ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка web-застосунку для ведення спільного списку покупок»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Артем РЯБОВ
(підпис)

Виконав: здобувач вищої освіти групи ~~ІД-44~~
Артем РЯБОВ

Керівник: Оксана ЗОЛОТУХІНА
к.т.н, доцент

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Рябову Артему Сергійовичу

1. Тема кваліфікаційної роботи: «Розробка web-застосунку для ведення спільного списку покупок»
керівник кваліфікаційної роботи к.т.н, доцент Оксана ЗОЛОТУХІНА,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи розробки веб-застосунків, опис роботи веб-додатків, документація фреймворку ASP.NET.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Огляд веб-застосунку для ведення спільного списку покупок та аналіз існуючих аналогів, визначення вимог до програмного забезпечення.
2. Проектування веб-застосунку для ведення спільного списку покупок.
3. Програмна реалізація та опис функцій застосунку для ведення спільного

списку покупок.

4. Тестування застосунку.

5. Перелік графічного матеріалу: *презентація*
   1. Мета, об’єкт та предмет дослідження.
   2. Задачі дипломної роботи.
   3. Аналіз аналогів.
   4. Вимоги до програмного забезпечення.
   5. Програмні засоби реалізації.
   6. Діаграма розгортання.
   7. Діаграма класів бази даних.
   8. Діаграма варіантів використання.
   9. Екранні форми.
   10. Апробація результатів дослідження.
   11. Висновки.

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Огляд веб-застосунку для ведення спільного списку покупок та аналіз аналогів	13.03-21.03.2025	
4	Проектування веб-застосунку для ведення спільного списку покупок	22.03-01.04.2025	
5	Програмна реалізація та опис функцій застосунку для ведення спільного списку покупок	01.04-24.04.2025	
6	Тестування застосунку	25.04-27.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Артем РЯБОВ
(підпис)

Керівник
кваліфікаційної роботи _____ Оксана ЗОЛОТУХІНА
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 45 стор., 2 табл., 18 рис., 11 джерел.

Мета роботи – спрощення процесу ведення спільного списку покупок шляхом використання сучасних веб-технологій.

Об'єкт дослідження – процес організації та ведення спільного списку покупок.

Предмет дослідження – web-застосунок, де застосовані технології веб-розробки, інтерфейси взаємодії користувачів та засоби синхронізації даних у реальному часі.

Короткий зміст роботи: В роботі був проведений аналіз сучасних варіантів організації спільної роботи користувачів у веб-застосунках. Розроблено веб-застосунок для ведення спільного списку покупок із підтримкою реєстрації, автентифікації та синхронізації даних у реальному часі. В якості середовища реалізації обрано ASP.NET Core, Razor Pages та SignalR. Реалізовано основні функціональні можливості: створення списків покупок, додавання та редагування елементів у реальному часі з використанням технології SignalR, управління правами доступу, спільний доступ до списків між користувачами, а також використання стандартних сторінок автентифікації та керування обліковим записом (Login, Register, Manage). Приділено увагу інтерфейсу для інших мобільних пристроях.

Сфера використання застосунку — спільне планування покупок у родині, гуртожитку або між друзями.

КЛЮЧОВІ СЛОВА: список покупок, ASP.NET Core, SignalR, веб-застосунок, спільний доступ, синхронізація, користувач, база даних, Entity Framework, MVC.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – Програмне забезпечення

UI - User Interface

MVC - Model-View-Controller

API – Application Programming Interface

EF Core - Entity Framework Core

HTTP - Hypertext Transfer Protocol

CSS – Cascading Style Sheets

SQL – Structured Query Language

ВСТУП

Актуальність: у сучасному світі технології відіграють ключову роль у повсякденному житті людини. Однією з побутових проблем, з якою зустрічаються користувачі є організація та ведення списку покупок – особливо в родинах або компаніях друзів. Традиційне використання паперових записничків або повідомлень в мессенджері є неефективними варіаціями і часто призводять до плутанини, забутих пунктів та дублювань. Використання спеціалізованного веб-застосунку дозволить підвищити зручність та ефективність побуту користувачів, особливо коли сервіс дозволить використовувати список водночас декільком людям – перший додає позицію, другий відмічає виконання. Завдяки технологіям веброзробки, таким як ASP.NET Core, можна створити безпечну і зручну систему, яка відповідатиме сучасним вимогам користувачів.

Об'єктом дослідження є процес організації та ведення спільного списку покупок.

Предметом дослідження є веб-застосунок, де застосовані технології веб-розробки, інтерфейси взаємодії користувачів та засоби синхронізації даних у реальному часі.

Метою роботи є спрощення процесу ведення спільного списку покупок шляхом використання сучасних веб-технологій

Методи дослідження: для початку був проаналізован ринок рішень у сфері керування списками покупок для формулювання функціональних та нефункціональних вимог до ПЗ. Надалі був проведений огляд стеку технологій реалізації додатку, що будуть відповідати вимогам до ПЗ, і на його основі було обрано фреймворк ASP.NET Core для серверної частини, SignalR для реалізації спільного використання списків користувачами, Entity Framework Core для взаємодії з базою даних через C# код, ASP.NET Core Identity для можливості реєстрації та логіну до сервісу, PostgreSQL як об'єктно-реляційну систему бази даних, JavaScript як клієнтська частина веб-застосунку.

Дослідження реалізації застосунку проводилося під час безпосередньої

розробки.

Наукова новизна роботи визначається наступними функціональними складовими: миттєва передача даних завдяки SignalR, надійна автентифікація завдяки інтеграції ASP.NET Core Identity.

Практична значущість результатів полягає в можливості використання веб-застосунку для полегшення ведення списку покупок в побуті.

1 АНАЛІЗ ЗАСТОСУНКУ ДЛЯ ВЕДЕННЯ СПІЛЬНОГО СПИСКУ ПОКУПОК

1.1 Вебсайт для ведення спільного списку покупок

Вебсайт для ведення спільного списку покупок – це спеціалізований онлайн сервіс, що надає доступ до використання списків покупок задля забезпечення виконання кожного пункту. Схожі вебсайти допомагають користувачам вести власні списки завдяки зручному дизайну та можливості створювати пункти з коментарями. Основною метою вебсайту для ведення спільного списку покупок є забезпечення зручного та ефективного доступу до списків покупок та надання спільного доступу до них.

Основні компоненти вебсайту для ведення спільного списку покупок:

- Авторизація користувачів: підтримання безпеки та виключення витоку приватної інформації.
- Списки: можливість створення та редагування пунктів.
- Спільний доступ: надання певного списку та функції відмітки виконання іншому авторизованому користувачу.
- Основні цілі вебсайту для ведення спільного списку покупок:
- Надання користувачам зручного інструменту для створення та ведення списків покупок;
- Забезпечення можливості спільного доступу до списків між кількома користувачами;
- Синхронізація даних у реальному часі;
- Забезпечення безпечного доступу до системи;
- Можливість доступу до системи з будь-якого пристрою;

Переваги:

- Спільна робота в реальному часі: декілька користувачів можуть одночасно редагувати один список з миттєвим оновленням.
- Кросплатформеність: доступ з будь-якого пристрою через браузер (ПК, планшет, смартфон).

- Простий інтерфейс: зручність використання навіть для недосвідчених користувачів.
- Безпека: захищена система автентифікації та авторизації.
- Гнучкість: можна легко додавати нові функції (фільтрація, нагадування, категорії тощо).
- Масштабованість: можливість подальшого розвитку до мобільного додатку або розширення функціоналу.

Недоліки:

- Залежність від інтернет-з’єднання: для роботи застосунку потрібен постійний доступ до мережі.
- Навантаження на сервер: при великій кількості одночасних користувачів може виникнути потреба в масштабуванні серверної інфраструктури.
- Обмеження браузера: деякі функції можуть працювати не однаково у різних браузерах.
- Складність реалізації синхронізації: підтримка реального часу через SignalR потребує додаткових технічних ресурсів і тестування.

1.2 Цільова аудиторія

Цільовою аудиторією web-застосунку для ведення спільного списку покупок є широке коло користувачів, яким необхідна зручна та ефективна система для організації процесу планування покупок. Основні категорії користувачів включають:

- Сім’ї та домогосподарства:

Члени однієї родини можуть колективно формувати список необхідних товарів, відслідковувати, що вже куплено, та хто за що відповідальний. Це спрощує комунікацію та зменшує кількість непотрібних покупок.

- Студенти, які проживають у гуртожитках чи знімають житло спільно:

Для спільного проживання важливо мати узгоджений список покупок побутових товарів або продуктів. Застосунок дозволяє уникнути дублювання покупок та забезпечити прозорість витрат.

- Колективи працівників (наприклад, в офісах):

У спільних просторах часто виникає потреба в організації закупівель (чай, кава, канцтовари тощо). Доступ до спільного списку дозволяє колегам швидко оновлювати інформацію.

- Користувачі, які планують покупки самостійно:

Навіть при індивідуальному використанні застосунок дозволяє зручно структурувати покупки, групувати їх, створювати шаблони, зберігати історію.

- Люди похилого віку та користувачі з низьким рівнем цифрової грамотності:

Завдяки простому та зрозумілому інтерфейсу, застосунок підходить для широкого кола користувачів, незалежно від технічного рівня. Основні дії (додавання, редагування, поділ) інтуїтивно зрозумілі.

1.3 Дослідження існуючих аналогів

На сьогоднішній день одними з найпопулярніших вебсайтів для ведення списків покупок є Google Keep, Microsoft To Do, OurGroceries та AnyList. До них детальніше.

1.3.1 Google Keep

Google Keep — це безкоштовний вебсервіс і мобільний застосунок від Google для створення, зберігання та синхронізації нотаток, списків справ, зображень і голосових нагадувань. Сервіс інтегровано в екосистему Google Workspace, що дозволяє користувачам легко зберігати нотатки, ділитися ними, і синхронізувати між пристроями.

Основні функції Google Keep включають:

- Нотатки: створення текстових нотаток з миттєвим збереженням;
- Списки справ: можливість створення чек-листів (to-do lists) із

можливістю відмічати виконані;

- Мітки і кольори: категоризація нотаток за допомогою міток і кольорів;
- Нагадування: додавання нагадувань за часом або місцем (геолокація);
- Спільна робота: спільне редагування нотаток з іншими користувачами Google;
- Голосовий ввід: диктування нотаток, що автоматично перетворюється на текст;
- Інтеграція з Google: можливість вставки в Google Docs, доступ через Google Assistant;
- Мобільність: кросплатформеність: Android, iOS, Web;
- Пошук: потужний пошук за ключовими словами, мітками, типом вмісту.

Переваги:

- Інтуїтивний та мінімалістичний інтерфейс — простота у використанні навіть для новачків.
- Повна синхронізація між пристроями — зміни зберігаються в реальному часі.
- Інтеграція з іншими сервісами Google — Google Docs, Calendar, Assistant.
- Швидке створення нотаток — мінімум кліків для фіксації ідей.
- Спільна робота — зручна взаємодія в реальному часі з іншими користувачами.
- Підтримка різних типів вмісту — текст, списки, зображення, аудіо.

Недоліки:

- Відсутність ієрархічної структури — немає підтримки вкладених списків, папок або піднотаток.
- Обмежена організація — тільки прості мітки, без категорій чи фільтрів за датою створення.
- Немає повноцінної системи управління завданнями — сервіс не замінює професійні task-менеджери.

- Обмежені функції форматування — відсутність жирного, курсиву, заголовків тощо.

- Неможливість офлайн-роботи в десктопному браузері без попередньої підготовки.

- Приватність та безпека — відсутність шифрування нотаток, доступ лише через Google-акаунт.

