукти потрібні сім'ї. Потім він перевіряє, що вже є вдома, і на основі цього складає список. Після цього список може уточнюватися: додається кількість товару, категорія, приблизна ціна. Під час

покупки користувач поступово відмічає позиції, які вже придбані. Після завершення покупки список може бути очищений або використаний як основа для наступного разу.

Окремо слід зазначити, що планування покупок пов'язане не лише зі зручністю, а й з економією. Якщо сім'я купує продукти без попереднього списку, зростає ризик придбати зайве або, навпаки, забути необхідні товари. Через це можуть збільшуватися витрати, а частина продуктів може залишатися невикористаною. Раціональне планування допомагає краще контролювати покупки та зменшувати кількість харчових відходів [1], [2].

На практиці люди використовують різні способи ведення списків. Найпростіший варіант — паперовий список. Він зрозумілий і не потребує технічних навичок, але його легко загубити або забути вдома. Інший поширений варіант — нотатки у смартфоні. Це вже зручніше, але звичайна нотатка не завжди дозволяє швидко структурувати список, розділити товари за категоріями або автоматично порахувати приблизну суму покупки. Також часто використовуються повідомлення в месенджерах, але в такому випадку список може змішуватися з іншими повідомленнями.

Для сімейного використання важливо, щоб список покупок був не просто набором назв продуктів. Бажано, щоб кожен товар мав хоча б мінімальну структуру: назву, кількість, одиницю вимірювання, категорію та орієнтовну ціну. Наприклад, запис «молоко» менш інформативний, ніж «молоко — 2 л — молочні продукти — 35 грн». Такий підхід дозволяє точніше планувати покупку і краще розуміти майбутні витрати.

Ще одна важлива особливість предметної області — потреба у швидкому оновленні списку. Під час походу до магазину користувач повинен мати можливість одразу відмітити куплений товар. Якщо список великий, без такої можливості легко заплутатися. Тому функція позначення товару як купленого є однією з базових для подібного програмного забезпечення.

Категоризація товарів також має практичне значення. У магазинах продукти зазвичай розміщені за відділами: овочі, молочні продукти, м'ясо, напої,

хлібобулочні вироби тощо. Якщо список покупок теж поділений за такими категоріями, користувачеві простіше орієнтуватися і не повертатися кілька разів до одного й того самого відділу. Це особливо помітно, коли список містить багато позицій.

Для розроблюваної системи предметна область може бути подана як процес, у якому користувач вводить інформацію про потрібні товари, а програма допомагає впорядкувати ці дані. Програмне забезпечення не повинно ускладнювати сам процес покупки. Навпаки, воно має зменшити кількість дрібних дій, які користувач зазвичай виконує вручну: запис, виправлення, викреслення, приблизний підрахунок суми.

У межах дипломної роботи доцільно створити саме демонстраційний веб-застосунок. Для такої теми не обов'язково одразу розробляти складну систему з реєстрацією користувачів, сервером і синхронізацією між пристроями. На першому етапі достатньо реалізувати основні функції: додавання товарів, редагування, видалення, позначення куплених позицій, фільтрацію за категоріями, підрахунок приблизної суми та збереження даних у браузері.

Такий підхід є виправданим і з технічного погляду. Якщо дані зберігаються локально у браузері, користувач може працювати зі списком без створення акаунта. Це спрощує розробку і робить систему зрозумілою для демонстраційного використання. У майбутньому, за потреби, таку систему можна розширити: додати спільний доступ для членів сім'ї, авторизацію, історію покупок або синхронізацію між пристроями.

Аналіз предметної області показує, що планування покупок продуктів харчування є актуальною задачею для сімейного побуту. Основні проблеми пов'язані з неструктурованими списками, забутими товарами, відсутністю контролю приблизної суми та складністю швидкого оновлення списку. Тому розробка простого програмного забезпечення для ведення списку покупок є доцільною. Воно має бути зрозумілим, швидким у використанні та орієнтованим на реальні побутові сценарії.

1.2 Огляд існуючих програмних рішень для планування покупок

Перед розробкою власного програмного забезпечення потрібно розглянути вже існуючі рішення, які користувачі застосовують для планування покупок. Це допомагає зрозуміти, які функції справді потрібні, а які можуть бути зайвими для невеликого навчального проєкту. Також такий огляд дозволяє не створювати систему «з нуля» без урахування реального досвіду користувачів.

У повсякденному житті люди часто користуються найпростішими інструментами. Наприклад, список покупок можна створити у звичайних нотатках смартфона. Це швидко і не потребує додаткових застосунків. Користувач просто записує потрібні продукти й поступово видаляє їх або викреслює після покупки. Проте цей варіант має очевидні обмеження. У звичайній нотатці складно автоматично порахувати суму, розділити товари за категоріями або швидко відфільтрувати список.

Іншим поширеним рішенням є Google Keep. Його можна використовувати для створення простих списків із можливістю позначати виконані пункти. Для побутових задач це досить зручно, оскільки сервіс простий і доступний на різних пристроях. Крім того, нотаткою можна поділитися з іншими користувачами. Але Google Keep не є спеціалізованим застосунком саме для продуктових покупок. У ньому немає окремих полів для ціни, кількості, одиниці вимірювання або категорії товару [3].

Microsoft To Do також може використовуватися для створення списків покупок. За своєю логікою це менеджер завдань, у якому можна створювати окремі списки, додавати пункти й позначати їх як виконані. Для короткого списку продуктів цього може бути достатньо. Проте основне призначення такого сервісу — керування справами, а не планування закупівлі продуктів. Через це в ньому відсутні функції, які важливі саме для продуктової тематики, наприклад підрахунок приблизної вартості покупки або групування товарів за продуктовими категоріями [4].

Більш спеціалізованим рішенням є AnyList. Цей застосунок уже орієнтований саме на списки покупок. У ньому зручніше працювати з продуктами, повторюваними позиціями та спільними списками. Таке рішення добре підходить для сімейного використання, оскільки список можуть переглядати кілька людей. Але для дипломної роботи не потрібно повністю повторювати функціональність подібного застосунку. Достатньо взяти з нього основну ідею: список покупок має бути структурованим і пристосованим саме до продуктів.

Ще одним прикладом є Bring!. Це застосунок для планування покупок із візуальним інтерфейсом і можливістю спільного використання списків. Його перевагою є те, що він створений саме для продуктових покупок, тому користувачеві простіше додавати товари й працювати з категоріями. Водночас реалізація повного аналога такого сервісу потребувала б складнішої архітектури, зокрема серверної частини, авторизації та синхронізації між користувачами [5].

Порівняння розглянутих рішень доцільно подати у вигляді таблиці, оскільки тут важливо не просто описати кожен програму, а побачити різницю між ними за однаковими критеріями (табл. 1.1).

Таблиця 1.1 – Порівняння існуючих програмних рішень

Програмне рішення	Основне призначення	Переваги	Недоліки для планування покупок
Звичайні нотатки смартфона	Швидкий запис інформації	Простота, доступність, не потрібно встановлювати додаткові програми	Немає категорій, підрахунку суми та спеціальної структури товару
Google Keep	Нотатки та прості списки	Швидке створення списків, позначення виконаних пунктів, спільний доступ	Не орієнтований саме на продуктові покупки
Microsoft To Do	Керування завданнями	Зручне позначення виконаних пунктів, окремі списки	Більше підходить для задач, ніж для списків продуктів
AnyList	Списки покупок	Категорії, повторювані товари, спільне використання	Частина функцій є зайвою для простого дипломного прототипу
Bring!	Планування продуктових покупок	Зручний візуальний інтерфейс, категорії, спільні списки	Для повного аналога потрібна складніша реалізація

З таблиці видно, що універсальні сервіси можуть використовуватися для списків покупок, але вони не враховують усі особливості цієї задачі. Вони зручні тоді, коли потрібно швидко записати кілька позицій. Але якщо список стає більшим, з'являється потреба у категоріях, фільтрації та підрахунку суми. Саме цих функцій часто не вистачає у звичайних нотатках або менеджерах завдань.

Спеціалізовані застосунки краще пристосовані до планування покупок. Вони мають категорії, підтримують сімейне використання та часто пропонують зручніше додавання товарів. Проте для студентського дипломного проєкту не обов'язково реалізовувати всі можливості таких систем. Якщо додати занадто багато функцій, застосунок стане складнішим, а основна задача роботи може втратити чіткість.

Для власного програмного забезпечення варто взяти найкорисніші ідеї з розглянутих рішень. Від звичайних нотаток — простоту. Від менеджерів завдань — можливість позначати позиції як виконані. Від спеціалізованих застосунків — категоризацію товарів і орієнтацію саме на продуктові списки. При цьому система має залишатися невеликою і зрозумілою, оскільки її головна мета — показати логіку планування покупок, а не замінити повноцінний комерційний сервіс.

У межах дипломної роботи доцільно реалізувати базові функції: додавання товару, редагування, видалення, позначення як купленого, вибір категорії, фільтрацію, підрахунок приблизної суми та локальне збереження списку. Цього достатньо, щоб продемонструвати практичну цінність системи і водночас не перевантажити її зайвими можливостями.

Огляд існуючих програмних рішень показує, що на практиці користувачі мають різні варіанти для планування покупок. Прості нотатки зручні, але обмежені. Спеціалізовані застосунки функціональніші, але складніші. Тому для дипломної роботи оптимальним рішенням є створення власного frontend-прототипу, який поєднує простоту використання з базовими функціями, потрібними саме для сімейного списку покупок.

1.3 Визначення вимог до програмного забезпечення

Після аналізу предметної області та огляду існуючих рішень можна перейти до визначення вимог до майбутнього програмного забезпечення. На цьому етапі важливо не просто перелічити бажані функції, а зрозуміти, що саме має робити система, щоб бути корисною для користувача. Якщо вимоги сформульовані нечітко, то під час розробки легко додати зайві можливості або, навпаки, пропустити важливі функції.

У цій роботі розробляється веб-застосунок для планування покупок продуктів харчування сім'ї. Тому система повинна допомагати користувачу створювати список товарів, змінювати його під час підготовки до покупки, контролювати виконання списку в магазині та бачити приблизну суму витрат. При цьому застосунок не повинен бути складним. Його основна перевага має полягати в тому, що користувач швидко розуміє, як додати продукт і як працювати зі списком.

Основним користувачем системи є член сім'ї, який планує покупку продуктів. Це може бути будь-яка людина без спеціальних технічних знань, тому інтерфейс має бути простим. Користувачеві не потрібно проходити реєстрацію, налаштовувати профіль або виконувати складні дії перед початком роботи. Він відкриває застосунок і одразу може формувати список покупок.

Функціональні вимоги описують конкретні дії, які має виконувати програмне забезпечення. Для цієї системи основними є робота зі списком товарів, категоріями, статусом купівлі, підсумковою сумою та локальним збереженням даних. Ці вимоги пов'язані з реальним сценарієм використання: користувач планує покупку вдома, а потім використовує список у магазині (табл.1.2, рис. 1.2).

Таблиця 1.2 – Основні вимоги до програмного забезпечення

Група вимог	Вимога	Зміст вимоги
Робота зі списком	Додавання товару	Користувач може додати продукт із назвою, кількістю, категорією та ціною

Кінець таблиці 1.2

Група вимог	Вимога	Зміст вимоги
Робота зі списком	Редагування товару	Користувач може змінити дані вже доданого продукту
Робота зі списком	Видалення товару	Користувач може прибрати зайву або помилкову позицію
Контроль покупки	Позначення як купленого	Користувач може відмітити товар, який уже придбано
Організація списку	Категорії товарів	Товари можна розподіляти за продуктовими категоріями
Організація списку	Фільтрація	Користувач може переглядати товари певної категорії
Розрахунки	Підрахунок суми	Система показує приблизну загальну вартість покупки
Збереження	localStorage	Список зберігається у браузері після перезавантаження сторінки
Інтерфейс	Адаптивність	Застосунок має бути зручним на комп'ютері та смартфоні
Інтерфейс	Простота використання	Основні дії мають виконуватися без зайвих переходів



Рисунок 1.2 – Діаграма варіантів використання системи

Окрім функціональних можливостей, потрібно визначити вимоги до даних. Кожен товар у списку має містити мінімальний набір інформації, достатній для роботи системи. До такого набору належать назва продукту, категорія, кількість, одиниця вимірювання, приблизна ціна та статус купівлі. Наприклад, якщо користувач додає товар «молоко», йому бажано вказати не лише назву, а й кількість, наприклад 2 літри, і приблизну ціну. Тоді система зможе коректно врахувати цей товар у загальній сумі.

Категорії товарів краще зробити фіксованими. Це спростить роботу користувача і зменшить кількість помилок під час введення. До основних категорій можна віднести овочі та фрукти, молочні продукти, м'ясо та рибу, хлібобулочні вироби, крупи та макарони, напої, побутові товари та інше. Такого набору достатньо для базового прототипу, а в майбутньому його можна розширити.

Важливою вимогою є перевірка введених даних. Система не повинна додавати товар без назви, з нульовою або від'ємною кількістю, а також із від'ємною ціною. Це прості перевірки, але вони впливають на якість роботи програми. Якщо дозволити некоректні значення, підрахунок суми може бути неправильним, а список стане незручним для використання.

Нефункціональні вимоги стосуються не окремих дій, а загальної якості програмного забезпечення. Для цієї роботи важливо, щоб застосунок працював швидко, мав зрозумілий інтерфейс і не вимагав додаткових налаштувань. Оскільки система призначена для побутового використання, користувач не повинен витрачати час на вивчення складного меню або інструкцій.

Окремо варто звернути увагу на адаптивність. Список покупок часто створюють вдома на комп'ютері, але використовують уже в магазині зі смартфона. Тому інтерфейс має нормально відображатися на різних екранах. На великому екрані можна показувати більше інформації в одному рядку, а на мобільному пристрої краще використовувати компактні блоки або картки.

Ще одна важлива вимога це збереження стану списку. Якщо користувач випадково закrije сторінку або перезавантажить браузер, список не повинен зникати. Для цього в межах дипломного прототипу доцільно використати локальне сховище браузера. Це дозволяє обійтися без backend і водночас зберігати дані між сесіями роботи [6].

У межах цієї дипломної роботи не планується реалізовувати всі можливості, які можуть бути у повноцінній комерційній системі. Наприклад, не передбачається авторизація користувачів, спільний сімейний акаунт, серверна база даних, синхронізація між пристроями або інтеграція з онлайн-магазинами.

Ці функції є корисними, але для демонстраційного frontend-прототипу вони ускладнили б проєкт і змістили б увагу з основної задачі.

Розроблюване програмне забезпечення має бути достатньо простим, але завершеним. Це означає, що користувач повинен мати змогу пройти повний сценарій: додати товари, переглянути список, відмітити куплені позиції, побачити суму та зберегти список. Якщо ці дії працюють коректно, прототип можна вважати таким, що виконує поставлену задачу.

Отже, вимоги до програмного забезпечення визначають основу подальшого проєктування. Система повинна забезпечувати роботу зі списком покупок, підтримувати категорії, виконувати прості розрахунки, зберігати дані локально та мати зручний адаптивний інтерфейс. Такі вимоги відповідають темі дипломної роботи й дозволяють перейти до постановки задачі розробки.

1.4 Постановка задачі дипломної роботи

Після аналізу предметної області, огляду існуючих програмних рішень і визначення вимог можна сформулювати задачу дипломної роботи. Загальна ідея роботи полягає в тому, щоб створити простий веб-застосунок, який допомагає сім'ї планувати покупки продуктів харчування. Такий застосунок не повинен замінювати великі комерційні сервіси, але має виконувати основні дії, які потрібні користувачу в побуті.

Основна проблема, яку потрібно вирішити, пов'язана з неструктурованим веденням списків покупок. Якщо список записується на папері, у звичайній нотатці або в повідомленнях, користувач не завжди може швидко його оновити, розподілити товари за категоріями чи оцінити приблизну суму. Через це покупки можуть бути менш організованими, а частина товарів може бути забута або куплена повторно.

У межах дипломної роботи потрібно розробити демонстраційний frontend-застосунок для планування покупок продуктів харчування сім'ї. Застосунок має працювати у браузері, не вимагати реєстрації та не

використовувати серверну частину. Для збереження списку покупок доцільно застосувати локальне сховище браузера, щоб дані не зникали після перезавантаження сторінки [6] (рис. 1.3).

