1. Формування вимог до ПЗ

Формування вимог є один із найважливіших процесів розробки програмного забезпечення. Вимоги допомагають зрозуміти потреби користувачів, визначають стандарти та очікування для системи чи програми, надають основу для розробки та управління змінами у процесі. Вони також служать критеріями для оцінки продуктивності та якості, допомагають знижувати ризики та сприяють ефективній комунікації між стейкхолдерами. Загалом, формування вимог визначає фундамент для успішної розробки продуктів та систем, враховуючи потреби і очікування всіх зацікавлених сторін.

Перший крок до формування вимог є розробка користувацьких історій.

Переглянемо отримані користувацькі історії:

1. Як користувач, я хочу мати можливість переглядати графіки з результатами продажів, щоб візуально відстежувати динаміку продажів та виявляти потенційні тенденції.
2. Як користувач, я хочу мати можливість розділяти дані на тренувальні та тестові, щоб ефективно використовувати їх для аналізу та моделювання.
3. Як користувач, я хочу бачити графік, який відображає кількість проданих одиниць товару в залежності від дати після кожного шторму, щоб визначати вплив погодних умов на продажі.
4. Як користувач, я хочу мати можливість навести курсор на графіку для перегляду конкретних дат та кількості продажів, щоб отримати детальну інформацію.
5. Як користувач, я хочу отримувати доступ до тренувальних та тестових даних для вибраного товару, щоб вивчати їх та проводити аналіз ефективності.

6. Як користувач, я хочу мати можливість сортувати дані за датою, номером магазину, кількістю продажів, номером станції, кодом погоди та опадами, щоб легко знаходити та аналізувати потрібну інформацію.
7. Як користувач, я хочу мати можливість переглядати загальну кількість продажів, щоб швидко оцінювати ефективність роботи магазину.
8. Як користувач, я хочу отримувати оцінку помилок на тренувальних та тестових даних, щоб визначити точність моделей та їхню придатність для прогнозування.
9. Як користувач, я хочу побудувати графік, який відображає залежність між кількістю днів в році та проданими товарами, щоб визначити сезонні варіації в попиті.

Завдяки даним користувацьким історіям маємо змогу побудувати use-case. Use-case діаграма використовується для створення чіткої та доступної моделі функціональності системи, що полегшує розуміння, планування та розробку програмного забезпечення. Представлено на рис. 1.