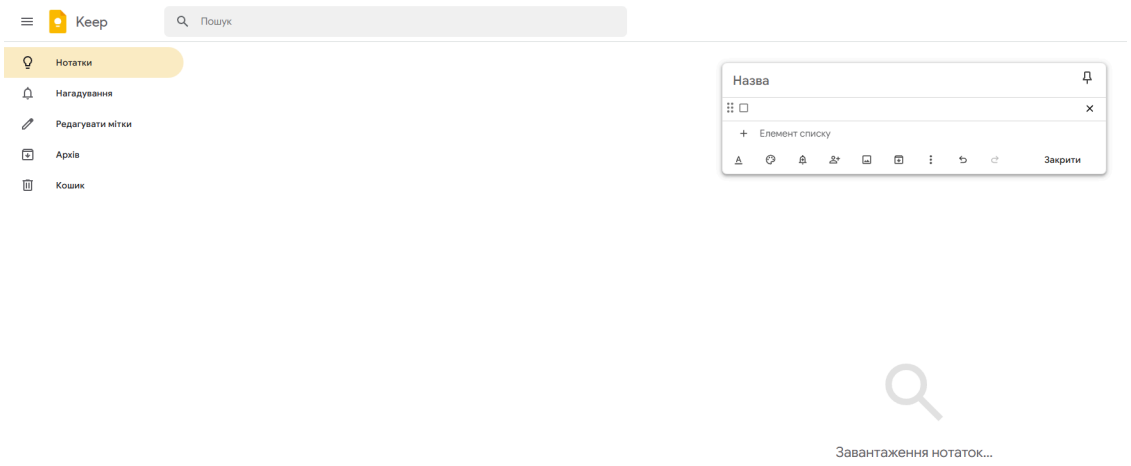


Рис.1.1 Приклад створення списку в Google Keep

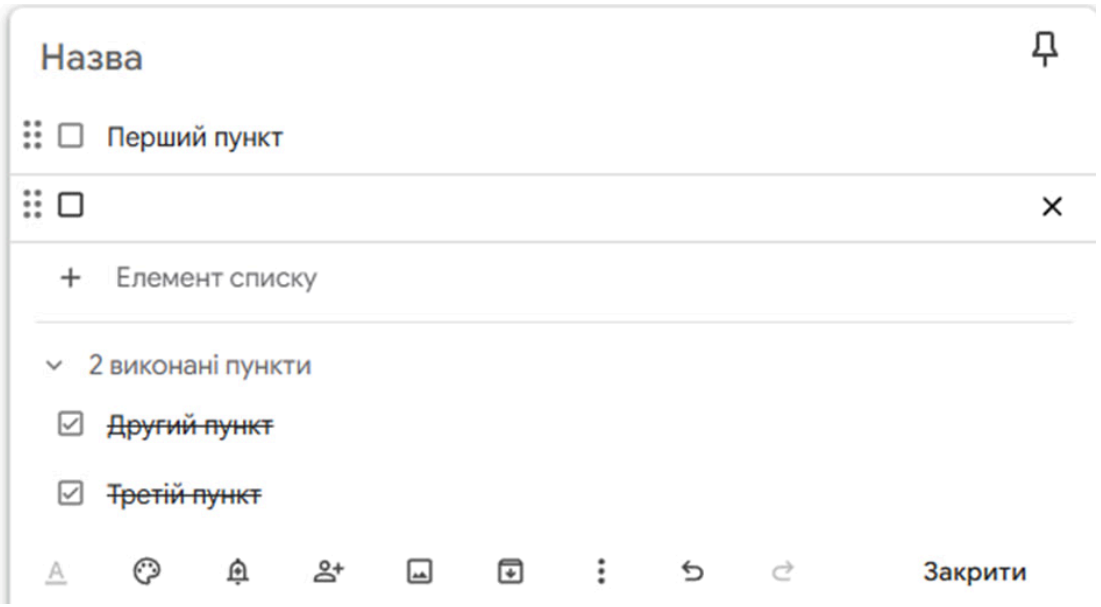


Рис.1.2 Приклад вигляду списку с виконаними частинами в Google Keep

1.3.2 Microsoft To Do

Microsoft To Do — це додаток для управління завданнями, який дозволяє користувачам створювати, організовувати та відстежувати списки справ, включаючи спільні списки, такі як списки покупок. Основними функціями Microsoft To Do є:

– Створення та організація списків: користувачі можуть створювати окремі списки та взаємодіяти з ними методами редагування, сортування або категоризації.

– Спільна робота: дозволяє ділитися списками з іншими користувачами, які мають обліковий запис Microsoft, Усі зміни (додавання, видалення чи позначення елементів як виконаних) синхронізуються миттєво між усіма учасниками списку

– Інтеграція з екосистемою Microsoft: To Do інтегрується з Outlook Tasks, що дозволяє автоматично імпортувати завдання з електронної пошти, Користувачі можуть налаштувати нагадування для списків покупок за часом або геолокацією.

– Додаткові функції: примітки та вкладення, мітки та теги, функція "Мій день", голосовий режим.

– Кастомізація: користувачі можуть змінювати фонові зображення та кольорові теми для кожного списку, інтеграція з Microsoft Power Automate дозволяє створювати автоматизовані робочі процеси, наприклад, автоматично додавати елементи до списку покупок на основі певних тригерів.

Переваги:

– Зручність і простота використання: додаток має мінімалістичний і зрозумілий дизайн, який підходить для користувачів будь-якого рівня технічної підготовки, та дозволяє використання всіма основними платформами(Windows, macOS, iOS, Android, Web).

– Ефективна синхронізація: синхронізація змін між учасниками списку відбувається миттєво, що запобігає дублюванню покупок або плутанин, дані зберігаються в хмарі Microsoft, що забезпечує надійність і доступність.

– Інтеграція з Microsoft 365: інтеграція з Outlook, Teams і Power Automate

робить To Do універсальним інструментом для тих, хто вже використовує продукти Microsoft.

– Гнучкість для спільної роботи: можливість ділитися списками з іншими користувачами робить To Do ідеальним для сімей, пар чи груп, які разом планують покупки, функція нагадувань за місцем розташування дозволяє отримувати сповіщення, коли користувач перебуває поблизу магазину.

– Безпека та конфіденційність: Microsoft To Do використовує хмарну інфраструктуру Azure із шифруванням даних, що забезпечує безпеку списків покупок.

Недоліки:

– Обмежені функції для спільної роботи: у Microsoft To Do немає можливості призначати різні ролі учасникам, у To Do немає вбудованого чату чи функції коментування.

– Обмеження функціоналу: не пропонує вбудованих функцій для створення підкатегорій у списку, інтеграція з іншими платформами (наприклад, Google Calendar або сторонніми додатками для покупок) є обмеженою, повноцінна робота в офлайн-режимі обмежена.

– Залежність від облікового запису Microsoft: потрібен обліковий запис Microsoft, що може бути незручним для користувачів, спільний доступ до списків можливий лише з іншими користувачами Microsoft.

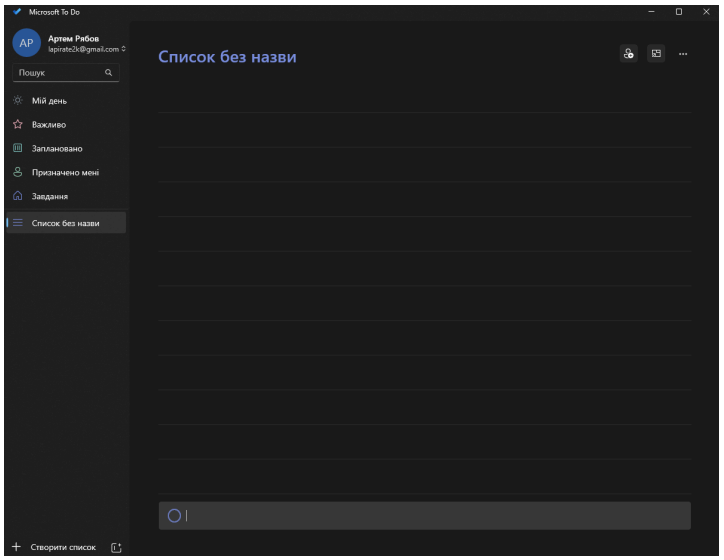


Рис.1.3. Приклад створення списку в Microsoft To Do

1.3.3 AnyList

AnyList — це спеціалізований додаток для створення списків покупок і планування їжі, який підтримує спільну роботу та має розширені функції для організації покупок. Він доступний на iOS, Android, macOS, Windows і у веб-версії.

Основними функціями додатку є:

– Створення та організація списків: користувачі можуть створювати списки покупок із автоматичною категоризацією (наприклад, "Молочні продукти", "Овочі"), підтримка штрих-кодів, дозволяє імпортувати списки покупок із рецептів на вебсайтах.

– Спільна робота: списками можна ділитися з іншими користувачами через запрошення за email або посилання, Зміни (додавання, видалення, позначення елементів) синхронізуються миттєво між усіма учасниками, AnyList надсилає сповіщення учасникам про зміни в списку.

– Планування їжі: користувачі можуть додавати рецепти до додатку, дозволяє планувати меню на тиждень і автоматично генерувати списки покупок.

– Додаткові функції: Офлайн-режим, до елементів можна додавати примітки, підтримка голосових команд, користувачі можуть створювати власні категорії та змінювати порядок їх відображення.

– Преміум-функції: додаткові теми та оформлення, розширений імпорт рецептів, інтеграція з календарями, додавання цін до позицій.

Переваги:

- Спеціалізація на списках покупок: AnyList створений саме для покупок і планування їжі, що робить його більш орієнтованим на цей сценарій порівняно з універсальними додатками, такими як Microsoft To Do.
- Автоматична категоризація: Автоматичне групування елементів за категоріями (наприклад, "М’ясо", "Бакалія") значно полегшує організацію великих списків.
- Інтеграція з рецептами: Можливість імпортувати інгредієнти з рецептів і планувати страви робить AnyList ідеальним для сімей, які планують харчування.

- Офлайн-доступ: Повноцінний офлайн-режим дозволяє працювати зі списками в магазинах без Wi-Fi чи мобільного інтернету.
- Зручний інтерфейс: Інтуїтивний і чистий дизайн, який підходить для користувачів будь-якого рівня.
- Кросплатформність: Доступність на всіх основних платформах, включаючи веб-версію, забезпечує гнучкість.

Недоліки:

- Платні функції: Деякі корисні можливості, такі як імпорт рецептів із будь-яких сайтів або відстеження цін, доступні лише в преміум-версії (вартість підписки — близько $9.99/рік для однієї особи або $14.99/рік для сім’ї).
- Обмеження для неанглійськомовних користувачів: Хоча додаток підтримує українську мову, автоматична категоризація та імпорт рецептів краще працюють для англійськомовного контенту.
- Залежність від облікового запису: Для спільного доступу потрібен обліковий запис AnyList, що може бути незручним для користувачів, які не хочуть реєструватися.
- Відсутність аналітики: AnyList не пропонує функцій для відстеження історії покупок або аналізу витрат (окрім преміум-функції додавання цін).
- Обмежена інтеграція: На відміну від Microsoft To Do, AnyList має меншу інтеграцію з іншими екосистемами (наприклад, Microsoft 365 або Google).

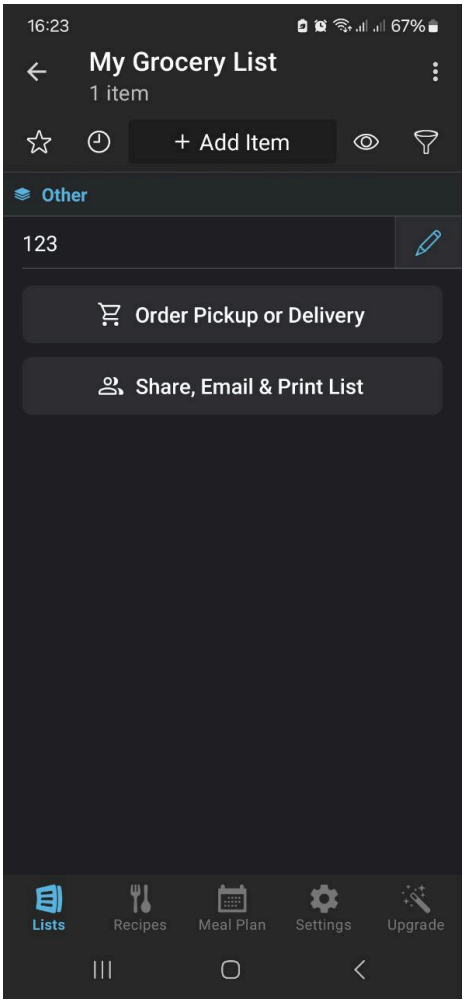


Рис 1.4. Приклад списку покупок в AnyList

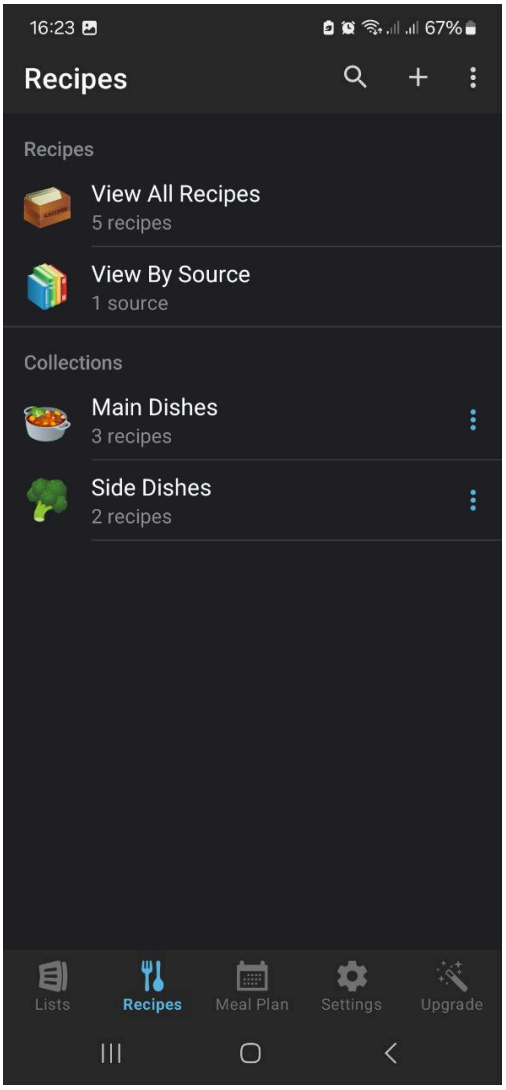


Рис 1.5. Приклад меню «Рецепти» в AnyList

1.3.4 OurGroceries

OurGroceries — це ще один спеціалізований додаток для створення списків покупок із підтримкою спільної роботи. Він доступний на iOS, Android, веб-версії, а також інтегрується з голосовими помічниками, такими як Alexa.