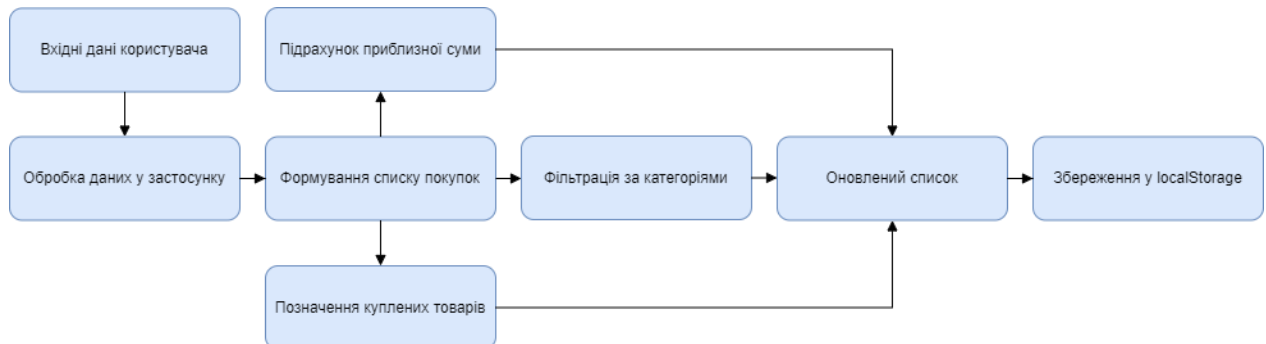


Рисунок 1.3 – Загальна схема роботи програмного забезпечення

Метою розробки є створення програмного забезпечення, яке дозволяє користувачу формувати список покупок, керувати товарами, контролювати виконання списку та переглядати приблизну суму витрат.

Для досягнення цієї мети необхідно виконати такі завдання: спроектувати структуру даних для товарів; визначити категорії продуктів; розробити інтерфейс користувача; реалізувати додавання, редагування та видалення товарів; додати можливість позначення товарів як куплених; реалізувати фільтрацію за категоріями; забезпечити підрахунок загальної суми; організувати локальне збереження даних; виконати тестування основних сценаріїв роботи системи.

Вхідними даними для системи є інформація, яку вводить користувач. До неї належать назва продукту, категорія, кількість, одиниця вимірювання та орієнтовна ціна. На основі цих даних система створює запис про товар і додає його до списку покупок. Користувач може змінювати ці дані, якщо помилився або якщо потрібно уточнити кількість чи ціну.

Вихідними даними є сформований список покупок, статус кожного товару, кількість куплених і некуплених позицій, а також приблизна загальна сума покупки. Саме ці дані користувач бачить під час роботи із застосунком.

Вони допомагають швидше орієнтуватися у списку і контролювати процес покупки.

Розроблювана система має реалізовувати базовий, але завершений сценарій роботи. Спочатку користувач відкриває застосунок і додає потрібні продукти. Далі він може переглянути список, відфільтрувати товари за категорією, змінити окремі позиції або видалити зайві. Під час покупки користувач відмічає придбані товари. Після завершення покупки список можна очистити або залишити як основу для наступного використання.

Важливо, що в межах цієї роботи не ставиться задача створення складної багатокористувацької системи. Не передбачається реєстрація користувачів, серверна база даних, спільний доступ для кількох членів сім'ї та синхронізація між пристроями. Такі можливості можна розглядати як подальший розвиток проєкту, але для дипломного прототипу достатньо локальної роботи в браузері.

Для реалізації програмного забезпечення планується використати сучасні frontend-технології: React, TypeScript, Vite та Tailwind CSS. Такий стек дозволяє створити компонентний інтерфейс, описати структуру даних, швидко розробити демонстраційний застосунок і забезпечити адаптивне відображення на різних екранах [7], [8].

Практичним результатом дипломної роботи має стати веб-застосунок, який можна запустити локально у браузері. Користувач повинен мати можливість додавати продукти до списку, редагувати їх, видаляти, позначати як куплені, фільтрувати за категоріями, бачити приблизну суму покупки та зберігати список між сесіями роботи.

Задача дипломної роботи полягає у розробці простого й зрозумілого програмного забезпечення для планування покупок продуктів харчування сім'ї. Запропонований застосунок має усунути основні недоліки традиційного ведення списків: відсутність структури, складність оновлення, неможливість швидкого підрахунку суми та ризик втрати інформації. Сформульована постановка задачі є основою для подальшого вибору технологій, проєктування архітектури та реалізації програмного прототипу.

2 ВИБІР ТЕХНОЛОГІЙ ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕСПЕЧЕННЯ

2.1 Обґрунтування вибору технологій розробки

Після постановки задачі потрібно визначити, за допомогою яких технологій буде реалізовано програмне забезпечення. Вибір технологій впливає не тільки на процес розробки, а й на подальшу підтримку системи, її швидкодію, масштабованість і зручність використання. Для цієї дипломної роботи важливо було обрати такі інструменти, які дозволяють створити повноцінний веб-застосунок для планування покупок продуктів харчування сім'ї, але при цьому не роблять архітектуру надмірно складною.

Оскільки розроблювана система має працювати як веб-застосунок, її доцільно реалізувати за клієнт-серверною архітектурою. Frontend відповідає за інтерфейс користувача та взаємодію зі списками покупок, backend — за обробку запитів, авторизацію, роботу з бізнес-логікою та збереження даних, а база даних — за постійне зберігання користувачів, сімейних груп, списків покупок і товарів.

Для реалізації frontend-частини обрано React, TypeScript, Vite і Tailwind CSS. Для backend-частини доцільно використати Node.js, Express, TypeScript, Prisma та PostgreSQL. Такий стек технологій добре підходить для студентського full stack проєкту, оскільки є сучасним, поширеним, має достатню документацію та дозволяє розділити систему на зрозумілі частини.

Основною бібліотекою для створення інтерфейсу користувача обрано React. Його зручно використовувати для такого проєкту, тому що інтерфейс застосунку можна поділити на окремі компоненти: форму додавання товару, список покупок, елемент товару, фільтр категорій, підсумкову панель, сторінку авторизації та сторінку історії покупок. Завдяки компонентному підходу код стає більш організованим, а окремі частини інтерфейсу легше змінювати або повторно використовувати [9]-[11].

Для типізації даних обрано TypeScript. У системі є багато структурованих даних: користувач, сім'я, список покупок, товар, категорія, кількість, ціна, статус купівлі. Якщо ці дані описати за допомогою типів та інтерфейсів, код стає

зрозумілішим і надійнішим. Наприклад, ціна товару повинна бути числом, статус купівлі — логічним значенням, а ідентифікатор списку — рядком. TypeScript допомагає виявляти частину помилок ще під час розробки, а не після запуску програми [12].

Для створення та запуску frontend-проєкту обрано Vite [13]. Він забезпечує швидкий запуск локального середовища розробки, просту структуру проєкту та зручну збірку готового застосунку. Для дипломної роботи це важливо, оскільки розробник може швидко створити проєкт, запустити його в браузері та поступово додавати потрібні компоненти без складного налаштування інструментів збірки.

Для оформлення інтерфейсу використовується Tailwind CSS. Цей інструмент дозволяє швидко створювати адаптивний дизайн за допомогою готових CSS-класів. У межах цієї роботи не потрібно створювати складний дизайнерський інтерфейс, але сторінки мають бути акуратними, зрозумілими та зручними для користувача. Tailwind CSS дає змогу оформити форми, кнопки, списки, картки товарів і підсумкові блоки без написання великої кількості окремого CSS-коду [14]-[17].

Для backend-частини обрано Node.js та Express. Node.js дозволяє виконувати JavaScript або TypeScript на сервері, а Express є простим і поширеним фреймворком для створення REST API [18]-[22]. У цій системі backend потрібен для реєстрації та входу користувачів, обробки запитів до списків покупок, перевірки доступу до сімейної групи та роботи з базою даних.

Для роботи з базою даних обрано Prisma. Це ORM-інструмент, який дозволяє описувати структуру бази даних у вигляді моделей і працювати з ними через TypeScript-код. Prisma зменшує кількість ручного SQL-коду, спрощує створення міграцій і робить роботу з БД [23]-[25].

Як систему керування базами даних доцільно використати PostgreSQL [26], [27]. Вона добре підходить для зберігання структурованих даних, таких як користувачі, сімейні групи, списки покупок і товари. На відміну від localStorage, база даних дозволяє зберігати інформацію централізовано, працювати з нею з різних пристроїв і в майбутньому реалізувати спільний доступ членів сім'ї [28].

Для авторизації користувачів можна використати JWT. Після входу користувач отримує токен, який надалі передається разом із запитом до backend. Це дозволяє захистити маршрути й не дозволяти стороннім користувачам переглядати або змінювати чужі списки покупок. Паролі при цьому не повинні зберігатися у відкритому вигляді. Для їхнього захисту доцільно використовувати хешування, наприклад за допомогою bcrypt.

Для тестування окремих функцій можна використати Vitest або Jest [29], [30]. У frontend-частині доцільно тестувати функції підрахунку загальної суми, кількості куплених товарів і фільтрації за категоріями. У backend-частині можна перевіряти валідацію вхідних даних і окремі сервісні функції. Такі тести допомагають переконатися, що базова логіка системи працює правильно. Щоб обґрунтувати вибір технологій, доцільно порівняти їх з альтернативами, які також могли бути використані для реалізації цієї системи (табл. 2.1).

Таблиця 2.1 – Порівняння обраних технологій з альтернативними варіантами

Частина системи	Обрана технологія	Альтернативні технології	Обґрунтування вибору
Frontend-бібліотека	React	Vue, Angular, Svelte	React має компонентний підхід, велику спільноту, багато документації та добре підходить для створення інтерактивного інтерфейсу списків покупок
Мова розробки	TypeScript	JavaScript	TypeScript забезпечує типізацію даних, що зменшує кількість помилок під час роботи з товарами, списками, користувачами та API
Інструмент збірки frontend	Vite	Webpack, Parcel, Create React App	Vite швидко запускається, має просте налаштування і добре підходить для сучасних React-проектів
CSS-фреймворк	Tailwind CSS	Bootstrap, Material UI, звичайний CSS	Tailwind CSS дозволяє швидко створити адаптивний інтерфейс без великої кількості окремих CSS-файлів
Backend-середовище	Node.js	PHP, Python, Java, .NET	Node.js дозволяє використовувати TypeScript і на frontend, і на backend, що робить проєкт більш єдиним за технологічним підходом

Кінець таблиці 2.1

Частина системи	Обрана технологія	Альтернативні технології	Обґрунтування вибору
Backend-фреймворк	Express	NestJS, Fastify, Koa	Express має просту структуру, достатній функціонал для REST API і не перевантажує дипломний проєкт зайвою складністю
База даних	PostgreSQL	MySQL, SQLite, MongoDB	PostgreSQL добре підходить для структурованих зв'язків між користувачами, сім'ями, списками покупок і товарами
ORM	Prisma	Sequelize, TypeORM, ручний SQL	Prisma має зрозумілу схему моделей, міграції та зручну роботу з TypeScript
Авторизація	JWT	Session-based auth, OAuth	JWT підходить для REST API та дозволяє зручно захищати запити до backend
Тестування	Vitest / Jest	Mocha, Jasmine	Vitest добре підходить для frontend-проєктів на Vite, а Jest може використовуватися для backend-тестів

З таблиці видно, що для кожної частини системи існує кілька можливих варіантів реалізації. Наприклад, замість React можна було б використати Vue або Angular. Vue також є зручним для створення інтерфейсів, а Angular має більш повну архітектуру з великою кількістю вбудованих можливостей. Проте для цієї роботи React є більш доцільним, оскільки він достатньо гнучкий і не змушує використовувати складну структуру там, де вона не потрібна.

Так само для backend можна було б використати NestJS. Він має більш сувору архітектуру і добре підходить для великих систем. Однак для дипломного проєкту Express є простішим і зрозумілішим. Він дозволяє швидко створити REST API для користувачів, сімейних груп, списків покупок і товарів, не перевантажуючи проєкт додатковими абстракціями [31].

Для збереження даних можна було б використати SQLite, особливо для локального прототипу. Але якщо система позиціонується як повноцінний full stack проєкт, краще обрати PostgreSQL. Така база даних краще підходить для подальшого розвитку, роботи з кількома користувачами та реалізації сімейних груп. MongoDB також могла б використовуватися, але в цій системі дані мають чіткі зв'язки, тому реляційна база є логічнішим вибором.

Окремо варто зазначити, що `localStorage` у такій архітектурі може використовуватися лише для допоміжних задач, наприклад для збереження JWT-токена на стороні клієнта. Основні дані системи — користувачі, сім'ї, списки покупок і товари — мають зберігатися в базі даних. Це робить систему більш серйозною та придатною для реального використання.

Обраний стек технологій дозволяє реалізувати не просто демонстраційну сторінку, а повноцінну інформаційну систему. Frontend забезпечує зручну взаємодію користувача із системою, backend відповідає за обробку даних і правила доступу, а PostgreSQL зберігає інформацію у структурованому вигляді. Така архітектура краще відповідає темі дипломної роботи, оскільки показує повний цикл розробки програмного забезпечення.

Для реалізації програмного забезпечення для планування покупок продуктів харчування сім'ї обрано React, TypeScript, Vite, Tailwind CSS, Node.js, Express, Prisma, PostgreSQL, JWT і засоби тестування. Ці технології відповідають поставленій задачі, дозволяють створити зручний веб-інтерфейс, REST API, базу даних і механізм авторизації.

2.2 Проектування архітектури програмного забезпечення

Після вибору технологій потрібно визначити, як саме буде побудоване програмне забезпечення. Архітектура застосунку має бути достатньо простою, щоб її можна було реалізувати в межах дипломної роботи, але водночас зрозумілою і придатною для подальшого розвитку. Для цієї системи найкраще підходить компонентний підхід, оскільки застосунок розробляється на React і складається з окремих частин інтерфейсу [32]- [34].

Розроблюване програмне забезпечення є односторінковим веб-застосунком. Це означає, що всі основні дії користувач виконує на одній сторінці: додає товар, переглядає список, фільтрує позиції, відмічає куплені продукти й бачить підсумкову суму. Для такої задачі не потрібно створювати

багато сторінок або складну навігацію. Навпаки, чим менше зайвих переходів, тим зручніше користуватися списком покупок у реальних умовах.

Головним компонентом системи є `App`. У ньому зберігається основний стан застосунку — список товарів, активна категорія фільтрації та функції для зміни списку. Саме цей компонент керує основними діями: додаванням, редагуванням, видаленням товарів, зміною статусу купівлі та очищенням списку. Такий підхід зручний тим, що вся основна логіка зібрана в одному місці, а дочірні компоненти отримують тільки потрібні їм дані та функції.

Компонент `Header` відповідає за верхню частину сторінки. У ньому розміщується назва застосунку і короткий опис. Цей компонент не має складної логіки, але він робить сторінку зрозумілішою для користувача.

Компонент `ShoppingForm` використовується для додавання нового товару. У ньому розміщуються поля введення: назва продукту, категорія, кількість, одиниця вимірювання та приблизна ціна. Після заповнення форми користувач натискає кнопку додавання, а дані передаються до головного компонента `App`. Там створюється новий об'єкт товару, який додається до списку.

Компонент `ShoppingList` відповідає за відображення списку покупок. Він отримує масив товарів і передає кожен окремий товар до компонента `ShoppingItem`. Якщо список порожній, користувачеві показується повідомлення про те, що ще не додано жодного продукту. Такий порожній стан потрібен, щоб інтерфейс не виглядав незавершеним після першого запуску.

Компонент `ShoppingItem` відображає один товар у списку. У ньому показуються назва, категорія, кількість, одиниця вимірювання, ціна та сума за позицію. Також тут розміщуються кнопки для редагування, видалення і позначення товару як купленого. Якщо товар уже куплений, його вигляд змінюється, наприклад назва може бути закреслена або менш помітна.

Компонент `CategoryFilter` потрібен для фільтрації товарів. Якщо у списку багато позицій, користувач може переглянути лише певну категорію, наприклад

молочні продукти або овочі та фрукти. Фільтр не змінює сам список, а тільки впливає на те, які товари показуються на екрані.

Компонент `SummaryPanel` показує коротку підсумкову інформацію. У ньому відображається загальна кількість товарів, кількість куплених і некуплених позицій, а також приблизна загальна сума покупки. Цей компонент важливий, бо користувач одразу бачить, наскільки заповнений список і яку суму приблизно потрібно витратити.

Окрім візуальних компонентів, у структурі застосунку передбачаються службові модулі. Модуль `storageService` відповідає за роботу з `localStorage`: завантаження списку, збереження списку та очищення даних. Це краще винести окремо, щоб не змішувати інтерфейс із технічною логікою збереження.

Також потрібен модуль `calculations`, у якому розміщуються функції підрахунку. Наприклад, обчислення загальної суми, кількості куплених товарів, кількості некуплених товарів і фільтрація за категорією. Такі функції можна використовувати в різних частинах застосунку і зручно тестувати окремо. Основні частини архітектури можна подати у вигляді таблиці (табл. 2.2).

