Computer Science Department

Software Engineering & Business Analysis

Bachelor's Thesis

ReFood

Design and Implementation of a Native iOS Application for Conscious Food Choices and Waste Sorting Optimization.

Kateryna Shashkina & Yaroslava Mala

Supervisor
KSE, Ihor Pysmennyi, ipysmennyi@kse.org.ua

Expert
Company, Expert Name, expert@domain.ua

Submission date
14 August 2025

Academic Integrity Statement

I, undersigned, hereby declare that this capstone project is the result of my own work.

- All ideas, data, figures and text from other authors have been clearly cited and listed in the bibliography.
- No part of this project has been submitted previously for academic credit in this or any other institution.
- All code, diagrams, and third-party materials are either my original work or are used with permission and properly referenced.
- I have not engaged in plagiarism or any form of academic dishonesty.
- Any assistance received (e.g. from peers, tutors, or online forums) is acknowledged in the acknowledgements section.

I understand that failure to comply with these declarations constitutes academic misconduct and may lead to disciplinary action.

Place, date	Kyiv, 03.06.2026
Signature	_____

Contents

Acknowledgements

> **Individual Contribution Note:** This acknowledgements section reflects the personal academic journey and gratitude of Kateryna Shashkina.

During my studies at Kyiv School of Economics, I had the honor of meeting incredible people whose knowledge and guidance influenced my development both as a person and as a professional.

First of all, I would like to express my sincere gratitude to my Academic Director, Artem Korotenko, whose knowledge and support shaped my entire learning journey. I am extremely grateful to Artem for his work as an Academic Director, as well as for the fact that every year we had courses where he was our lecturer. These were definitely courses that I will remember for the rest of my life, and they gave me valuable knowledge that I use today and will continue to use in the future.

I would also like to express my sincere gratitude to my supervisor, Ihor Pysmennyi. I am very thankful to Ihor for believing in our project and agreeing to guide us throughout the long journey of writing this thesis. I would like to thank him for his creativity and enthusiasm. Every meeting we had regarding our project ended with our heads full of new ideas that we wanted to implement in the application. I would also like to highlight the Cloud Architecture course. During this course, I gained very useful knowledge that I had not had the opportunity to learn before, and it helped me greatly while working on this thesis.

I would also like to express my special thanks to Vadym Yaremenko and Volodymyr Skochko. The courses taught by these lecturers will always remain in my heart as some of the most interesting moments of my university life. I learned so many new and exciting things that it is impossible to describe them all in words. Whenever someone asked me which courses I enjoyed the most, I always confidently mentioned the courses taught by these lecturers. I am grateful for the knowledge, meaningful conversations, and guidance that I received from them.

Finally, I would like to thank KSE for the time, the friends, the knowledge, the connections, and the opportunities it gave me. I am grateful for every day spent on campus, for every piece of knowledge I gained during my studies, and for every wonderful and inspiring person I had the chance to meet. Kyiv School of Economics will always remain one of the best periods of my life, and I will remember it with great warmth in my heart.

Collaborative Work Acknowledgement: Finally, I would like to express my deepest and most sincere gratitude to my dear friend and teammate, Yaroslava Mala. I cannot imagine my university life without Yaroslava, and I am certain that I would not have been able to create this project without her contribution. I am truly grateful for her knowledge, hard work, dedication, and perseverance throughout our journey. Most importantly, I am grateful for her friendship. Working together on this project was not only a valuable learning experience but also one of the most meaningful parts of my university years.

> **Individual Contribution Note:** This acknowledgements section reflects the personal academic journey and gratitude of Yaroslava Mala.

Main Text

Collaborative Work Acknowledgement: I would like to express my sincere gratitude to Kateryna Shashkina .....

Abstract

Although there are countless mobile solutions in the fields of modern digital health and environmental sustainability, the market lacks a comprehensive tool that combines instant analysis of a product's nutritional value with clear instructions on how to recycle its packaging. This capstone project documents the entire design and development process of **ReFood** - an iOS mobile application that fills this fundamental gap. The product allows users to scan product barcodes to obtain clear information about their ingredients, nutritional value and an AI-generated assessment, as well as data on packaging materials and the nearest collection and recycling points. This promotes a culture of conscious consumption and simplifies the waste sorting process for every user.

Drawing inspiration from existing systems like Open Food Facts, Yuka and Junker, we identified an opportunity to create a comprehensive system amid the global trend toward a healthy lifestyle and people's desire to control the quality of their diet. Our initial concept arose from the observation that manufacturers often hide critical ingredients (such as sugar under various names) behind complex wording, marketing tactics or fine print. Unlike traditional online searches or waiting for responses from AI chatbots, ReFood provides instant interpretation of product characteristics without unnecessary scientific jargon. Accordingly, this project aimed to develop a functional mobile application that includes: (1) a barcode scanning system with instant product recognition, (2) a module for detailed analysis and "translation" of ingredient lists into a form understandable to the consumer, (3) an interactive flowchart for the proper disposal and sorting of different types of packaging, (4) a data enrichment module that allows users to add new products to the database, (5) a scalable microservices architecture capable of supporting high query processing speeds and (6) the deployment of a cloud infrastructure based on AWS services.

To achieve these goals, we conducted a discovery of the FoodTech market and an analysis of competitive solutions to validate the concept and identify functional gaps. The system was developed using the Agile methodology, structured around consecutive three-month sprints. We implemented a service-oriented serverless architecture, consisting of the following core cloud modules: User Handler for user authentication and profile management; Product Handler & AI Service for barcode scanning, AI analysis and translation; Map Service for geolocating processing points; News Service for news in the field of nutrition and products; Metrics Service for collecting user metrics. Technical implementation of the backend: a Lambda function on the Node.js (v24) platform using NoSQL DynamoDB and an S3 bucket to optimize data storage. The frontend was implemented as an iOS app in Swift using the declarative SwiftUI framework, which allowed for the implementation of modern UI/UX design principles and adaptive animation.

As a result, the project successfully created a production-ready mobile application with full functionality that meets all defined business requirements and quality criteria, and prepared for deployment to the official Apple App Store. Ultimately, the implementation demonstrates the successful application of best software engineering practices, creating an attractive alternative to existing food monitoring methods and laying a solid foundation for further expansion of the application's functionality.

Keywords:

KSE, Software Engineering, ReFood, IOS Mobile Application, FoodTech, Barcode Scanning, Nutritional Analysis, Artificial Intelligence, Waste Sorting, Food Transparency

1 Introduction

In this era of rapid digital transformation, consumers face a challenge: while technology makes it easier to access information, the process of choosing products in the supermarket that truly align with the principles of a healthy lifestyle and environmental responsibility remains unsystematized. As global food production continues to grow, misleading marketing tactics and complex ingredient lists are obscuring the transparency of product information. Consumers spend too much time decoding the small print on labels and ambiguous symbols indicating waste disposal methods. Despite advances in generative artificial intelligence, general chatbots remain too broad and require lengthy text queries to provide accurate information. This calls for an intelligent, integrated personal assistant that can instantly interpret product data and provide immediate, practical recommendations on both nutrition and sustainable waste management.

Unlike existing similar apps, which typically prioritize only one isolated function (such as purely counting a product's calories or providing general reference information about packaging), ReFood addresses the lack of a comprehensive platform that combines in-depth nutritional analysis with powerful geolocation and AI features available in both English and Ukrainian languages. The app combines elements of an interactive scanner with the AI analysis of product ingredients and the convenience of mapping services, allowing users to identify products by barcode, view ingredient lists translated into plain language without marketing jargon, identify the type of packaging material and plan optimal routes to the nearest recycling collection points. Additionally, the app provides users with news and articles on healthy eating and environment.

This paper describes the journey of two software engineering students who, over an intensive three-month development period, transformed a conceptual solution into a fully functional mobile application ready for production deployment on the Apple App Store. The development process included detailed research of the FoodTech market, competitor analysis, the design of architectural solutions and the implementation of a scalable cloud system using best software engineering practices.

1.1 Project Objectives

The primary objectives of this capstone project are:

1. To develop a fully functional iOS mobile app that facilitates the analysis of food ingredients and simplifies the waste sorting process.
2. To implement an intelligent ingredient translation system that provides instant, AI-powered analysis of composition and identifies hidden components.
3. To build reliable geolocation features that enable users to find the nearest recycling collection and processing centers, with routing capabilities.
4. To integrate with open product databases and implement a quick barcode scanning module to obtain comprehensive product metadata.
5. To build a data enrichment logic that allows users to manually add missing products and/or update outdated information, thereby expanding the database.

6. To implement a news feed with articles on healthy eating and environment.
7. To build a scalable architecture capable of supporting growth in both users and processed data.
8. To deploy a backend infrastructure using the AWS cloud and complete the deployment pipeline for the application's publication on the Apple App Store.

These goals guided our development process throughout the project lifecycle, from initial research to implementation and deployment.

1.2 Relevance and Significance

This project holds significance in several dimensions:

Technical Relevance: The development of ReFood demonstrates the application of best software engineering methods in the creation of a mobile application with a wide range of features. The project illustrates the practical implementation of a serverless microservices architecture based on AWS Lambda, comprehensive integration with diverse third-party APIs and cloud-based artificial intelligence services, as well as the development of a flexible and adaptive user interface using the SwiftUI framework.

Market relevance: Current data reveals a steep divide between consumer intentions and actions: while up to 79% of consumers prefer sustainable packaging, more than half misinterpret recycling labels, causing severe batch contamination at recycling centers. Furthermore, global applications lack localization for the Ukrainian market, failing to index local brands or address regional specificities like the absence of official Nutri-Score standards. ReFood addresses this market void by delivering real-time, localized data and an intuitive interface.

Academic relevance: This capstone project integrates knowledge from various courses in the software engineering and business analysis curriculum, including mobile development, software architecture, database design, user experience design and market research. It demonstrates our ability to integrate theoretical concepts to solve practical, real-world problems.

1.3 Methodology

Our approach to developing **ReFood** followed a structured methodology combining thorough research with agile development practices:

Discovery Phase: We conducted market research on FoodTech and eco-friendly digital solutions, analyzed competitors' platforms, identified market opportunities and defined the key requirements for the future product.

Iterative Development: The project was implemented using the Agile methodology in the form of sequential sprints, each with specific goals and defined deliverables:

Sprint 1: Interface design and basic scanning flow. The main goal of this phase was to finalize the UX/UI design and create an interactive, clickable prototype of the app. During this phase, we implemented the splash screen with user onboarding logic, basic functionality for scanning barcodes, displaying the product page and showing initial information about its recycling.