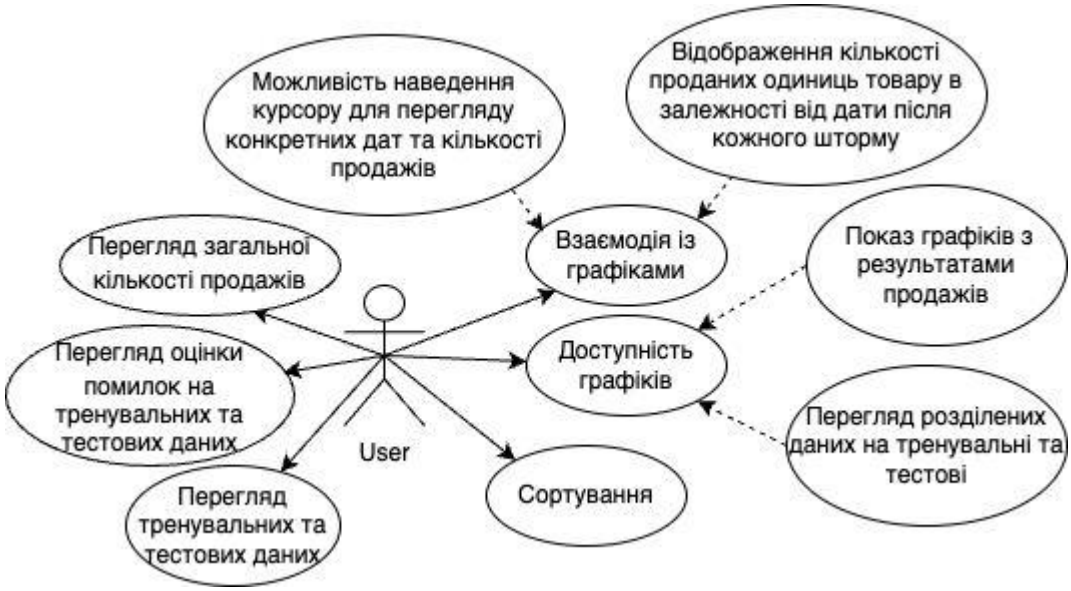

Рис. 1 – Use-case діаграма

В результаті побудови даної діаграми маємо змогу сформувати попередні функціональні вимоги, що надалі можуть уточнюватися.

Функціональні вимоги:

1. Доступність графіків:

А) Показ графіків з результатами продажів.

Б) Розділення даних на тренувальні та тестові.

2. Деталі графіків:

А) Відображення кількості проданих одиниць товару в залежності від дати після кожного шторму.

Б) Можливість наведення курсору для перегляда конкретних дат та кількості продажів.

3. Виведення тренувальних та тестових даних для вибраного товару.

4. Можливість сортування за датою, номером магазину, кількістю продажів, номером станції, кодом погоди та опадами.

5. Виведення загальної кількості продажів.

6. Виведення оцінки помилок на тренувальних та тестових даних

7. Побудова графіка залежності кількості днів в році від проданих товарів.

2. Попередня архітектура додатку на основі діаграми пакетів

Перш за все, необхідним етапом є вибір архітектури.

Серед предсталених архітектурних шаблонів, вибір було зупинено на клієнт-серверній архітектурі веб-додатку.

Клієнт-серверна архітектура є однією з основних архітектурних моделей для розробки розподілених систем. Вона передбачає взаємодію між двома основними компонентами: клієнтом, який ініціює запити, і сервером, який відповідає на ці запити. Ось декілька переваг клієнт-серверної архітектури:

1. Розподілена модель:

- Клієнт і сервер можуть розташовуватися на різних фізичних машинах чи пристроях, що дозволяє ефективно розподіляти ресурси та покращувати масштабованість системи.

2. Масштабованість:

- Ця архітектура дозволяє легко масштабувати систему, додаючи нові клієнти чи сервери в залежності від потреб.

3. Зменшення навантаження на мережу:

- Клієнт-серверна архітектура дозволяє зменшити обсяг трафіку мережі, оскільки лише необхідна інформація передається між клієнтом і сервером.

4. Можливість використання різних платформ:

- Клієнти та сервери можуть бути реалізовані за допомогою різних технологій та платформ, що дозволяє їм працювати незалежно один від одного.

5. Простота управління та підтримці:

- Розподілена природа архітектури спрощує управління та підтримку системи, оскільки розробка, тестування та впровадження можуть проводитися поетапно.

6. Забезпечення безпеки:

- Ця архітектура дозволяє легше управляти доступом до ресурсів, так як сервер може здійснювати контроль доступу та автентифікацію.

7. Можливість оновлення окремих компонентів:

- Клієнтські та серверні компоненти можуть бути оновлювані незалежно один від одного, що полегшує роботу з системами, які мають декілька клієнтів.

8. Спрощення розробки:

- Розділення функціональності між клієнтом і сервером спрощує розробку та обслуговування коду, зменшуючи залежність між компонентами.

Клієнт-серверна архітектура застосовується в широкому спектрі систем, включаючи веб-додатки, бази даних, електронну пошту та багато інших додатків.

Для формування діаграми пакетів необхідно визначитися із розроділом функціональних вимог по пакетам.

Ось функціональні вимоги, поділені на пакети за їхньою призначеністю:

1. Пакет "Відображення інтерфейсу"

- 1.1. Показ графіків з результатами продажів.