Його основними функціями є:

- Створення та організація списків: користувачі можуть створювати кілька списків покупок, пропонує пропозиції елементів на основі попередніх покупок, елементи можна групувати за категоріями.

- Спільна робота: списки можна ділитися з іншими користувачами через email або унікальний код списку, синхронізація в реальному часі, не потрібен обліковий запис для всіх учасників.

- Інтеграція з голосовими помічниками: підтримує інтеграцію з Amazon Alexa, Google Assistant і Siri, підтримка смарт-пристроїв.
- Додаткові функції: користувачі можуть зберігати рецепти та додавати їхні інгредієнти до списків покупок, офлайн-режим, сканування штрих-кодів
- Преміум-функції: за підписку користувачі отримують безрекламний досвід, розширені категорії та можливість додавати фото до елементів.

Переваги:

- Простота та доступність: OurGroceries має мінімалістичний інтерфейс, який легко освоїти навіть для нетехнічних користувачів.
- Гнучкість спільного доступу: На відміну від AnyList, OurGroceries дозволяє ділитися списками без необхідності створення облікового запису для всіх учасників, що спрощує співпрацю.
- Інтеграція з голосовими помічниками: Підтримка Alexa, Google Assistant і Siri робить додаток зручним для швидкого додавання елементів.
- Офлайн-доступ: Повноцінна робота в офлайн-режимі ідеально підходить для використання в магазинах із поганим інтернет-з’єднанням.
- Низька ціна преміум-версії: Преміум-підписка дешевша, ніж у AnyList, і пропонує корисні функції, такі як безрекламний режим.

Недоліки:

- Обмежений функціонал: OurGroceries менш функціональний порівняно з AnyList, особливо в плані планування їжі та імпорту рецептів.
- Реклама в безкоштовній версії: Безкоштовна версія містить рекламу, яка може бути нав’язливою.
- Менш розвинений дизайн: Інтерфейс OurGroceries виглядає застарілим порівняно з AnyList або Microsoft To Do, що може впливати на користувацький

досвід.

- Відсутність розширеної аналітики: Як і AnyList, OurGroceries не пропонує функцій для відстеження витрат або аналізу покупок.
- Обмежена інтеграція: Додаток не інтегрується з іншими екосистемами (наприклад, Microsoft 365 або Google Calendar), що обмежує його універсальність.

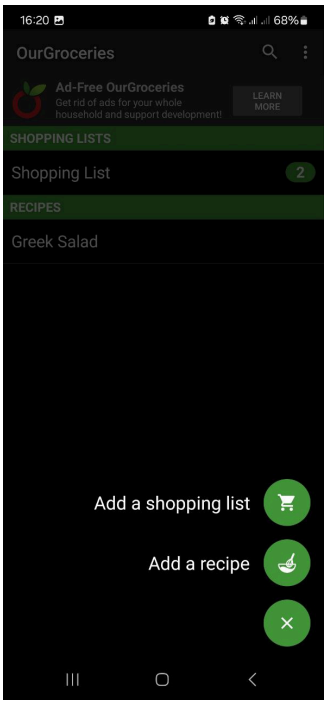


Рис.1.6. Приклад меню створення списку та рецепту в OurGroceries

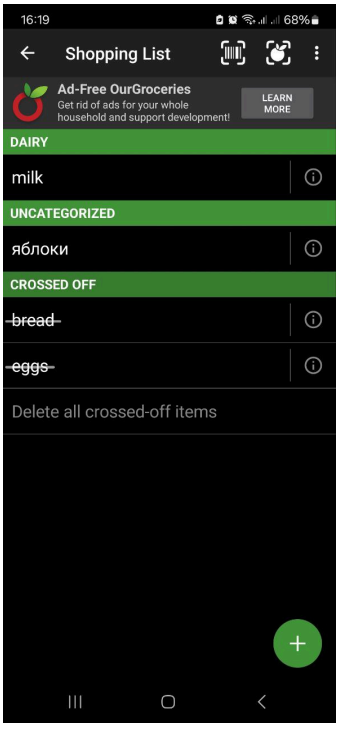


Рис.1.7. Приклад створеного списку покупок в OurGroceries

В таблиці 1.1 зведено результати аналізу існуючих застосунків для створення списку покупок та їх порівняння.

Таблиця 1.1

Зведена таблиця аналізу існуючих застосунків для створення списку покупок та їх порівняння

Показник	Google Keep	Microsoft To Do	AnyList	OurGroceries
Автоматична категоризація	Ні	Ні	Так (розширена)	Так (налаштовувана)
Спільний доступ	Через Google-акаунт	Через Microsoft-акаунт	Через email/посилання (потрібен акаунт)	Через email/код (акаунт не обов’язковий для всіх)
Синхронізація	Реальний час	Реальний час	Реальний час	Реальний час
Офлайн-режим	Так (з обмеженнями на редагування)	Обмежений (перегляд без редагування)	Повноцінний	Повноцінний
Інтеграція з рецептами	Ні	Ні	Розширена (імпорт із вебсайтів)	Базова (додавання рецептів вручну)
Голосовий ввід	Через Google Assistant	Siri, Google Assistant	Siri, Google Assistant	Alexa, Google Assistant, Siri
Планування їжі	Ні	Ні	Так (календар страв)	Базове (рецепти без календаря)

Продовження таблиці 1.1

Зведена таблиця аналізу існуючих застосунків для ведення спільного списку покупок та їх порівняння

Показник	**Google Keep**	**Microsoft To Do**	**AnyList**	**OurGroceries**
Преміум-функції	Безкоштовно	Безкоштовно	преміум	преміум
Інтеграція з екосистемами	Google (Google Calendar, Drive)	Microsoft 365 (Outlook, Teams)	Обмежена (Google Calendar у преміум)	Обмежена (голосові помічники)
Нагадування за геолокацією	Так	Так	Ні	Ні
Недоліки	Відсутність спеціалізованих функцій для покупок	Відсутність автоматичної категоризації та функцій для рецептів	Платність багатьох корисних функцій	Застарілий інтерфейс

1.4 Визначення вимог до застосунку

Проаналізувавши сутність застосунку, його специфіку, цільову аудиторію та аналогів, було визначено наступні вимоги до застосунку для спільного списку покупок:

- Реєстрація та авторизація користувачів: можливість створення облікового запису;
- Управління списками покупок: створення, редагування, видалення та перейменування списку, надання спільного доступу;
- Взаємодія з об’єктами списку: редагування, додавання та видалення товарів, відмітка виконання.

- Зручність інтерфейсу: інтуїтивно зрозумілий дизайн для різних пристроїв (ПК, смартфони, планшети);
- Безпека: використання HTTPS, авторизація за допомогою JWT, захист від SQL-ін’єкцій через ORM;
- Швидкий доступ до списків та позицій завдяки індексації в базі даних, підтримка навантаження від кількох користувачів одночасно.

2 ЗАСОБИ РЕАЛІЗАЦІЇ ТА ПРОЄКТУВАННЯ ЗАСТОСУНКУ ДЛЯ ВЕДЕННЯ СПІЛЬНОГО СПИСКУ ПОКУПОК

2.1 Середовище розробки

Інтегроване середовище розробки (IDE) – це програмний комплекс, що об’єднує інструменти для створення, редагування, компіляції, налагодження та тестування програмного забезпечення. Таке середовище дозволяє розробнику ефективно працювати з кодом в єдиному інтерфейсі, полегшуючи процес розробки завдяки автодоповненню, перевірці помилок, управлінню проєктами та взаємодії з системами контролю версій.

2.1.1 Visual Studio 2022

Для реалізації веб-застосунку було використано інтегроване середовище розробки Visual Studio 2022 — потужне IDE від компанії Microsoft, призначене для розробки програмного забезпечення на різних мовах, зокрема C#, C++, Python та JavaScript. Visual Studio 2022 забезпечує повну підтримку створення веб-додатків на базі ASP.NET Core, включаючи інструменти для проєктування інтерфейсів, управління базами даних, інтеграції з Git, відлагодження та профілювання продуктивності. Однією з ключових переваг Visual Studio є можливість роботи з багатьма типами проєктів у межах одного рішення, що забезпечує зручну структуру при реалізації багатокомпонентних додатків. IDE включає інтегрований сервер IIS Express, що дозволяє запускати та тестувати веб-застосунки локально без потреби встановлення окремого веб-сервера. Visual Studio також забезпечує гнучку систему управління NuGet-пакетами, через яку реалізується підключення необхідних бібліотек, таких як SignalR, Entity Framework Core, ASP.NET Identity тощо.

2.1.2 SQL Server Management Studio

Для управління базою даних використовувалась програма SQL Server Management Studio (SSMS) — офіційне середовище для адміністрування Microsoft SQL Server. SSMS дозволяє створювати, змінювати та переглядати бази даних, виконувати SQL-запити, управляти транзакціями, користувачами, ролями та іншими об’єктами бази даних. SSMS було використано для перегляду згенерованих Entity Framework Core моделей, перевірки цілісності збережених даних, виконання ручних SQL-запитів при налагодженні, а також для резервного копіювання та відновлення бази під час тестування.

2.2 Мови програмування

Мова програмування — це формалізована мова, призначена для комунікації інструкцій до машини, зокрема комп'ютера. Вона використовується для створення програм, які виконують алгоритми. Більшість мов програмування складається з інструкцій для комп'ютерів. Вони дозволяють розробникам ефективно створювати програми, виражаючи розрахунки та управління даними.

2.2.1 C#

Основною мовою програмування при реалізації серверної частини веб-застосунку є C# — об’єктно-орієнтована, типізована мова, що розроблена Microsoft у межах платформи .NET. C# є сучасною мовою програмування з великою екосистемою, що підтримує високий рівень абстракції, інкапсуляцію, наслідування та поліморфізм. Вона дозволяє створювати надійні та масштабовані веб-додатки, включаючи API, системи автентифікації та взаємодію з базами даних. C# активно використовується в екосистемі ASP.NET Core і має тісну інтеграцію з Entity Framework Core, що забезпечує зручну роботу з даними через ORM-механізм.

2.2.2 JavaScript

На клієнтській стороні веб-застосунку використовується JavaScript — мова сценаріїв, яка виконується в браузері та забезпечує інтерактивність користувацького

інтерфейсу. За допомогою JavaScript реалізується взаємодія з сервером через AJAX, динамічне оновлення DOM-елементів, а також обробка подій, пов’язаних з додаванням або оновленням елементів списку покупок у реальному часі через SignalR. У застосунку JavaScript використовується переважно у вигляді власноруч написаних скриптів у Razor-сторінках, без застосування сторонніх фреймворків (React, Angular), що дозволило спростити реалізацію та покращити контроль над логікою взаємодії.

2.3 Веб-фреймворк

Веб-фреймворк - це програмний фреймворк, призначений для підтримки розробки веб-додатків, включаючи веб-сервіси, веб-ресурси та веб-API. Веб-фреймворки надають стандартний спосіб будувати та розгортати веб-додатки в Інтернеті. В додатку використовується ASP.NET Core. ASP.NET Core — це кросплатформний, високопродуктивний веб-фреймворк з відкритим кодом від Microsoft, який дозволяє створювати сучасні веб-додатки, RESTful API, веб-сервіси та програми реального часу. Його основні переваги — швидкодія, масштабованість, модульність та гнучкість налаштування залежностей і сервісів через вбудовану систему впровадження залежностей (DI). У проєкті ASP.NET Core MVC використовується для створення структурованої архітектури з розділенням моделі, представлення і контролера. Аутентифікація та авторизація користувачів реалізовані за допомогою бібліотеки ASP.NET Identity. Синхронізація списків у реальному часі здійснюється через бібліотеку SignalR.

2.4 Система управління базою даних

У проєкті використовується Microsoft SQL Server як система управління реляційною базою даних. База даних містить таблиці для зберігання інформації про користувачів, списки покупок, позиції списку та права доступу. Для взаємодії із базою даних застосовується ORM-фреймворк Entity Framework Core, який забезпечує зручне створення, читання, оновлення і видалення даних через C#-класи. Робота з базою даних реалізується за допомогою міграцій, що дозволяють створювати схему бази автоматично на основі опису моделей.

2.5 Інструменти проєктування інтерфейсу користувача

Figma — це веб-орієнтований інструмент для дизайну інтерфейсів, який дає змогу командам розробляти, співпрацювати та обмінюватися дизайн-проєктами та прототипами. Створений у 2012 році Діланом Філдом та Еваном Воллесом, цей інструмент здобув популярність серед дизайнерів завдяки своїй зручності, адаптивності та широкому функціоналу.

Одна з ключових переваг Figma — повна інтеграція з веб-платформами, що дозволяє користувачам працювати на будь-якому комп'ютері без потреби встановлювати програмне забезпечення. Це забезпечує зручний доступ і сприяє командній роботі незалежно від географічного розташування учасників.

