Mingle Project — це розподілена вебсистема, яка складається з клієнтської частини (React), бекенду (NestJS) із мікросервісною архітектурою та централізованої бази даних (PostgreSQL).

Система забезпечує реєстрацію користувачів, створення та перегляд рецептів, коментування, оцінювання, планування меню.

Основні компоненти системи

1. Client Application
    a. Реалізує інтерфейс користувача.
    b. Взаємодіє з API Gateway через HTTPS.
    c. Використовує Redux для контролю станів застосунку.
    d. Основні сторінки:
        i. реєстрація / вхід;
        ii. список рецептів;
        iii. сторінка рецепту;
        iv. створення рецепту;
        v. меню користувача;
2. API Gateway
    a. Єдина точка входу для клієнта.
    b. Проксіює запити до відповідних мікросервісів.
    c. Виконує авторизацію (JWT).
    d. Реалізований на NestJS як Gateway Service.
3. Auth Service
    a. Керує реєстрацією, входом, виходом.
    b. Підтримує ролі користувачів: Admin, Author, User.
    c. Зберігає дані про користувачів у власній таблиці бази даних.
    d. Генерує та валідує JWT-токени.

4. Recipe Service
    a. Відповідає за операції з рецептами.
    b. Обробляє дані: опис, інгредієнти, калорійність, категорії.
    c. Підтримує пошук і фільтрацію.
5. Comment Service
    a. Обробляє коментарі до рецептів.
    b. Відправляє події у Notification Service.
6. Notification Service
    a. Відповідає за email-сповіщення.
7. Menu Planning Service
    a. Формує план харчування користувача.
8. Database Layer
    a. Основна БД — PostgreSQL.
    b. Кожен сервіс має власну схему.
    c. Використовується ORM — TypeORM.
9. Message Broker
    a. Взаємодія між мікросервісами через Redis PubSub.
    b. Підтримує асинхронну обробку подій.
    c. Підвищує відмовостійкість і масштабованість системи.
10. Logging
    a. Використовується Winston (на NestJS) для логування.

Функціональні вимоги

1. Реєстрація та автентифікація користувачів

Користувачі можуть створювати облікові записи, входити в систему, виходити та відновлювати пароль.

2. Керування ролями (адміністратор, автор, користувач)

Система має визначати права доступу відповідно до ролі користувача.

3. Створення та редагування рецептів

Автор може додавати нові рецепти із назвою, описом, інгредієнтами, інструкціями, калорійністю, часом приготування та фото.

4. Перегляд рецептів іншими користувачами

Усі користувачі можуть переглядати рецепти з докладною інформацією.

5. Пошук і фільтрація рецептів

Користувач може шукати рецепти за назвою, типом страви, інгредієнтами або дієтичними обмеженнями.

6. Система оцінок і коментарів

Користувачі можуть оцінювати рецепти і залишати коментарі.

7. Збереження рецептів у “Вибране”

Користувач може додавати улюблені рецепти до персонального списку.

8. Планування раціону

Користувач може створювати меню або план харчування на день/тиждень, використовуючи вибрані рецепти.

9. Модерація контенту адміністратором

Адміністратор має змогу переглядати, схвалювати або видаляти рецепти та коментарі, що порушують правила.

10. Система сповіщень

Користувачі отримують сповіщення про коментарі, лайки або схвалення рецептів.

Нефункціональні вимоги

1. Продуктивність

Система повинна витримувати одночасну роботу не менше 1000 користувачів без суттєвого зниження швидкодії.

2. Масштабованість

Завдяки мікросервісній архітектурі система має бути здатною до горизонтального масштабування.

3. Безпека

Дані користувачів мають бути захищені за допомогою JWT для авторизації, bcrypt для хешування паролів і HTTPS для шифрування запитів.

