Software Engineering Department

Braude College

Final Project Phase B

CartGenie – A smartphone application for Smart Grocery Shopping

Project Code: 25-2-D-20

Developers:

Alex Segal — *alex.segal@e.braude.ac.il*

Nir Froiman (Froimovich) — *nir.froimovich@e.braude.ac.il*

Project Drive: CartGenie - Final Project

GitHub : https://github.com/CoderxX22/CartGenie.git

Project facilitator: Ronen Zilber

Semester A Winter 2026

Contents

1. Abstract

Purchasing groceries is an integral part of daily life, but the preparatory process involves a series of manual tasks—alerting household members, checking existing supplies, and drawing up lists. Despite the assistance from digital tools like phone applications, chatbots, or memo papers, users often recall essential items late and do not synchronize food consumption with their medical needs. We created a smart grocery management platform that moves beyond traditional mechanisms. The system analyzes users' blood test reports to identify potential health risks and generates individualized shopping lists according to health-based dietary advice. It also scans supermarket receipts to check whether purchased items align with the user's health-based dietary recommendations. The platform reduces manual effort, automates core processes, and improves the effectiveness of grocery shopping by incorporating health and nutrition guidance into the purchase process.

2. Introduction

Grocery shopping is a routine part of everyday life, yet food choices made in a supermarket often do not reflect a person's health needs. Many people live with chronic or borderline medical conditions that could be better managed—or even prevented—through healthier dietary decisions. In practice, however, applying medical advice while shopping is difficult and often impractical.

After a medical diagnosis, healthcare professionals usually provide general dietary recommendations, such as reducing sugar or avoiding saturated fats. These guidelines are rarely translated into specific, product-level choices that users can easily follow in real shopping environments. As a result, decisions are typically driven by habit, convenience, or time pressure rather than health considerations.

Most existing grocery and nutrition applications focus on list management, reminders, or manual food tracking. They rarely integrate clinical data, such as blood test results, and do not provide immediate, personalized feedback on purchased products. CartGenie addresses this gap by connecting medical information directly to shopping decisions. The system:

1. analyzes blood test reports to identify potential health risks,
2. extracts product and receipt information using OCR
3. provides clear, explainable feedback for individual items and entire shopping carts based on the user's profile.

CartGenie is intended for users with chronic conditions or borderline biomarkers, health-conscious individuals, caregivers assisting family members, and anyone who prefers simple, intuitive guidance without manually analyzing nutrition labels.

2.1 Technical Approach (High Level)

- The mobile application, developed using React Native, Expo, and TypeScript, handles user onboarding, profile management, uploads, and scanning interactions.
- A backend service built with Node.js and Express provides REST APIs for authentication, OCR processing, blood test analysis, AI consultation, and history management.
- MongoDB Atlas database on Azure is used as the cloud database to store user data, profiles, blood test analyses, and scan or receipt history.
- OCR processing is performed on the backend using Tesseract on uploaded images. Blood tests and receipts are submitted as images only (no PDF support), and blood test analysis supports uploading multiple images in a single request.
- Decision logic follows a hybrid approach: deterministic medical thresholds are applied to blood biomarkers, while an LLM-based AI agent generates food and cart safety recommendations.

3. System description

3.1 Project Overview

CartGenie is implemented as a client–server system composed of a mobile application and a backend service. The mobile client is designed as a thin layer responsible for user interaction, onboarding, and data capture, including camera-based scans and file uploads. All computationally intensive and sensitive operations are handled on the backend. The backend centralizes key system logic such as optical character recognition (OCR), blood test interpretation, AI-based consultation, and data persistence. This architectural separation simplifies maintenance, improves security, and enables updates to OCR or AI logic without requiring redeployment of the mobile application. The overall design supports scalability and allows the system to evolve independently on the client and server sides.