Sprint 2: Integration of the recycling map and database enrichment tools. This phase focused on expanding interactive capabilities. We developed a page for comparing multiple products and implemented the logic for users to create and edit products. In addition, we integrated an interactive map of waste sorting points with custom markers, geolocation support and a filtering system by material type. That allows us to link product scan results directly to the nearest recycling points.

Sprint 3: Users profile, advanced analytics, interactive features and finalization. The final sprint resulted in the creation of a full-scale map with detailed information about collection points and route planning to them. We implemented a user profile with secure authentication via Apple ID, a scan history and an achievement system. The product creation functionality was expanded to allow users to upload photos. We also fully localized the app and integrated an analytics event tracking system using Amplitude Tracking to monitor user behavior.

Technology Stack: We carefully selected a technology stack based on project requirements, the team's experience and industry best practices. The frontend was implemented as a mobile app in Swift using the SwiftUI framework, while the backend uses cloud-based Lambda functions built on Node.js (v24). AWS provides the microservices infrastructure and cloud storage. We selected specific services to optimize request processing speed, scalability and minimize system maintenance costs.

1.4 Structure of this paper

This thesis is structured to provide both a comprehensive technical reference and an engaging narrative of the development process:

Domain Research and Analysis (Chapter 3) examines the current FoodTech market based on a thorough analysis of competitors, global market research and the identification of functional gaps that justify the need for our comprehensive solution.

System Design and Architecture (Chapter 4) details the complete design of our product, including software architecture decisions, the selection and rationale for the technology stack, and considerations regarding user experience (UI/UX) design.

ReFood Implementation Process (Chapter 5) describes the intensive three-month development process, documenting the goals of each sprint, technical challenges, results and retrospective findings.

Validation and Testing (Chapter 6) shows how we verified that our iOS mobile app met all initial business and technical requirements.

Conclusions and Future Perspectives (Chapter 7) contains reflections on the project's final achievements, lessons learned during the engineering cycle and potential directions for future expansion of the app's functionality.

2 Research

2.1 Domain Overview

Today's eating habits harm both human health and the environment. Eating many processed foods leads to obesity and other related health problems [1]. Industrial food production also contributes to biodiversity loss, water shortages, climate change, and increasing amounts of packaging waste [1]. Plastic recycling rates remain low, ranging from 4% to 32% depending on the region. Many consumers struggle with proper waste sorting, and more than half of them do not correctly understand recycling labels. Among those who are unsure, approximately 42% dispose of packaging incorrectly, contaminating recyclable materials, while 22% do not recycle at all [2].

Despite the declaration of "green" and healthy values, actual consumer habits often do not correspond to them. Research shows that people want to have a healthy and sustainable diet, but face difficulties in choosing the right products. The reasons are lack of knowledge, small print on labels, marketing gimmicks, limited time and budget. For example, only 42% of consumers regularly buy "healthy" and sustainable products, although 79% express a desire for environmentally friendly packaging. The main barriers are high price, skepticism about marketing promises and lack of transparency about the composition and origin of products [3].

A movement for "open food data" has emerged, allowing consumers to learn the truth about products. Users want to understand exactly what they are eating and how it affects their health and the planet. This has created a demand for applications that "digitize" information from labels and make it understandable. Such services satisfy a deep demand for transparency in society: they close the gap between consumers' perceptions of "healthy" and the actual contents of a product. There is also a growing trend towards conscious consumption - people are ready to give preference to products without harmful additives, organic, in environmentally friendly packaging, etc., if they have reliable information

To help consumers act responsibly, smartphones are entering the scene. They have become "personal advisors" during shopping: they allow you to instantly obtain product information by scanning a barcode or QR code. Thus, even in a supermarket, a person can find out the composition, calorie content, allergens or ecological rating of a product in seconds, without having to decipher small print and complex chemical names [4]

To sum up, the development of technology opens up opportunities for conscious food choices. Global challenges (diabetes and obesity epidemics, plastic crisis) are driving the demand for tools that help people make healthier and more environmentally friendly choices every day. Solutions that provide transparent food information and recommendations in real time are becoming increasingly relevant.

2.2 Analysis of Existing Solutions

There are already several solutions on the market that partially cover the mentioned needs. Let's consider the main players and their characteristics:

2.2.1 Open Food Facts

Open Food Facts (OFF) is a non-profit open platform known as the “Wikipedia of food”. It has been around since 2012 and contains data on over 3 million food products worldwide. OFF allows you to scan product barcodes and receive detailed information about their impact on health and the environment. In particular, the application shows Nutri-Score (nutritional quality) and NOVA (processing level) ratings so that the user can quickly assess the nutritional value and the level of ultra-processing of the product.

Uniquely, OFF also integrates environmental indicators: it calculates the Eco-Score (eco-complex rating), which takes into account the carbon footprint, the type of packaging and its processing instructions, the presence of eco-certificates, the origin of the ingredients, etc. That is, the user immediately sees how useful the product is for him and how “green” it is for the planet. OFF is an open source and open data project: the information is populated by a community of volunteers, and manufacturers can also add their products under an open data license. This approach ensures brand independence and API availability for third-party developers

Strengths:

- Worldwide coverage
- Detailed product information
- Scientific ratings (Nutri-Score, NOVA)
- Transparent and non-profit project
- Open-source and open-data approach
- Available on multiple platforms
- Supports multiple languages

Disadvantages:

- Depends on community-contributed data
- Some local products may be missing
- Less attractive UX/UI compared to commercial competitors
- Does not provide direct recommendations or clear judgments
- Requires users to interpret the information themselves

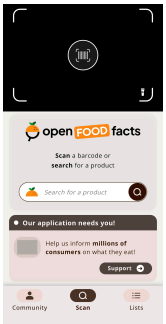

Figure 1: Open Food Facts IOS interface.

2.2.2 Yuka

Yuka is one of the most popular mobile product scanners, launched in France (2017) and currently has over 80 million users worldwide. Yuka scans both food and cosmetics, "decoding" their composition and assessing their impact on health on a simple scale. Each product is assigned a score of 0 - 100 points and a color indicator that reflects the degree of risk. The Yuka rating formula is built as follows: 60% of the score depends on nutritional value, 30% on the presence of harmful additives, and 10% on organic quality.

Yuka's interface is very user-friendly: after scanning, you can see a short verdict and a list of reasons. The application also offers healthier alternatives: if a product has received a low score, Yuka will show similar products with a better rating. For cosmetics, Yuka highlights dangerous ingredients and also gives a safety rating. It is important that the project declares 100% independence. That means that Yuka does not show advertising and does not cooperate with manufacturers, so brands cannot influence the ratings or buy promotions. Monetization is carried out through a premium subscription and the release of its own products, like a recipe book [5]

Advantages

- Simple and easy-to-understand product ratings
- Clear recommendations for consumers
- Supports healthier food choices
- Significant time savings during shopping
- Independent and advertisement-free platform
- Covers both food products and cosmetics
- User-friendly interface

Disadvantages

- Oversimplified evaluation methodology
- Product scores may not accurately reflect overall health benefits
- Limited consideration of individual dietary needs
- Strong reliance on Nutri-Score and additive penalties
- Minimal focus on environmental impact
- No dedicated eco-score or recycling recommendations
- Limited coverage of local products
- More complete database for Europe and North America
- No official Ukrainian language support

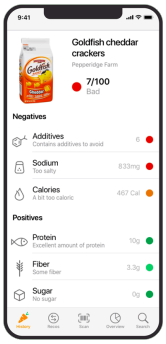

Figure 2: Yuka IOS interface.

2.2.3 Fooducate

Fooducate is one of the first mobile apps in this area (launched in 2010 in the US). It was initially positioned as a personal nutritionist on your phone. Fooducate allows you to scan products and get a letter grade from A (best) to D or F (worst) - similar to school grades. The algorithm mainly takes into account nutritional content: the ratio of calories, sugar, fats, proteins, etc. For example, fried potatoes get a "C" for high calories and low protein, while an apple gets an "A" as a fortified low-calorie product. If a product gets a bad rating, Fooducate offers alternatives - other products in the same category with a higher rating. Many users noted that this feature helped them decide what to buy instead of a harmful option, right when going to the store. Unlike Yuka, Fooducate has an advanced calorie and physical activity tracker. The user can keep a food diary, set weight goals, track calories, and BV. The application provides tips, recipes, and supports the community - there are forums where participants share progress. Thus, Fooducate is focused on long-term improvement of eating habits and weight loss [6]

Advantages

- Combines food scanning and health-tracking features
- Supports healthier eating habits and diet management
- Personalized recommendations based on user preferences
- Large community and educational nutrition content
- User-friendly interface and fast scanning

Disadvantages

- Focused mainly on the US market
- Limited support for local products and languages
- Product ratings may oversimplify nutritional value
- No environmental impact or recycling information
- Some advanced features require a paid subscription

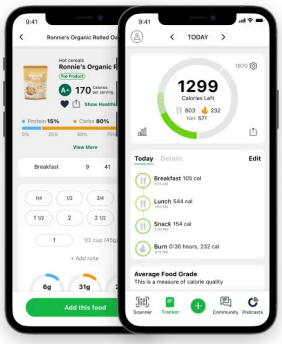

Figure 3: Fooducate IOS interface.

2.2.4 Junker

Junker is a specialized ecological application from Italy that solves the problem of "Where to throw away the packaging?" Its slogan is "virtual assistant for sorting waste". Junker scans the barcode of a product and recognizes all the parts of the packaging it consists of and the materials of these parts. Then the application clearly instructs which container each part should be disposed of in according to local recycling rules. Junker's strong point is its integration with municipalities: the application uses geolocation to provide instructions specifically for the community where the user is located (since sorting rules may differ by region). It also reminds about the days of collection of certain fractions via push notifications. Junker's database contains 1.7 million products, covering most of the Italian market. If the product is missing, the user can take a photo - and the AI algorithms will recognize the packaging [7].

Advantages

- Provides clear recycling and waste-sorting guidance
- Reduces consumer confusion about packaging disposal
- Supports environmentally responsible behavior
- Large and accurate recycling database
- Includes additional features such as waste collection schedules
- Community-driven data contribution

Disadvantages

- Focused only on recycling and waste management
- Does not provide food, nutrition, or health information
- Requires additional apps for complete product awareness
- Limited support for regions outside Italy

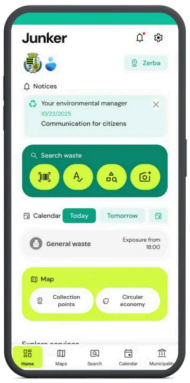

Figure 4: Junker IOS interface.