Figma пропонує інструменти для створення інтерактивних прототипів без залучення додаткових програм. Завдяки цьому дизайнери можуть швидко розробляти прототипи з анімованими переходами та мікровзаємодіями, а також тестувати їх безпосередньо в середовищі Figma.

Крім того, Figma підтримує розробку та управління дизайн-системами, даючи змогу дизайнерам зберігати стилі, компоненти та інші ресурси в зручних бібліотеках. Це полегшує підтримку узгодженості та повторне використання елементів у різних проєктах.

Інструмент Figma забезпечив зручну розробку макетів майбутнього інтерфейсу, що дозволило заздалегідь узгодити структуру користувацьких екранів, логіку переходів та загальний стиль оформлення веб-додатку. Figma дозволила створити інтерактивний прототип інтерфейсу з підтримкою мікровзаємодій, а також підготувати компоненти, які згодом були адаптовані у вигляді Razor-сторінок у Visual Studio.

2.6 Проєктування архітектури застосунку

Розроблений веб-застосунок базується на клієнт-серверній архітектурі з використанням технологій ASP.NET Core, що дозволяє забезпечити гнучкість, масштабованість та розширюваність проєкту. Архітектура побудована на основі

шаблону MVC (Model-View-Controller), який чітко розділяє логіку обробки даних, візуальне відображення та управління запитами користувача.

Застосунок реалізовано у вигляді трирівневої архітектури:

- Презентаційний рівень (View) — відповідає за взаємодію з користувачем. Реалізований за допомогою Razor Views, HTML5, CSS3 та JavaScript. На цьому рівні відображається користувацький інтерфейс, а також відбувається обробка подій користувача, які надсилаються до контролерів.

- Рівень бізнес-логіки (Controller та Service Layer) — забезпечує обробку запитів від клієнта, виконання відповідних бізнес-правил та координацію взаємодії між моделями та представленнями. Основна логіка, пов'язана з керуванням списками покупок, правами доступу, а також взаємодією з базою даних, зосереджена саме тут.

- Рівень доступу до даних (Data Access Layer) — реалізовано за допомогою Entity Framework Core. Цей рівень взаємодіє з реляційною базою даних та відповідає за зберігання, отримання й оновлення інформації про списки покупок, користувачів та доступи.

Додатково в архітектуру інтегровано SignalR, який реалізує двосторонній зв’язок між клієнтом і сервером, дозволяючи здійснювати синхронізацію змін у списках покупок у режимі реального часу. Завдяки цьому кілька користувачів можуть одночасно працювати над одним списком і миттєво бачити зміни.

Таким чином, застосунок побудований із урахуванням принципів модульності, інкапсуляції, розширюваності та масштабованості. Така архітектура дає змогу легко вносити зміни, додавати нові функції або адаптувати застосунок до змін у середовищі використання.

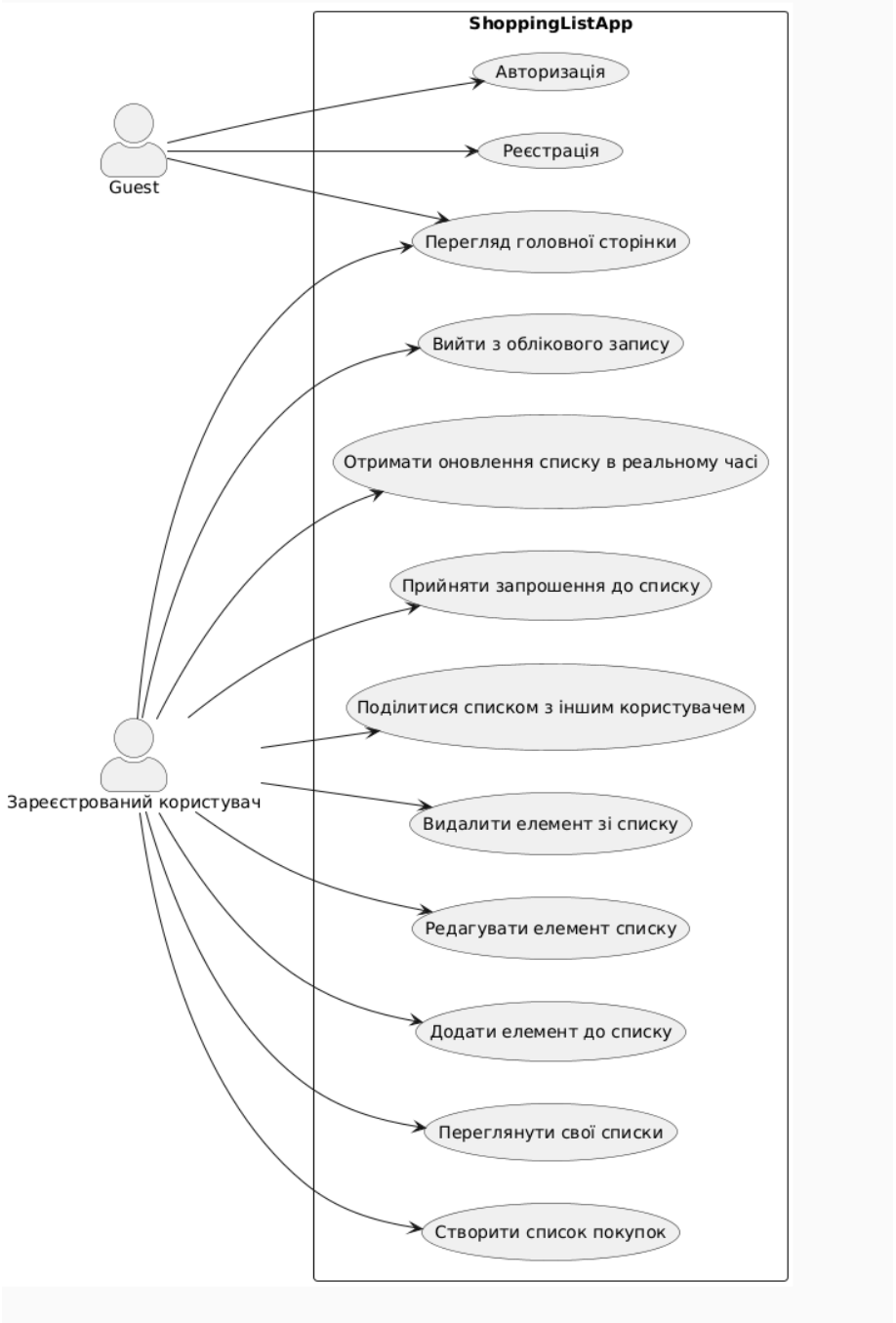


Рис. 3.1 Діаграма варіантів використання

2.7 Архітектура клієнтської частини

Клієнтська частина реалізована за допомогою Razor Pages та JavaScript, що забезпечує динамічну взаємодію користувача з веб-застосунком. Основна мета клієнтської частини — надати зручний інтерфейс для перегляду, редагування та спільного використання списків покупок.

Під час розробки клієнтського інтерфейсу використовувалися такі технології:

- Razor View Engine — механізм генерації HTML-сторінок на основі шаблонів Razor, що дозволяє інтегрувати C#-код безпосередньо у розмітку.
- JavaScript — відповідає за обробку подій на стороні клієнта, взаємодію з SignalR-хабом, динамічне оновлення DOM-елементів без перезавантаження сторінки.
- HTML5 + CSS3 — забезпечують структуру і базове оформлення елементів інтерфейсу.

Користувач має змогу виконувати такі дії:

- Створювати нові списки покупок та додавати в них елементи.
- Редагувати або видаляти окремі позиції у списках.
- Ділитися списками з іншими користувачами через електронну пошту або логін.
- В режимі реального часу бачити, як інші користувачі додають або змінюють пункти у спільному списку.

Важливою особливістю клієнтської частини є інтеграція з SignalR, яка забезпечує двостороннє з’єднання WebSocket між браузером користувача і сервером. Це дозволяє миттєво оновлювати вміст списку для всіх користувачів, які мають доступ до нього.

Комунікація з сервером відбувається як через традиційні HTTP POST-запити (наприклад, при створенні списку чи додаванні елемента), так і через SignalR-канал — для синхронізації змін у реальному часі.

Таким чином, клієнтська частина веб-застосунку є гнучкою, інтерактивною та повністю орієнтованою на покращення зручності взаємодії з користувачем.

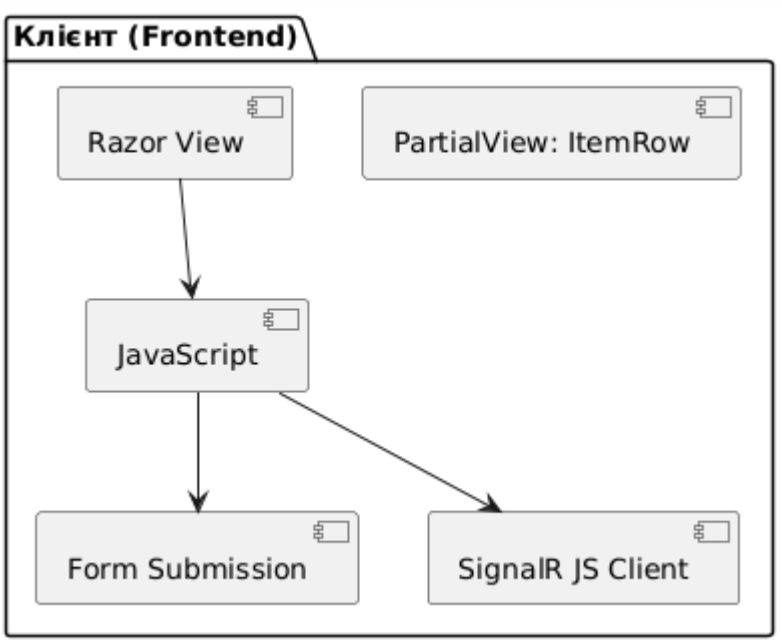


Рис. 3.2 Діаграма пакетів клієнтської частини

2.8 Архітектура серверної частини

Серверна частина розроблена з використанням ASP.NET Core — сучасного, кросплатформного фреймворку для створення високопродуктивних веб-застосунків. У її структурі виокремлено кілька ключових компонентів:

- Контролери (Controllers) — відповідають за обробку HTTP-запитів від клієнта. Кожен контролер реалізує набір методів (Action Methods), які відповідають за обробку конкретних сценаріїв — створення, редагування списків, надання доступу, тощо.

- Сервіси (Services) — інкапсулюють бізнес-логіку застосунку. Вони використовуються контролерами та можуть взаємодіяти з репозиторіями (або напряму з контекстом бази даних через EF Core). Поділ відповідальностей між контролерами й сервісами дозволяє покращити тестованість і підтримку коду.

- SignalR Hub — спеціальний клас, який реалізує обробку повідомлень у режимі реального часу. Кожного разу, коли користувач додає, видаляє або змінює

елемент у списку, сервер через SignalR надсилає повідомлення іншим підключеним користувачам, які працюють із тим самим списком.

- ASP.NET Identity — відповідає за автентифікацію користувачів, реєстрацію, вхід/вихід, а також забезпечення авторизації доступу до списків. Кожен користувач має унікальний обліковий запис із паролем, захищеним хеш-функцією.

- Entity Framework Core — ORM, яка дозволяє працювати з реляційною базою даних за допомогою C#-класів (моделей). Весь доступ до таблиць бази реалізовано через контекст даних ApplicationDbContext.

- Middleware-компоненти — фільтри, обробники помилок, логування запитів та конфігурація HTTP-пайплайна реалізовані у файлаі Program.cs.

Такий підхід дозволяє побудувати надійний, безпечний і розширюваний серверний застосунок із підтримкою високої навантаженості та сучасних вимог до веб-розробки.

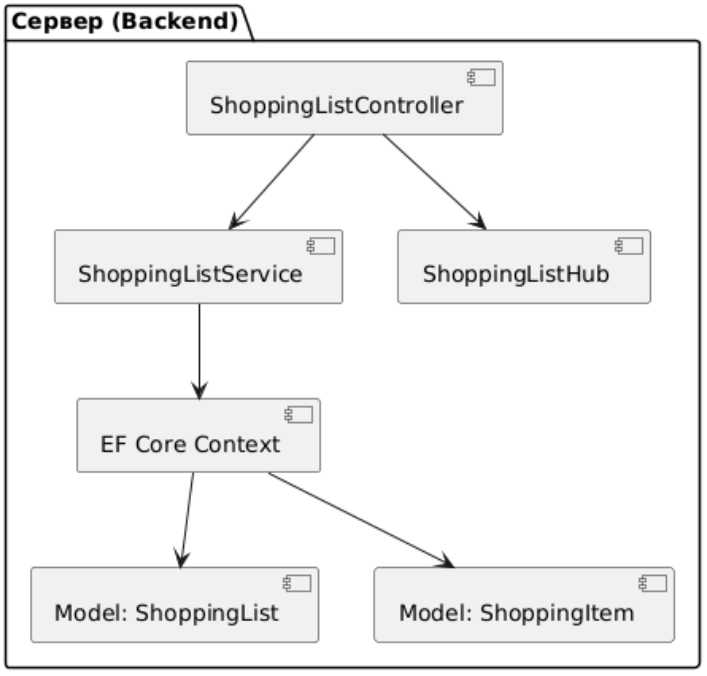


Рис. 3.3 Діаграма пакетів серверної частини

2.9 Архітектура бази даних

База даних, яка використовується в проєкті, реалізована у вигляді реляційної структури з використанням системи керування базами даних (СКБД) SQL Server. Для взаємодії з базою даних використовується ORM Entity Framework Core, що дозволяє автоматично створювати таблиці на основі моделей C# та здійснювати всі операції доступу до даних у strongly-typed вигляді.