4. Доступність

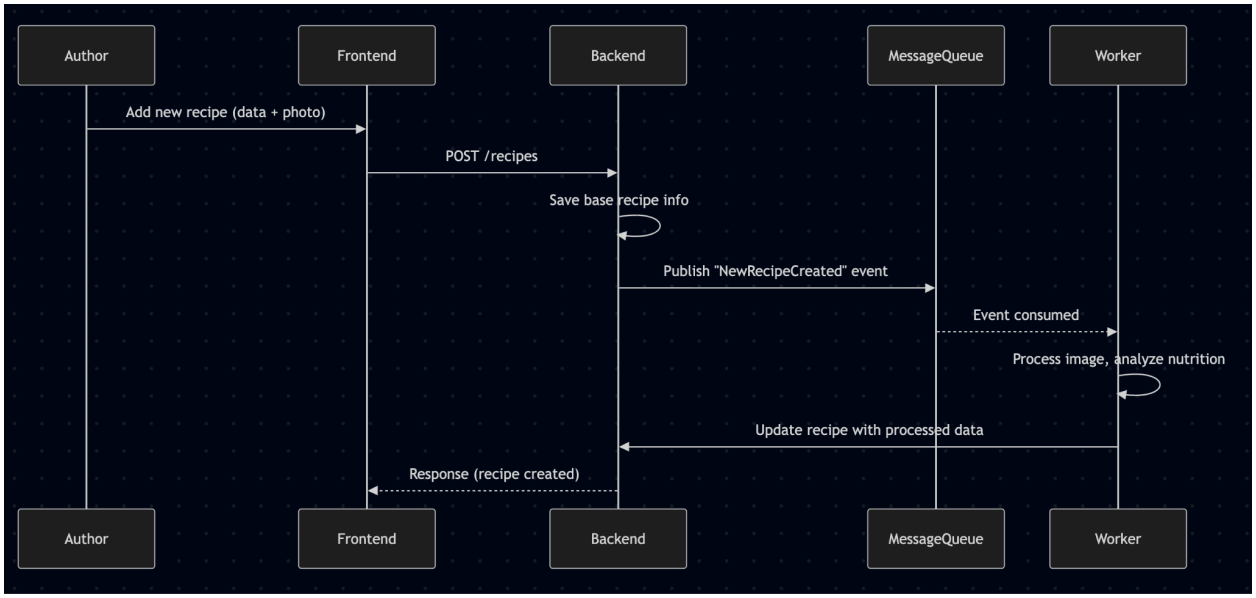
Сервіс має бути доступним 24/7 із гарантованим uptime не менше 99.5%.

5. Зручність інтерфейсу

Інтерфейс React має бути інтуїтивно зрозумілим, адаптивним і доступним для користувачів на мобільних пристроях.

Concurrency flows

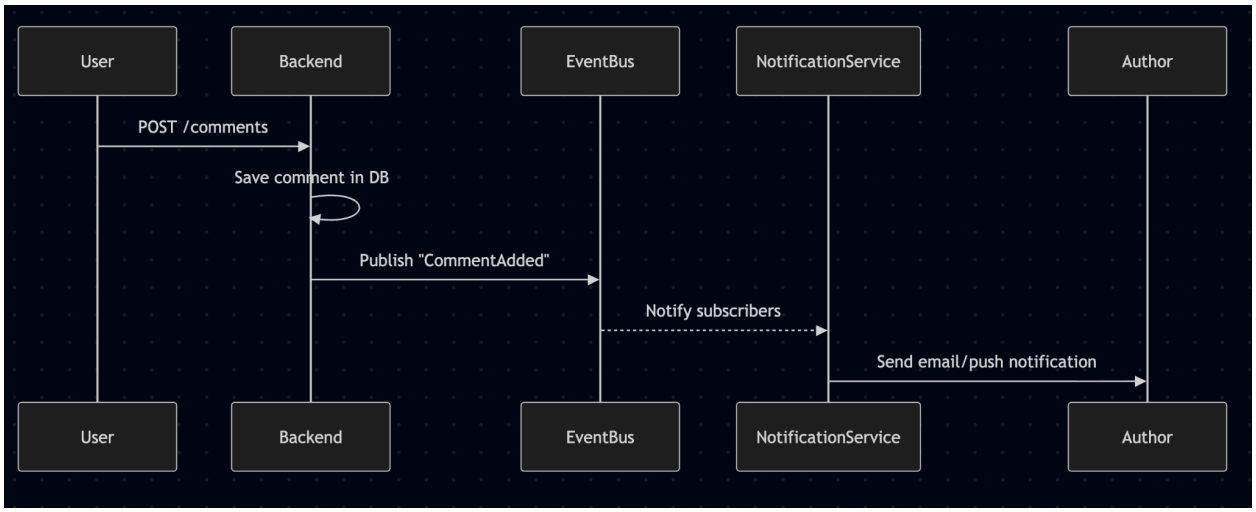
1. Asynchronous Messaging
    - Автор надсилає запит через фронтенд.
    - Backend створює запис рецепту з базовими даними.
    - Подія “NewRecipeCreated” надсилається у чергу.
    - Окремі worker-сервіси обробляють фото, обчислюють калорійність і зберігають результати.



2. Observer

Коли користувач залишає коментар, інші зацікавлені сторони (автор рецепту) отримують сповіщення.