2.2.5 Comparative Analysis

Feature	OFF	Yuka	Fooducate	Junker	ReFood
Food Scanning	✓	✓	✓	✗	✓
Waste Scanning	△	✗	✗	✓	✓
Health Analysis	✓	✓	✓	✗	✓
Environmental Analysis	✓	△	✗	✓	✓
AI Explanations	✗	✗	✗	✗	✓
Ukrainian Localization	△	✗	✗	△	✓
Recycling Guidance	△	✗	✗	✓	✓

Table 1: Comparison of existing solutions and the proposed ReFood feature set.

Legend: ✓ = supported; △ = partially supported or limited; ✗ = not supported.

As shown in Table 1, all the solutions analyzed in Market Research focus on one specific goal. Open Food Facts mainly focuses on providing information about the product, while also containing additional information on how to properly sort the packaging from this product. Yuka and Fooducate mainly provide information exclusively about how a product affects health and how it is a good choice for the consumer. At the same time, Junker provides information exclusively on how to properly sort waste and dispose of packaging.

Thus, we see the problem in the fact that at the moment there is no single solution that would combine all the necessary functions and accompany the user on the entire path - from buying a product in a store to its proper disposal and sorting.

2.3 Gap Analysis

After conducting a review, we can identify needs that open up opportunities for a new ReFood product:

2.3.1 Gap 1: Lack of Combined Health and Environmental Assessment

None of the existing services provides a comprehensive assessment of both the impact on health and the impact on the environment at the same time. Consumers have to choose: either an application about the composition and calories, or separately about waste recycling. Even Yuka only partially takes into account environmental friendliness (organic), and Junker does not show anything about food quality at all. Therefore, ReFood has the potential to become an integrated solution that combines both perspectives.

2.3.2 Gap 2: Lack of Localization for Ukrainian Consumers

Global applications do not always work well with Ukrainian products: absence in the database, no translation, irrelevance of some ratings (for example, Nutri-Score is not yet officially used on packaging in Ukraine [8]). There is a demand for a solution adapted to the local market. ReFood can use OpenFoodFacts data and its own content to focus on products from Ukrainian stores, including local brands.

2.3.3 Gap 3: Limited Personalization and AI-Based Recommendations

Existing apps mostly provide static information or assessments based on a fixed formula. However, different users have different needs: allergy sufferers need to filter out certain ingredients, athletes need to see proteins and calories, and eco-activists need to minimize plastic. Some apps (OFF, Fooducate) allow for some customization, but there is no flexible tool that would explain the ingredients specifically for the user's needs. Modern technologies like GPT allow you to do exactly that - generate explanations or recommendations based on ingredients. None of the analyzed solutions provide AI explanations as a core feature. Therefore, ReFood has a chance to implement an intelligent assistant.

2.3.4 Gap 4: Absence of Integrated Recycling Guidance

Even when the packaging says it is recyclable, it may be unclear to the consumer where to take this container. Junker solves this through local instructions, but there is no single solution globally. An open map based on OSM data shows that there are actually hundreds of thousands of recycling collection points around the world [9]. However, most people do not know about them. By implementing the map in ReFood, you can give the user a final hint about where to take the container. Thus, ReFood will close the cycle: from choosing a product to proper disposal.

2.4 Chapter Conclusion

Conducted market research and gap analysis demonstrate that currently existing solutions cover only one consumer need at a time - either health care or environmental care. None of the analyzed solutions combines all these aspects in one application.

As a result, users have to switch between several applications to get complete information about the product: starting with the right choice on the store shelf and ending with how to properly dispose of the packaging without harming the environment. This gap creates an excellent opportunity for the development of ReFood - a smart assistant in the pocket that will accompany the user at every stage of the product's life cycle.

3 System Design

3.1 System Architecture and Design Rationale

Based on the results of our marketing research and gap analysis, we identified a number of important user needs that remain unmet by existing solutions on the market. These requirements formed the basis for developing key functional and non-functional requirements for the ReFood app.

Our primary goal was to create a scalable and maintainable solution that would combine the propper food selection and recycling process within a single platform.

The following sections detail how the identified functional and non-functional requirements were translated into specific engineering solutions and influenced the product's architecture and implementation.

3.1.1 Mapping Requirements to Architectural Components

Functional Requirements and Corresponding Architectural Decisions

- **Scanning and Manual Entry**: The system must allow users to scan barcodes or manually enter them to instantly retrieve all the data about the product (Nutri-Score, Eco-Score, ingredients, allergens, nutrients) and an provide an AI analysis of the fetched product.

  – Architectural Decision: We utilize a protocol-oriented AVFoundation service for high-performance barcode decoding, coupled with a repository-based API layer to fetch and process product data.

- **Recycling Navigation and Map Integration**: Users must be able to view packaging components, transition to a "How to sort" flow and locate nearby recycling points filtered by material (e.g., plastic, glass). The map must support routing, "search this area" functionality and default locations if geolocation is disabled.

  – Architectural Decision: We integrated Apple's MapKit framework on the frontend to manage client-side rendering, visualization grouping and searches outside a specific area. The backend delegates computing workloads onto the transient **`MapService`** Lambda, which interacts with Geoapify APIs on demand to compute real-time routing structures without introducing local data storage overhead and to comply with API usage rules.

- **User Dashboard and History Management**: The Home screen must display dynamic content (Eco Tip, News, recent scans, current statistics), while the Search screen provides a complete history with filtering (all/favorites) and like/dislike capabilities

  – Architectural design: The server-side handles content generation asynchronously by scheduling daily Amazon EventBridge events that fill the **`News`** table in DynamoDB. Scanned products are stored in a separate **`Scans`** table and are recorded with every product search. The **`ProductsFavorites`** table is filled separately when the like button is clicked on the client. In this way, we divide the tables into separate logical components that are independent and easy to modify and scale.

- **Seamless Authentication and Data Portability**: Users must be able to log in via Apple ID to securely save their data. If the app is deleted and reinstalled, the user's entire scan history, favorites, and statistics must be fully restored upon login.

  – Architectural Decision: We implemented a dual-identity model using the AWS Cognito Identity Pool for temporary anonymous device tracking and the AWS Cognito User Pool for authentication via Apple ID. The backend service uses a merge logic to combine the device associated history (anonymous user) with the persistent Apple credentials (registered user) without data loss.

- **Gamification & Progress Tracking**: The system must track user interactions, displaying a daily streak, scanned/sorted counts and an Achievements tab with 8 distinct unlockable milestones.

  – Architectural Decision: User metric updates are processed in an asynchronous way that managed by a dedicated Lambda function - **`MetricsService`** to avoid blocking core user interactions. This service analyzes incoming action signals, calculates and updates data in the **`UserMetrics`** table, and evaluates achievements based on a static configuration of achievements.

- **Community Crowdsourcing (Add and Edit)**: If a product is missing or contains incorrect/poor-quality data (e.g., bad photos), the user must be able to submit new information or edits via a form for backend validation.

  – Architectural Decision: We implemented an AI validation process that runs when a user fills out a form, using **`AIService`** to check for suspicious, unusual or invalid entered data. Attached images are transferred to a temporary S3 bucket, which publishes **`ObjectCreated`** triggers to EventBridge, which invokes the AI before the records are moved to the public bucket. Thus, the AI result for the provided image is stored for 24 hours in **`UploadJobs`** and are pinged by the client until a REJECT or APPROVED status is received.

- **Smart Product Comparison**: The app must support a "Compare Mode" where a user can scan a second product, confirm the comparison, and view a side-by-side UI of nutrients. An AI analysis must determine and explain the "winner."

  – Architectural Decision:мThe comparison mechanism uses **`AIService`**, which sends comparison requests via Vertex AI (Gemini LLM), compares the features of two products and returns the comparison results to the client.

Non-Functional Requirements and Corresponding Architectural Decisions

- **Availability and Offline Data**: Users frequently experience poor cellular network connectivity inside supermarkets. The application must provide continuous access to previously scanned items, favorites, and gamification progress regardless of the current internet connection.

  – Architectural Decision: To ensure uninterrupted access to data, an "Offline-First" approach has been implemented using the native SwiftData framework. All scanned products and interaction history are immediately stored in the device's local database using **`@Model`** macros. The user interface is subscribed to local data via the **`@Query`** macro.

This means that the app always reads data from the phone's memory (which works instantly and without an internet connection), so the user can always view their scan history even in "airplane mode".

- **Performance and UI Responsiveness**: The mobile application must maintain a highly responsive interface (targeting 60 FPS) and avoid any freezing or UI blocking during heavy API requests, AI processing or data synchronization.

  – Architectural Decision: To prevent the interface from freezing while loading images or waiting for a response from the backend, several architectural decisions were made. All network requests, JSON parsing and image processing are implemented using the async/await mechanism. This allows "heavy" tasks to be executed in background threads, completely freeing up the UI thread. The `@MainActor` attribute is used for ViewModel classes. This ensures that any state change (e.g. displaying a loading indicator) that affects the UI is safely executed exclusively on the main thread, preventing crashes and animation stuttering.

- **Security and Data Privacy**: Personal user data, scan history and authentication tokens must be strictly isolated, securely transmitted and protected against unauthorized access.

  – Architectural Decision: All data transmission from the client to the server is secured using the HTTPS protocol at the AWS API Gateway level, which validates short lived Cognito authentication tokens. DynamoDB tables and S3 buckets are controlled by detailed access policies through the appropriate IAM policies to restrict data access and mitigate the risk of data leaks.

3.2 C4 Context Diagram

The system context diagram illustrates the ReFood app's position at the highest level of abstraction and its interaction with the external environment - users and third-party integration services (APIs). The diagram highlights key external entities that interact with the system, including users who scan products, view recycling information and manage their profiles, as well as third-party services such as product API, geolocation API, scientific research API and AI-based data processing API. This contextual overview provides a clear understanding of the system's scope of application and its interaction with external entities.