Таблиця 2.2 – Основні компоненти та модулі програмного забезпечення

Компонент або модуль	Призначення
App	Керує станом застосунку та основними діями зі списком
Header	Відображає назву і короткий опис системи
ShoppingForm	Забезпечує додавання нового товару
ShoppingList	Відображає список покупок
ShoppingItem	Показує один товар і кнопки дій
CategoryFilter	Дозволяє фільтрувати товари за категоріями
SummaryPanel	Показує підсумкову інформацію про список
storageService	Зберігає та завантажує дані з <code>localStorage</code>
calculations	Містить функції підрахунку та фільтрації

Взаємодія між компонентами відбувається за принципом односпрямованого потоку даних. Головний компонент зберігає список товарів і передає його дочірнім компонентам. Якщо користувач виконує дію, наприклад додає або видаляє товар, дочірній компонент викликає функцію, передану з App. Після цього стан оновлюється, а інтерфейс автоматично відображає нові дані (рис. 2.1)

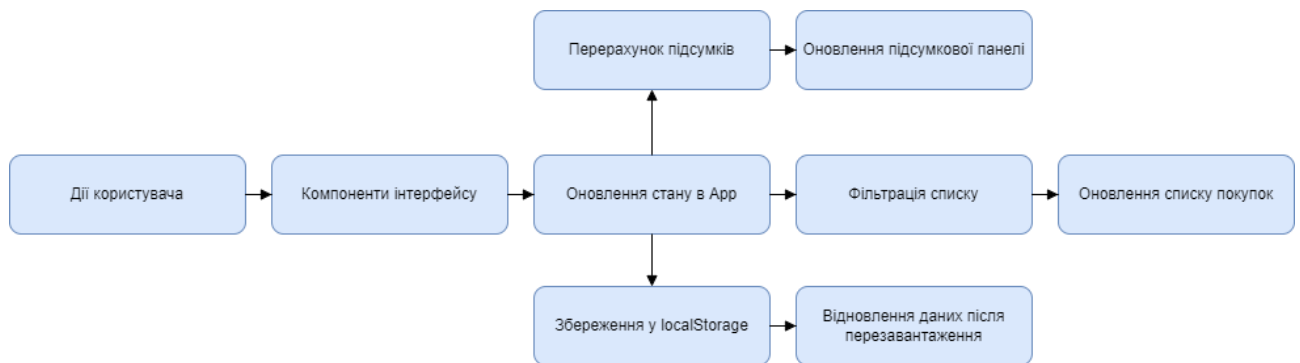


Рисунок 2.1 – Схема потоку даних у застосунку

Такий підхід робить поведінку системи передбачуваною. Коли всі зміни проходять через головний компонент, легше зрозуміти, де саме оновлюються дані. Це особливо важливо для дипломного проєкту, оскільки архітектуру потрібно не лише реалізувати, а й пояснити в тексті роботи.

Під час першого запуску застосунок звертається до localStorage і перевіряє, чи є раніше збережений список. Якщо дані є, вони завантажуються в стан застосунку. Якщо даних немає, система показує порожній список. Після кожної зміни — додавання, редагування, видалення або позначення товару як купленого — оновлений список зберігається у браузері.

Обрана архітектура не передбачає backend, API або серверної бази даних. Для поточної задачі це не є недоліком, оскільки створюється демонстраційний frontend-прототип. Якщо в майбутньому потрібно буде додати спільний доступ для сім'ї або синхронізацію між пристроями, архітектуру можна буде розширити. Наприклад, замість localStorage можна підключити серверне API, а компоненти інтерфейсу залишити майже без змін.

Архітектура програмного забезпечення побудована за компонентним принципом. Вона включає головний компонент, набір візуальних компонентів для роботи зі списком покупок і службові модулі для збереження та обробки даних. Такий підхід є достатньо простим для реалізації, але водночас дозволяє підтримувати зрозумілу структуру проєкту і розширювати систему в майбутньому.

2.3 Проєктування бази даних

Після визначення архітектури потрібно описати, які саме дані буде зберігати та обробляти система. Для застосунку планування покупок це особливо важливо, тому що вся робота програми фактично будується навколо списку товарів. Якщо структура даних буде незручною, то складніше реалізувати редагування, фільтрацію, підрахунок суми та збереження інформації.

У межах цієї дипломної роботи не створюється повноцінна база даних, оскільки застосунок працює без backend. Дані зберігаються у браузері користувача за допомогою localStorage. Проте навіть для такого простого варіанта потрібно заздалегідь продумати логічну структуру даних. Це дозволяє зробити код більш зрозумілим і підготувати систему до можливого розширення в майбутньому (рис. 2.2).

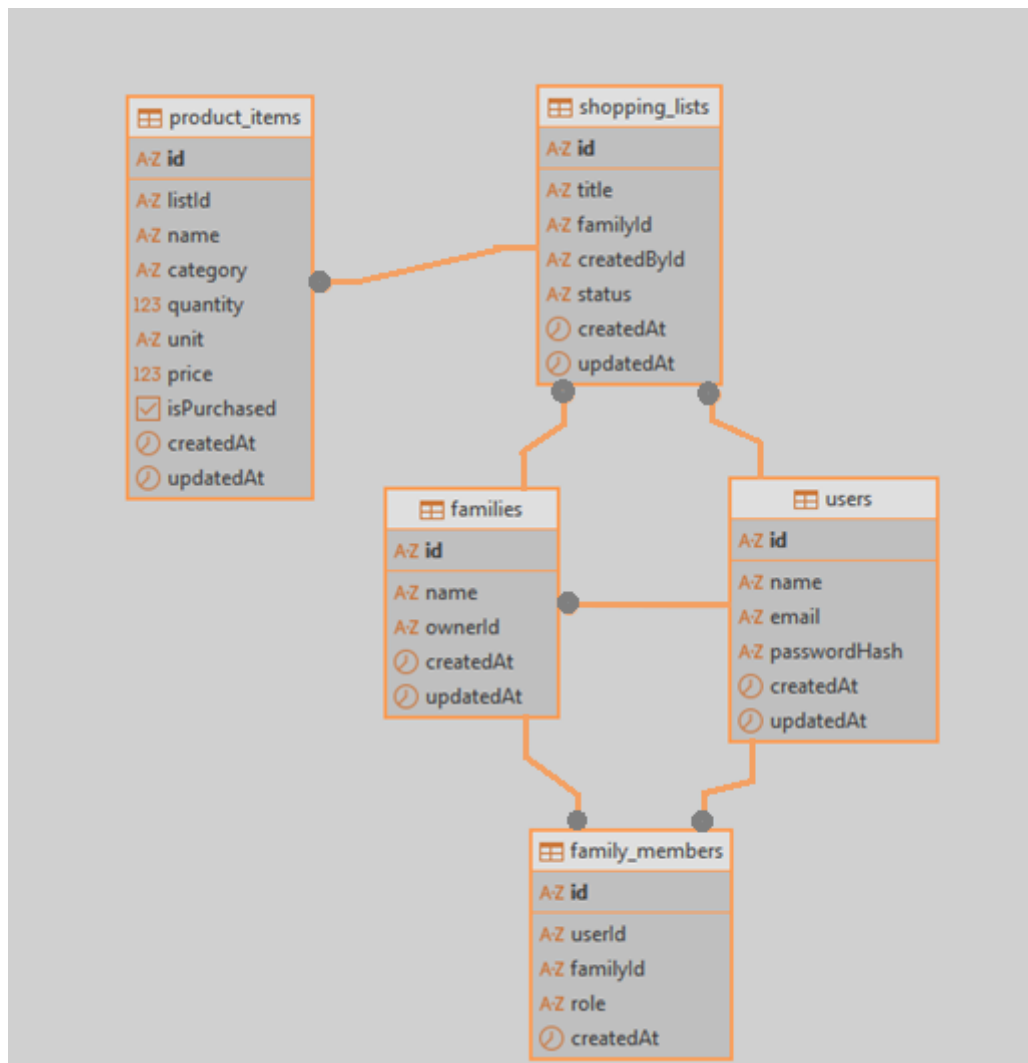


Рисунок 2.2 – Логічна модель даних програмного забезпечення

Основним об'єктом у системі є товар. Кожен товар у списку повинен містити не тільки назву, а й додаткову інформацію, яка потрібна для нормальної роботи застосунку. Наприклад, якщо зберігати лише текст «молоко», система не зможе порахувати суму або відфільтрувати товар за категорією. Тому для кожного продукту потрібно зберігати кількість, одиницю вимірювання, категорію, приблизну ціну та статус покупки.

У структурі товару передбачається поле `id`. Воно потрібне для того, щоб система могла відрізнити один товар від іншого. Це особливо важливо, якщо в списку є схожі або однакові назви. Наприклад, користувач може додати «молоко» двічі, але з різною кількістю або ціною. За допомогою ідентифікатора програма зможе правильно редагувати або видаляти конкретний запис.

Поле `name` зберігає назву продукту. Це основна інформація, яку бачить користувач у списку. Назва не повинна бути порожньою, тому під час додавання товару потрібно виконувати перевірку введення.

Поле `category` відповідає за категорію продукту. Воно потрібне для фільтрації та зручнішої організації списку. У програмі доцільно використати фіксований набір категорій: овочі та фрукти, молочні продукти, м'ясо та риба, хлібобулочні вироби, крупи та макарони, напої, побутові товари та інше. Такого набору достатньо для базового прототипу, а в разі подальшого розвитку його можна розширити.

Поля `quantity` і `unit` потрібні для уточнення кількості товару. Наприклад, користувач може вказати 2 літри молока, 1 кілограм картоплі або 3 упаковки макаронів. Одиниці вимірювання також краще задавати через фіксований список, щоб не було різних варіантів написання одного й того самого значення.

Поле `price` використовується для збереження приблизної ціни за одиницю товару. На основі цього поля і кількості система може розрахувати суму за окрему позицію та загальну суму покупки. Ціна є орієнтовною, тому користувач може вводити її приблизно. Головне, щоб система не дозволяла вводити від'ємні значення.

Поле `isPurchased` зберігає статус товару. Якщо значення дорівнює `false`, товар ще потрібно купити. Якщо значення дорівнює `true`, товар уже позначений як придбаний. Завдяки цьому користувач може бачити, які позиції вже виконані, а які залишилися.

Поле `createdAt` містить дату створення запису. У базовому прототипі воно не є критично необхідним, але може бути корисним для сортування товарів або подальшого розвитку системи. Наприклад, у майбутньому це поле можна використати для історії покупок.

Основну структуру об'єкта товару можна подати у вигляді таблиці (табл. 2.3)

Таблиця 2.3 – Структура об’єкта товару

Поле	Тип даних	Призначення
id	string	Унікальний ідентифікатор товару
name	string	Назва продукту
category	string	Категорія товару
quantity	number	Кількість товару
unit	string	Одиниця вимірювання
price	number	Орієнтовна ціна за одиницю
isPurchased	boolean	Статус купівлі
createdAt	string	Дата створення запису

У програмній реалізації цю структуру можна описати через інтерфейс TypeScript. Це дозволяє чітко визначити, які поля має містити товар і які типи даних для них очікуються. Наприклад, кількість і ціна мають бути числами, а статус купівлі — логічним значенням. Такий підхід допомагає уникнути помилок під час написання коду.

Список покупок у цій системі є масивом об’єктів товару. Коли користувач додає продукт, у масив додається новий об’єкт. Коли товар видаляється, відповідний об’єкт прибирається з масиву. Якщо користувач редагує товар, змінюються окремі поля потрібного об’єкта. Якщо товар позначається як куплений, змінюється лише поле isPurchased.

Для збереження списку в localStorage масив товарів потрібно перетворити у формат JSON. Під час наступного відкриття сторінки дані зчитуються з localStorage і перетворюються назад у масив об’єктів. Це простий і зручний спосіб зберегти інформацію без серверної частини.

Водночас потрібно враховувати можливі помилки. Наприклад, якщо в localStorage немає збережених даних або вони були пошкоджені, застосунок не повинен припиняти роботу. У такому випадку краще повернути порожній список і дозволити користувачу почати роботу заново. Це робить систему більш надійною.

Запропонована структура даних є простою, але достатньою для реалізації всіх основних функцій. Вона дозволяє додавати товари, змінювати їх, видаляти, фільтрувати за категоріями, рахувати суму та зберігати список у браузері. Якщо в майбутньому буде додано backend, цю модель можна буде перенести до бази даних майже без суттєвих змін.

Отже, у розроблюваному програмному забезпеченні основною структурою даних є список товарів. Кожен товар має набір полів, потрібних для його відображення, редагування, фільтрації та підрахунку суми. Така модель відповідає задачі дипломної роботи й добре підходить для реалізації frontend-прототипу.

2.4 Розробка алгоритмів роботи програмного забезпечення

Після проєктування структури даних потрібно описати основні алгоритми, за якими працюватиме програмне забезпечення. Для цієї системи алгоритми не є надто складними, але вони важливі, бо саме через них реалізується вся логіка роботи зі списком покупок. Користувач бачить простий інтерфейс, але всередині застосунок постійно оновлює список, перевіряє введені дані, рахує суму та зберігає інформацію у браузері.

Основним алгоритмом є додавання товару до списку. Користувач вводить назву продукту, обирає категорію, указує кількість, одиницю вимірювання та приблизну ціну. Після цього система перевіряє, чи правильно заповнені поля. Наприклад, назва не повинна бути порожньою, кількість має бути більшою за нуль, а ціна не може бути від'ємною. Якщо дані введені правильно, створюється новий об'єкт товару, якому присвоюється унікальний ідентифікатор. Далі цей об'єкт додається до списку, а оновлений список зберігається у localStorage. Блок схема в додатку А.

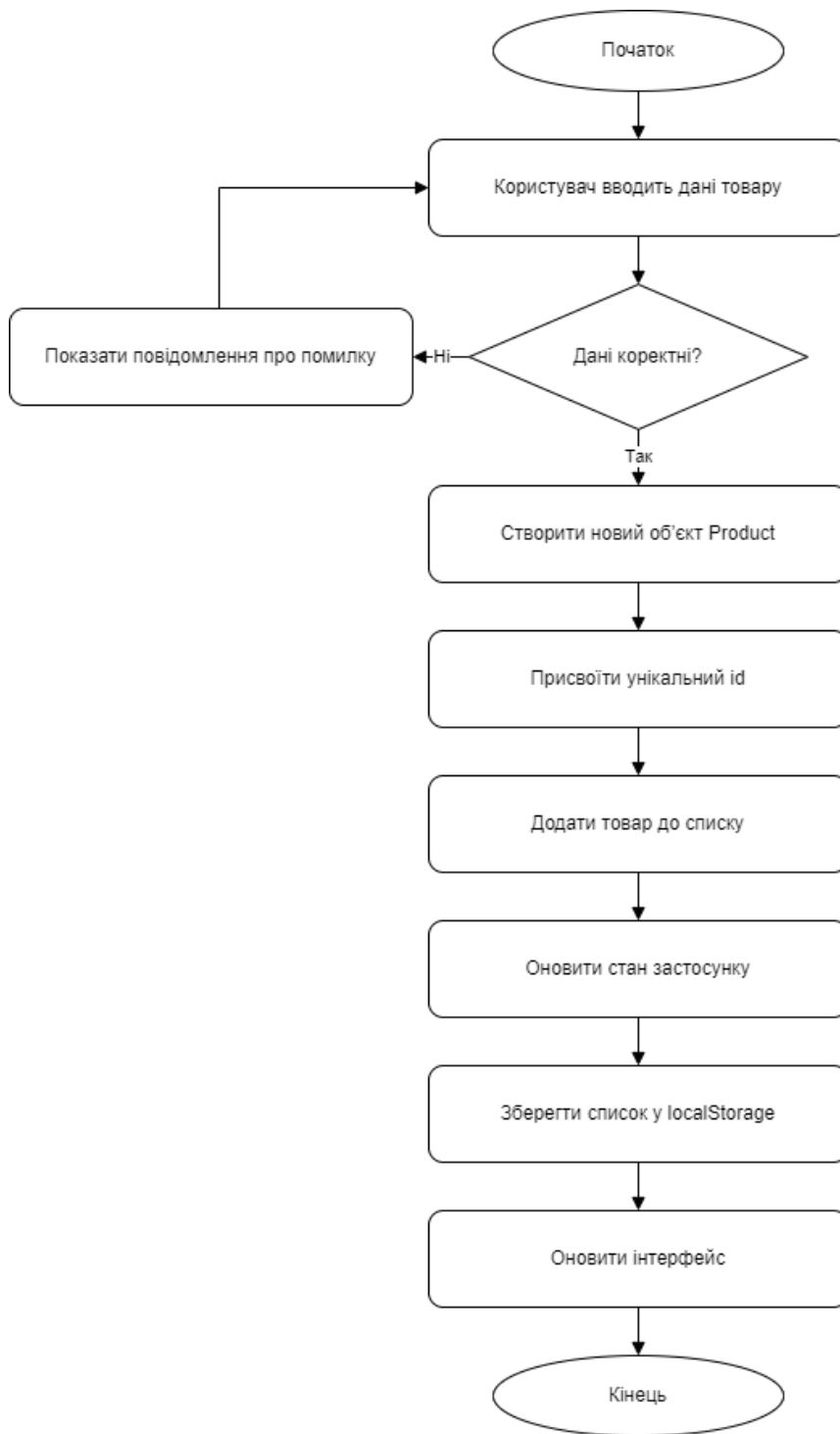


Рисунок 2.3 – Блок схема алгоритмів роботи програмного забезпечення

Алгоритм редагування товару використовується тоді, коли користувач хоче виправити або уточнити вже додану позицію. Наприклад, можна змінити кількість, категорію або орієнтовну ціну. Для цього система повинна знайти потрібний товар за його id. Після внесення змін програма оновлює відповідний

об'єкт у списку та знову зберігає дані. Важливо, що редагування не повинно створювати новий товар, а має змінювати саме той запис, який обрав користувач.