- 2.1. Відображення кількості проданих одиниць товару в залежності від дати після кожного шторму.

- 2.2. Можливість наведення курсору для перегляда конкретних дат та кількості продажів.

- 5.1. Виведення загальної кількості продажів.

- 7.1. Побудова графіка залежності кількості днів в році від проданих товарів.

2. Пакет "Відображення таблиць"

- 3.1. Виведення тренувальних та тестових даних для вибраного товару.

- 4.1. Можливість сортування за датою, номером магазину, кількістю продажів, номером станції, кодом погоди та опадами.

3. Пакет "Аналіз товару"

- 6.1. Виведення оцінки помилок на тренувальних та тестових даних.

4. Пакет "Модуль побудови графіків"

- 1.1. Показ графіків з результатами продажів.

- 2.1. Відображення кількості проданих одиниць товару в залежності від дати після кожного шторму.

- 2.2. Можливість наведення курсору для перегляда конкретних дат та кількості продажів.

- 7.1. Побудова графіка залежності кількості днів в році від проданих товарів.

5. Пакет "Модуль обробки даних"

- 3.1. Виведення тренувальних та тестових даних для вибраного товару.

- 4.1. Можливість сортування за датою, номером магазину, кількістю продажів, номером станції, кодом погоди та опадами.

- 6.1. Виведення оцінки помилок на тренувальних та тестових даних.

Така організація допомагає логічно розподілити функціональні вимоги між різними компонентами системи, що полегшує розробку та підтримку(Рис. 2).

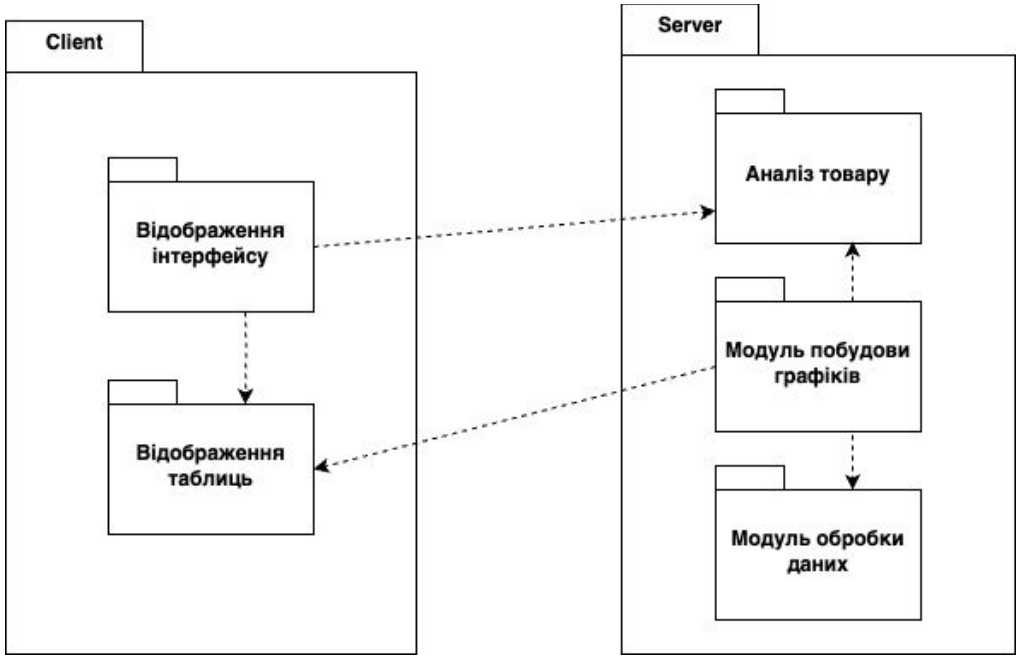

Рис. 1 – Діаграма пакетів ПЗ

На даній діаграмі читко видні залежності компонентів друг від друга та взаємодія клієнтської частини із серверною.

Проаналізуємо коректність сформованої архітектури.