Основні сутності бази даних:

1. Users (AspNetUsers) — таблиця, що містить облікові записи користувачів. Автоматично створюється системою Identity. Зберігає логін, email, хешований пароль, тощо.

2. ShoppingLists

- Id — первинний ключ.
- Name — назва списку.
- OwnerId — зовнішній ключ на таблицю Users.
- CreatedAt — дата створення.

3. ShoppingItems

- Id — первинний ключ.
- ListId — зовнішній ключ на таблицю ShoppingLists.
- Name — назва позиції.
- Quantity — кількість.
- IsCompleted — ознака, чи куплено товар.

- AddedBy — Id користувача, який додав.

4. SharedAccess

- Id — первинний ключ.

- ListId — зовнішній ключ на таблицю ShoppingLists.

- UserId — Id користувача, який отримав доступ.

Зв’язки між таблицями:

- Один користувач може мати багато списків (зв’язок один-до-багатьох).

- Один список може мати багато позицій (один-до-багатьох).

- Один список може бути доступний багатьом користувачам через SharedAccess (багато-до-багатьох із додатковими атрибутами).

Завдяки використанню Entity Framework Core весь процес взаємодії з базою даних (створення, оновлення, видалення записів) реалізується за допомогою об’єктів C# і LINQ-запитів, що значно спрощує код і зменшує ризик SQL-ін’єкцій.

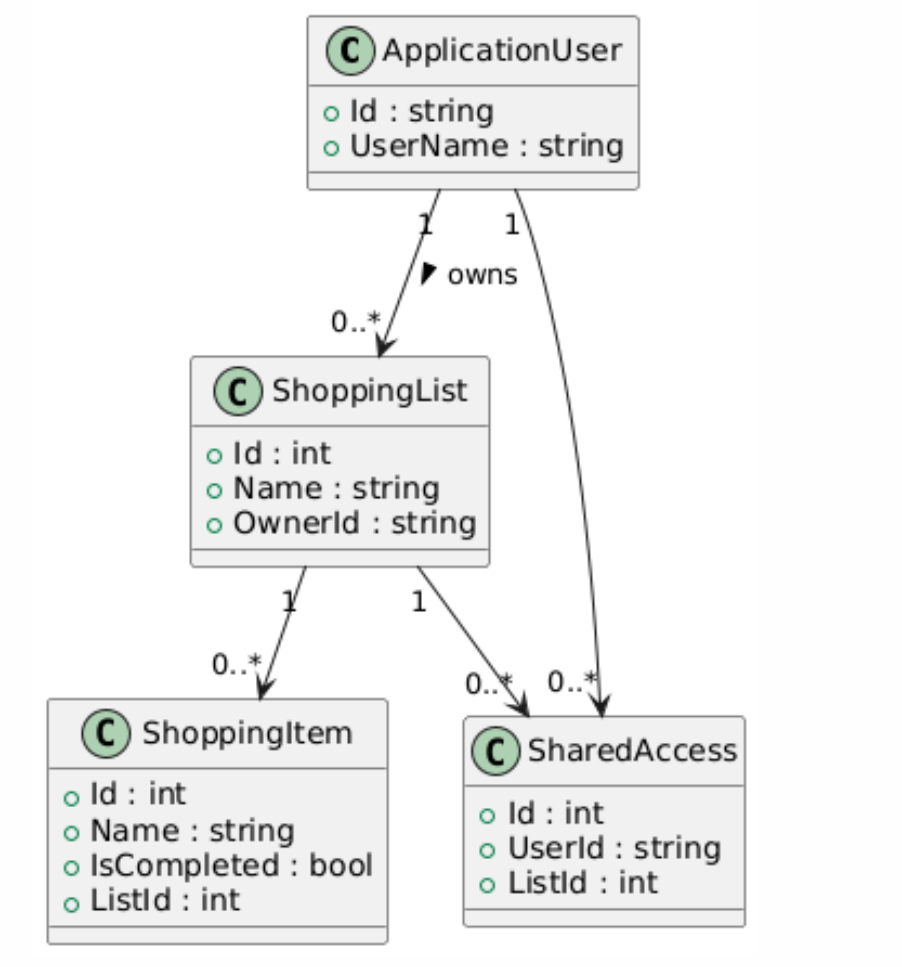


Рис. 3.4 Діаграма класів бази даних

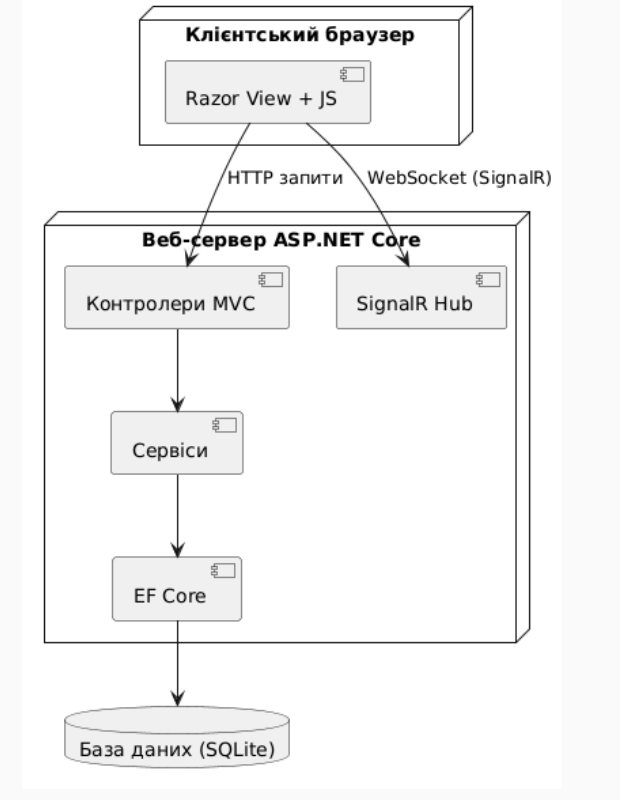


Рис. 3.5 Діаграма розгортання клієнт-серверної моделі

3 РЕАЛІЗАЦІЯ ЗАСТОСУНКУ ДЛЯ ВЕДЕННЯ СПІЛЬНОГО СПИСКУ ПОКУПОК

3.1 Дизайн інтерфейсу

Інтерфейс користувача (UI, User Interface) є ключовим елементом будь-якого програмного продукту, адже саме через нього користувач взаємодіє з додатком. У контексті розробки веб-застосунку для спільного списку покупок основний акцент був зроблений на простоті, інтуїтивності, адаптивності та максимальній зручності для користувачів. UI-дизайн у цьому проєкті — це не лише про візуальну привабливість, а й про забезпечення швидкого доступу до основних функцій, таких як створення списків, додавання товарів чи спільний доступ. Дизайн мав бути зрозумілим як для досвідчених користувачів, так і для новачків, які вперше працюють із подібними додатками.

Процес планування інтерфейсу розпочався з розробки прототипів у Figma — сучасному інструменті для створення UI/UX-дизайну, який дозволяє командам працювати спільно та швидко вносити зміни. У Figma було створено кілька макетів для основних сторінок додатка: головна сторінка з переліком доступних списків, форма для входу та реєстрації, інтерфейс перегляду конкретного списку покупок, сторінка додавання нових товарів, а також налаштування доступу для спільної роботи. Кожен макет був ретельно продуманий, щоб забезпечити логічну послідовність дій користувача та уникнути надмірної складності.

Дизайн базувався на визнаних стандартах — принципах Material Design від Google та Human Interface Guidelines від Apple. Це допомогло створити інтерфейс, який відповідає сучасним очікуванням користувачів і забезпечує знайомий досвід незалежно від платформи. Для візуального оформлення використовувалися пастельні кольори, які створюють м’який і ненав’язливий вигляд, не відволікаючи від основного контенту. Великі кнопки були спеціально адаптовані для зручної взаємодії на мобільних пристроях, де точність натискання може бути нижчою. Ієрархія шрифтів (наприклад, більший розмір для заголовків і менший для другорядних елементів) допомогла розставити акценти та спростити сприйняття інформації.

Ключові елементи інтерфейсу:

- Панель навігації: Розташована у верхній частині екрана, містить логотип додатка, основні посилання (наприклад, "Мої списки", "Налаштування") та кнопку профілю користувача. Панель забезпечує швидкий доступ до всіх основних розділів.
- Карточки списків покупок: Кожен список відображається у вигляді карточки з назвою, датою створення, кількістю елементів і кнопками для швидких дій (наприклад, "Поділитися", "Редагувати", "Видалити").
- Модальні вікна: Використовуються для додавання чи редагування товарів, а також для налаштування спільного доступу. Модальні вікна дозволяють користувачам зосередитися на конкретній дії, не залишаючи основну сторінку.
- Візуальні індикатори: Показують статус синхронізації (наприклад, іконка "оновлення" під час збереження змін) та зміни в реальному часі, що були внесені іншими користувачами (реалізовано через SignalR). Це допомагає уникнути плутанини під час спільної роботи.
- Респонсивна верстка: Інтерфейс адаптований для різних пристроїв за допомогою CSS-фреймворків Bootstrap і Tailwind. Наприклад, на маленьких екранах панель навігації трансформується в бургер-меню, а карточки списків розташовуються вертикально.

Для оцінки зручності дизайну проводилося тестування прототипів у Figma. Кілька груп користувачів (включаючи цільову аудиторію — сім’ї, студентів, пари) протестували макети, виконуючи типові завдання: створення списку, додавання товарів, надання доступу іншому користувачу. Зворотний зв’язок допоміг виявити UX-помилки, такі як недостатня контрастність кнопок або незручне розташування елементів на маленьких екранах. Усі зауваження були враховані, що дозволило значно покращити інтерфейс ще до початку програмної реалізації. Підсумовуючи, UI-дизайн був розроблений з урахуванням потреб користувачів, які прагнуть швидко й ефективно керувати списками покупок, а тестування допомогло мінімізувати помилки на ранніх етапах.

3.2 Реалізація клієнтської частини

Клієнтська частина (frontend) додатка була розроблена з використанням стеку технологій: HTML, CSS (з фреймворком TailwindCSS) та JavaScript (з бібліотекою jQuery для спрощення маніпуляцій із DOM). Основна мета полягала в створенні динамічного та інтерактивного інтерфейсу, який би дозволив користувачам комфортно взаємодіяти з додатком у режимі реального часу, без затримок чи необхідності перезавантаження сторінок.

Клієнтська логіка була спроєктована для забезпечення основних функцій:

- Відображення списків покупок: На головній сторінці користувач бачить перелік своїх списків у вигляді карточок, кожна з яких містить основну інформацію (назва, кількість елементів, статус синхронізації).
- Створення нового списку: Через просту HTML-форму користувач може ввести назву списку та одразу створити його. Форма включає валідацію на клієнтському рівні (наприклад, перевірка на мінімальну довжину назви).
- Редагування списків: Додавання, редагування та видалення товарів у списку відбувається асинхронно через AJAX-запити, що дозволяє уникнути перезавантаження сторінки.
- Синхронні зміни: Завдяки інтеграції з SignalR, усі зміни, зроблені іншими користувачами (наприклад, додавання нового товару), миттєво відображаються на екрані без додаткових дій із боку користувача.

JavaScript та SignalR логіка:

- Після завантаження сторінки ініціалізується підключення до SignalR Hub за допомогою JavaScript-бібліотеки SignalR. Це дозволяє клієнту отримувати оновлення від сервера в реальному часі.
- Клієнт підписується на ключові події: ItemAdded (додавання товару), ItemUpdated (оновлення товару), ItemDeleted (видалення товару). Для кожної події визначені обробники, які динамічно оновлюють DOM.
- Локальні дії користувача (наприклад, натискання кнопок "Додати", "Редагувати", "Видалити") викликають відповідні методи SignalR, які передають

зміни на сервер, а той, у свою чергу, розсилає оновлення всім підключеним клієнтам.

Адаптивність:

- Використання TailwindCSS значно спростило створення респонсивного дизайну. Завдяки вбудованим класам, таким як sm:, md:, lg:, інтерфейс автоматично адаптується до різних розмірів екранів. Наприклад, на десктопі списки покупок відображаються в сітці 3×3, а на смартфоні — у вертикальному списку.
- Оптимізація для мобільних пристроїв була пріоритетом, адже за оцінками, близько 50% користувачів працюватимуть із додатком зі смартфонів або планшетів. Для цього були враховані особливості сенсорного введення: кнопки зроблені достатньо великими, а відстані між елементами — комфортними для натискання пальцем.
- Додатково проводилося тестування на різних пристроях (iPhone, Android-смартфони, iPad), щоб переконатися, що інтерфейс виглядає і працює коректно незалежно від розміру екрана чи операційної системи.