Алгоритм видалення товару є простішим. Користувач натискає кнопку видалення біля певної позиції, після чого система отримує ідентифікатор цього товару. Далі зі списку прибирається об'єкт із відповідним id. Після видалення список оновлюється на екрані, а його новий стан записується у локальне сховище. Якщо після цього список стає порожнім, користувачу потрібно показати повідомлення, що товари ще не додані або список очищено.

Позначення товару як купленого також виконується через зміну одного поля. Коли користувач відмічає товар, система змінює значення `isPurchased`. Якщо товар ще не був куплений, він переходить у стан купленого. Якщо користувач натискає повторно, статус можна повернути назад. Це зручно, бо під час покупки легко випадково натиснути не на той товар, тому можливість скасувати дію є корисною.

Після зміни статусу товару має оновлюватися не тільки його зовнішній вигляд, а й підсумкова інформація. Наприклад, якщо один товар позначено як куплений, кількість куплених позицій збільшується, а кількість некуплених зменшується. Такий результат користувач повинен бачити одразу, без перезавантаження сторінки.

Окремий алгоритм відповідає за фільтрацію товарів за категоріями. Він потрібен тоді, коли список стає більшим і користувачу незручно переглядати всі позиції одразу. Якщо вибрано фільтр «Усі», система показує повний список. Якщо вибрана конкретна категорія, програма відображає тільки ті товари, у яких поле `category` збігається з обраним значенням. Важливо, що фільтрація не змінює сам список товарів. Вона лише впливає на те, що показується на екрані.

Алгоритм підрахунку загальної суми працює з кількістю та ціною кожного товару. Для кожної позиції система множить `quantity` на `price`, після чого додає отримані значення. У результаті користувач бачить приблизну вартість усієї покупки. Якщо ціна певного товару не вказана або дорівнює нулю, такий товар залишається у списку, але фактично не впливає на загальну суму.

Крім загальної суми, система повинна рахувати кількість усіх товарів, кількість куплених і некуплених позицій. Загальна кількість визначається за довжиною масиву товарів. Кількість куплених позицій можна отримати через підрахунок товарів, у яких `isPurchased` має значення `true`. Кількість некуплених товарів можна рахувати окремо або визначати як різницю між загальною кількістю та кількістю куплених.

Важливе місце займає алгоритм збереження та відновлення даних. Після кожної зміни списку система перетворює масив товарів у JSON-рядок і записує його у `localStorage`. Коли користувач відкриває застосунок знову, система перевіряє, чи є збережені дані. Якщо вони є, програма зчитує їх і перетворює назад у масив товарів. Якщо дані відсутні або некоректні, застосунок починає роботу з порожнім списком. Такий підхід робить програму більш стабільною, оскільки вона не залежить від сервера.

Усі описані алгоритми пов'язані між собою. Наприклад, після додавання або видалення товару потрібно не тільки оновити список, а й перерахувати підсумкову суму та зберегти дані. Після зміни статусу купівлі також оновлюється підсумкова панель. Тому в реалізації важливо, щоб стан застосунку був єдиним джерелом даних, а всі компоненти отримували актуальну інформацію з нього.

Якщо коротко описати загальну логіку роботи системи, вона виглядає так: користувач виконує дію, програма перевіряє або обробляє введені дані, оновлює список товарів, перераховує підсумкові значення, зберігає новий стан і показує оновлений результат на екрані. Така послідовність повторюється майже для всіх основних дій у застосунку.

Запропоновані алгоритми є достатніми для реалізації демонстраційного frontend-прототипу. Вони не потребують складної серверної логіки, але повністю покривають основні сценарії роботи користувача. Крім того, частину цих алгоритмів можна винести в окремі функції й протестувати. Наприклад, підрахунок суми, фільтрацію та підрахунок куплених товарів зручно перевіряти за допомогою модульних тестів.

Алгоритми роботи програмного забезпечення забезпечують повний цикл взаємодії користувача зі списком покупок. Вони дозволяють додавати, редагувати й видаляти товари, змінювати статус купівлі, фільтрувати список, рахувати приблизну суму та зберігати дані в браузері. На основі цих алгоритмів можна переходити до опису практичної реалізації програмного забезпечення.

3 РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Розробка основних модулів програмного забезпечення

Після проєктування архітектури, структури даних і основних алгоритмів можна перейти до практичної реалізації програмного забезпечення. У межах дипломної роботи було розроблено frontend-застосунок для планування покупок продуктів харчування сім'ї. Його основна задача — дати користувачу простий інструмент для створення списку покупок, редагування товарів, позначення куплених позицій і перегляду приблизної суми.

Розробка виконувалася з урахуванням того, що застосунок має бути невеликим і зрозумілим. Через це логіку було поділено на окремі компоненти й модулі. Такий поділ допомагає не змішувати все в одному файлі: інтерфейс знаходиться в компонентах, структура даних описується окремо, функції підрахунку винесені в допоміжний модуль, а робота з localStorage — у службовий файл (рис. 3.1).

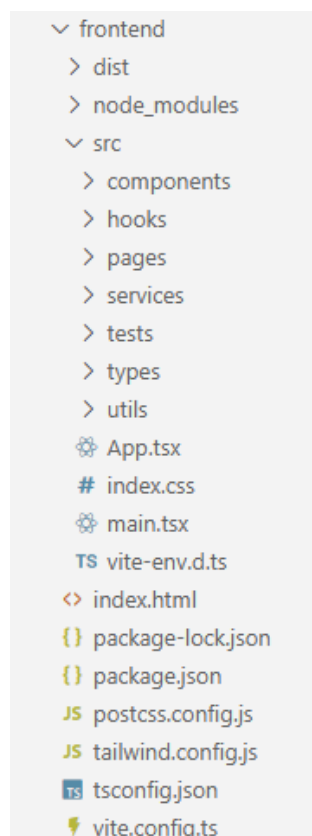


Рисунок 3.1 – Структура файлів frontend-проєкту

Основним файлом застосунку є `App.tsx`. У ньому зберігається список покупок і функції, які змінюють цей список. Саме тут виконується додавання нового товару, редагування, видалення, зміна статусу купівлі та очищення списку. Також у цьому компоненті відбувається завантаження даних із `localStorage` під час запуску програми та збереження оновленого списку після змін.

Компонент `Header` відповідає за верхню частину сторінки. Він містить назву застосунку і короткий опис його призначення. Цей компонент не виконує складних дій, але потрібен для того, щоб користувач одразу розумів, з якою системою працює.

Компонент `ShoppingForm` реалізує форму додавання товару. У ній користувач вводить назву продукту, обирає категорію, зазначає кількість, одиницю вимірювання та приблизну ціну. Перед додаванням товару виконується проста перевірка. Якщо назва порожня або кількість введена некоректно, товар не повинен додаватися до списку. Це допомагає уникнути помилкових записів.

Компонент `ShoppingList` відповідає за відображення всіх товарів. Він отримує список із головного компонента і показує його користувачу. Якщо список порожній, замість порожнього місця відображається повідомлення, що товари ще не додані. Це робить інтерфейс більш зрозумілим при першому запуску застосунку.

Для кожного окремого товару використовується компонент `ShoppingItem`. У ньому показується назва продукту, категорія, кількість, одиниця вимірювання, ціна та сума за позицію. Також у цьому компоненті є кнопки для редагування, видалення та позначення товару як купленого. Якщо товар уже куплений, його вигляд змінюється, щоб користувач одразу бачив виконані позиції.

Компонент `CategoryFilter` потрібен для фільтрації товарів за категоріями. Наприклад, користувач може переглянути тільки молочні продукти або тільки овочі та фрукти. При цьому сам список не змінюється, змінюється лише те, що показується на екрані. Це зручно, коли список покупок містить багато товарів.

Компонент `SummaryPanel` показує підсумкову інформацію. У ньому відображається загальна кількість позицій, кількість куплених і некуплених товарів, а також приблизна загальна сума покупки. Цей блок оновлюється автоматично після кожної зміни списку, тому користувач завжди бачить актуальний стан покупки.

Основні модулі та компоненти програмного забезпечення подано в таблиці (табл. 3.1)

Таблиця 3.1 – Основні модулі програмного забезпечення

Модуль або компонент	Призначення
<code>App.tsx</code>	Керує станом застосунку та основними діями зі списком
<code>ShoppingForm.tsx</code>	Забезпечує додавання нового товару
<code>ShoppingList.tsx</code>	Відображає список покупок
<code>ShoppingItem.tsx</code>	Показує один товар і кнопки дій
<code>CategoryFilter.tsx</code>	Фільтрує товари за категоріями
<code>SummaryPanel.tsx</code>	Відображає підсумкову інформацію
<code>storageService.ts</code>	Зберігає та завантажує список із <code>localStorage</code>
<code>calculations.ts</code>	Містить функції підрахунку і фільтрації
<code>product.ts</code>	Описує структуру товару

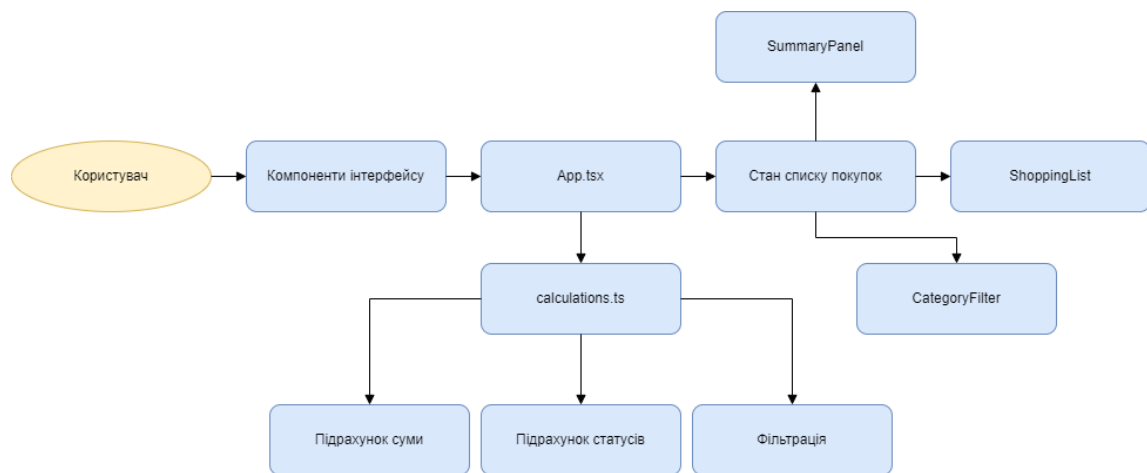
Окремо було створено файл `product.ts`, у якому описується структура товару. У TypeScript це можна зробити за допомогою інтерфейсу. Такий підхід дає змогу чітко визначити, які саме поля має містити кожен товар [11].

Для підрахунку загальної суми було винесено окрему функцію. Це зручніше, ніж рахувати суму безпосередньо в компоненті інтерфейсу. Якщо логіка підрахунку знаходиться в окремому файлі, її простіше перевірити і за потреби змінити.

Функція проходить по всіх товарах у списку, множить кількість на ціну і додає результат до загальної суми. Саме це значення потім відображається у підсумковій панелі.

Для збереження даних використовується `localStorage`. Щоб не повторювати однаковий код у різних частинах програми, роботу зі сховищем було винесено в окремий модуль `storageService.ts` [13].

Під час завантаження даних потрібно враховувати, що `localStorage` може бути порожнім. Також дані можуть бути некоректними, якщо вони були змінені вручну або пошкоджені. Тому функція завантаження має повертати порожній список у разі помилки, щоб застосунок не припиняв роботу (рис. 3.2).



Рисутнок 3.2 – Схема взаємодії програмних модулів

У розробленому застосунку всі основні дії користувача проходять через головний компонент. Наприклад, коли користувач додає товар, форма передає дані до `App.tsx`. Головний компонент створює новий об'єкт товару, додає його до списку, після чого оновлений список передається в компоненти відображення. Одночасно список зберігається у `localStorage`.

Такий підхід робить роботу системи зрозумілою. Дані не розкидані по різних частинах застосунку, а змінюються через конкретні функції. Це спрощує підтримку коду і допомагає швидше знаходити помилки під час тестування.

Реалізовані модулі забезпечують виконання всіх основних функцій, які були визначені раніше. Користувач може додавати товари, редагувати їх, видаляти, позначати як куплені, фільтрувати за категоріями та переглядати

підсумкову інформацію. Крім того, список зберігається у браузері, тому після перезавантаження сторінки дані залишаються доступними.

Отже, основні модулі програмного забезпечення реалізовано відповідно до спроектованої архітектури. Компонентний підхід дозволив розділити інтерфейс на окремі частини, а службові модулі зробили код більш організованим. Це створює основу для подальшого опису інтерфейсу користувача і тестування роботи системи.

3.2 Розробка інтерфейсу користувача

Після реалізації основних модулів потрібно описати інтерфейс користувача, оскільки саме через нього відбувається вся робота із застосунком. Для програмного забезпечення планування покупок інтерфейс має бути простим і зрозумілим. Користувач не повинен витрачати час на пошук потрібної кнопки або розбиратися зі складною навігацією. У побутових задачах це особливо важливо, бо список покупок часто створюється швидко, іноді вже перед самим походом до магазину (рис. 3.3).

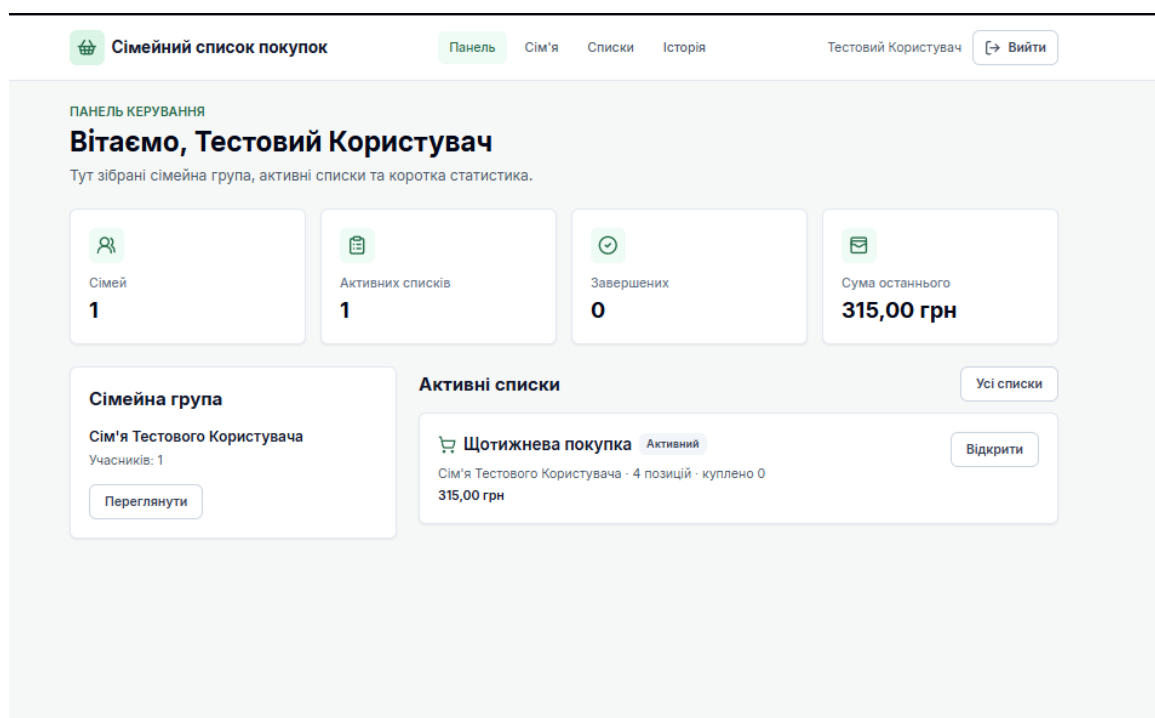


Рисунок 3.3 – Головна сторінка веб-застосунку

Інтерфейс розробленого застосунку побудований як одна сторінка. На ній розміщено всі основні елементи: заголовок, форму додавання товару, фільтр категорій, список покупок і підсумковий блок. Такий підхід є зручним, тому що користувач одразу бачить усю потрібну інформацію і може виконувати дії без переходу між різними сторінками.

У верхній частині сторінки розміщується заголовок застосунку. Він містить назву системи та коротке пояснення її призначення. Наприклад, назва може бути «Сімейний список покупок», а під нею можна додати короткий опис: «Плануйте продукти, контролюйте покупки та приблизну суму витрат». Такий блок не є складним з технічного погляду, але він робить інтерфейс більш завершеним.

Одним із головних елементів сторінки є форма додавання товару. Вона повинна бути помітною і зручною, бо саме з неї починається основна робота користувача. У формі передбачено поля для назви продукту, категорії, кількості, одиниці вимірювання та приблизної ціни. Для категорій і одиниць вимірювання краще використовувати випадаючі списки, а не звичайне текстове введення. Це зменшує кількість помилок і робить дані більш однаковими (рис. 3.4).

The screenshot displays the 'Щотижнева покупка' (Weekly Shopping) app interface. At the top, there is a navigation bar with a back arrow labeled 'До списків' and two buttons: 'Завершити список' (Complete list) and 'Архівувати' (Archive). Below the title, the status is 'Статус: активний'.