Для аналізу сформуємо діаграму послідовності загальної роботи додатку. (Рис. 3)

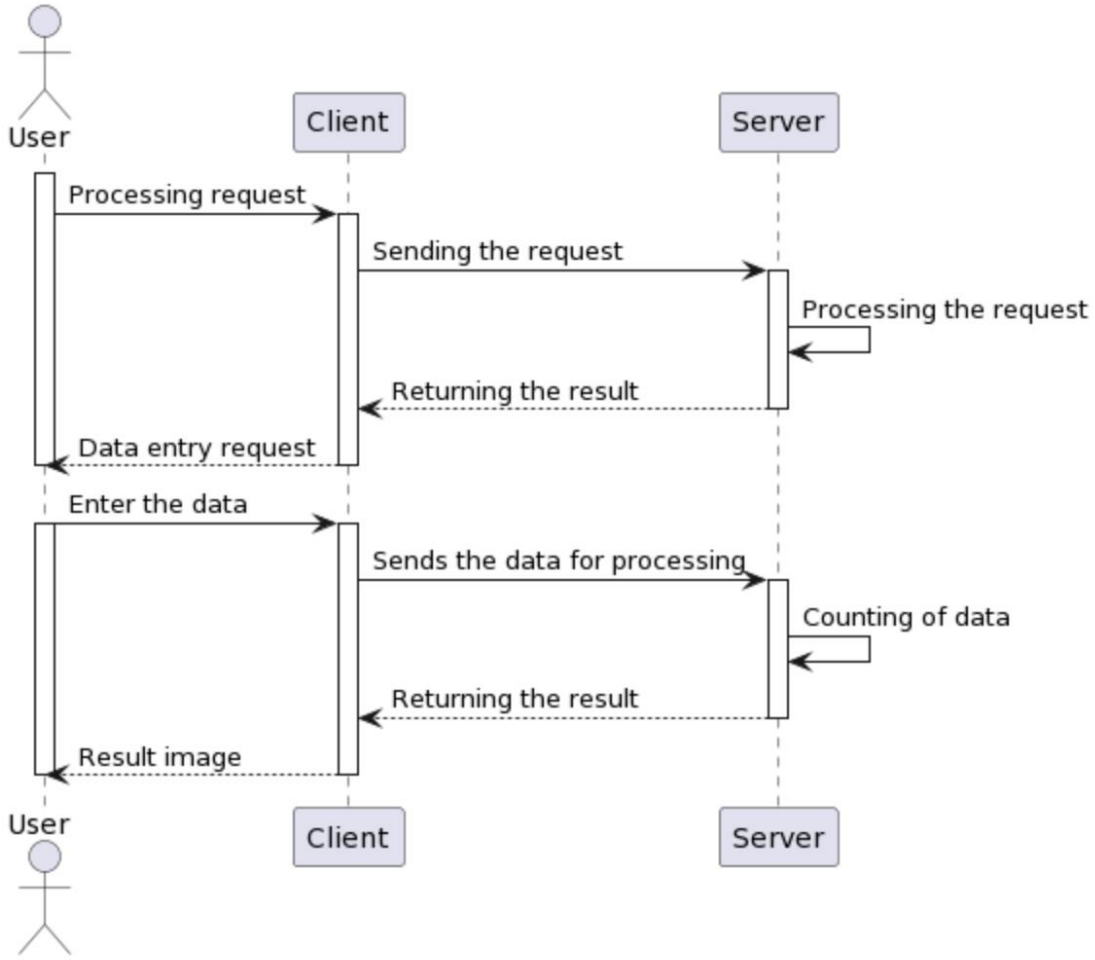

Рис. 3 – Діаграма послідовності загальної роботи додатку

Проаналізувавши дану діаграму перекриттів виявлено не було, що дає підстави зробити висновок, що архітектура була сформована коректно та уточнення не потребує.