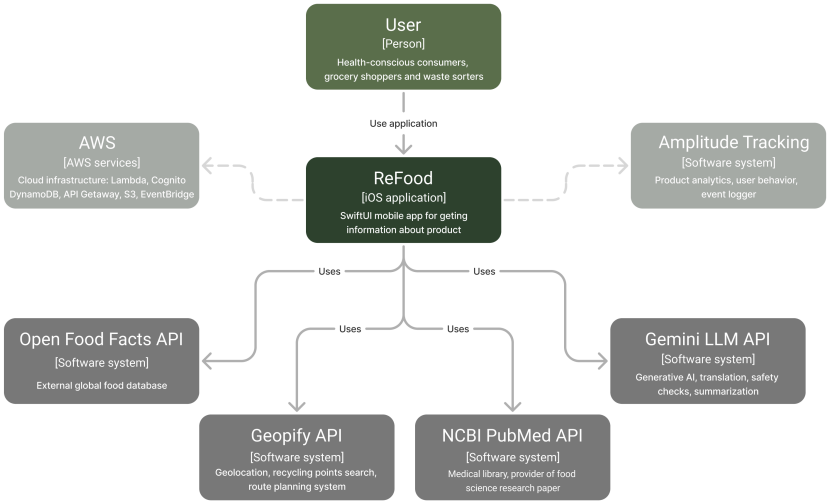

Figure 5: System Context Diagram - ReFood App Ecosystem

Justification for the selection of external integrations and analysis of alternatives

To ensure the application'ss business functionality we conducted an analysis of available solutions and selected the following external systems:

- **Open Food Facts API**
  - **Selected option:** The Open Food Facts (OFF) API - a global open crowdsourced database for barcode recognition, retrieving Nutri-Score and Eco-Score ratings, product ingredients and detailed packaging information. This is how we populate our database, ensuring we have the latest product data and integrating both local and outsourced sources.
  - **Alternatives:** Commercial product catalogs (Barcode Lookup API, Go-UPC API) or building a custom database from scratch by parsing online supermarket data.
  - **Comparison:** Commercial alternatives have strict limits on free queries (mostly paid commercial plans) and a small database of Ukrainian local brands. Building our own database from scratch at the start of the project would have resulted in a persistent **`404 Not Found`** error for 95% of products. In contrast, the Open Food Facts API is completely free, supports millions of products from around the world (including the Ukrainian market) and allows us to use the "Cache-Aside" architectural pattern. If a product isn't in our DynamoDB database - we fetch it from OFF, localize it using AI and store it locally in database, thus expanding our own product catalog.
- **Geoapify API**
  - **Selected option:** Integration of the Geoapify API to enable real-time search for map points, filtering them by material type and the creation of walking, cycling or driving routes.
  - **Alternatives:** Direct fetching and parsing of raw geodata from OpenStreetMap (OSM) or deploying a custom server based on OSRM (Open Source Routing Machine).
  - **Comparison:** Direct data extraction from OSM would have required writing complex, rather heavy queries on the Lambda side, which led to significant response delays and user interface freezes. Initially, we tried using direct fetching, but this

held our system back in development. It also complicated the logic for storing and building routes, which led to increased product maintenance (which we cannot afford at the MVP stage). Therefore, working with geodata directly forced us to rethink our approach to constantly updating coordinates and our limited AWS cloud budget. Instead, we found a suitable alternative that we are currently using - the Geoapify service. It provides a ready to user, fast fetch (response < 200ms), handles coordinate validation and builds accurate route graphs. Thus, at this stage of product development, we can provide users with a convenient, fast interface that displays processing points and routes to them.

Geographic Data Storage and Licensing Compliance: Under the Geoapify API license agreement, caching or persistently storing geographic coordinate points, spatial indexes and routing details in a local database is explicitly prohibited. The ReFood architecture strictly enforces this compliance by treating responses as transient data: they are fetched on-demand, processed, transformed in memory and immediately transmitted to the client. Additionaly, this architectural design eliminates geospatial table maintenance and reduces DynamoDB storage requirements.

- **Gemini LLM API**
  - **Selected option:** Using Gemini models via the API to translate complex ingredients into understandable language, detect hidden sugars, compare two products and protect the system against prompt injection and inappropriate content when users add products.
  - **Alternatives:** OpenAI services (GPT-4o) or our own LLM.
  - **Comparison:** OpenAI (GPT) has a higher token cost, which is critical for us given the limited budget for our diploma project. Deploying our own LLM model would require constantly running server resources and thus, the maintenance cost would exceed several hundred dollars per month. The Gemini service provided the best free request limit for development, high speed generation of structured JSON response, processing of both text and images, and excellent performance with Ukrainian language.
- **NCBI PubMed API**
  - **Selected option:** Integration with the official API of the U.S. National Center for Biotechnology Information for the daily automatic collection of the latest medical and nutritional articles.
  - **Alternatives:** Manually populating the dashboard with articles from the internet.
  - **Comparison:** Manual data entry contradicts the principles of software engineering automation. Additionally, unverified data obtained from the internet or AI chatbots would significantly undermine trust in the application. In contrast, the PubMed API provides a standardized, validated and strictly structured XML data format. This allowed us to implement a fully autonomous service: a Lambda function retrieves pure scientific facts once a day (at nignt), passes them to AI to filter out complex medical terminology and displays verified information to the user in the feed without any human intervention. We also provide a direct link to the article, which increases user trust.
- **Amplitude Tracking**

- **Selected option:** Amplitude, a platform for collecting customer metrics and behavioral events in real time.
- **Alternatives:** A custom analytics table in DynamoDB or Firebase Analytics integration.
- **Comparison:** Firebase Analytics is geared toward general marketing and the Google ecosystem, which makes it difficult to build custom conversion funnels for SwiftUI interfaces. Recording every user click in a custom DynamoDB table would place a massive financial burden on the database due to the high frequency of write operations. Amplitude provides a ready to use SDK for iOS that bundles events for free, sends them in the background without taxing the device's CPU and gives us dashboards for UX analysis without having to write visualization code.

3.3 C4 Container Diagram

The container diagram details ReFood's high-level architecture, dividing the system into logical functional blocks that are deployed independently. This level of architectural design reflects the division of responsibilities among the mobile client, the serverless cloud infrastructure and data storage systems, and defines the technical protocols for their interaction.

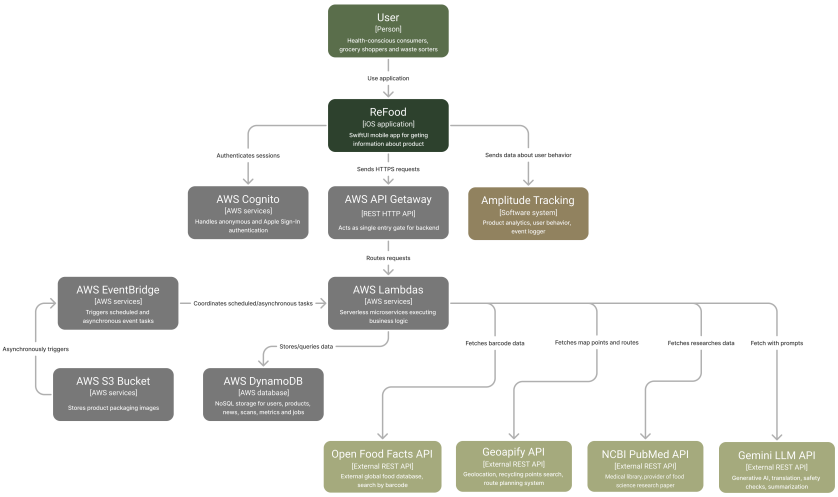

Figure 6: System Context Diagram - ReFood App Ecosystem

Rationale for choosing a cloud infrastructure and analysis of technical trade-offs

The serverless computing pattern formed the basis of the architectural choice for the ReFood backend platform. The main goal of this solution is to ensure linear scaling of the system in line with user request intensity, completely eliminate financial costs for maintaining infrastructure during the periods of low activity (we pay only for actual usage) and minimize operational overhead.

- **AWS API Gateway**
  - **Selected option:** REST HTTP gateway that serves as the single entry point for the mobile app, providing CORS validation, token authorization and request routing.

  - **Benefits:** AWS API Gateway integrates with the rest of the AWS servvices, automatically scales to handle millions of requests and charges only for calls actually processed.

- **AWS Lambda**
  - **Selected option:** A serverless code execution environment (Node.js v24) in the form of isolated microservices that are triggered in response to events from API Gateway, EventBridge or a direct invocation Lambda.
  - **Alternatives:** Traditional monolithic or containerized architecture running on AWS EC2.
  - **Comparison and benefits:** Classic servers run 24/7, which means a constant cost for computing power, even when no one is using the application (for example, during periods of lowest demand - at night). Using AWS Lambda allowed us to break down the logic into atomic, independent functions (`UserHandler`, `ProductHandler` and et.). This ensures fault isolation (if the news service goes down, the scanning continues to work) and automatic horizontal scaling of each function separately.
- **AWS DynamoDB**
  - **Selected option:** A fully managed key-value NoSQL database (DynamoDB) that delivers consistent response times of a few milliseconds at any scale.
  - **Alternatives:** Relational databases such as PostgreSQL or MySQL.
  - **Comparison and benefits:** Relational databases have limited horizontal scaling capabilities and require complex replication and connection pooling mechanisms. Product data in ReFood (ingredients, allergens, nutrients and etc.) has a flexible JSON structure from Open Food Facts, which fits perfectly into NoSQL databse without the need to write heavy schema migrations. Since we use composite keys and global secondary indexes, DynamoDB allows us to instantly retrieve product and user profile data. Therefore, it showed us high read speeds at a fixed cost.
- **AWS S3 Bucket and AWS EventBridge**
  - **Selected option:** Cloud object storage (S3) for uploading photos of packages that integrated with EventBridge to manage asynchronous events.
  - **Alternatives:** Storing images as Blob arrays directly in the database or synchronously processing uploaded photos via API Gateway.
  - **Comparison and benefits:** Storing binary files in the database would quickly exhaust its limits and slow down indexing. Synchronous image verification via AI during an HTTP request would force the user to wait for a response for more than 5-10 seconds, which ruins the UX and lead to an API Gateway timeout. The combination of S3 and EventBridge allowed us to implement an asynchronous pattern. The user uploads a photo directly to the S3 bucket via a pre-signed URL -> the bucket generates an `ObjectCreated` event -> EventBridge instantly intercepts it and triggers a background Lambda function for AI verification. The user receives a response based on the JobStatus (the AI response, which we store separately).
- **AWS Cognito**

- **Selected option:** An authentication service (Cognito) that provides Identity Pools for securely managing temporary IAM roles for anonymous devices and User Pools for verifying Sign-In with Apple tokens.
- **Alternatives:** Building a custom JWT-based authentication service, storing hashed passwords in a database and manually refreshing session tokens.
- **Comparison and benefits:** A custom security implementation carries critical risks of data leaks and requires significant effort to meet Apple ID security standards. AWS Cognito handles cryptographic verification of Apple signatures, token lifecycle management and account isolation. This allows the application to recognize access rights for both temporary anonymous device sessions and JWT tokens for authorized users.