The main form for adding a product includes the following fields:

- Товар** (Product): A text input field with the placeholder 'Назва продукту'.
- Категорія** (Category): A dropdown menu currently showing 'Інше'.
- Кількість** (Quantity): A numeric input field with the value '1'.
- Одиниця** (Unit): A dropdown menu currently showing 'шт.'.
- Ціна** (Price): A numeric input field with the value '0'.
- Додати** (Add): A green button to submit the form.

Below the form, a summary bar displays the following statistics:

усього позицій	куплено	залишилось купити	приблизна сума	сума купленого
4	0	4	315,00 грн	0,00 грн

At the bottom, there is a 'Товари' (Products) section with a filter bar. The 'Усі' (All) filter is selected. Other filters include 'Овочі та фрукти', 'Молочні продукти', 'М'ясо та риба', 'Хлібобулочні вироби', 'Крупи та макарони', and 'Напої'. Below these, there are two more filters: 'Побутові товари' and 'Інше'.

The bottom of the screen shows a table header with columns: 'СТАТУС ТОВАРУ', 'КІЛЬКІСТЬ', 'ЦІНА', 'СУМА', and 'ДІЇ'.

Рисунок 3.4 – Форма додавання нового продукту

Поля форми мають бути підписані зрозумілою мовою. Наприклад: «Назва продукту», «Категорія», «Кількість», «Одиниця вимірювання», «Ціна за одиницю». Якщо користувач вводить некоректні дані, система не повинна додавати товар до списку. Для такого застосунку достатньо базової перевірки: назва не може бути порожньою, кількість має бути більшою за нуль, а ціна не повинна бути від'ємною.

Після додавання товар одразу з'являється у списку покупок. Це важливо, тому що користувач повинен бачити результат своєї дії без перезавантаження сторінки. Якщо список був порожній, повідомлення про порожній стан зникає, а замість нього відображається перша позиція. Така поведінка робить роботу із застосунком більш природною.

Список покупок є центральною частиною інтерфейсу. У ньому показуються всі додані товари. Для кожної позиції потрібно відобразити назву, категорію, кількість, одиницю вимірювання, ціну та суму за конкретний товар. Також біля кожного товару мають бути кнопки для редагування, видалення та позначення товару як купленого.

Для широкого екрана список можна зробити у вигляді таблиці або рядків, схожих на таблицю. Так користувач бачить багато інформації компактно. Для смартфона краще використовувати картковий вигляд, коли кожен товар показується окремим блоком. Це зручніше для натискання пальцем і краще пристосовується до малої ширини екрана. Оскільки покупки часто контролюють саме зі смартфона, адаптивність є важливою частиною інтерфейсу [№].

Куплені товари повинні відрізнятися від тих, які ще потрібно придбати. Найпростіший спосіб — закреслити назву або зробити картку менш помітною. Це допомагає користувачу швидко бачити, які товари вже куплені, а які ще залишилися. При цьому бажано залишити можливість повернути товар у стан некупленого, якщо користувач випадково натиснув не ту кнопку.

Фільтр категорій використовується для зручнішої роботи з великим списком. Якщо користувач додав багато товарів, йому може бути незручно переглядати все одразу. Тому він може вибрати конкретну категорію, наприклад

«Молочні продукти» або «Овочі та фрукти». Після цього на екрані залишаються тільки товари цієї категорії, але сам список у пам’яті не змінюється (рис. 3.5).

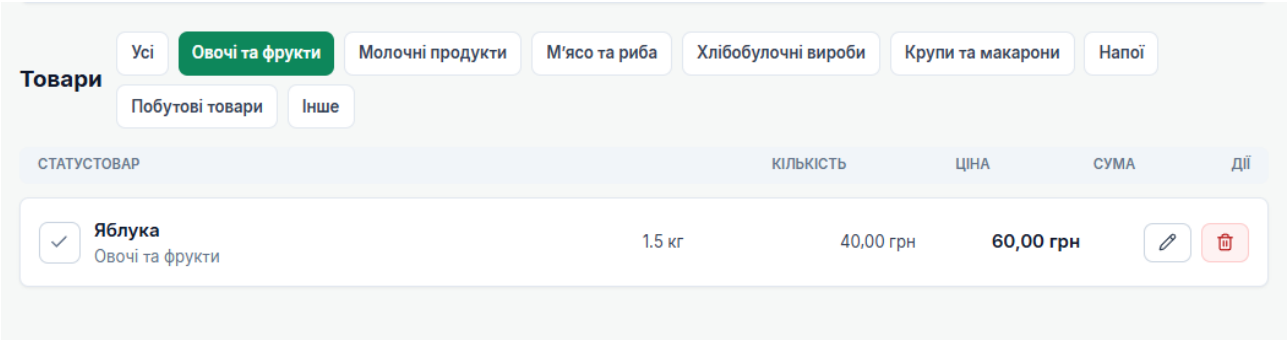


Рисунок 3.5 – Відображення списку покупок за категоріями

Підсумкова панель допомагає користувачу швидко оцінити стан списку. У ній варто показувати загальну кількість товарів, кількість куплених позицій, кількість некуплених позицій і приблизну загальну суму. Цей блок повинен оновлюватися автоматично після кожної дії користувача. Наприклад, якщо товар видалено або змінено його ціну, підсумкова сума має одразу перерахуватися.

Основні елементи інтерфейсу можна коротко подати в таблиці (табл. 3.2).

Таблиця 3.2 – Основні елементи інтерфейсу користувача

Елемент інтерфейсу	Призначення
Заголовок	Показує назву та призначення застосунку
Форма додавання	Дозволяє ввести новий товар
Фільтр категорій	Допомагає переглядати товари певного типу
Список покупок	Відображає додані товари
Кнопки дій	Дозволяють редагувати, видаляти та відмічати товари
Підсумкова панель	Показує кількість товарів і приблизну суму
Порожній стан	Повідомляє, що список ще не заповнений

Окремо потрібно передбачити порожній стан списку. Якщо користувач відкрив застосунок уперше або очистив список, не варто залишати центральну

частину сторінки пустою. Краще показати коротке повідомлення, наприклад: «Список покупок поки порожній. Додайте перший продукт». Це невелика деталь, але вона робить інтерфейс зрозумілішим.

Під час розробки інтерфейсу важливо не перевантажувати сторінку. Для цієї системи не потрібні складні меню, багато вкладок або зайві декоративні блоки. Головне — щоб користувач міг швидко додати товар, побачити список і відмітити куплені позиції. Саме тому дизайн має бути стриманим, а всі елементи повинні мати практичне призначення.

Завдяки використанню Tailwind CSS інтерфейс можна зробити адаптивним без великої кількості окремого CSS-коду. Класи для відступів, розміщення блоків, ширини, кольорів і станів кнопок задаються безпосередньо в компонентах. Це прискорює розробку і дозволяє швидко змінювати зовнішній вигляд окремих елементів [11].

Отже, інтерфейс користувача в розробленому застосунку побудований за принципом простоти й швидкого доступу до основних функцій. Користувач бачить форму додавання товару, список покупок, фільтр категорій і підсумкову інформацію на одній сторінці. Такий підхід відповідає реальному сценарію використання системи та робить програмне забезпечення зручним як для комп'ютера, так і для смартфона.

3.3 Тестування програмного забезпечення

Після розробки основних модулів та інтерфейсу користувача потрібно перевірити, чи правильно працює програмне забезпечення. У цій роботі тестування виконується не тільки для пошуку помилок, а й для підтвердження того, що застосунок відповідає поставленим вимогам. Користувач повинен мати можливість пройти повний сценарій роботи: додати товар, переглянути список, змінити дані, позначити товар як куплений, відфільтрувати позиції за категорією та побачити правильну підсумкову суму.

Оскільки розроблений застосунок є frontend-прототипом без backend, основна увага приділяється перевірці інтерфейсу, коректності обробки даних, роботі з localStorage і правильності розрахунків. У межах тестування не перевіряються серверні запити, авторизація користувачів або синхронізація між пристроями, оскільки ці функції не входять до поточної реалізації.

Першим етапом було перевірено додавання нового товару до списку. Для цього у форму вводиться назва продукту, категорія, кількість, одиниця вимірювання та приблизна ціна. Якщо всі дані введено правильно, після натискання кнопки додавання товар має з'явитися у списку покупок. Також повинна оновитися підсумкова панель, у якій відображається кількість позицій і загальна сума (рис. 3.6).

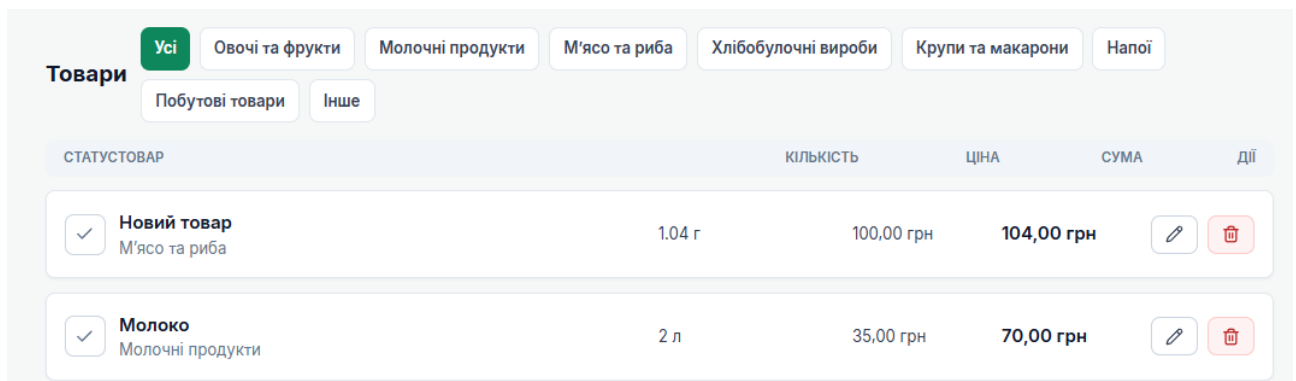


Рисунок 3.6 – Додавання нового товару до списку покупок

Окремо перевірялася обробка некоректних даних. Наприклад, якщо користувач намагається додати товар без назви, із кількістю 0 або з від'ємною ціною, система не повинна додавати таку позицію до списку. Це важливо, оскільки неправильні дані можуть вплинути на підрахунок загальної суми та зробити список незручним для використання.

Наступним етапом було перевірено відображення списку покупок. Після додавання кількох товарів різних категорій система повинна показувати їх у зрозумілому вигляді. Для кожної позиції мають бути видимі назва, категорія, кількість, одиниця вимірювання, ціна та кнопки дій. Це дозволяє переконатися, що дані не тільки зберігаються, а й коректно відображаються для користувача.

Редагування товару перевірялося шляхом зміни вже доданої позиції. Наприклад, користувач може змінити кількість продукту або його орієнтовну ціну. Після збереження змін система повинна оновити саме вибраний товар, не змінюючи інші позиції у списку. Для цього використовується унікальний ідентифікатор товару, який дозволяє програмі правильно знайти потрібний запис (рис. 3.7).

Статус	Товар	Кількість	Ціна	Сума	Дії
☒	Куряче філе М'ясо та риба	1 кг	160,00 грн	160,00 грн	
☒	Яблука				

Рисунок 3.7 – Редагування даних товару у списку покупок

Також було перевірено видалення товару. Після натискання кнопки видалення відповідна позиція має зникнути зі списку. Після цього підсумкова інформація повинна перерахуватися автоматично. Якщо видаляється останній товар, система має показати повідомлення про порожній список. Це підтверджує, що інтерфейс правильно реагує на зміну стану списку (рис. 3.8).

Статус	Товар	Кількість	Ціна	Сума	Дії
☒	Молоко Молочні продукти	2 л	35,00 грн	70,00 грн	
☒	Куряче філе М'ясо та риба	1 кг	160,00 грн	160,00 грн	
☒	Яблука				

Рисунок 3.8 – Видалення товару зі списку покупок

Однією з важливих функцій є позначення товару як купленого. Під час тестування перевірялося, що після натискання відповідної кнопки або чекбокса товар змінює свій статус. Куплений товар повинен візуально відрізнятися від некупленого, наприклад через закреслену назву або менш помітний стиль. Якщо користувач натискає повторно, товар має повернутися до стану некупленого (рис. 3.9).

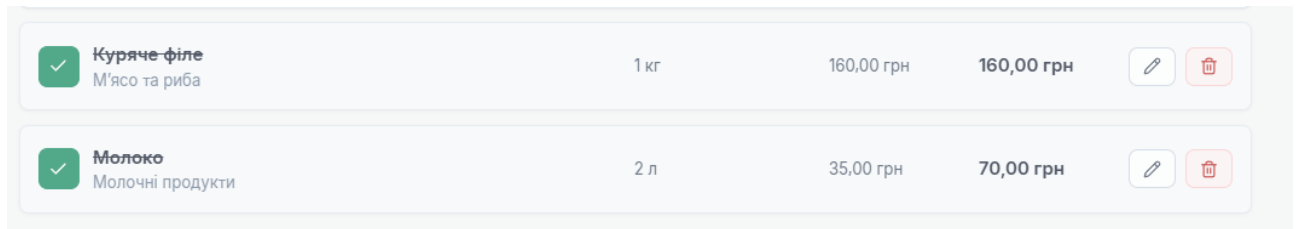


Рисунок 3.9 – Позначення товару як купленого

Фільтрація за категоріями перевірялася на списку, який містить товари різних типів. Наприклад, якщо у списку є молоко, хліб, яблука та вода, при виборі категорії «Молочні продукти» має відображатися лише молоко. При поверненні до фільтра «Усі» система повинна знову показати повний список. Важливо, що фільтрація не повинна видаляти товари, а лише змінювати їх відображення на екрані (рис. 3.10).

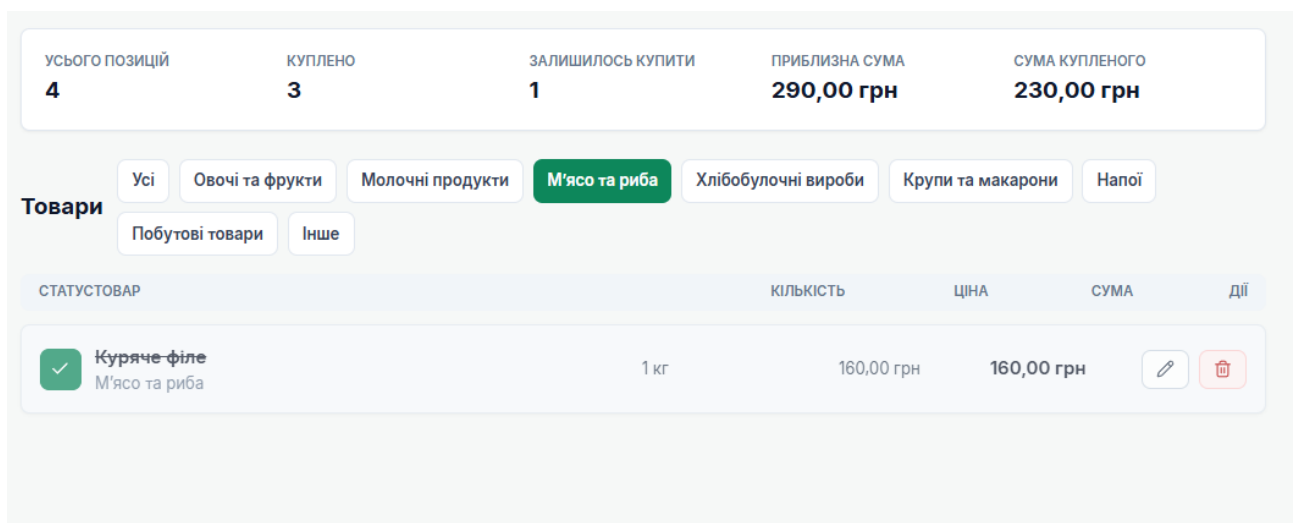


Рисунок 3.10 – Фільтрація товарів за категорією

Окремо перевірявся підрахунок підсумкової інформації. Система повинна правильно рахувати загальну кількість товарів, кількість куплених і некуплених позицій, а також приблизну суму покупки. Наприклад, якщо додано два товари по 35 грн і один товар за 25 грн, загальна сума повинна відповідати результату арифметичного підрахунку. Якщо товар редагується або видаляється, підсумкова панель має оновлюватися автоматично.

Важливою частиною тестування була перевірка збереження даних у браузері. Після додавання товарів сторінку було перезавантажено. Якщо localStorage працює правильно, список покупок повинен залишитися на місці. При цьому мають зберігатися не тільки назви товарів, а й категорії, кількість, ціна та статус купівлі. Це підтверджує, що користувач не втратить список після випадкового закриття сторінки (рис. 3.11).