Таким чином, клієнтська частина забезпечує швидкий, зручний і безперебійний доступ до всіх функцій додатка, створюючи ефект "живого" застосунку, який реагує на дії користувача в реальному часі. Адаптивність, інтерактивність і оптимізація для мобільних пристроїв роблять додаток доступним для широкої аудиторії.

3.3 Реалізація серверної частини

Серверна частина додатка була створена з використанням ASP.NET Core 8.0 — сучасного фреймворку, який відомий своєю високою продуктивністю, кросплатформністю та можливістю масштабування. Архітектура проєкту базується на шаблоні MVC (Model–View–Controller), що забезпечує чіткий розподіл відповідальностей між логікою, даними та відображенням. Такий підхід спрощує підтримку коду, його тестування та подальший розвиток.

Основні компоненти серверної частини:

- Controllers: Відповідають за обробку HTTP-запитів від клієнтів. Вони координують роботу між моделями, сервісами та представленнями, а також керують логікою авторизації, CRUD-операціями (створення, читання, оновлення, видалення) та маршрутизацією. Наприклад, ShoppingListController обробляє запити на створення нового списку чи додавання товару.
- Services: Містять бізнес-логіку додатка. Наприклад, сервіс ShoppingListService перевіряє права доступу користувача до списку, обробляє логіку спільного доступу та взаємодіє з SignalR для відправлення оновлень клієнтам.
- Repositories: Реалізують доступ до даних через патерн "Репозиторій", абстрагуючи роботу з базою даних. Використовується Entity Framework Core для спрощення операцій із базою даних, таких як вибірка, оновлення чи видалення записів.
- Models: Включають два типи моделей — DTO (Data Transfer Objects) для передачі даних між клієнтом і сервером, та Entity для представлення сутностей у базі даних. Основні сутності: User (користувач), ShoppingList (список покупок), Item (елемент списку), SharedAccess (доступ до списків).

Технологічна база:

- ASP.NET Core 8.0: Забезпечує високу швидкість обробки запитів і підтримку асинхронного програмування.
- Entity Framework Core: Використовується як ORM для взаємодії з базою даних, дозволяючи працювати з даними через об’єктно-реляційне відображення.
- ASP.NET Identity: Відповідає за автентифікацію та реєстрацію користувачів, включаючи управління ролями, паролями та токенами.
- SignalR: Використовується для реалізації синхронізації в реальному часі, дозволяючи серверу повідомляти клієнтів про зміни (наприклад, додавання нового товару до списку).

Безпека:

- Авторизація запитів: Кожен HTTP-запит перевіряється на авторизованість користувача через ASP.NET Identity. Якщо користувач не авторизований, доступ до захищених ресурсів (наприклад, редагування списку) забороняється.
- Контроль доступу: Доступ до списків покупок регулюється на основі ідентифікатора користувача. Наприклад, користувач може редагувати лише ті списки, де він є власником або має права спільного доступу.
- Захист від атак: ASP.NET Core за замовчуванням включає захист від CSRF (Cross-Site Request Forgery) через токени, а також від XSS (Cross-Site Scripting) завдяки вбудованим механізмам Razor Pages та middleware. Додатково використовується валідація вхідних даних для запобігання SQL-ін’єкціям.
- Шифрування даних: Чутливі дані, такі як паролі, зберігаються в зашифрованому вигляді за допомогою ASP.NET Identity, а передача даних між клієнтом і сервером захищена через HTTPS.

Продуктивність:

- Використання асинхронного програмування (async/await) дозволяє ефективно обробляти велику кількість запитів без блокування потоків.
- Кешування даних (наприклад, через Redis або вбудований IMemoryCache) було впроваджено для зменшення навантаження на базу даних при частих запитах, таких як вибірка списків покупок.
- Оптимізація запитів до бази даних через використання LINQ та попереднього завантаження пов’язаних даних (Eager Loading) за допомогою Entity Framework Core.

Таким чином, серверна частина забезпечує надійний, безпечний і масштабований фундамент для роботи додатка. Вона ефективно обробляє запити користувачів, підтримує синхронізацію в реальному часі та гарантує безпеку даних, що є критично важливим для спільного використання списків покупок.

3.4 Створення бази даних та взаємодія з нею

Сховище даних для додатка було створене на основі реляційної бази даних Microsoft SQL Server, а для взаємодії з нею використовувався ORM-фреймворк Entity Framework Core. Такий підхід дозволив спростити роботу з даними, забезпечивши об’єктно-реляційне відображення та автоматизацію багатьох рутинних завдань, таких як створення таблиць чи управління зв’язками. Створення бази даних здійснювалося через механізм міграцій Entity Framework Core: команди Add-Migration і Update-Database дозволили автоматично генерувати та застосовувати зміни до структури бази даних на основі коду.

Основні сутності та зв’язки:

- User: Представляє унікального користувача додатка. Містить поля, такі як ідентифікатор, ім’я, email, хешований пароль (через ASP.NET Identity). Один користувач може мати кілька списків покупок.

- ShoppingList: Список покупок, який включає назву, дату створення, ідентифікатор власника (зв’язок із User) та колекцію елементів (Items). Один список може належати одному власнику, але бути доступним для кількох користувачів через спільний доступ.

- Item: Елемент списку покупок, який містить назву, кількість, статус покупки (наприклад, "куплено" чи "не куплено") та ідентифікатор списку, до якого він належить.

- SharedAccess: Таблиця для реалізації зв’язку "багато до багатьох" між користувачами (User) та списками (ShoppingList). Вона визначає, які користувачі мають доступ до певного списку, і містить ідентифікатор користувача, ідентифікатор списку та рівень доступу (наприклад, "перегляд" чи "редагування").

Ключові принципи проєктування бази даних:

- Цілісність даних: Використання зовнішніх ключів (Foreign Keys) забезпечує зв’язність між таблицями та запобігає появі "осиротілих" записів (наприклад, елементів без списку).

- Fluent API: Для точного налаштування зв’язків і обмежень у базі даних використовувалася конфігурація через Fluent API в Entity Framework Core. Наприклад, було чітко визначено каскадне видалення: якщо список видаляється, усі пов’язані елементи (Items) також видаляються автоматично.

- Seed-дані: Для тестування бази даних були створені початкові тестові дані через метод HasData. Наприклад, додаток ініціалізується з кількома користувачами, списками та товарами, щоб розробники могли одразу перевірити функціонал.

Взаємодія з базою даних:

- Усі операції з базою даних здійснюються через сервіси та репозиторії, що забезпечує інкапсуляцію логіки доступу до даних. Наприклад, ShoppingListRepository містить методи для створення, оновлення та видалення списків.

- Для підвищення продуктивності використовуються асинхронні методи (async/await), що дозволяють уникнути блокування потоків під час виконання запитів до бази даних.

- Механізм відстеження змін (ChangeTracker) в Entity Framework Core оптимізує запити, оновлюючи лише ті записи, які дійсно змінилися, що зменшує навантаження(OnConnectedAsync, Groups.AddToGroupAsync(...)) на базу.

- Для складних запитів, таких як вибірка списків із пов’язаними елементами та користувачами, використовується попереднє завантаження даних (Eager Loading) через метод Include.

Безпека даних:

- Захист паролів: ASP.NET Identity автоматично хешує паролі користувачів за допомогою алгоритму PBKDF2, що забезпечує захист від витоку даних.

- Контроль доступу: Репозиторії перевіряють права доступу перед виконанням будь-яких операцій. Наприклад, користувач не може змінити список, якщо він не є власником або не має відповідних прав через SharedAccess.

- Шифрування: Усі з’єднання з базою даних захищені через SSL, а чутливі дані (наприклад, токени авторизації) зберігаються в зашифрованому вигляді.

Оптимізація:

- Індекси були додані до часто використовуваних полів (наприклад, UserId у таблиці ShoppingList), щоб прискорити пошук.

- Для зменшення кількості запитів використовувалася пакетна обробка (batching) в Entity Framework Core, що дозволяє групувати кілька операцій в один SQL-запит.

- Логування запитів до бази даних (через ILogger) допомагало виявляти повільні запити та оптимізувати їх.

Підсумовуючи, створення бази даних і її взаємодія є основою архітектури додатка, яка забезпечує надійне зберігання даних, швидкий доступ до них і високу безпеку. Завдяки ретельному проєктуванню та оптимізації база даних ефективно підтримує всі операції, необхідні для роботи спільного списку покупок.

3.5 Реалізація механізму синхронізації

Механізм синхронізації в додатку був реалізований за допомогою ASP.NET Core SignalR — потужної бібліотеки для двосторонньої комунікації між клієнтом і сервером у реальному часі. Це ключова технологія, яка забезпечує миттєве

оновлення інтерфейсу для всіх користувачів, які мають доступ до певного списку покупок, без необхідності перезавантаження сторінки. Такий підхід значно підвищує зручність спільної роботи, адже зміни (наприклад, додавання нового товару) одразу відображаються у всіх учасників.

Принцип роботи:

Після входу в додаток кожен клієнт (браузер користувача) встановлює підключення до SignalR Hub через JavaScript-клієнт SignalR. Це створює WebSocket-з'єднання (або альтернативний транспорт, якщо WebSocket недоступний), яке забезпечує безперервний зв'язок із сервером.

При кожній дії, яка змінює список (додавання, редагування, видалення товару), сервер викликає метод Clients.Group(listId).SendAsync(...), який надсилає оновлення всім клієнтам, підключеним до групи цього списку.

На стороні клієнта JavaScript-обробник отримує повідомлення від сервера і оновлює DOM, додаючи, змінюючи або видаляючи елементи списку в реальному часі.

Організація груп:

Кожен список покупок у додатку асоціюється з унікальною групою в SignalR, яка ідентифікується за listId. Це дозволяє надсилати оновлення лише тим користувачам, які переглядають конкретний список, і не перевантажувати інших клієнтів.

При підключенні до списку (наприклад, коли користувач відкриває сторінку списку) клієнт додається до відповідної групи через методи OnConnectedAsync і Groups.AddToGroupAsync(...). При виході зі списку клієнт видаляється з групи.

Такий підхід забезпечує ефективне розмежування потоків даних і масштабованість, адже сервер не надсилає зайвих повідомлень.

Обробка подій:

Основні методи для передачі оновлень: ItemAdded, ItemUpdated, ItemDeleted. Наприклад, коли користувач додає новий товар, сервер викликає ItemAdded, передаючи дані про товар (назва, кількість) усім клієнтам у групі.

Перед відправленням оновлень сервер перевіряє права доступу користувача до списку через сервіс ShoppingListService. Це гарантує, що оновлення отримують лише авторизовані учасники.

На клієнтській стороні JavaScript-код оновлює DOM: наприклад, додає новий рядок у список товарів або змінює статус існуючого товару. Для покращення користувацького досвіду додано анімації оновлення (наприклад, плавна поява нового елемента).

Переваги:

Низька затримка: Середній час доставки оновлень становить менше 100 мс, що забезпечує відчуття "миттєвості" при спільній роботі.

Масштабованість: SignalR підтримує інтеграцію з Azure SignalR Service, що дозволяє масштабувати додаток для великої кількості користувачів без втрати продуктивності.

Гнучкість: Технологія SignalR дозволяє легко додати нові функції, такі як чат між учасниками списку, сповіщення про нові дії (наприклад, "Користувач X додав товар") або live-редагування (одночасне редагування одного елемента кількома користувачами).

Кросбраузерність: SignalR автоматично обирає оптимальний транспорт (WebSocket, Server-Sent Events або Long Polling) залежно від можливостей браузера, що забезпечує сумісність із усіма сучасними платформами.

Оптимізація та безпека:

Для зменшення навантаження на сервер використовується стиснення даних (compression) під час передачі оновлень через SignalR.

Усі з’єднання захищені через HTTPS, а авторизація клієнтів здійснюється через JWT-токени, що передаються під час підключення до SignalR Hub.

Логування подій (наприклад, підключення/відключення клієнтів) дозволяє швидко виявляти проблеми, такі як втрата з’єднання або помилки авторизації.

Обробка помилок:

У разі втрати з’єднання SignalR автоматично намагається перепідключитися

(reconnect) протягом заданого часу. Клієнт отримує сповіщення про втрату з’єднання, а після відновлення синхронізується з останніми змінами.

Якщо оновлення не доходить до клієнта (наприклад, через мережеві проблеми), клієнтська логіка періодично запитує сервер (через REST API) для синхронізації даних.

Таким чином, механізм синхронізації через SignalR є основою для реалізації концепції "спільного списку" у додатку. Він забезпечує швидке, надійне та безпечне оновлення даних, підвищуючи зручність спільної роботи та створюючи безшовний користувацький досвід.

4 ТЕСТУВАННЯ ЗАСТОСУНКУ ДЛЯ ВЕДЕННЯ СПІЛЬНОГО СПИСКУ ПОКУПОК

У рамках розробки веб-застосунку для ведення спільного списку покупок було проведено мануальне функціональне тестування основних модулів системи. Метою тестування є перевірка відповідності реалізованої функціональності технічним вимогам, виявлення помилок та забезпечення стабільної роботи системи в типових сценаріях використання.