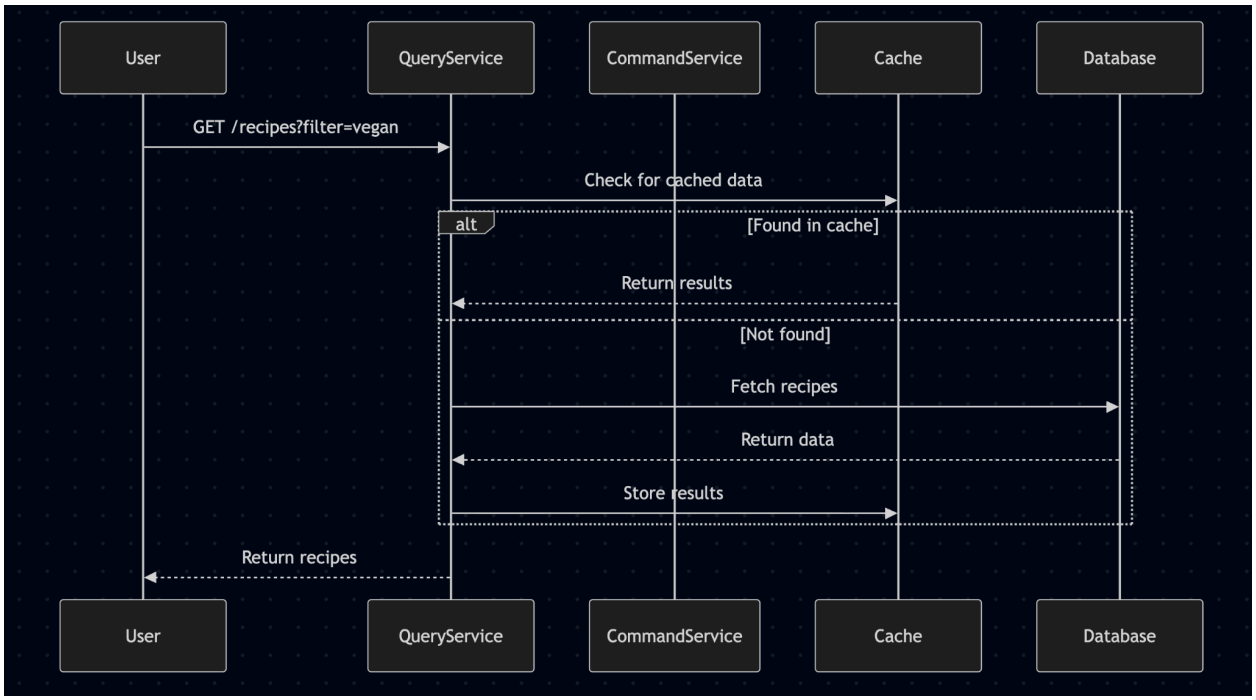
- Користувач надсилає коментар.
- Подія “CommentAdded” публікується у систему подій.
- Сервіс сповіщень отримує подію і надсилає push/email автору рецепту.



3. Command Query Responsibility Segregation

Система розділяє команди (створення, оновлення даних) і запити (пошук рецептів), щоб забезпечити швидкий доступ до даних навіть при високому навантаженні.

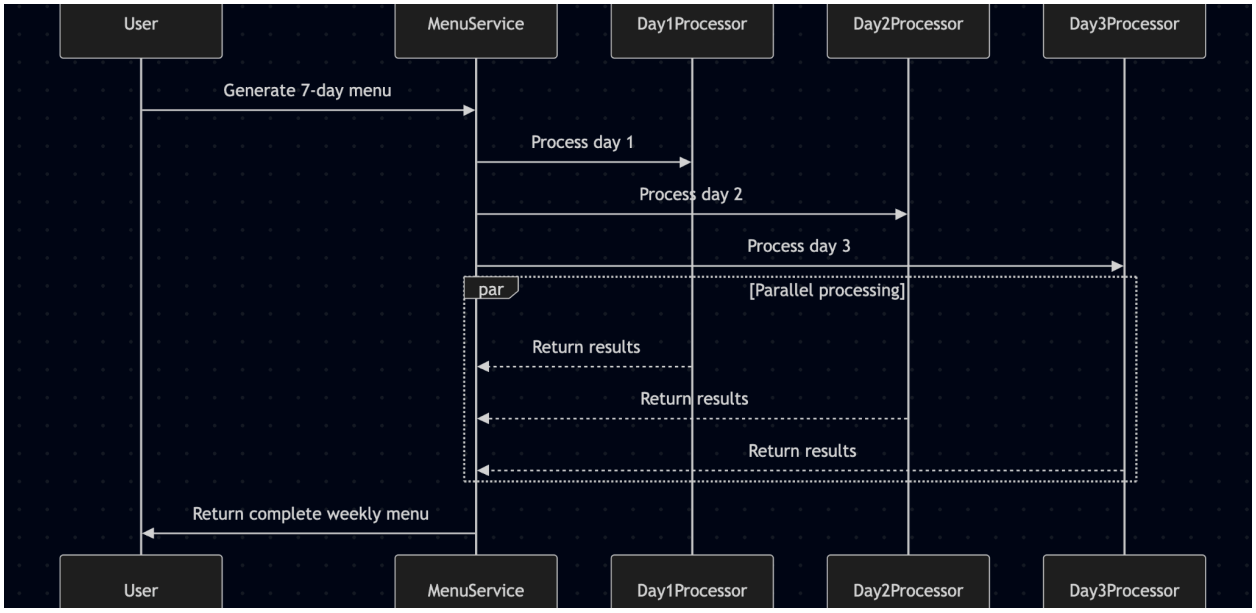
- Один сервіс відповідає за запис даних (Command service).
- Інший — за пошук і кешування даних (Query service).
- Оновлення надходять асинхронно, але користувач завжди бачить актуальну інформацію з кешу.