3.4 C4 Component Diagram

The component diagram illustrates the highest level of detail in ReFood's serverless backend. This level of architectural description shows how the overall business logic is broken down into atomic, independent Lambda functions (considered microservices) and how they interact with other system components, including database tables and integration APIs.

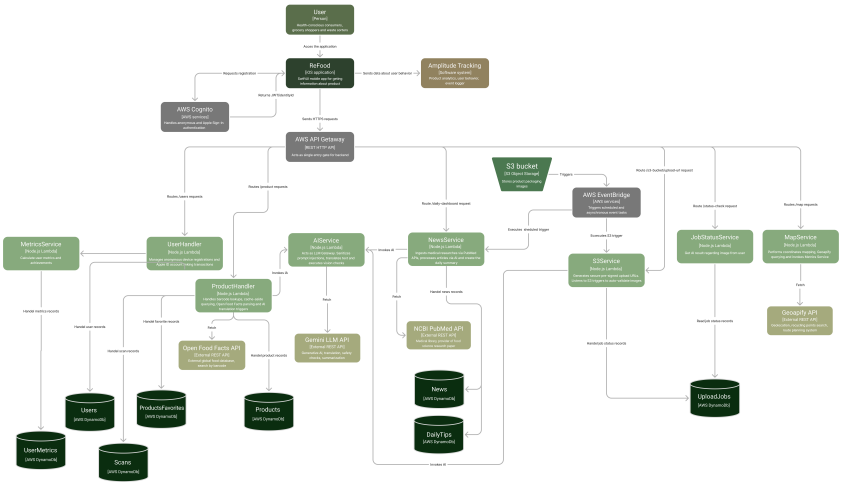

Figure 7: System Context Diagram - ReFood App Ecosystem

Functional description of system components and the logic of their interaction

ReFood's backend architecture is built on the Single Responsibility Principle and context isolation, where each microservice is represented by a separate Lambda function that has access only to the resources that it needs:

- **UserHandler**
  - **Purpose and relationships:** Handles initial anonymous device registration, updates session ID and manages logic for linking anonymous data to a permanent account when switching to an Apple ID. Has direct access to the `Users` table for storing profiles. This component also interacts with `MetricsService` to display user achievement statistics.

- **ProductHandler**
  - **Purpose and relationships:** Implements the logic for processing barcode scans and manual searches. It follows the "Cache-Aside" pattern: fi it checks for the code in the local **`Products`** table -> in case of a cache miss, it makes a request to the Open Food Facts API -> initiates an internal synchronous call to **`AIService`** for translation and analysis -> writes the final result back to **`Products`**. It implements product comparison through AI analysis of each product. It also handles the addition of a new product by a user input, passes it to **`AIService`** for processing and, based on the result, either saves or prevents the saving of the new product. In addition, the component logs every scan in the **`Scans`** table and manages product likes and dislikes via the **`ProductsFavorites`** table.
- **MapService**
  - **Purpose and Interactions:** Processes client requests to search for waste sorting points and build routes. It validates the user's coordinates and dynamically retrieves data from the **`Geoapify API`**.
- **NewsService**
  - **Purpose and relationships:** A separate component that does not depend on direct user requests for generating news. It runs once a day on a schedule via the **`AWS EventBridge Scheduler`**. The service makes a request to the **`NCBI PubMed API`**, parses data and filters out duplicates already stored in our database. The cleaned data is sent for processing to **`AIService`** and tobtained results are recorded in **`News`** table.
- **AIService**
  - **Purpose and connections:** An isolated component with no direct connection to the API Gateway. It acts as an internal proxy for interacting with the **`Gemini LLM API`**. Three lambdas call it synchronously: **`ProductHandler`** (for translating and analyzing product descriptions, as well as validating user text input), **`NewsService`** (for simplifying articles) and **`S3Service`** (for visually checking images uploaded by users).
- **S3Service and JobStatusService (Checker)**
  - **Purpose and relationships:** Manage the lifecycle of photos that users upload when adding new products. When a client uploads a file, **`S3 Bucket`** generates an **`ObjectCreated`** event which **`AWS EventBridge`** forwards to **`S3Service`**. This function registers a new job in the **`UploadJobs`** table and triggers an analysis in **`AIService`** to check the photo. The result is recorded in **`UploadJobs`**. The **`JobStatusChecker`** component receives requests from the mobile app, allowing the iOS client to poll for the readiness status and approve the product addition.
- **MetricsService**
  - **Purpose and relationships:** Collects analytics events from other lambdas (such as successful scans or routes showing) and updates counters, daily streaks and user eco-achievement statuses in the **`UserMetrics`** table.

3.5 Technology Stack Summary

Component	Technology	Justification	Trade-offs
Frontend	Swift / SwiftUI	Native iOS compilation	Platform exclusivity (iOS only)
Backend	Node.js v24	Team expertise, performance	Not for complex calculations
Serverless Infrastructure	AWS Lambda	No cost for downtime, easy linear scaling	"Cold start" effect, time limit on function execution
API Routing	AWS API Gateway	Managed HTTPS proxy, built-in JWT and CORS validation	Cloud provider dependency
Authentication	AWS Cognito	Built-in support for Apple ID and anonymous device sessions	Complex data migration in future
Database	AWS DynamoDB	Fast response, flexible NoSQL schema	No JOIN queries, requires early index planning (GSI)
File storage	AWS S3 Bucket	Low storage cost, Object-Created event generation	Requires implementation of pre-signed URL logic on the client
Product Data	Open Food Facts API	Free access, millions of products, support for the Ukrainian market	Unreliable quality of crowdsourced data
Generative AI	Gemini LLM API	Generous developer free tiers, low maintenance costs	Vendor dependency, unstable latency under load
Geodata	Geoapify API	Fast route calculation, large data volume	Need for constant monitoring of free request quotas
Research Data	NCBI PubMed API	Authoritative medical database, standardized catalog	Results returned in outdated XML format
Analytics	Amplitude SDK	Client-side event batching, built-in funnels	Paid premium plans for complex tracking systems

Table 2: Technology Stack Justification

4 Implementation

4.1 Development Methodology and Team Organization

4.1.1 Agile Development Approach

The ReFood development followed the Agile methodology, organized into three sprints, each lasting one month. Given the small team size, we chose Agile over the formal Scrum approach, which in allowed us to quickly adapt to changes in the project and implement the feedback received into the app.

Sprint Organization

- Each sprint began with planning and prioritizing tasks for the following month using the RICE framework.
- During the sprint, we held meetings every two days to determine whether we were on track and to identify potential issues and areas where additional support was needed.
- At the end of each sprint, we received feedback from our supervisor and conducted a retrospective analysis of the previous sprint.

Feature	Reach	Impact	Confidence	Effort	RICE Score
Product Comparison Flow	9	8	9	4	162
Recycling Map Flow	8	9	7	7	72
End-to-End Product - Recycling Flow	7	8	8	5	90
Add/Edit Product	5	6	8	4	60
Backend Refactoring	4	7	8	5	45
User Profiles & Authentication	6	8	7	8	42

Figure 8: Sprint 2 feature prioritization using the RICE framework.

Team Responsibilities Distribution:

- **Yaroslava Mala:** Responsible for the entire serverless backend architecture, AWS cloud engineering and API integrations. This included developing all AWS Lambda microservices, configuring NoSQL DynamoDB database, managing Amazon S3 bucket triggers and storage layout parameters, API gateway logic and executing third-party API integrations (Open Food Facts endpoints, Geoapify, PubMed and AI model).
- **Kateryna Shashkina:** Responsible for the entire iOS mobile client application and user experience layer. This included implementing the responsive iOS frontend using Swift and the declarative SwiftUI framework, designing interface components, managing layout views and setting up the complete client-side event tracking architecture via Amplitude Tracking to log, monitor and evaluate production analytics and user behavior patterns.

4.1.2 Iterative Design

Before implementing the final version of ReFood, several architectural and product decisions were validated through iterative prototyping.

Architecture Validation

- An early version of the application was developed following Clean Architecture principles to validate the separation between the user interface, business logic, and data layer.

Database Design Validation
- Different entity relationships, data models and storage approaches were tested to determine which product information should be stored locally and which data should remain cloud-based.

AI Service Evaluation
- Multiple AI providers and prompting strategies were tested to evaluate response quality, consistency and cost efficiency. Different prompt structures were compared in order to achieve stable and predictable product explanations across multiple requests.

Recycling Infrastructure Research
- Different providers of recycling location data were evaluated and compared based on data quality, coverage and integration complexity. The selected solution demonstrated the best balance between performance and geographic coverage.

User Interface and Responsive Layout Prototyping
- We created several design prototypes before the final one was selected. Main attention was given to responsive layouts and usability across different devices of different sizes.

Local Data Storage Validation
- SwiftData prototypes were developed to validate local storage performance. This approach was ultimately adopted to ensure that frequently accessed information, such as scan history, could be loaded instantly without requiring a network request.

This iterative prototyping processes significantly reduced risks and allowed us be sure about or key architectural decisions before development began.

4.2 Architectural Patterns and Coding Standards

Since the project has a distributed client-server architecture, architectural patterns and coding standards were adapted to the specifics of the server (Backend) and mobile (iOS Frontend) parts of the system.

4.2.1 Server-side architecture and patterns

The server-side component of the ReFood application is built using a serverless architecture with AWS Lambda. Each Lambda function is an independent microservice with a clearly defined scope of responsibilitya according the Single Responsibility Principle.

Design and Cloud Architectural Patterns Implementation

- **Front Controller:** Each Lambda function has a centralized entry point (**`index.mjs`**) that acts as a front controller. It receives invocation events from AWS API Gateway, checks the corresponding HTTP methods and resource routing paths, and directs execution to the appropriate isolated function handler.
- **Cache-Aside Pattern:** This strategy is implemented in the product data gathering (**`ProductHandler`**). When a request for a product barcode arrives, we first check our own internal cache in DynamoDB. In the event of a cache miss, it fetches data from the Open Food Facts API, runs it through AI processing and stores the record in the database so that subsequent requests are retrieved from the database and are faster.

- **Data Mapper Pattern:** We useed separate mapper modules to transform external data models into internal ones (for example, from the Open Food Facts API into an our product model). This isolates the data transformation logic from the business logic.
- **Event-Driven Pattern:** Lambda function (**`news-service`**) is triggered not by HTTP requests, but by AWS events. It runs on a schedule via AWS EventBridge to automatically collect scientific publications. Additionally, s3-service responds to S3 events to validate uploaded images.