4.1 Вибір методики тестування

Для перевірки функціональності використовувались сценарії ручного тестування, що охоплюють основні користувацькі дії:

1. створення списку покупок;
2. додавання, редагування та видалення позицій у списку;
3. перемикання стану "куплено/не куплено" для кожної позиції;
4. спільний доступ до списку іншим користувачам;
5. синхронізація змін у списку між користувачами через SignalR.

Тестування проводилось на локальному сервері в середовищі розробки Visual Studio 2022 з використанням браузера Google Chrome.

4.2 Результати тестування

Таблиця 1.2

Таблиця мануального функціонального тестування застосунку

№	Тестовий сценарій	Очікуваний результат	Фактичний результат	Статус	Демонстрація результату
1	Створення нового списку	Список з’являється у переліку	Список створено успішно	Пройдено	Рис. 5.1
2	Додавання позиції у список	Позиція з'являється у таблиці	Позиція додана коректно	Пройдено	Рис. 5.2
3	Позначення товару як "куплений"	Стан змінюється, стилізація оновлюється	Функціональність працює	Пройдено	Рис. 5.3

Продовження таблиці 1.2

Таблиця мануального функціонального тестування застосунку

№	Тестовий сценарій	Очікуваний результат	Фактичний результат	Статус	Демонстрація результату
4	Видалення позиції	Позиція зникає зі списку	Видалено успішно	Пройдено	Рис. 5.4
5	Надання спільного доступу іншому користувачу	Користувач бачить список у своєму кабінеті	Список доступний для другого користувача	Пройдено	Рис. 5.5
6	Одночасна робота кількох користувачів зі списком	Зміни відображаються в реальному часі	SignalR працює стабільно	Пройдено	Рис. 5.6

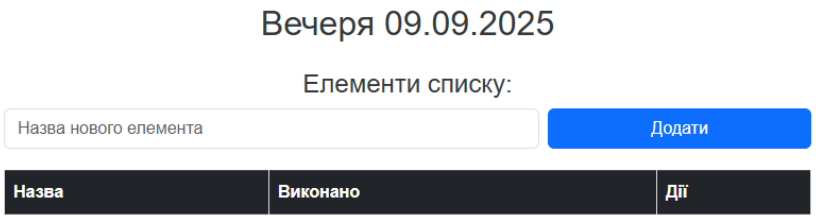


Рис. 5.1 Тестування створення нового списку

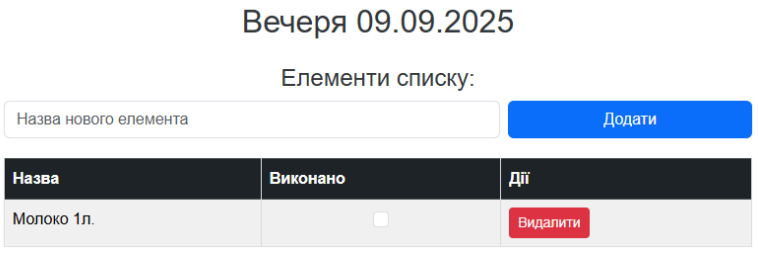


Рис. 5.2. Тестування додавання позиції у список

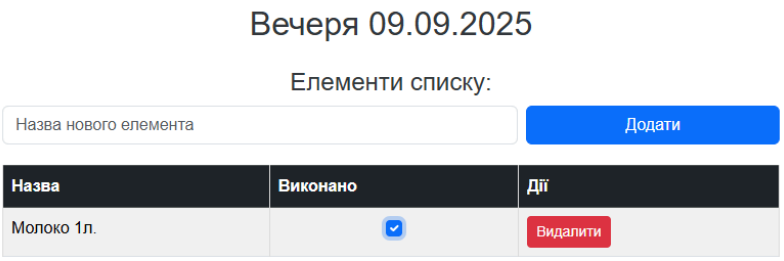


Рис. 5.3. Тестування додавання позиції у список

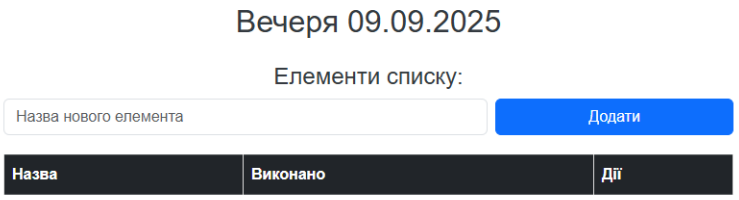


Рис. 5.4. Тестування видалення позиції

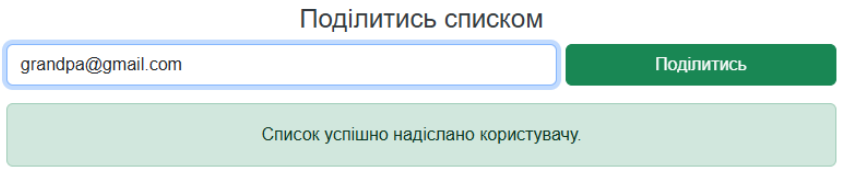


Рис. 5.5. Тестування надання спільного доступу іншому користувачу

Рис. 5.6 Тестування одночасної роботи кількох користувачів зі списком

Проведене тестування підтвердило коректну реалізацію ключової функціональності застосунку. Всі протестовані сценарії завершилися успішно. Недоліків або критичних помилок виявлено не було. Таким чином, веб-застосунок відповідає функціональним вимогам і готовий до подальшого використання або розгортання на продакшн-сервері.

ВИСНОВКИ

Результатами виконаної роботи стало створення web-застосунку для ведення спільного списку покупок із синхронізацією між користувачами, що дало змогу підвищити ефективність планування та організації покупок у групі користувачів. Для реалізації проєкту було використано мову програмування C# та фреймворк ASP.NET Core, SignalR для синхронізації в реальному часі, а також інструменти Entity Framework Core і JavaScript.

Було виконано наступні задачі:

– Проаналізовано предметну область: визначено основні потреби користувачів у веденні списків покупок із можливістю спільного доступу; охарактеризовано цільову аудиторію, яка потребує ефективного інструменту для організації спільних покупок; проведено аналіз існуючих рішень та виявлено їх недоліки — зокрема, відсутність реального сповіщення про зміни, обмеженість у спільному доступі та недостатня адаптивність інтерфейсу. На основі цього сформульовано вимоги до системи: реєстрація й автентифікація користувачів, створення особистих і спільних списків, додавання та редагування елементів, візуальне позначення статусу товарів, а також миттєва синхронізація даних між користувачами.

– Обрано засоби реалізації: Visual Studio як основне середовище розробки, C# як основна мова програмування для серверної частини, ASP.NET Core MVC як фреймворк для побудови backend-частини, Entity Framework Core як ORM для доступу до бази даних, JavaScript і jQuery для клієнтської частини, SignalR для реалізації синхронізації у реальному часі, GitHub для контролю версій, Microsoft SQL Server як система управління базою даних, Figma як інструмент для проєктування інтерфейсу.

– Зроблено архітектурне проєктування системи: створено діаграми варіантів використання, класів, послідовностей і розгортання відповідно до стандартів UML, що дозволило структуровано підійти до реалізації логіки веб-застосунку.

– Реалізовано систему, використовуючи обрані технології: створено UI-прототипи в Figma, налаштовано середовище ASP.NET Core і створено серверний проєкт, реалізовано механізми реєстрації, автентифікації та авторизації користувачів, реалізовано CRUD-операції для списків та позицій у списках, впроваджено можливість надання спільного доступу до списків за email, налаштовано двосторонню синхронізацію змін між користувачами за допомогою SignalR, здійснено базову адаптацію клієнтської частини до мобільних пристроїв, протестовано основний функціонал на коректність і стабільність.

За результатами роботи було опубліковано таку тезу:

1. Рябов А.С. ЗАХИСТ ІНФОРМАЦІЇ У WEB-ЗАСТОСУНКУ ДЛЯ СТВОРЕННЯ СПИСКУ ПОКУПОК. //XII Всеукраїнська науково-практичної конференція молодих учених «Інформаційні технології – 2025» (IT-2025). – Київ: КИЇВСЬКИЙ СТОЛИЧНИЙ УНІВЕРСИТЕТ ІМЕНІ БОРИСА ГРІНЧЕНКА, 2025. – С.320-322

ПЕРЕЛІК ПОСИЛАНЬ

1. Smith J. Web Development with ASP.NET Core: Building Modern Applications. Berkeley, CA : Apress, 2022. 320 p.

2. Johnson R., Smith T. Real-Time Web Applications with SignalR. Birmingham : Packt Publishing, 2021. 245 p.

3. ASP.NET Core documentation. URL: https://docs.microsoft.com/en-us/aspnet/core (дата звернення: 25.05.2025).

4. Brown L. et al. Collaborative Tools in Web Development. Journal of Software Engineering and Applications. 2023. Vol. 16, no. 2. P. 45–60. URL: https://doi.org/10.4236/jsea.2023.162003 (дата звернення: 26.05.2025).

5. Taylor K. Database Design for Web Applications. 3rd ed. New York : Springer, 2019. 180 p.

6. Lee H. et al. Responsive Design and User Experience in E-Commerce Platforms. Proceedings of the 2024 International Conference on Human-Computer Interaction, Lisbon, Portugal, 10–12 April 2024. P. 112–125. URL: https://doi.org/10.1007/978-3-031-56709-8_9 (дата звернення: 27.05.2025).

7. Davis M. Mastering Tailwind CSS: Building Modern Web Interfaces. Boca Raton : CRC Press, 2023. 150 p.

8. Patel S. et al. Security Challenges in Real-Time Collaborative Systems. IEEE Transactions on Software Engineering. 2022. Vol. 48, no. 5. P. 789–802. URL: https://doi.org/10.1109/tse.2021.3098765 (дата звернення: 26.05.2025).

9. Wilson A. Entity Framework Core in Action. 2nd ed. Manning Publications, 2021. 400 p.

10. SignalR documentation. URL: https://docs.microsoft.com/en-us/aspnet/signalr/overview (дата звернення: 25.05.2025).

11. Bootstrap documentation. URL: https://getbootstrap.com/docs/5.3/ (дата звернення: 27.05.2025).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка web-застосунку для ведення спільного списку покупок

Виконав студент 4 курсу

групи ПД-44

Рябов Артем Сергійович

Керівник роботи

доцент, кандидат технічних наук, доцент кафедри, Золотухіна Оксана Анатоліївна

Київ – 2025

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи:** спрощення процесу ведення спільного списку покупок шляхом використання сучасних веб-технологій
- **Об’єкт дослідження:** процес організації та ведення спільного списку покупок.
- **Предмет дослідження:** web-застосунок, де застосовані технології веб-розробки, інтерфейси взаємодії користувачів та засоби синхронізації даних у реальному часі.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести огляд літератури та наявних рішень у межах обраної предметної області.
2. Дослідити програмне забезпечення, яке може бути застосоване для розв’язання задач цієї галузі.
3. Сформулювати об’єкт, предмет дослідження та визначити мету кваліфікаційної роботи.
4. Створити функціональні та нефункціональні вимоги до майбутнього програмного застосунку.
5. Розробити програмний застосунок відповідно до визначених вимог.
6. Виконати тестування основних функціональних можливостей створеного застосунку.

АНАЛІЗ АНАЛОГІВ

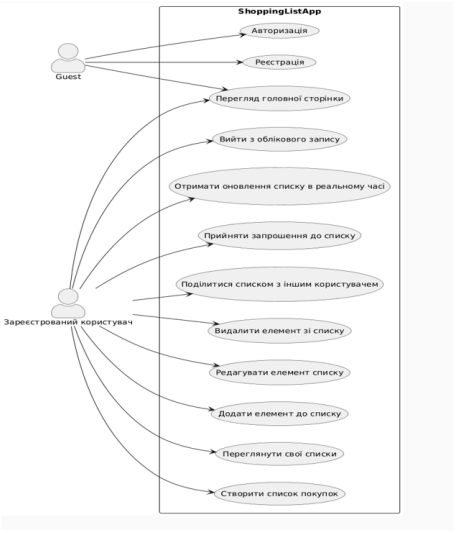
	Google Keep	Microsoft To Do	OurGroceries	AnyList	Shop Together
Спільний доступ до списків покупок	+	+	+	+	+
Управління списками покупок	+	+	+	+	+
Взаємодія зі списками покупок	+	+	+	+	+
Реєстрація та авторизація	+	+	-	-	+

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

- **Функціональні вимоги:**
- Реєстрація та авторизація користувачів: можливість створення облікового запису;
- Управління списками покупок: створення, редагування, видалення та перейменування списку;
- Взаємодія з об’єктами списку: редагування, додавання та видалення товарів, відмітка виконання;
- Спільне використання: надання доступу до списку певному користувачу.
- **Нефункціональні вимоги:**
- Зручність інтерфейсу: інтуїтивний та адаптивний дизайн для різних пристроїв (ПК, смартфони, планшети);
- Безпека: використання HTTPS, авторизація за допомогою JWT, захист від SQL-ін’єкцій через ORM.