3. Проектування користуватцького інтерфейсу ПЗ

Користувацький інтерфейс (КІ) - ключовий елемент програм та систем, який визначає місце взаємодії з користувачем. Важливість КІ полягає в декількох аспектах.

По-перше, зручність і легкість використання гарантують приємний досвід користувача. Далі, задоволені користувачі стають лояльнішими. Підвищення продуктивності відбувається завдяки ефективному інтерфейсу. Мінімізація помилок та невірних дій допомагає уникнути непорозумінь. Адаптованість до потреб користувачів та естетичний дизайн роблять інтерфейс особистим та приємним. Зменшення навантаження на підтримку виникає від інтуїтивно зрозумілого інтерфейсу. Узагальнюючи, конкурентоспроможність продукту залежить від якості його інтерфейсу. Сформуємо макет головної сторінки додатку на рис. 4.

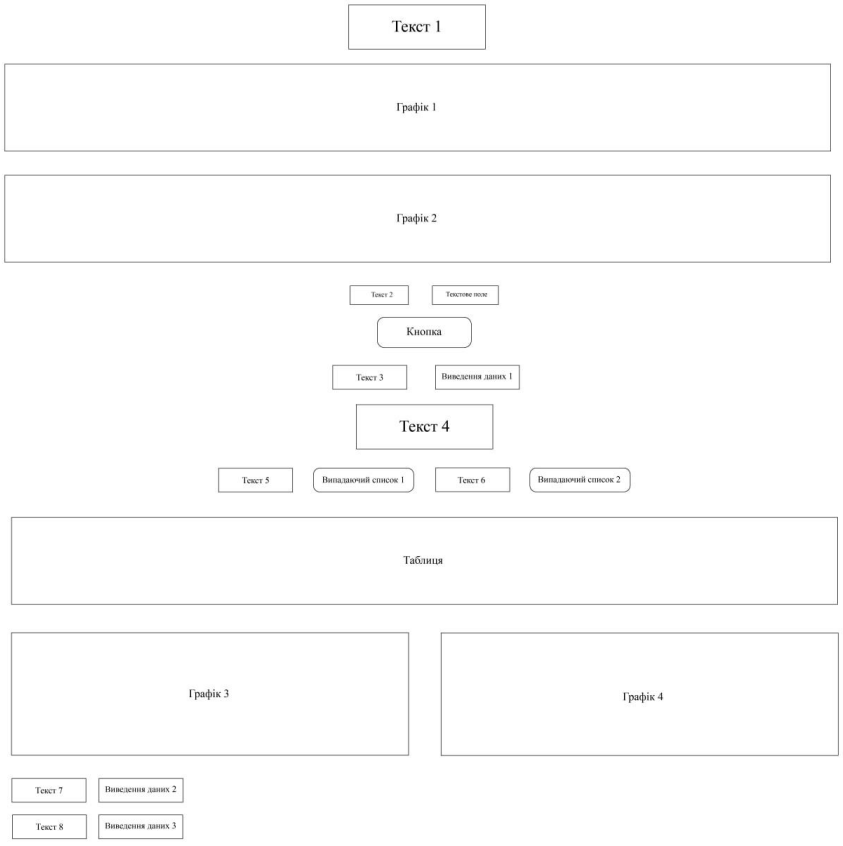

Рис. 2 – Макет головної сторінки ПЗ

На даному макеті зображено основні елементи інтерфейсу, деякі із них є динамічними, наприклад елемент «Таблиця», а також «Графік 1» та «Графік 2».

Висновок до проведеної роботи:

Було розглянуто користувацькі історії, в результаті чого було сформовано вимоги. Було побудовано use-case діаграму та проаналізовано завдяки діаграмі послідовності. Проведено розподілення вимог за пакетами та проведено попереднє проектування на основі діаграмі пакетів, використовуючи архітектуру клієнт-сервер. Виконанвши формування вимог, було сформовано макет головної сторінки додатку та передано в подальшу розробку.