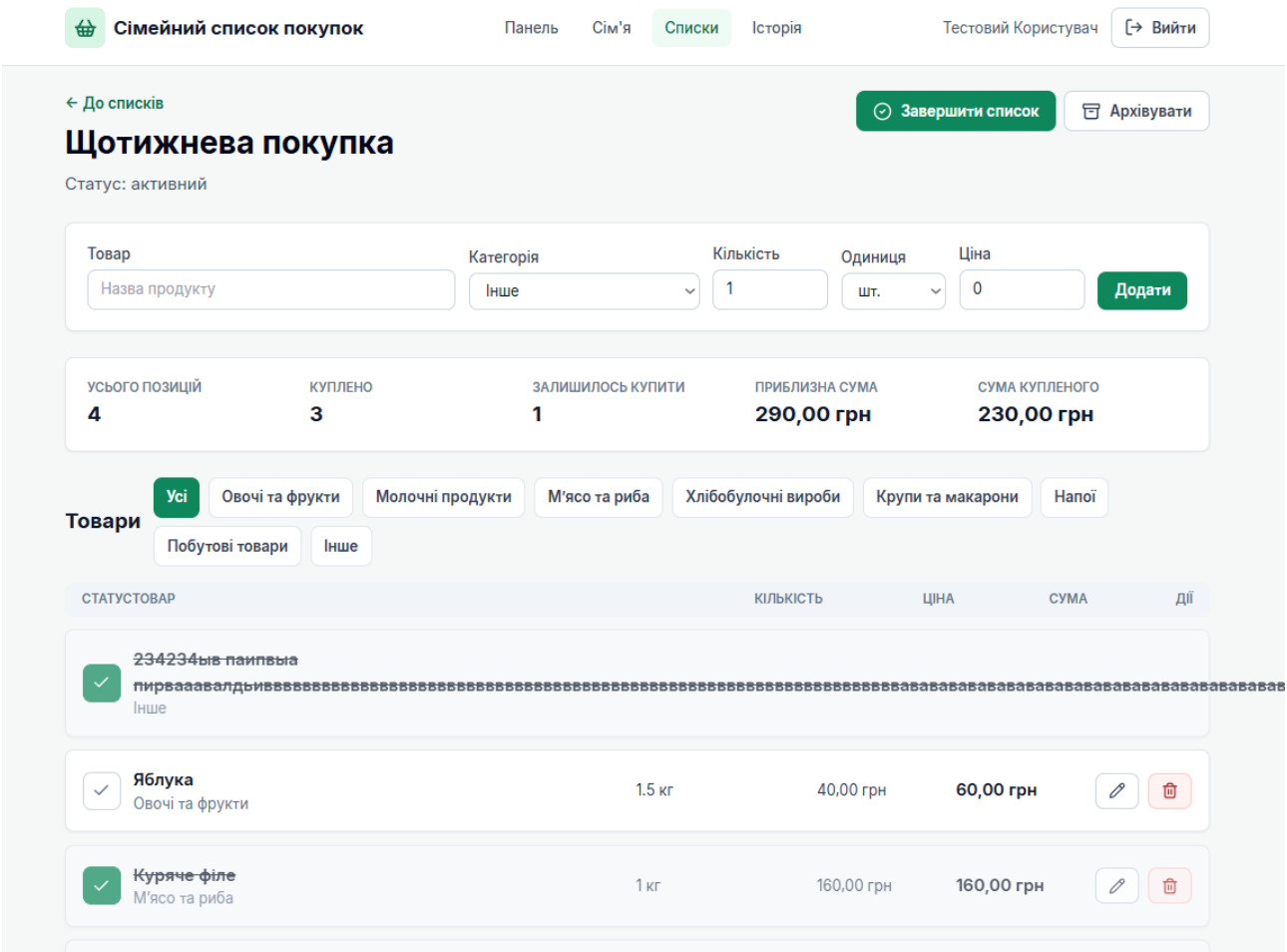


Рисунок 3.11 – Відновлення списку покупок після перезавантаження сторінки

Також було перевірено адаптивність інтерфейсу. Оскільки список покупок може використовуватися не тільки на комп'ютері, а й на смартфоні, застосунок повинен коректно відображатися на різних розмірах екрана. На мобільному пристрої форма, список і кнопки мають бути достатньо зручними для натискання. Елементи не повинні виходити за межі екрана або накладатися один на одного (рис. 3.12).

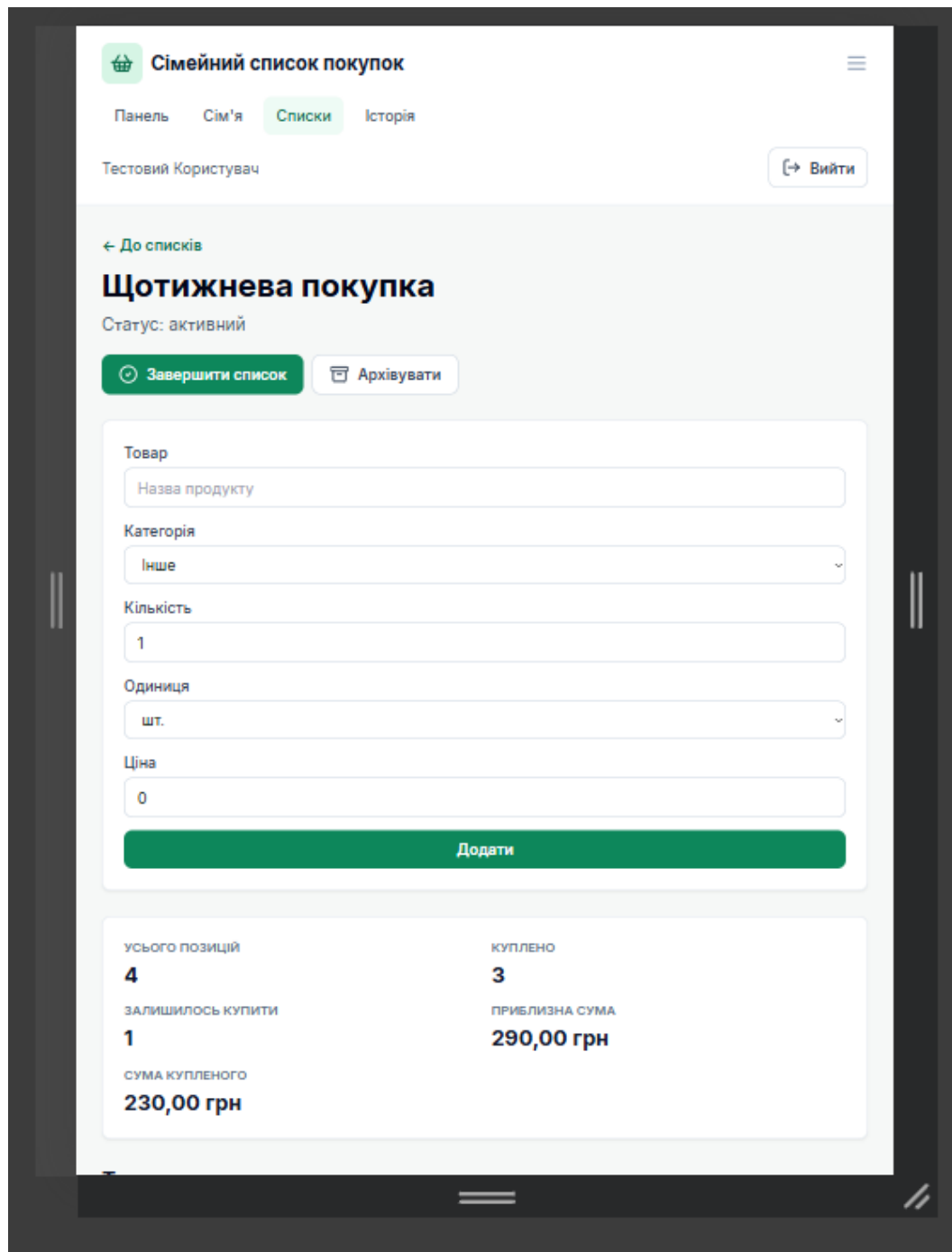


Рисунок 3.12 – Вигляд застосунку на мобільному пристрої

Основні тестові сценарії програмного забезпечення подано в таблиці (табл. 3.3).

Таблиця 3.3 – Тестові сценарії програмного забезпечення

№	Сценарій перевірки	Очікуваний результат
1	Додавання товару з коректними даними	Товар з'являється у списку, підсумкова панель оновлюється
2	Додавання товару без назви	Товар не додається до списку
3	Додавання товару з кількістю 0	Товар не додається до списку
4	Додавання товару з від'ємною ціною	Товар не додається до списку
5	Редагування товару	Дані вибраного товару змінюються
6	Видалення товару	Товар зникає зі списку
7	Позначення товару як купленого	Статус і зовнішній вигляд товару змінюються
8	Повторне натискання на куплений товар	Товар повертається до стану некупленого
9	Фільтрація за категорією	Відображаються тільки товари вибраної категорії
10	Повернення до фільтра «Усі»	Відображається повний список товарів
11	Підрахунок загальної суми	Сума відповідає кількості та ціні товарів
12	Перезавантаження сторінки	Список відновлюється з localStorage
13	Очищення списку	Усі товари видаляються, показується порожній стан
14	Перевірка мобільної версії	Інтерфейс коректно відображається на вузькому екрані

Крім ручного тестування, доцільно використати модульні тести для окремих функцій. Це стосується тих частин програми, які не залежать від інтерфейсу. Наприклад, функція підрахунку загальної суми отримує масив товарів і повертає число. Її можна перевірити окремо, без запуску повного інтерфейсу. Для цього підходить Vitest, який використовується для тестування функцій у frontend-проектах [29], [30].

Також можна протестувати функції підрахунку куплених і некуплених товарів. Наприклад, якщо у списку є три товари, з яких один має статус `isPurchased = true`, функція підрахунку куплених товарів повинна повернути 1, а функція підрахунку некуплених — 2. Такі прості тести допомагають швидко перевірити правильність логіки.

Під час тестування було підтверджено, що застосунок виконує основні дії коректно. Товари додаються до списку, редагуються, видаляються, фільтруються за категоріями, а підсумкова інформація оновлюється після кожної зміни. Збереження у `localStorage` також працює правильно: після перезавантаження сторінки список відновлюється.

Разом із тим потрібно враховувати межі проведеного тестування. Оскільки система не має серверної частини, у ній не перевірялася робота з акаунтами користувачів, базою даних, API або синхронізацією між пристроями. Ці функції можна тестувати лише після майбутнього розширення системи.

Отже, тестування показало, що розроблене програмне забезпечення відповідає основним вимогам. Користувач може створити список покупок, змінювати його, відмічати куплені товари, переглядати суму та зберігати дані у браузері. Наявність кількох скріншотів тестових сценаріїв дозволяє краще продемонструвати роботу практичної частини дипломної роботи.

3.4 Аналіз результатів роботи програмного забезпечення

Після розробки та тестування програмного забезпечення потрібно оцінити отримані результати. На цьому етапі важливо визначити, чи відповідає створений веб-застосунок поставленій задачі, чи реалізовано основні функції та які обмеження має розроблене рішення.

У межах дипломної роботи було створено демонстраційний frontend-застосунок для планування покупок продуктів харчування сім'ї. Застосунок дозволяє користувачу створювати список покупок, додавати нові товари, редагувати їх, видаляти непотрібні позиції, позначати товари як куплені,

фільтрувати список за категоріями та переглядати приблизну загальну суму покупки.

Головним результатом є те, що система автоматизує базовий процес ведення списку покупок. Замість паперового списку або звичайної нотатки користувач отримує структурований інструмент, у якому кожен товар має назву, категорію, кількість, одиницю вимірювання, ціну та статус купівлі. Це робить список зрозумілішим і зручнішим для використання.

Розроблений застосунок відповідає вимогам, визначеним у першому розділі. Користувач може пройти повний сценарій роботи: від створення списку до завершення покупки. Він додає товари, переглядає їх у списку, за потреби фільтрує за категоріями, відмічає куплені позиції та бачить підсумкову суму. Усі зміни зберігаються локально у браузері, тому список не зникає після перезавантаження сторінки.

Однією з переваг системи є простота інтерфейсу. Усі основні елементи розміщені на одній сторінці, тому користувачу не потрібно переходити між різними розділами. Форма додавання товару, список покупок, фільтр і підсумкова панель доступні одразу після відкриття застосунку. Для побутової задачі це зручно, оскільки список покупок зазвичай потрібно створювати швидко.

Ще однією важливою перевагою є категоризація товарів. Завдяки категоріям список стає більш упорядкованим. Користувач може переглядати не всі товари одразу, а лише певну групу, наприклад молочні продукти, овочі та фрукти або напої. Це особливо корисно під час покупки в магазині, де товари також розташовані за відділами.

Підрахунок приблизної суми дозволяє краще контролювати витрати. Користувач може ще до покупки побачити орієнтовну загальну вартість списку. Якщо сума виходить занадто великою, можна змінити кількість товарів або видалити частину позицій. Для сімейного бюджету така функція є корисною, навіть якщо ціни вводяться приблизно.

Розроблене програмне забезпечення також має адаптивний інтерфейс. Це важливо, тому що список покупок може створюватися на комп'ютері, а використовуватися вже зі смартфона в магазині. На вузькому екрані елементи інтерфейсу повинні залишатися зручними для перегляду та натискання.

Попри реалізацію основних функцій, система має певні обмеження. Найперше обмеження пов'язане з локальним збереженням даних. Список покупок зберігається тільки в браузері конкретного пристрою. Якщо користувач відкриє застосунок на іншому телефоні або комп'ютері, список автоматично не перенесеться.

Також у поточній версії не реалізовано спільний доступ для кількох членів сім'ї. Це означає, що кілька користувачів не можуть одночасно редагувати один список. Для реалізації такої можливості потрібно було б додати серверну частину, авторизацію та базу даних. У межах цієї дипломної роботи ці функції не передбачалися, оскільки основна мета полягала у створенні frontend-прототипу.

Ще одним обмеженням є те, що ціни вводяться користувачем вручну. Система не отримує актуальні ціни з магазинів і не порівнює вартість товарів у різних торгових мережах. Проте для демонстраційного застосунку цього достатньо, оскільки головна задача полягає не в інтеграції з магазинами, а в організації списку покупок.

У майбутньому програмне забезпечення можна розширити. Насамперед можна додати backend і базу даних, щоб зберігати списки на сервері. Це дозволить користувачу відкривати один і той самий список із різних пристроїв. Також можна реалізувати сімейні акаунти, щоб кілька людей могли додавати товари до спільного списку.

Іншим напрямом розвитку є історія покупок. Користувач міг би переглядати попередні списки, повторювати типові покупки або аналізувати, які продукти купуються найчастіше. Також можна додати шаблони списків, наприклад «щотижнева покупка», «сніданки», «побутові товари» або «святковий список».

Корисним доповненням могла б бути аналітика витрат за категоріями. Наприклад, система могла б показувати, скільки коштів витрачається на молочні продукти, овочі, м'ясо, напої або побутові товари. Така функція зробила б застосунок корисним не лише для складання списку, а й для контролю сімейного бюджету.

Загалом розроблений застосунок виконує поставлену задачу. Він не є складною комерційною системою, але для дипломного проєкту має завершений вигляд і демонструє основні можливості програмного забезпечення для планування покупок. Реалізовані функції відповідають темі роботи, а архітектура дозволяє розширювати систему в майбутньому.

Отже, результати розробки показують, що створене програмне забезпечення є працездатним і відповідає визначеним вимогам. Застосунок дозволяє організувати список покупок, контролювати статус товарів, бачити приблизну суму та зберігати дані у браузері. Попри певні обмеження, розроблений frontend-прототип може бути використаний як основа для подальшого розвитку повноцінної інформаційної системи для сімейного планування покупок.

ВИСНОВКИ

У дипломній роботі було розглянуто розробку програмного забезпечення для планування покупок продуктів харчування сім'ї. Тема є актуальною, оскільки планування покупок належить до регулярних побутових задач, з якими стикається більшість сімей. Якщо список продуктів ведеться без чіткої структури, користувач може забути частину товарів, купити зайві продукти або не врахувати приблизну суму витрат. Через це виникає потреба у простому цифровому інструменті, який допомагає зробити процес покупок більш організованим.

У першому розділі було проаналізовано предметну область планування покупок продуктів харчування. Було визначено, що цей процес складається не лише зі звичайного запису продуктів, а й з перевірки залишків удома, уточнення кількості, розподілу товарів за категоріями, оцінки приблизної вартості та контролю виконання списку під час покупки. Також було розглянуто поширені способи ведення списків, зокрема паперові записи, нотатки у смартфоні та спеціалізовані застосунки.

У роботі було виконано огляд існуючих програмних рішень для планування покупок. Було розглянуто звичайні нотатки смартфона, Google Keep, Microsoft To Do, AnyList і Bring!. Аналіз показав, що універсальні сервіси є простими й доступними, але вони не завжди враховують особливості саме продуктових списків. Спеціалізовані застосунки мають більше потрібних функцій, але для дипломного прототипу їхня функціональність є надмірною. Тому було зроблено висновок, що доцільно створити власний простий frontend-застосунок із базовими можливостями.

На основі аналізу було визначено вимоги до програмного забезпечення. До основних функціональних вимог належать додавання товарів, редагування, видалення, позначення товарів як куплених, вибір категорії, фільтрація, підрахунок приблизної суми та локальне збереження списку. До нефункціональних вимог було віднесено простоту використання, адаптивність,

швидкість роботи, зрозумілу структуру інтерфейсу та можливість подальшого розвитку системи.