4. Fork/Join (Parallel Processing)

Коли користувач формує тижневе меню, система паралельно обробляє інформацію про кожен день.

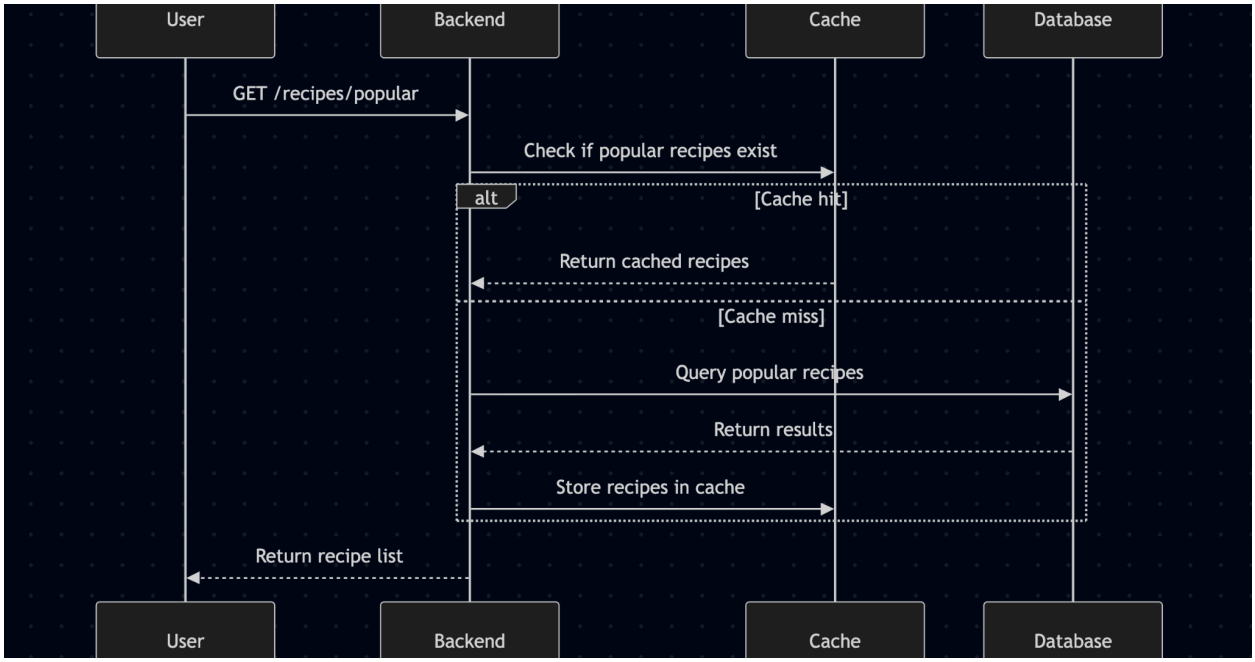
- Користувач задає параметри (кількість калорій, дні тижня, обмеження).
- Сервіс розділяє задачу на підзадачі — по одному дню.
- Обробка кожного дня відбувається у паралельних потоках.
- Результати збираються у фінальне меню.



5. Cache-Aside / Lazy Loading

Щоб зменшити навантаження на базу даних, система кешує найпопулярніші рецепти.

- Користувач запитує популярні рецепти.
- Сервіс спочатку перевіряє кеш.
- Якщо даних немає — бере з бази даних і додає в кеш.
- Наступні запити обслуговуються швидше.


User
Backend
Cache
Database
GET /recipes/popular
Check if popular recipes exist
alt
[Cache hit]
Return cached recipes
[Cache miss]
Query popular recipes
Return results
Store recipes in cache
Return recipe list
User
Backend
Cache
Database