Naming Conventions

- CamelCase was used for files naming (for example **`fetchOffApi.mjs`**).
- Directory names reflect the modul's role in the architecture: handlers/ - request handlers, services/ - interaction with infrastructure, helpers/ - utility functions, mappers/ - data transformation.
- Environment variable names are formatted in SCREAMING_SNAKE_CASE style.

4.2.2 Client-side architecture and patterns

MVVM Architecture Implementation

The ReFood application is built on the Model-View-ViewModel (MVVM) architectural pattern, which is one of the most common approaches when developing SwiftUI applications. This architecture provides a clear division of responsibility between the layers of the system and simplifies further support and development of the application.

- **Model** is responsible for storing and providing data. This layer includes product models and user data.
- **ViewModel** contains the business logic of the application. It is responsible for data processing and also prepares data for display in the user interface.
- **View** consists of declarative SwiftUI components responsible for displaying data and interacting with the user. The View is automatically updated when the state of the data in the ViewModel changes

Key benefits:

- Separation of Concerns: The user interface is completely isolated from the logic of working with the network or database.
- Reactive UI: The use of state properties (**`@State`**, **`@Published`**, **`@Bindable`**) allows the interface to update automatically without direct manipulation of elements.

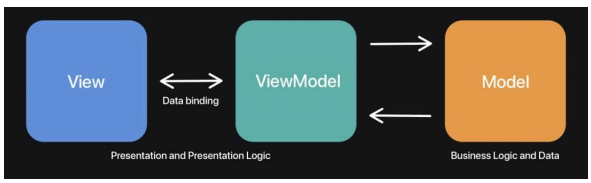

Figure 9: MVVM architectural pattern used in the ReFood.

Naming Conventions

- PascalCase style was used for named structures, classes, protocols and enums (for example, **`ProductPreviewScreen`**, **`AnalyticsServiceProtocol`**).

- CamelCase style was used for variables, constants and methods (for example, **`scanBarcode()`**).
- For localization keys, the snake_case style was used in accordance with the accepted structure of localization files.

Design Pattern Implementation

- **Observer / Reactive Pattern:** Used at the core of SwiftUI and Combine. UI components "subscribe" to changes in data models and react to them instantly.
- **Repository Pattern:** Implemented using SwiftData (**`@Query`** and **`@Model`**), which provides an offline approach to storing and quickly accessing scan history without constantly accessing the network.

4.3 Critical Code Implementations

This section presents the key Lambda function implementations that form the core of the ReFood backend. Each subsection highlights a critical component, its flow and the architectural decisions that bring business value.

4.3.1 User Handler: Implementation of User Services and Authentication

The ReFood app implements a hybrid identity model designed to maximize user retention. Users can launch and use the app completely anonymously, with their scan history and settings tracked instantly. At any time, the user can choose to upgrade to a fully authenticated account via Apple Sign-In without losing any accumulated data.

Architectural Identity Lifecycle Flows:

- Anonymous Session Onboarding: App Launch ⟶ AWS Amplify (Cognito Identity Pool) ⟶ POST /users/register ⟶ AWS Lambda (registerUser) ⟶ *the session is established using physical deviceId and AWS IAM temporary credentials attached to the identityId.*
- Authenticated Account Linking Upgrade: Apple OAuth Sign-In ⟶ AWS Cognito User Pool Validation ⟶ JWT token (cognitoSub within claims) ⟶ POST /users/register/link-account ⟶ AWS Lambda (registerLinkAccount).

In this way, when a user deletes the app and reinstalls it their anonymous data reappears because it is tied to deviceId. When a user signs in with Apple on any other IOS device - their full history and favorites are restored via cognitoSub.

Anonymous Registration Implementation

The following code snippet of **`registerUser`** processes initial device checks, updates outdated identityId tokens and guarantees database entity uniqueness using DynamoDB conditional checks:

```
...
  try {
    const existingUser = await findUserByDevice(body.deviceId);

    if (existingUser) {
      // if identityId changed (e.g. Cognito refreshed it) - update it
      if (existingUser.identityId !== body.identityId) {
        await updateIdentityId(existingUser.userId, body.identityId);
```

```
      }
      return response(200, { message: "User recognized" });
    }

    const newUser = toUserDBModel(body.deviceId, body.identityId);

    try {
      await createUser(newUser);
    } catch (error) {
      // race condition guard: if two requests hit simultaneously
      if (error.name === 'ConditionalCheckFailedException') {
        const findedUser = await findUserByDevice(body.deviceId);
        if (findedUser) {
          return response(200, { message: "User recognized" });
        }
      }
      throw error;
    }

    return response(201, { message: "New user created" });
  }
...
```

User DB Model

```
export const toUserDBModel = (deviceId, identityId) => {
    return {
        userId: randomUUID(),       // our internal primary key
        deviceId: deviceId,         // physical device identifier
        identityId: identityId,     // AWS Cognito Identity Pool ID
        isPremium: false,
        scansCount: 0,
        createdAt: new Date().toISOString()
    };
};
```

Apple ID Account Linking Logic

When transitioning into an authenticated state, the app sends a request to the /link-account endpoint. This handler transfers the anonymous data to a permanent user account.

```
...
  try {
    // case 1: returning Apple user on a new/same device
    const existingAppleUser = await findUserByCognitoSub(cognitoSub);

    if (existingAppleUser) {
      await updateUserDevice(existingAppleUser.userId, deviceId);
      return response(200, { message: "Welcome back" });
    }

    // case 2: first Apple sign-in — link to the existing anonymous record
    const anonymousUser = await findUserByDevice(deviceId);
```

```
      if (anonymousUser) {
        await linkUserToApple(anonymousUser.userId, { cognitoSub, email,
givenName });

        console.log(`Anonymous user linked to Apple: ${anonymousUser.userId}
`);

        return response(200, { message: "Account linked" });
      }

      return response(409, { error: "No user session found for this device.
Call /users/register first." });
    }
    ...
```

In that way, a shared helper extracts the caller's identity from either authentication method allowing all handlers to resolve a user uniformly.

```
  export const getRequestIdentity = (event) => {
    // authenticated via Apple ID (JWT from Cognito User Pool)
    const jwtSub = event.requestContext.authorizer?.claims?.sub ||
event.requestContext.authorizer?.jwt?.claims?.sub;

    if (jwtSub) {
      return { type: 'jwt', id: jwtSub };
    }

    // anonymous user authenticated via Cognito Identity Pool (IAM)
    const identityId = event.requestContext.identity?.cognitoIdentityId;

    if (identityId) {
      return { type: 'identityId', id: identityId };
    }

    // for testing auth out of the prod stage
    const mockIdentityId = event.headers?.['x-mock-identity-id'];
    if (mockIdentityId && process.env.STAGE !== 'prod') {
      return { type: 'identityId', id: mockIdentityId };
    }

    return null;
  };
```

Benefits:

- **Easy to start:** No registration required to start scanning products.
- **Data persistence:** Anonymous scan history survives app reinstalls.
- **Seamless upgrade:** Apple Sign-In merges with existing anonymous data, no data loss.

4.3.2 Product Handler & AI Service: Intelligent Product Analysis

When a user scans a barcode, the system must: retrieve product information, translate it into Ukrainian and English, and analyze the product's nutritional value using our AI Service. In addition, the system builds its own product database by collecting information

from the Open Food Facts (OFF) API and enhancing it with translations generated by AI. This allows for a reduction in the number of external API calls in the future and shorter response times for repeat scans.

Product Query and Fallback Cache Execution

The getProduct handler manages cache queries, coordinates external data access and maps localized response:

```
  ...
    // step 1: check our own database first
    const dbProduct = await getLatestProductFromDB(barcode);
    if (dbProduct) {
      console.log(`Found product in DB for barcode: ${barcode}`);
      return response(200, {
        source: "local",
        product: dbProduct
      });
    }

    // step 2: fetch from Open Food Facts if not in local DB
    console.log(`DB miss, fetching from OFF: ${barcode}`);
    const offProduct = await fetchFromOpenFoodFacts(barcode);

    if (!offProduct) {
      return response(404, {
        error: "Product not found"
      });
    }

    // step 3: translate and structure via AI
    const translated = await translateProduct(offProduct);
    const { categories_tags, allergens_tags, ingredients_text,
packaging, ...baseProduct } = offProduct;
    const product = { ...baseProduct, ...translated };

    // step 4: persist to our own database - next scan will be local
    await saveProductToDB(product);

  ...
```

Fetching from Open Food Facts

If the product is not found in the local database, the system queries the Open Food Facts API. This external dataset serves as the primary source for product information.

```
  export async function fetchFromOpenFoodFacts(barcode) {
    const url = `${OFF_API_URL}/${barcode}.json`;

    try {
        const res = await fetch(url, {
            headers: {
                "User-Agent": USER_AGENT,
                Accept: "application/json",
            },
```

```
        });

        if (!res.ok) {
            console.error(`OFF API returned ${res.status}`);
            return null;
        }

        const data = await res.json();
        if (data.status !== 1) return null;

        const product = data.product || {};
        const barcodeFromAPI = product.barcode || product.code || data.code ||
barcode;

        return offProductToProduct(barcodeFromAPI, product);
    } catch (error) {
        console.error("OFF API Error:", error);
        return null;
    }
}
```

Collaborative Data Contribution with AI Validation

When a user manually enters data about a missing product, this creates serious security risks, such as unreliable text data and unverified images. The platform uses a two-tier verification process:

Prompt Injection Defense & AI Input Validation: Before analysis, the user's text input is checked using special classification filters to detect malicious "prompt injection" attacks aimed at disrupting the artificial intelligence model. After that, our AI function checks each input field for unacceptable wording or unrealistic phrases.

Asynchronous Event-Driven Media Analysis: User submitted product images cannot be trusted or served immediately. The system decouples this workflow via an asynchronous event-driven pattern using AWS S3, Lambda AIService and AWS EventBridge.

Image Verification Lifecycle Flow:

Step 1: The iOS client requests a pre-signed URL and uploads the packaging image directly to a restricted transient directory (/temp/) within the AWS S3 bucket.

Step 2: The successful S3 image upload automatically emits an ObjectCreated state change event.

Step 3: AWS EventBridge intercepts this event and safely routes it to trigger an asynchronous background validation function.