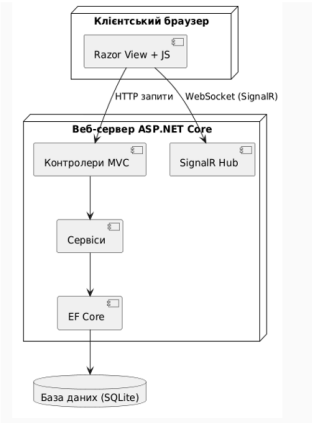
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Діаграма варіантів використання



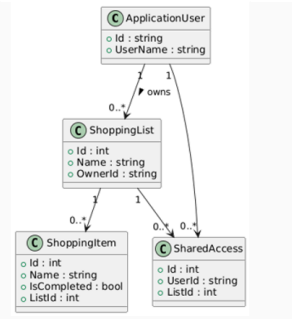
7

Діаграма розгортання клієнт-серверної моделі

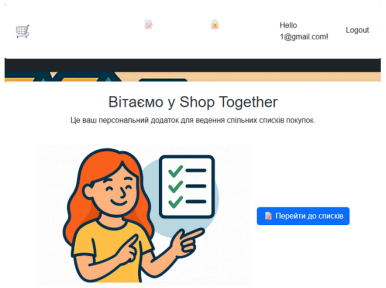


8

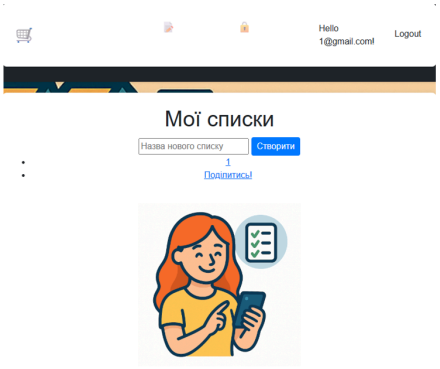
Діаграма класів бази даних



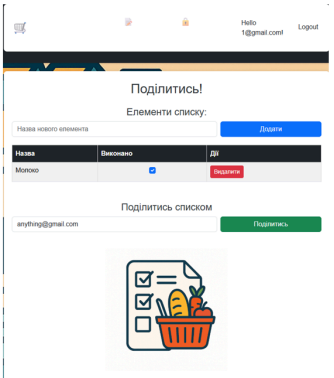
ЕКРАННІ ФОРМИ



Головний екран додатку



Екран зі списками



Екран з редагуванням та поширенням списку

ВИСНОВКИ

1. Проведено аналіз предметної галузі — спільне ведення списків покупок потребує синхронізації в реальному часі, зручного доступу та безпечної авторизації.
2. Проаналізовано існуючі аналоги (Google Keep, Microsoft To Do, OurGroceries, AnyList) та виявлено їх ключові недоліки.
3. Проаналізовано виконання функціональних та нефункціональних вимог до застосунку.
4. Обрано та використано засоби розробки: ASP.NET Core, SignalR, EF Core, SQL Server, JavaScript, Bootstrap.
5. Розроблено основні компоненти системи: моделі, інтерфейс, обробка запитів, синхронізація.
6. Підготовлено документацію для захисту дипломної роботи.

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

- Рябов А.С. ЗАХИСТ ІНФОРМАЦІЇ У WEB-ЗАСТОСУНКУ ДЛЯ СТВОРЕННЯ СПИСКУ ПОКУПОК. //XII Всеукраїнська науково-практичної конференція молодих учених «Інформаційні технології – 2025» (ІТ-2025). – Київ: КИЇВСЬКИЙ СТОЛИЧНИЙ УНІВЕРСИТЕТІМЕНІ БОРИСА ГРІНЧЕНКА, 2025. – С.320-322

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

ShoppingList.cs

```
using System.Collections.Generic;

namespace ShoppingListApp.Models
{
   public class ShoppingList
   {
      public int Id { get; set; }
      public string Name { get; set; }

      public string OwnerId { get; set; }
      public ApplicationUser Owner { get; set; }

      public ICollection<ShoppingItem> Items { get; set; } =
new List<ShoppingItem>();
      public ICollection<SharedShoppingList> SharedUsers {
get; set; } = new List<SharedShoppingList>();
   }

   public class SharedShoppingList
   {
      public int Id { get; set; }

      public int ListId { get; set; }
      public ShoppingList List { get; set; }

      public string UserId { get; set; }
      public ApplicationUser User { get; set; }
   }
}
```

ShoppingItem.cs

```
namespace ShoppingListApp.Models
{
   public class ShoppingItem
   {
      public int Id { get; set; }
      public string Name { get; set; }
      public bool IsBought { get; set; }
      public int ListId { get; set; }
      public ShoppingList List { get; set; }
   }
}
```

ShoppingListController.cs

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using ShoppingListApp.Models;
using ShoppingListApp.Services;
using System.Security.Claims;
using System.Threading.Tasks;

namespace ShoppingListApp.Controllers
{
   [Authorize]
   public class ShoppingListController : Controller
   {
      private readonly IShoppingListService
_shoppingListService;
      private readonly UserManager<ApplicationUser>
_userManager;

      public ShoppingListController(
         IShoppingListService shoppingListService,
         UserManager<ApplicationUser> userManager)
      {
         _shoppingListService = shoppingListService;
         _userManager = userManager;
      }

      private string GetCurrentUserId() =>
         User.FindFirstValue(ClaimTypes.NameIdentifier);

      public async Task<IActionResult> Index()
      {
         ViewData["PageImage"] =
"/images/shoppinglist.png";
         var userId = GetCurrentUserId();
         var lists = await
_shoppingListService.GetListsForUserAsync(userId);
         return View(lists);
      }


      public async Task<IActionResult> Details(int id)
      {
         ViewData["PageImage"] = "/images/inside.png";
         var userId = GetCurrentUserId();
         var list = await
_shoppingListService.GetListWithItemsAsync(id, userId);
         if (list == null)
            return Forbid();

         return View(list);
      }

      [HttpPost]
      public async Task<IActionResult> Create(string name)
      {
         var userId = GetCurrentUserId();
         var list = await
_shoppingListService.CreateListAsync(name, userId);
         return RedirectToAction("Details", new { id = list.Id
});
      }

      [HttpPost]
      [ValidateAntiForgeryToken]
      public async Task<IActionResult> Share(int listId, string
userEmail)
      {
         var ownerId = _userManager.GetUserId(User);
         var result = await
_shoppingListService.ShareListWithUserAsync(listId,
ownerId, userEmail);

         if (result)
         {
            TempData["Success"] = "Список успішно
надіслано користувачу.";
```

```
        }
        else
        {
            TempData["Error"] = "Не вдалося поділитися списком. Можливо, користувача з таким email не існує або ви не є власником.";
        }

        return RedirectToAction("Details", new { id = listId });
    }

    [HttpPost]
    public async Task<IActionResult> UpdateItem(int itemId, bool isBought, string name)
    {
        var userId = GetCurrentUserId();
        var success = await _shoppingListService.UpdateItemAsync(itemId, isBought, name, userId);
        if (!success)
            return Forbid();

        var item = await _shoppingListService.GetItemByIdAsync(itemId);
        if (item == null) return NotFound();

        return RedirectToAction("Details", new { id = item.ListId });
    }

    [HttpPost]
    public async Task<IActionResult> AddItem(int listId, string name)
    {
        if (string.IsNullOrWhiteSpace(name))
            return BadRequest("Назва не може бути порожньою.");

        var userId = GetCurrentUserId();

        var item = await _shoppingListService.AddItemAsync(listId, name, userId);
        if (item == null)
            return Forbid();

        return PartialView("_ShoppingItemPartial", item);
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> ToggleItemBought(int itemId)
    {
        var userId = GetCurrentUserId();
        var item = await _shoppingListService.ToggleItemAsync(itemId, userId);

        if (item == null)
            return NotFound();

        return Ok(new { item.Id, item.IsBought });
    }
  }
}
```

ShoppingListHub

```
using Microsoft.AspNetCore.SignalR;
using ShoppingListApp.Services;
using System.Threading.Tasks;

namespace ShoppingListApp.Hubs
{
  public class ShoppingListHub : Hub
  {
    private readonly IShoppingListService _service;

    public ShoppingListHub(IShoppingListService service)
    {
      _service = service;
    }

    public async Task JoinList(string listId)
    {
      await Groups.AddToGroupAsync(Context.ConnectionId, listId);
    }

    public async Task AddItem(int listId, string name)
    {
      var item = await _service.AddItemAsync(listId, name);
      if (item != null)
        await Clients.Group(listId.ToString()).SendAsync("ItemAdded", item);
    }

    public async Task ToggleItem(int itemId)
    {
      // Потрібен userId для перевірки доступу
      var userId = Context.UserIdentifier;
      var item = await _service.ToggleItemAsync(itemId, userId);
      if (item != null)
        await Clients.Group(item.ListId.ToString()).SendAsync("ItemToggled", item);
    }

    public async Task RemoveItem(int itemId)
    {
      // Потрібен userId для перевірки доступу
      var userId = Context.UserIdentifier;

      var item = await _service.GetItemByIdAsync(itemId); // потрібен додатковий метод у сервісі
      var success = await _service.RemoveItemAsync(itemId, userId);

      if (success && item != null)
        await Clients.Group(item.ListId.ToString()).SendAsync("ItemRemoved", itemId);
    }
  }
}
```

Shoppinglist.js

```
const listId = document.getElementById('listId').value;
const connection = new signalR.HubConnectionBuilder()
  .withUrl('/shoppingListHub')
  .build();

connection.on('ItemAdded', item => {
  const li = document.createElement('li');
  li.setAttribute('data-id', item.id);
  li.innerHTML = `
    <input type="checkbox" disabled />
    <span>${item.name}</span>
    <button class="remove-btn">Видалити</button>
  `;
  document.getElementById('itemsList').appendChild(li);
```

```
});

connection.on('ItemToggled', item => {
    const li =
document.querySelector(`li[data-id='${item.id}']`);
    const span = li.querySelector('span');
    span.classList.toggle('bought', item.isBought);
    li.querySelector('input[type=checkbox]').checked =
item.isBought;
});

connection.on('ItemRemoved', itemId => {
    const li =
document.querySelector(`li[data-id='${itemId}']`);
    li.remove();
});

connection.start().then(() => {
    connection.invoke('JoinList', listId);
});


document.getElementById('addForm').addEventListener('sub
mit', e => {
    e.preventDefault();
    const name =
document.getElementById('newItemName').value;
    connection.invoke('AddItem', parseInt(listId), name);
    document.getElementById('newItemName').value = '';
});



document.getElementById('itemsList').addEventListener('clic
k', e => {
    if (e.target.classList.contains('remove-btn')) {
        const id =
parseInt(e.target.closest('li').getAttribute('data-id'));
        connection.invoke('RemoveItem', id);
    }
});
fetch('/ShoppingList/ToggleItemBought', {
    method: 'POST',
    headers: {
        'Content-Type': 'application/x-www-form-urlencoded'
    },
    body: `itemId=${encodeURIComponent(itemId)}`
})
```

Program.cs

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using ShoppingListApp.Data;
using ShoppingListApp.Hubs;
using ShoppingListApp.Models;
using ShoppingListApp.Services;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<ApplicationDbContext>(opti
ons =>

options.UseSqlite(builder.Configuration.GetConnectionString
("DefaultConnection")));

builder.Services.AddDefaultIdentity<ApplicationUser>(optio
ns => options.SignIn.RequireConfirmedAccount = false)
    .AddEntityFrameworkStores<ApplicationDbContext>();
builder.Services.AddScoped<IShoppingListService,
ShoppingListService>();
builder.Services.AddSignalR();
builder.Services.AddControllersWithViews();
builder.Services.AddRazorPages();

var app = builder.Build();

using (var scope = app.Services.CreateScope())
{
    var db =
scope.ServiceProvider.GetRequiredService<ApplicationDbCo
ntext>();
    db.Database.Migrate();
}

app.UseStaticFiles();
app.UseRouting();
app.UseAuthentication();
app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
app.MapRazorPages();
app.MapHub<ShoppingListHub>("/shoppingListHub");

app.Run();
```

ApplicationDbContext.cs

```
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using ShoppingListApp.Models;

namespace ShoppingListApp.Data
{
    public class ApplicationDbContext :
IdentityDbContext<ApplicationUser>
    {
        public
ApplicationDbContext(DbContextOptions<ApplicationDbCo
ntext> options) : base(options) { }

        public DbSet<ShoppingList> ShoppingLists { get; set; }
        public DbSet<ShoppingItem> ShoppingItems { get; set;
}
        public DbSet<SharedShoppingList>
SharedShoppingLists { get; set; }

        protected override void OnModelCreating(ModelBuilder
builder)
        {
            base.OnModelCreating(builder);

            builder.Entity<SharedShoppingList>()
                .HasKey(ss => new { ss.ListId, ss.UserId });

            builder.Entity<SharedShoppingList>()
                .HasOne(ss => ss.List)
                .WithMany(l => l.SharedUsers)
                .HasForeignKey(ss => ss.ListId);

            builder.Entity<SharedShoppingList>()
                .HasOne(ss => ss.User)
                .WithMany()
                .HasForeignKey(ss => ss.UserId);
        }
    }
```