У другому розділі було обґрунтовано вибір технологій розробки. Для реалізації застосунку обрано React, TypeScript, Vite, Tailwind CSS, localStorage і Vitest. Такий набір технологій дозволяє створити компонентний веб-застосунок, описати структуру даних, швидко розробити інтерфейс, зберігати дані локально у браузері та перевіряти окремі функції за допомогою тестів.

Було спроектовано архітектуру програмного забезпечення. Застосунок побудовано за компонентним принципом. Основними частинами системи є головний компонент, форма додавання товару, список покупок, елемент окремого товару, фільтр категорій, підсумкова панель і службові модулі для збереження та обробки даних. Така структура є достатньо простою для дипломного проєкту, але водночас дає можливість розширювати систему в майбутньому.

Також у роботі було спроектовано структуру даних. Основним об'єктом системи є товар, який має унікальний ідентифікатор, назву, категорію, кількість, одиницю вимірювання, приблизну ціну, статус купівлі та дату створення. Список покупок представлено як масив таких об'єктів. Така модель дозволяє реалізувати всі основні функції застосунку: додавання, редагування, видалення, фільтрацію, підрахунок суми та локальне збереження даних.

У третьому розділі було описано практичну реалізацію програмного забезпечення. Було розроблено основні компоненти frontend-застосунку, службові модулі для роботи з localStorage і функції для підрахунків. Застосунок дозволяє користувачу створювати список покупок, додавати товари, редагувати їх, видаляти зайві позиції, позначати товари як куплені, фільтрувати список за категоріями та переглядати підсумкову інформацію.

Інтерфейс користувача було розроблено з орієнтацією на простоту й швидке виконання основних дій. Усі головні елементи розміщені на одній сторінці: заголовок, форма додавання товару, фільтр категорій, список покупок і підсумкова панель. Такий підхід є зручним для побутового використання,

оскільки користувач може швидко створити список і контролювати його виконання без переходу між різними сторінками.

Було проведено тестування програмного забезпечення. Перевірялися основні сценарії роботи: додавання товару з коректними даними, спроба додати некоректний товар, редагування, видалення, позначення товару як купленого, фільтрація за категоріями, підрахунок загальної суми, збереження даних після перезавантаження сторінки та адаптивність інтерфейсу. Також було передбачено модульне тестування окремих функцій підрахунку й фільтрації.

Результати тестування показали, що розроблене програмне забезпечення виконує основні функції коректно. Користувач може пройти повний сценарій роботи зі списком покупок: додати товари, змінити їх, відмітити куплені позиції, переглянути суму та зберегти список у браузері. Це підтверджує, що поставлену задачу дипломної роботи виконано.

Практичним результатом роботи є демонстраційний frontend-застосунок для планування покупок продуктів харчування сім'ї. Він не потребує backend, реєстрації користувачів або складного налаштування. Дані зберігаються локально у браузері, тому застосунок можна використовувати як простий прототип системи для ведення списку покупок.

Розроблене програмне забезпечення має певні обмеження. У поточній версії список доступний тільки на тому пристрої та в тому браузері, де його було створено. Також не реалізовано спільний доступ для кількох членів сім'ї, синхронізацію між пристроями, серверне збереження даних та інтеграцію з онлайн-магазинами. Ці обмеження пояснюються тим, що в межах роботи створювався саме демонстраційний frontend-прототип.

У майбутньому систему можна розширити. До можливих напрямів розвитку належать додавання backend, створення акаунтів користувачів, реалізація сімейних груп, синхронізація списків між пристроями, збереження історії покупок, створення шаблонів списків і аналітика витрат за категоріями. Такі функції могли б перетворити прототип на повноцінну інформаційну систему для сімейного планування покупок.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Policy Support and Governance Gateway. Policy themes Policy support. Fao. URL: <https://www.fao.org/policy-support/policy-themes/food-loss-and-food-waste/en> (date of access: 04.05.2026)
2. Food Waste Index Report 2024. Publication Resources. Unep. URL: <https://www.unep.org/resources/publication/food-waste-index-report-2024> (date of access: 04.05.2026)
3. Google Keep Help. Google. URL: <https://support.google.com/keep> (date of access: 04.05.2026)
4. Support Microsoft To Do. Microsoft. URL: <https://support.microsoft.com/todo> (date of access: 05.05.2026)
5. AnyList. AnyList. URL: <https://help.anylist.com> (date of access: 05.05.2026)
6. The easiest shopping list to share. Getbring. URL: <https://www.getbring.com/en/home> (date of access: 06.05.2026)
7. ISO/IEC/IEEE 29148:2018. Systems and software engineering — Life cycle processes — Requirements engineering. Edition 2, 2018. Iso. URL: <https://www.iso.org/standard/72089.html> (date of access: 06.05.2026)
8. ISO/IEC 25010:2023. Systems and software engineering — Systems and software Quality Requirements and Evaluation — Product quality model. Edition 2, 2023. Iso. URL: <https://www.iso.org/standard/78176.html> (date of access: 06.05.2026)
9. Quick Start. React. URL: <https://react.dev/learn> (date of access: 06.05.2026)
10. React Reference Overview. Reference. React. URL: <https://react.dev/reference/react> (date of access: 06.05.2026)
11. Overview HTTP. Guides Http Web Docs. Mozilla. URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview> (date of access: 12.05.2026)

12. TypeScript Documentation. Typescriptlang. URL: <https://www.typescriptlang.org/docs/> (date of access: 07.05.2026)
13. Getting Started. Vite. URL: <https://vite.dev/guide/> (date of access: 08.05.2026)
14. Get started with Tailwind CSS. Installation Docs. Tailwindcss. URL: <https://tailwindcss.com/docs/installation/using-vite> (date of access: 08.05.2026)
15. HTML: HyperText Markup Language. Web Docs. Mozilla. URL: <https://developer.mozilla.org/en-US/docs/Web/HTML> (date of access: 12.05.2026)
16. CSS: Cascading Style Sheets. Web Docs. Mozilla. URL: <https://developer.mozilla.org/en-US/docs/Web/CSS> (date of access: 12.05.2026)
17. Styling with utility classes. Docs. Tailwindcss. URL: <https://tailwindcss.com/docs/styling-with-utility-classes> (date of access: 08.05.2026)
18. Node.js v26.2.0 documentation. Latest Docs. Nodejs URL: <https://nodejs.org/docs/latest/api/> (date of access: 09.05.2026)
19. Introduction to Node.js. Getting started Learn. Nodejs. URL: <https://nodejs.org/learn/getting-started/introduction-to-nodejs> (date of access: 09.05.2026)
20. Fast, unopinionated, minimalist web framework for Node.js. Expressjs. URL: <https://expressjs.com/> (date of access: 10.05.2026)
21. JavaScript Guide. JavaScript Web Docs. Mozilla. URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide> (date of access: 12.05.2026)
22. Duckett J. JavaScript and JQuery: Interactive Front-End Web Development. Indianapolis: Wiley, 2014. 640 p.
23. Introduction to Prisma. Prisma. URL: <https://www.prisma.io/docs> (date of access: 10.05.2026)
24. Next-generation Node.js and TypeScript ORM. Prisma. URL: <https://www.prisma.io/orm> (date of access: 10.05.2026)
25. Intro to the TSConfig Reference. Typescriptlang. URL: <https://www.typescriptlang.org/tsconfig/> (date of access: 07.05.2026)

26. Documentation. Postgresql. URL: <https://www.postgresql.org/docs/> (date of access: 11.05.2026)
27. About PostgreSQL. Postgresql. URL: <https://www.postgresql.org/about/> (date of access: 11.05.2026)
28. Window: localStorage property. Window Api Web Docs. Mozilla URL: <https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage> (date of access: 08.05.2026)
29. Getting Started. Vitest. URL: <https://vitest.dev/guide/> (date of access: 08.05.2026)
30. Test. Api. Vitest. URL: <https://vitest.dev/api/test> (date of access: 08.05.2026)
31. Web Storage API. Api Web Docs. Mozilla. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API (date of access: 08.05.2026)
32. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures: Doctoral dissertation. University of California, Irvine, 2000. URL: <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
33. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston: Addison-Wesley, 1994. 395 p.
34. Sommerville I. Software Engineering. 10th ed. Boston: Pearson, 2016. 816 p.

ДОДАТОК А
Лістинг Коду

CategoryFilter.tsx

```
import { ALL_CATEGORIES_OPTION, PRODUCT_CATEGORIES } from '../utils/constants';

interface CategoryFilterProps {
  selectedCategory: string;
  onCategoryChange: (category: string) => void;
}

export function CategoryFilter({ selectedCategory, onCategoryChange }: CategoryFilterProps) {
  const categories = [ALL_CATEGORIES_OPTION, ...PRODUCT_CATEGORIES];

  return (
    <section className="rounded-lg border border-slate-200 bg-white p-4">
      <label htmlFor="category-filter" className="text-sm font-semibold text-academic-900">
        Фільтр за категорією
      </label>

      <select
        id="category-filter"
        value={selectedCategory}
        onChange={(event) => onCategoryChange(event.target.value)}
        className="mt-2 w-full rounded-md border border-slate-300 bg-white px-3 py-2 text-sm text-slate-800 outline-none transition focus:border-academic-700 focus:ring-2 focus:ring-academic-100"
      >
        {categories.map((category) => (
          <option key={category} value={category}>
            {category}
          </option>
        ))}
      </select>

      <div className="mt-3 hidden flex-wrap gap-2 lg:flex">
        {categories.map((category) => {
          const isActive = category === selectedCategory;

          return (
            <button
              key={category}
              type="button"
              onClick={() => onCategoryChange(category)}
              className={`rounded-md border px-3 py-2 text-sm transition ${
                isActive
                  ? 'border-academic-700 bg-academic-700 text-white'
                  : 'border-slate-300 bg-white text-slate-700 hover:border-academic-700'
              }`}
            >
              {category}
            </button>
          );
        })}
      </div>
    </section>
  );
}
```

ShoppingForm.tsx

```
import { FormEvent, useEffect, useState } from 'react';
import type { Product } from '../types/product';
import { PRODUCT_CATEGORIES, PRODUCT_UNITS } from '../utils/constants';

interface ShoppingFormProps {
  editingProduct: Product | null;
```

```

onAddProduct: (product: Product) => void;
onUpdateProduct: (product: Product) => void;
onCancelEdit: () => void;
}

interface FormErrors {
  name?: string;
  quantity?: string;
  price?: string;
}

const initialCategory = PRODUCT_CATEGORIES[0];
const initialUnit = PRODUCT_UNITS[0];

function createProductId(): string {
  return `${Date.now()}-${crypto.randomUUID()}`;
}

export function ShoppingForm({
  editingProduct,
  onAddProduct,
  onUpdateProduct,
  onCancelEdit,
}: ShoppingFormProps) {
  const [name, setName] = useState("");
  const [category, setCategory] = useState<string>(initialCategory);
  const [quantity, setQuantity] = useState('1');
  const [unit, setUnit] = useState<string>(initialUnit);
  const [price, setPrice] = useState('0');
  const [errors, setErrors] = useState<FormErrors>({});

  useEffect(() => {
    if (!editingProduct) {
      return;
    }

    setName(editingProduct.name);
    setCategory(editingProduct.category);
    setQuantity(editingProduct.quantity.toString());
    setUnit(editingProduct.unit);
    setPrice(editingProduct.price.toString());
    setErrors({});
  }, [editingProduct]);

  function resetForm(): void {
    setName("");
    setCategory(initialCategory);
    setQuantity('1');
    setUnit(initialUnit);
    setPrice('0');
    setErrors({});
  }

  function validateForm(): FormErrors {
    const nextErrors: FormErrors = {};
    const parsedQuantity = Number(quantity);
    const parsedPrice = Number(price);

    if (!name.trim()) {
      nextErrors.name = 'Введіть назву продукту.';
    }

    if (!Number.isFinite(parsedQuantity) || parsedQuantity <= 0) {

```

```

    nextErrors.quantity = 'Кількість має бути більшою за 0.';
  }

  if (!Number.isFinite(parsedPrice) || parsedPrice < 0) {
    nextErrors.price = 'Ціна не може бути від'ємною.';
  }

  return nextErrors;
}

function handleSubmit(event: FormEvent<HTMLFormElement>): void {
  event.preventDefault();

  const validationErrors = validateForm();
  setErrors(validationErrors);

  if (Object.keys(validationErrors).length > 0) {
    return;
  }

  const productData: Product = {
    id: editingProduct?.id ?? createProductId(),
    name: name.trim(),
    category,
    quantity: Number(quantity),
    unit,
    price: Number(price),
    isPurchased: editingProduct?.isPurchased ?? false,
    createdAt: editingProduct?.createdAt ?? new Date().toISOString(),
  };

  if (editingProduct) {
    onUpdateProduct(productData);
  } else {
    onAddProduct(productData);
  }

  resetForm();
}

function handleCancelEdit(): void {
  resetForm();
  onCancelEdit();
}

return (
  <section className="rounded-lg border border-slate-200 bg-white p-4 shadow-sm">
    <div className="flex flex-col gap-1">
      <h2 className="text-lg font-semibold text-academic-900">
        {editingProduct ? 'Редагування продукту' : 'Додавання продукту'}
      </h2>
      <p className="text-sm text-slate-600">
        Заповніть основні дані для планування кількості та приблизної вартості.
      </p>
    </div>

    <form onSubmit={handleSubmit} className="mt-5 grid gap-4 lg:grid-cols-12">
      <div className="lg:col-span-4">
        <label htmlFor="product-name" className="text-sm font-medium text-slate-700">
          Назва продукту
        </label>
        <input
          id="product-name"

```

```

    value={name}
    onChange={(event) => setName(event.target.value)}
    className="mt-1 w-full rounded-md border border-slate-300 px-3 py-2 text-sm outline-none transition
focus:border-academic-700 focus:ring-2 focus:ring-academic-100"
    placeholder="Наприклад, яблука"
  />
  {errors.name && <p className="mt-1 text-sm text-red-600">{errors.name}</p>}
</div>

<div className="lg:col-span-3">
  <label htmlFor="product-category" className="text-sm font-medium text-slate-700">
    Категорія
  </label>
  <select
    id="product-category"
    value={category}
    onChange={(event) => setCategory(event.target.value)}
    className="mt-1 w-full rounded-md border border-slate-300 bg-white px-3 py-2 text-sm outline-none transition
focus:border-academic-700 focus:ring-2 focus:ring-academic-100"
  >
    {PRODUCT_CATEGORIES.map((productCategory) => (
      <option key={productCategory} value={productCategory}>
        {productCategory}
      </option>
    ))}
  </select>
</div>

<div className="lg:col-span-2">
  <label htmlFor="product-quantity" className="text-sm font-medium text-slate-700">
    Кількість
  </label>
  <input
    id="product-quantity"
    type="number"
    min="0"
    step="0.01"
    value={quantity}
    onChange={(event) => setQuantity(event.target.value)}
    className="mt-1 w-full rounded-md border border-slate-300 px-3 py-2 text-sm outline-none transition
focus:border-academic-700 focus:ring-2 focus:ring-academic-100"
  />
  {errors.quantity && <p className="mt-1 text-sm text-red-600">{errors.quantity}</p>}
</div>

<div className="lg:col-span-1">
  <label htmlFor="product-unit" className="text-sm font-medium text-slate-700">
    Одиниця
  </label>
  <select
    id="product-unit"
    value={unit}
    onChange={(event) => setUnit(event.target.value)}
    className="mt-1 w-full rounded-md border border-slate-300 bg-white px-3 py-2 text-sm outline-none transition
focus:border-academic-700 focus:ring-2 focus:ring-academic-100"
  >
    {PRODUCT_UNITS.map((productUnit) => (
      <option key={productUnit} value={productUnit}>
        {productUnit}
      </option>
    ))}
  </select>
</div>

```

```

<div className="lg:col-span-2">
  <label htmlFor="product-price" className="text-sm font-medium text-slate-700">
    Ціна за од.
  </label>
  <input
    id="product-price"
    type="number"
    min="0"
    step="0.01"
    value={price}
    onChange={(event) => setPrice(event.target.value)}
    className="mt-1 w-full rounded-md border border-slate-300 px-3 py-2 text-sm outline-none transition
focus:border-academic-700 focus:ring-2 focus:ring-academic-100"
  />
  {errors.price && <p className="mt-1 text-sm text-red-600">{errors.price}</p>}
</div>

<div className="flex flex-col gap-2 sm:flex-row lg:col-span-12">
  <button
    type="submit"
    className="rounded-md bg-academic-700 px-4 py-2 text-sm font-semibold text-white transition hover:bg-
academic-900"
  >
    {editingProduct ? 'Зберегти зміни' : 'Додати продукт'}
  </button>
  {editingProduct && (
    <button
      type="button"
      onClick={handleCancelEdit}
      className="rounded-md border border-slate-300 bg-white px-4 py-2 text-sm font-semibold text-slate-700
transition hover:border-academic-700"
    >
      Скасувати
    </button>
  )}
</div>
</form>
</section>
);
}