Step 4: The background function passes the S3 image URI to the AIService using multimodal vision analysis to confirm if the file is a real product packaging image (checking for inappropriate content, blurring or mismatching data).

Step 5: The evaluation output is written to the UploadJobs table. Upon approval, the image is moved to the permanent production folder, updating the record status. If rejected, the job documents specific error details for client tracking.

The snippet code of **`validateImage function`**:

```
  ...
    try {
        const imageUrl = `https://${BUCKET_NAME}.s3.amazonaws.com/${s3Key}`;
        const aiResult = await checkPhoto(imageUrl);       // call AI to check
image

        const isValid = aiResult?.isValid ?? false;
        const error_en = aiResult?.error_en || null;
        const error_ua = aiResult?.error_ua || null;
        const status = isValid ? "APPROVED" : "REJECTED";

        // update for client to know about status of job
        await updateJobStatus(imageId, status, error_en, error_ua);

        results.push({ imageId, status });
    } catch (error) {
        await updateJobStatus(imageId, "REJECTED", {
            error_en: "Validation error. Please try again.",
            error_ua: "Помилка валідації. Спробуйте ще раз.",
        });
        results.push({ imageId, status: "REJECTED", error: error.message });
    }
  ...
```

Benefits:

- **Own product database:** Each OFF lookup is saved locally - subsequent scans are instant with no external API call.
- **Multilingual support:** AI translates product data to Ukrainian and English regardless of the original product language.
- **AI health analysis:** Users receive an immediate analysis of hidden sugars and harmful additives.
- **Safety gate:** User-submitted products must pass AI validation before entering the database.

4.3.3 Map Service: Geoinformation System and Routing

The Map Service provides recycling ponts search capabilities and navigation route generation for user. Both features integrate directly with the Geoapify API.

Geographic Data Storage and Licensing Compliance: Under the Geoapify API license agreement, caching or persistently storing geographic coordinate points, spatial indexes and routing details in a local database is explicitly prohibited. The ReFood architecture strictly enforces this compliance by treating responses as transient data: they are fetched on-demand, processed, transformed in memory and immediately transmitted to the client. Additionaly, this architectural design eliminates geospatial table maintenance and reduces DynamoDB storage requirements.

Route Request Handler

The following snippet of **`getRoute`** function represents the handler routing parameters, making external requests and returning simplified geometry coordinates:

```
export async function getRoute(event) {
    try {
        const params = event.queryStringParameters || {};
        const fromLat = params.fromLat;
        const fromLon = params.fromLon;
        const toLat = params.toLat;
        const toLon = params.toLon;
        const mode = params.mode;

        if (!fromLat || !fromLon || !toLat || !toLon) {
            return response(400, {
                message: 'Missing required parameters: fromLat, fromLon,
toLat, toLon'
            });
        }

        const rawData = await fetchRoute({
            fromLat,
            fromLon,
            toLat,
            toLon,
            mode: mode || 'walk'
        });

        const route = routeMapper(rawData);

        if (!route) {
            return response(404, {
                message: 'Route not found'
            });
        }

        // invoke metrics functions to update user statistics and achievements
        const identity = getRequestIdentity(event);
        const userId = await findUserIdByAnyMethod(identity);
        invokeMetrics('increment_sorted', userId);
        invokeMetrics('track_map_check', userId);

        return response(200, route);
    } catch (error) {
        console.error("Error occurred", error);
        return response(500, { message: "Internal Server Error" });
    }
}
```

Benefits:

- **License compliance:** All routing and geoinformation queries are processed without additional caching, ensuring full compliance with API provider licenses.
- **Zero storage overhead:** No local DynamoDB table storage or complex algorithm for routing is required on AWS.

4.3.4 Metrics Service: Gamification and Achievements Tracking

The ReFood backend implements a gamification system designed to increase user engagement. The gamification mechanics rely on a set of static achievements calculated dynamically from real-time user statistics stored in DynamoDB.

Incremental Metrics System

The following snippet showcases one of metric logic's implementation - the increment of scanned product in **`incrementScanned`** function:

```
  ...
    // just incrementation for metric
    let updateExpr = "ADD scannedCount :inc SET lastUpdated = :now";
    const exprAttrValues = { ":inc": 1, ":now": now };

    // check if scanning match any other achivements goals
    if (hour < 9) {
        updateExpr += ", earlyBirdUnlocked = :true";
        updateExpr += ", earlyBirdUnlockedAt =
if_not_exists(earlyBirdUnlockedAt, :now)";
        exprAttrValues[":true"] = true;
    }

    if (isWeekend) {
        updateExpr += ", weekendHasScanned = :true";
        exprAttrValues[":true"] = true;
    }

    await docClient.send(new UpdateCommand({
        TableName: USER_METRICS_TABLE,
        Key: { userId },
        UpdateExpression: updateExpr,
        ExpressionAttributeValues: exprAttrValues
    }));

    if (isWeekend) {
        await checkAndSetEcoWeekend(userId, now);
    }
  ...
```

4.3.5 News Service: Research and Nutritional Education Materials

The News Service provides research studies and nutritional education materials to the application's daily dashboard. The service architecture separates into a scheduled background research aggregation pipeline via PubMed APIs query and AWS EventBridge scheduler.

Automated Data Collection Process:

- Scheduled update cycle: AWS EventBridge (daily scheduled rule) ⟶ AWS Lambda (fetchNews).
- External data retrieval: Query to the external NCBI PubMed HTTP API.
- Processing cycle: Analysis of internal parameters using **`fast-xml-parser`** ⟶ Filtering existing records using a quick search by primary key in local NoSQL tables.

- Data Processing: Executing an internal lambda → AWS Lambda (AIService: "process_research" action).
- AI translation and simplification: AI runs automatic summarization algorithms, removing complex terms and creating clear, structured arrays and translate information both Ukrainian and English languages.
- Storage synchronization: Storing processed records in DynamoDB.

PubMed Data Retrieval and Processing

The following snippet represents the scheduled task that performs data retrieval and enrichment loop:

```
export const fetchNews = async () => {
    const rawResearches = await fetchNewsFromPubMed();
    console.log(`Step 1: Found in PubMed: ${rawResearches.length}`);

    if (rawResearches.length === 0) {
        return { message: "No researches found" };
    }

    const newResearches = [];
    for (const item of rawResearches) {
        const alreadyExists = await checkNewsExists(item.id);
        if (!alreadyExists) {
            newResearches.push(item);
        }
    }

    if (newResearches.length === 0) {
        return { message: "All fetched news already exist in database" };
    }

    console.log(`Step 2: New articles to process: ${newResearches.length}`);

    const aiResponse = await processNewsWithAI(newResearches);
    const processedArticles = Array.isArray(aiResponse) ? aiResponse :
(aiResponse.processed_news || []);

    console.log(`Step 3: AI responded with: ${JSON.stringify(aiResponse)}`);

    for (const processedItem of processedArticles) {
        const original = newResearches.find(r => r.id === processedItem.id);

        await saveNewsToDb({
            id: processedItem.id,
            date: original?.date || new Date().toISOString().split('T')[0],
            resource: original?.resource || "PubMed",
            ai_processed: processedItem
        });
    }

    return { message: `Successfully processed new articles` };
};
```

Daily Dashboard handler

When the client queries the daily summary, the database queries both the date-based tip and the latest 10 articles concurrently:

```
export const getSummary = async (event) => {
    const requestTimestamp = event.requestContext?.timeEpoch ? new
Date(event.requestContext.timeEpoch) : new Date();
    const tipDate = `${String(requestTimestamp.getUTCDate()).padStart(2,
'0')}.${String(requestTimestamp.getUTCMonth() + 1).padStart(2, '0')}`;

    const [tipResult, newsResult] = await Promise.all([
        getDailyTip(tipDate),
        getLatestNewsFromDb(10)
    ]);

    return response(200, {
        date_utc: requestTimestamp.toISOString().split('T')[0],
        tip: tipResult.Item || null,
        news: newsResult || []
    });
};
```

Benefits:

- **Autonomic Ingestion:** Aggregation runs entirely in the background, requiring zero manual operations or content moderation.
- **Low-Latency Serving:** Scientific article aggregation and AI translations are done ahead of time. Therefore client dashboard reads require only quick concurrent queries.

4.4 Testing Approach and Quality Assurance Measures

This section describes in detail the testing methods and tools that were applied to the entire project code base to ensure the stability and reliability of the client and server components of the system.

4.4.1 Server-side testing approach

To ensure the stability and reliability of the ReFood serverless backend, a testing strategy for modular testing of Lambda functions was developed using the **Vitest** framework.

Mocking Strategy

To avoid dependencies on actual cloud infrastructure during testing, a system of mock objects was implemented using the **`vi.mock()`** mechanism built into Vitest. All external dependencies are mocked:

- AWS SDK (**`@aws-sdk/client-dynamodb`**, **`@aws-sdk/lib-dynamodb`**, **`@aws-sdk/client-s3`**) to isolate from actual calls to DynamoDB and S3.
- Service layer modules (**`newsDatabase.mjs`**, **`userMetricsDatabase.mjs`**, **`aiService.mjs`**) - to verify the business logic of handlers independently of the infrastructure.
- Mock Lambda functions - for testing cross-service interactions without actual calls through AWS.

This approach allows us to run a full list of tests locally without access to the AWS environment and without any financial costs for cloud resources.

Unit Testing The core business logic of the server-side is covered by unit tests using the **Vitest** framework. Testing is performed at the level of individual Lambda functions. The main areas of testing include:

- **Routing:** Verifying the correct differentiation of input event types - HTTP requests, EventBridge triggers and direct Lambda invokation, and delegating them to the appropriate handlers with the return of correct status codes.
- **Business logic and calculations:** Validation of data transfer and processing logic and algorithms. Specifically, tracking user gamification progress for numerical and boolean achievement types.
- **Input data validation:** Verifying the validation of required parameters and their format before processing.
- **Testing positive and negative scenarios:** Each logic is covered by tests for both the expected successful outcome and failure scenarios like database unavailability, AI service errors and invalid input data.

Alternative Approaches and Rationale

When designing the QA strategy, alternative approaches were considered, including integration testing with actual AWS resources. However, the decision was made to focus on unit testing with full AWS mocking. This strategy is justified by several factors:
- It ensures predictable and reproducible behavior independent of the state of the cloud infrastructure.
- It provides significantly faster test execution.
- It allows us to test edge cases (DynamoDB unavailability, AI service errors) that are difficult to reproduce in a real environment.

4.4.2 Client-side testing approach

To ensure the stability and reliability of the ReFood mobile client, a testing strategy based on the MVVM architecture was implemented.