```

ShoppingItem.tsx

```
import type { Product } from '../types/product';
```

```
interface ShoppingItemProps {
  product: Product;
  view: 'table' | 'card';
  onTogglePurchased: (productId: string) => void;
  onEditProduct: (product: Product) => void;
  onDeleteProduct: (productId: string) => void;
}
```

```
function formatCurrency(value: number): string {
  return new Intl.NumberFormat('uk-UA', {
    style: 'currency',
    currency: 'UAH',
    maximumFractionDigits: 2,
  }).format(value);
}
```

```

export function ShoppingItem({
  product,
  view,
  onTogglePurchased,
  onEditProduct,
  onDeleteProduct,
}: ShoppingItemProps) {
  const itemTotal = product.quantity * product.price;
  const purchasedClass = product.isPurchased ? 'text-slate-400 line-through' : 'text-academic-900';

  if (view === 'table') {
    return (
      <tr className={product.isPurchased ? 'bg-slate-50' : 'bg-white'}>
        <td className="px-4 py-3">
          <div className="flex items-center gap-3">
            <input
              type="checkbox"
              checked={product.isPurchased}
              onChange={() => onTogglePurchased(product.id)}
              className="h-4 w-4 rounded border-slate-300 accent-academic-700"
              aria-label={`Позначити ${product.name} як куплене`}
            />
            <span className={`font-medium ${purchasedClass}`}>{product.name}</span>
          </div>
        </td>
        <td className="px-4 py-3 text-slate-600">{product.category}</td>
        <td className="px-4 py-3 text-slate-600">
          {product.quantity} {product.unit}
        </td>
        <td className="px-4 py-3 text-slate-600">{formatCurrency(product.price)}</td>
        <td className="px-4 py-3 font-semibold text-academic-900">{formatCurrency(itemTotal)}</td>
        <td className="px-4 py-3">
          <div className="flex justify-end gap-2">
            <button
              type="button"
              onClick={() => onEditProduct(product)}
              className="rounded-md border border-slate-300 px-3 py-1.5 text-sm font-medium text-slate-700 transition
hover:border-academic-700"
            >
              Редагувати
            </button>
            <button
              type="button"
              onClick={() => onDeleteProduct(product.id)}
              className="rounded-md border border-red-200 px-3 py-1.5 text-sm font-medium text-red-700 transition
hover:bg-red-50"
            >
              Видалити
            </button>
          </div>
        </td>
      </tr>
    );
  }

  return (
    <article
      className={`rounded-lg border border-slate-200 bg-white p-4 shadow-sm ${
        product.isPurchased ? 'opacity-70' : ''
      }`}
    >
      <div className="flex items-start justify-between gap-3">
        <div>

```

```

    <h3 className={`text-base font-semibold ${product.isPurchased}>{product.name}</h3>
    <p className="mt-1 text-sm text-slate-600">{product.category}</p>
  </div>
  <input
    type="checkbox"
    checked={product.isPurchased}
    onChange={() => onTogglePurchased(product.id)}
    className="mt-1 h-5 w-5 rounded border-slate-300 accent-academic-700"
    aria-label={`Позначити ${product.name} як куплене`}
  />
</div>

<div className="mt-4 grid grid-cols-2 gap-3 text-sm">
  <div>
    <p className="text-slate-500">Кількість</p>
    <p className="font-medium text-academic-900">
      {product.quantity} {product.unit}
    </p>
  </div>
  <div>
    <p className="text-slate-500">Ціна за од.</p>
    <p className="font-medium text-academic-900">{formatCurrency(product.price)}</p>
  </div>
  <div className="col-span-2">
    <p className="text-slate-500">Сума</p>
    <p className="text-lg font-semibold text-academic-900">{formatCurrency(itemTotal)}</p>
  </div>
</div>

<div className="mt-4 flex gap-2">
  <button
    type="button"
    onClick={() => onEditProduct(product)}
    className="flex-1 rounded-md border border-slate-300 px-3 py-2 text-sm font-medium text-slate-700"
  >
    Редагувати
  </button>
  <button
    type="button"
    onClick={() => onDeleteProduct(product.id)}
    className="flex-1 rounded-md border border-red-200 px-3 py-2 text-sm font-medium text-red-700"
  >
    Видалити
  </button>
</div>
</article>
);
}
SummaryPanel.tsx

import type { Product } from '../types/product';
import {
  calculatePurchasedTotal,
  calculateTotal,
  countNotPurchased,
  countPurchased,
} from '../utils/calculations';

interface SummaryPanelProps {
  products: Product[];
}

function formatCurrency(value: number): string {

```

```

return new Intl.NumberFormat('uk-UA', {
  style: 'currency',
  currency: 'UAH',
  maximumFractionDigits: 2,
}).format(value);
}

export function SummaryPanel({ products }: SummaryPanelProps) {
  const totalItems = products.length;
  const purchasedItems = countPurchased(products);
  const notPurchasedItems = countNotPurchased(products);
  const totalPrice = calculateTotal(products);
  const purchasedTotal = calculatePurchasedTotal(products);

  const summaryItems = [
    { label: 'Усього позицій', value: totalItems.toString() },
    { label: 'Куплено', value: purchasedItems.toString() },
    { label: 'Не куплено', value: notPurchasedItems.toString() },
    { label: 'Загальна сума', value: formatCurrency(totalPrice) },
    { label: 'Сума купленого', value: formatCurrency(purchasedTotal) },
  ];

  return (
    <section className="rounded-lg border border-academic-700 bg-white p-4 shadow-sm">
      <h2 className="text-base font-semibold text-academic-900">Підсумок покупок</h2>
      <div className="mt-4 grid gap-3 sm:grid-cols-2 lg:grid-cols-5">
        {summaryItems.map((item) => (
          <div key={item.label} className="rounded-md bg-academic-50 p-3">
            <p className="text-xs font-medium uppercase tracking-wide text-slate-500">
              {item.label}
            </p>
            <p className="mt-1 text-lg font-semibold text-academic-900">{item.value}</p>
          </div>
        ))}
      </div>
    </section>
  );
}

```

App.tsx

```

import { useEffect, useMemo, useState } from 'react';
import { CategoryFilter } from './components/CategoryFilter';
import { Header } from './components/Header';
import { ShoppingForm } from './components/ShoppingForm';
import { ShoppingList } from './components/ShoppingList';
import { SummaryPanel } from './components/SummaryPanel';
import { clearProducts, loadProducts, saveProducts } from './services/storageService';
import type { Product } from './types/product';
import { filterByCategory } from './utils/calculations';
import { ALL_CATEGORIES_OPTION } from './utils/constants';

function App() {
  const [products, setProducts] = useState<Product[]>(() => loadProducts());
  const [selectedCategory, setSelectedCategory] = useState(ALL_CATEGORIES_OPTION);
  const [editingProduct, setEditingProduct] = useState<Product | null>(null);

  useEffect(() => {
    saveProducts(products);
  }, [products]);

  const filteredProducts = useMemo(
    () => filterByCategory(products, selectedCategory),

```

```

    [products, selectedCategory],
  );

  function handleAddProduct(product: Product): void {
    setProducts((currentProducts) => [product, ...currentProducts]);
  }

  function handleUpdateProduct(updatedProduct: Product): void {
    setProducts((currentProducts) =>
      currentProducts.map((product) =>
        product.id === updatedProduct.id ? updatedProduct : product,
      ),
    );
    setEditingProduct(null);
  }

  function handleTogglePurchased(productId: string): void {
    setProducts((currentProducts) =>
      currentProducts.map((product) =>
        product.id === productId
          ? { ...product, isPurchased: !product.isPurchased }
          : product,
      ),
    );
  }

  function handleDeleteProduct(productId: string): void {
    setProducts((currentProducts) => currentProducts.filter((product) => product.id !== productId));

    if (editingProduct?.id === productId) {
      setEditingProduct(null);
    }
  }

  function handleClearProducts(): void {
    const shouldClear = window.confirm('Очистити весь список покупок?');

    if (!shouldClear) {
      return;
    }

    setProducts([]);
    setEditingProduct(null);
    clearProducts();
  }

  return (
    <div className="min-h-screen bg-academic-50">
      <Header />

      <main className="mx-auto grid max-w-6xl gap-5 px-4 py-6 sm:px-6 lg:grid-cols-[minmax(0,1fr)_320px]">
        <div className="grid gap-5">
          <ShoppingForm
            editingProduct={editingProduct}
            onAddProduct={handleAddProduct}
            onUpdateProduct={handleUpdateProduct}
            onCancelEdit={() => setEditingProduct(null)}
          />

          <div className="flex flex-col gap-3 sm:flex-row sm:items-center sm:justify-between">
            <div>
              <h2 className="text-lg font-semibold text-academic-900">Список продуктів</h2>
              <p className="text-sm text-slate-600">

```

```

        Показано позицій: {filteredProducts.length} із {products.length}
      </p>
    </div>
    <button
      type="button"
      onClick={handleClearProducts}
      disabled={products.length === 0}
      className="rounded-md border border-red-200 bg-white px-4 py-2 text-sm font-semibold text-red-700
transition hover:bg-red-50 disabled:cursor-not-allowed disabled:opacity-50"
    >
      Очистити список
    </button>
  </div>

  <ShoppingList
    products={filteredProducts}
    onTogglePurchased={handleTogglePurchased}
    onEditProduct={setEditingProduct}
    onDeleteProduct={handleDeleteProduct}
  />
</div>

<aside className="grid content-start gap-5">
  <SummaryPanel products={products} />
  <CategoryFilter
    selectedCategory={selectedCategory}
    onCategoryChange={setSelectedCategory}
  />
</aside>
</main>
</div>
);
}

export default App;

main.tsx
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import './index.css';

ReactDOM.createRoot(document.getElementById('root') as HTMLElement).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
);

```

ДОДАТОК Б
Слайди презентації

Національний університет
«Запорізька політехніка»
Кафедра ПРОГРАМНИХ ЗАСОБІВ

Розроблення програмного забезпечення для планування покупок продуктів для харчування сім'ї

Виконав: студент групи КНТ-132
Микита КОЛЯДА

Керівник: д.т.н., професор
Михайло ПОЛЯКОВ

Рисунок Б.1 – Слайд 1

Об'єкт, предмет, мета дослідження

Об'єкт дослідження - процес планування покупок продуктів харчування сім'ї.

Предмет дослідження - програмне забезпечення для автоматизації планування, ведення та контролю списків сімейних покупок продуктів харчування.

Мета роботи - розробка програмного забезпечення для планування покупок продуктів харчування сім'ї, яке забезпечує зручне створення, редагування, перегляд і контроль списків покупок.

Рисунок Б.2 – Слайд 2

Порівняння існуючих програмних рішень

Програмне рішення	Основне призначення	Переваги	Недоліки для планування покупок
Звичайні нотатки смартфона	Швидкий запис інформації	Простота, доступність, не потрібно встановлювати додаткові програми	Немає категорій, підрахунку суми та спеціальної структури товару
Google Keep	Нотатки та прості списки	Швидке створення списків, позначення виконаних пунктів, спільний доступ	Не орієнтований саме на продуктові покупки
Microsoft To Do	Керування завданнями	Зручне позначення виконаних пунктів, окремі списки	Більше підходить для задач, ніж для списків продуктів
AnyList	Списки покупок	Категорії, повторювані товари, спільне використання	Частина функцій є зайвою для простого дипломного прототипу
Bring!	Планування продуктових покупок	Зручний візуальний інтерфейс, категорії, спільні списки	Для повного аналога потрібна складніша реалізація

Рисунок Б.3 – Слайд 3

Порівняння обраних та альтернативних технологій

Частина системи	Обрана технологія	Альтернативні технології	Обґрунтування вибору
Frontend бібліотека	React	Vue, Angular, Svelte	React має компонентний підхід, велику спільноту, багато документації та добре підходить для створення інтерактивного інтерфейсу списків покупок
Мова розробки	TypeScript	JavaScript	TypeScript забезпечує типізацію даних, що зменшує кількість помилок під час роботи з товарами, списками, користувачами та API
Інструмент збірки frontend	Vite	Webpack, Parcel, Create React App	Vite швидко запускається, має просте налаштування і добре підходить для сучасних React-проектів
CSS фреймворк	Tailwind CSS	Bootstrap, Material UI, звичайний CSS	Tailwind CSS дозволяє швидко створити адаптивний інтерфейс без великої кількості окремих CSS-файлів
Backend середовище	Node.js	PHP, Python, Java, .NET	Node.js дозволяє використовувати TypeScript і на frontend, і на backend, що робить проект більш єдиним за технологічним підходом
Backend фреймворк	Express	NestJS, Fastify, Koa	Express має просту структуру, достатній функціонал для REST API і не перевантажує дипломний проект зайвою складністю
База даних	PostgreSQL	MySQL, SQLite, MongoDB	PostgreSQL добре підходить для структурованих зв'язків між користувачами, сім'ями, списками покупок і товарами

Рисунок Б.4 – Слайд 4

Діаграма варіантів використання системи



Рисунок Б.5 – Слайд 5

Загальна схема роботи програмного забезпечення

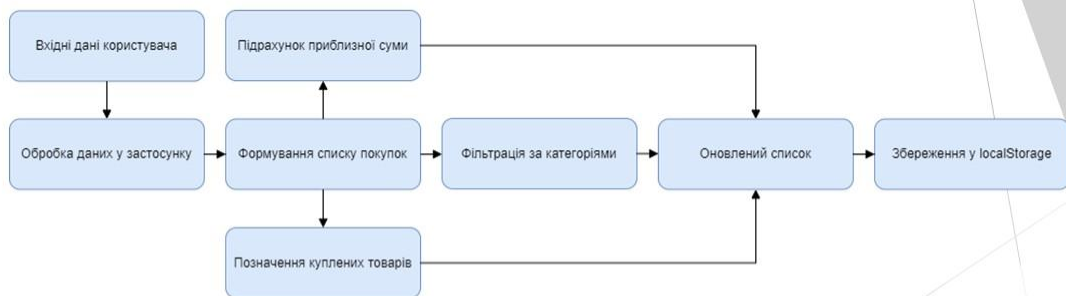


Рисунок Б.6 – Слайд 6

Основні модулі програмного забезпечення

Модуль або компонент	Призначення
App.tsx	Керує станом застосунку та основними діями зі списком
ShoppingForm.tsx	Забезпечує додавання нового товару
ShoppingList.tsx	Відображає список покупок
ShoppingItem.tsx	Показує один товар і кнопки дій
CategoryFilter.tsx	Фільтрує товари за категоріями
SummaryPanel.tsx	Відображає підсумкову інформацію
storageService.ts	Зберігає та завантажує список із localStorage
calculations.ts	Містить функції підрахунку і фільтрації
product.ts	Описує структуру товару

Рисунок Б.7 – Слайд 7

Схема взаємодії програмних модулів

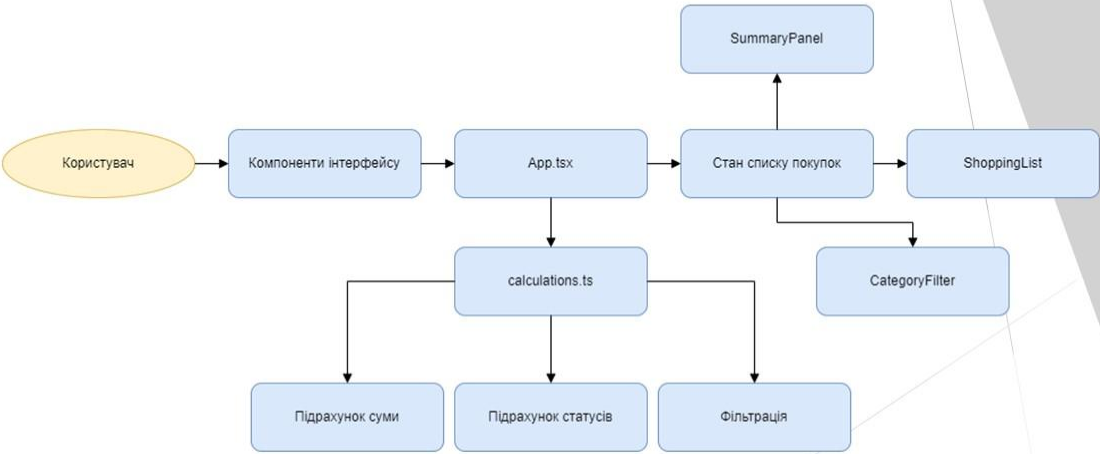


Рисунок Б.8 – Слайд 8

Результати розробки

- Проаналізовано предметну область планування покупок
- Розроблено frontend-застосунок для ведення списків покупок
- Реалізовано:
 - додавання та редагування товарів;
 - фільтрацію за категоріями;
 - підрахунок суми покупок;
 - локальне збереження даних
- Створено простий та адаптивний інтерфейс
- Проведено тестування основних функцій системи

Рисунок Б.9 – Слайд 9

Висновки

- Розроблено програмне забезпечення для планування сімейних покупок
- Досягнуто поставленої мети дипломної роботи
- Застосунок забезпечує зручне керування списками покупок
- Використані технології дозволили створити сучасний frontend-застосунок
- Систему можна розширити backend-частиною та синхронізацією між пристроями

Рисунок Б.10 – Слайд 10