Mocking Strategy

To avoid dependence on real network requests during testing, a system of mock objects was created, for example:
- A MockProductRepository was implemented, which mimics the behavior of the ProductRepository protocol.
- This mock object allows you to simulate asynchronous operations, such as receiving a product.

Unit Testing

The main business logic of the application is covered by unit tests using the native XCTest framework. Testing is performed at the ViewModel level. The main areas of testing include:

- **Localization and Fallback Logic**: Verifying the correct display of multilingual content (English and Ukrainian) using a `MockLanguageProvider` and ensuring that missing or incomplete backend data gracefully falls back to default values without causing application crashes.
- **Business Logic and Calculations**: Validating complex algorithms, such as tracking gamification progress for user achievements and determining the healthier product in the comparison module.
- **Asynchronous State Management**: Verifying the correct handling of concurrent tasks, which includes preventing duplicate network requests during barcode searches and ensuring UI loading indicators behave predictably during simulated API delays.
- **Data Validation**: Ensuring strict validation of user inputs before data is processed, such as trimming whitespaces from barcodes, verifying string lengths and correctly parsing decimal numbers in the product creation forms.

Alternative Approaches and Justification

When designing the QA strategy, alternative approaches were considered, including the use of Behavior-Driven Development frameworks such as Quick and Nimble, as well as libraries for auto-generation of mock objects (for example, Cuckoo or SwiftyMocky). However, it was decided to abandon third-party solutions in favor of the native XCTest framework and the creation of manual mocks. This choice was justified by the desire to minimize the number of external dependencies in the project, which guarantees higher stability, security and perfect compatibility with future updates of the Apple ecosystem.

In addition, testing business logic through the graphical interface using XCUITest (UI Testing) was considered as an alternative. However, given the MVVM architecture, priority was given to unit testing of the ViewModel level. This solution avoided the "unstable tests" problem inherent in UI testing and provided significantly higher test execution speed due to complete isolation of logic from interface rendering and real network requests.

4.5 Performance Optimizations

This section discusses performance problems and their solutions that arose during the development of both the server and client parts of the application.

4.5.1 Server-side performance optimization

On the server side, the main focus was on reducing request processing latency, optimizing calls to external services and avoiding overload on the cloud infrastructure.

Bottleneck of Repeated AI Calls During Product Scanning

- **Problem:** If AI product analysis were performed with every scan, this would result in two major operational costs—in terms of both time and money—and with a large number of users, these costs would increase proportionally to the number of scans of the same product. Additionally, generative models can produce different responses each time they are requested, which would mean that different users would receive differently worded analyses of the same product.

- **Solution:** AI product analysis is performed exactly once when the product is first saved to the database. The result (`analysis_ua`, `analysis_en`) is stored along with the product and subsequently returned to all users without recalls to AI. This ensures the consistent conclusion for every user and completely eliminates AI costs during subsequent scans of the same barcode.

Bottleneck in Long-Running Photo Processing

- **Problem:** AI analysis of an uploaded image is an asynchronous operation that can take several seconds. Keeping the HTTP connection open for the entire duration of processing is inefficient: it ties up client-side resources and increases the risk of a connection timeout.
- **Solution:** An asynchronous approach based on the Job Status pattern has been implemented. Upon receiving a presigned URL for upload, a task record with the status `PENDING` is created in the database. After the file is uploaded, an S3 event automatically triggers a Lambda function that performs AI validation and updates the task status to `APPROVED` or `REJECTED`. The client, in turn, periodically sends polling requests to a separate `job-status-checker` endpoint (`GET /image-validation`) to check the current processing status. This approach eliminates the need to keep a long-lived connection open.

Metrics Service Call Bottleneck

- **Problem:** After each scan or product creation, the user metrics achievement must be updated. Synchronously waiting for a response from `metrics-service` increased the API total response time by the duration of DynamoDB operations that are not critical to the main query result.
- **Solution:** The `metrics-service` call was switched to asynchronous mode (`InvocationType: "Event"`). The function initiates the metric update and immediately returns a response to the client without waiting for the operation to complete. This allowed us to reduce the response time of the main endpoints without losing data.

Bottleneck of Repeated AI Calls In NewsPipeline

- **Problem:** The daily news collection pipeline from PubMed could send articles that already existed in the database to the AI service for processing. Each call to the AI service is relatively slow and can be a costly operation, so processing duplicates led to wasted resources.
- **Solution:** A deduplication step was introduced before sending articles to the AI: for each received publication, we check for verify its existence in the database (`checkNewsExists`). Only new content is sent to the AI service, which significantly reduces the number of calls.

4.5.2 Client-side performance optimizations

In the case of the mobile client, the main attention was paid to optimizing the operation of hardware components, reducing the number of redundant network requests, and preventing interface freezes when processing large arrays of local data.

Scanner Bottleneck

- **Problem:** The constant search for a barcode by the camera heavily loaded the processor and could cause the device to overheat. In addition, if not interrupt the process after a successful scan, the application inevitably enters an endless cycle of recognizing the same code, blocking further logic.
- **Solution:** The scanner logic was designed in such a way to instantly stop the video capture session (**`AVCaptureSession`**) and disable frame analysis immediately after the first successful reading. Scanning resumes only after an explicit user action (for example, when closing the product details window or pressing the rescan button).

Map API Bottleneck

- **Problem:** Every time the user moves the map, the application could initiate dozens or hundreds of requests to the server to load recycling points. This would overload the network, exhaust API quotas, and lead to critical crashes of the graphical interface.
- **Solution:** Camera movement state tracking was implemented. Instead of automatically loading data with each screen shift, the application waits for the complete stop of the map and shows the search button. The request for recycling points in a specific radius is performed only once and at the explicit command of the user, which radically optimized the consumption of resources.

Search UI Bottleneck

- **Problem:** On the search screen (**`SearchView`**), the list of scanned history is reactively updated every time user enters a new character, change the focus, or switch tabs. Deserialization of "heavy" JSON data (productData) for hundreds or thousands of elements in real time with each such update of the interface would lead to a strong drop in performance.
- **Solution:** In **`SearchViewModel`**, the preliminary filtering strategy was applied. Matched text search and "favorite" status check are performed on lightweight properties of the local model (such as name and brand) before JSON processing begins. The resource-intensive operation JSONDecoder().decode is called only for those elements that have already passed filtering, which allows the search to work instantly regardless of the size of the scanned history.

4.6 Deployment and Configuration Management

4.6.1 Server-side deployment

The ReFood app deployment is based on AWS's public global infrastructure, where everything is geographically localized within a single region (**`eu-north-1`**, Stockholm) to ensure minimal network latency for European and Ukrainian users.

Automated Deployment via CI/CD

To automate the deployment of the application server-side components, a separate GitHub Actions pipeline has been set up for each Lambda function. Each workflow triggers automatically when changes are pushed to the **`main`** branch - but only if the changes affect the directory of the corresponding service. This means that changes in one service

do not trigger a redeployment of others, which minimizes the number of unnecessary deployment operations.

Each pipeline consists of four sequential steps: fetching code from the repository, installing production dependencies, packaging the function into a ZIP archive and uploading the new code to AWS Lambda via the AWS CLI. AWS access credentials are stored as encrypted GitHub Secrets and do not appear in text anywhere in the code. This approach ensures a fully automated and reproducible deployment process: any code change merged into the main branch is automatically deployed to the production environment without manual intervention.

4.6.2 Client-side deployment

The practical process of deploying the ReFood mobile client and managing its configurations had a clear sequence of actions, which was performed immediately after the development and testing of the code base was completed. This process consisted of several key stages:

Release Build

After completing the active coding phase in Xcode, using a registered Apple Developer account, the project archiving process was initiated. The result of this step was the creation of a ready-to-deploy installation package in **`.ipa`** format.

Testing via TestFlight

The generated .ipa file was exported directly from Xcode and uploaded to the App Store Connect cloud system. In order to test the application in real conditions, the TestFlight platform was configured. Through it, the build was sent to real iPhones of the development team members, which allowed us to identify and fix minor interface bugs before the official publication.

App configuration in App Store Connect

At the final stage, in parallel with testing, the application's product page was prepared for release. In the App Store Connect account, text descriptions and keywords were filled in, real screenshots of the application screens were uploaded, and the security section was configured.

5 Validation

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut.

Contents

5.1 Section 1

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut.

5.2 Section 2

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut.

5.3 Conclusion

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut.

6 Conclusion

6.1 Project summary

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut.

6.2 Comparison with the initial objectives

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut.

6.3 Encountered difficulties

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut.

6.4 Future perspectives

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut.

Glossary

Bibliography

[1] R. Benthem de Grave *et al.*, "Smartphone Apps for Food Purchase Choices: Scoping Review of Designs, Opportunities, and Challenges," *Journal of Medical Internet Research*, Mar. 2024, [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC10955402/

[2] M. Scott, "Ending Consumer Confusion Over Recycling Is ``Critical'' in Battle Against Plastic Waste." [Online]. Available: https://www.reuters.com/sustainability/boards-policy-regulation/ending-consumer-confusion-over-recycling-is-critical-battle-against-plastic-2023-07-19/

[3] FoodMinds, "FoodMinds Unveils its 2024 Conscious Consumption Index Global Report." [Online]. Available: https://foodminds.com/news/foodminds-unveils-its-2024-conscious-consumption-index-global-report/

[4] A. Basli, "The Open Data Revolution in Food." [Online]. Available: https://medium.com/@adelbasli/the-open-data-revolution-in-food-6686ebe4aa8e

[5] Yuka, "Yuka." [Online]. Available: https://yuka.io/en/

[6] B. M. Chaudhry, "Food for Thought," *mHealth*, vol. 5, p. 20, 2019, doi: 10.21037/mhealth.2019.06.02.

[7] F. Southey, "Italian Recycling App Tackles ``Confused'' Waste Sorting in Europe." [Online]. Available: https://www.foodnavigator.com/Article/2020/08/18/Junker-app-tackles-confused-waste-sorting-in-Europe/

[8] Castle Group, "Nutri-Score: Understanding Europe's Controversial Nutrition Labeling System in 2025." [Online]. Available: https://www.castle-group.eu/blog/our-blog-1/nutri-score-understanding-europe-s-controversial-nutrition-labeling-system-in-2025-10

[9] Geosemantica, "WasteFreeMap: Global Map for Recycling Collection Points." [Online]. Available: https://geosemantica.com/post/wastefreemap-global-map-for-recycling-collection-points