3.2 System Architecture

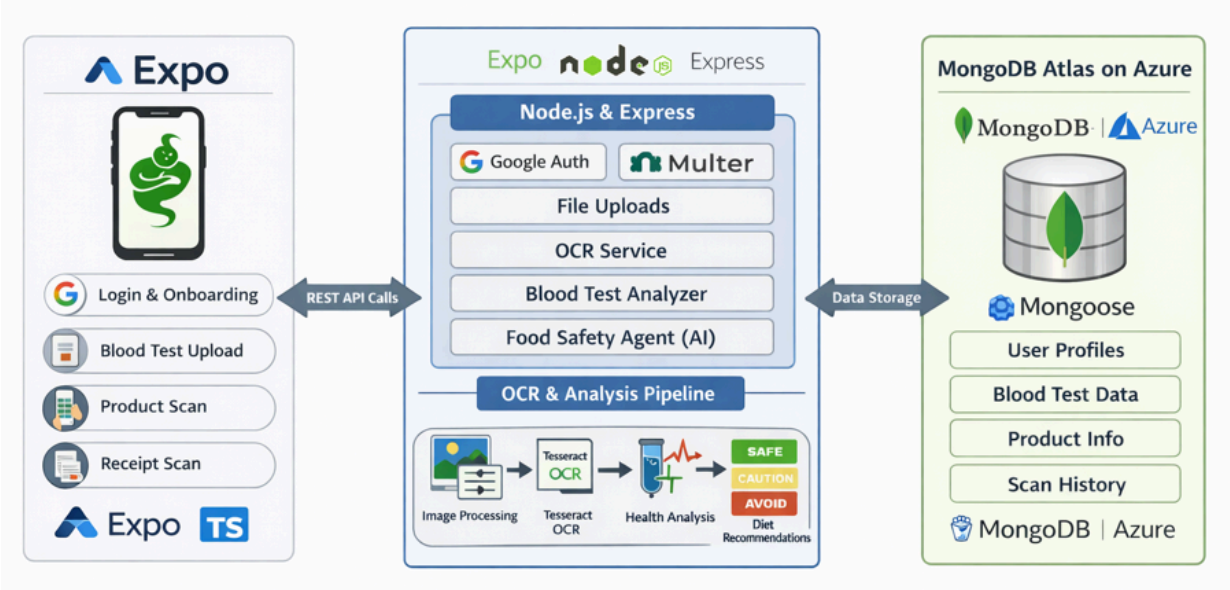

Figure 1: System Architecture Diagram (Mobile App ↔ Backend REST API ↔ MongoDB Atlas; OCR + Analysis pipeline on backend).

Main components of the system include:

- **Mobile Application (React Native / Expo):**
  Provides the user interface for login, onboarding, profile management, blood test uploads, product barcode scanning, receipt scanning, and viewing history. The mobile app focuses on usability and efficient data collection. We chose this platform due to the fact that it is single-source for both Android and iOS.

- **Backend API (Node.js + Express):**
  Exposes RESTful endpoints responsible for authentication, user and profile management, file uploads, OCR processing, blood test analysis, AI consultation, and history management. We found NodeJS is the most Optimal framework for our mobile app.
- **Blood Test Analyzer (Backend):**
  Parses OCR output and applies deterministic, clinically motivated threshold rules to identify abnormal biomarkers and generate a structured health-risk summary for the user.
- **OCR Service (Backend):** Executes Tesseract OCR on uploaded images (blood tests and receipts) and returns extracted text to downstream analyzers. Due to its accuracy in extracting text from images.
- **Food Safety Agent (LLM-based, Backend):**
  Performs health-based evaluation of individual products and entire shopping carts. The agent returns structured recommendation labels along with short, explainable justifications. The LLM is an AI agent that acts as a nutrition expert without the need for pre-training.
- **Persistence Layer (MongoDB Atlas):**
  Stores user accounts, personal profiles, the latest blood test baseline, product scan history, and receipt analysis history in a scalable cloud-based database.

4. Main Workflows

4.1 Onboarding and Profile Completion

The user registers or logs in using email/password or Google Sign-In. After authentication, the user completes a personal profile including basic details, body measurements, and medical conditions. This information is stored in the database and is used to personalize all future analyses and recommendations.

4.2 Blood Test Upload and Analysis

The user uploads blood test images. The user may upload multiple images at the same time (e.g., several screenshots/pages). The backend applies OCR using Tesseract, extracts key biomarkers, and analyzes them using predefined medical thresholds. Only the most recent blood test set is kept as the active baseline. Identified health risks are displayed in the app and used for personalized feedback.

4.3 Product Scan (Single Item)

The user scans a product barcode or enters a product name manually. Barcode scanning is the preferred method, as it provides reliable access to structured nutritional data. The backend

retrieves product details from the database and combines them with the user profile. The Food Safety Agent returns a recommendation (SAFE / CAUTION / AVOID / UNKNOWN) with a short explanation. The result is saved in the scan history.

4.4 Receipt / Cart Analysis

The user scans or uploads a grocery receipt image. The backend performs OCR and extracts product information, including barcodes when available. Identified products are matched against the database and analyzed together as a shopping cart. The system returns per-item recommendations and an overall cart summary. Results are stored in the receipt history for later review

4.5 Use Case Diagram

Figure 2: Use Case Diagram.

Use cases:

UC01 – Log In / Authenticate. Allows the user to authenticate and access the system in order to use personalized features and data.
UC02 – Create / Manage Profile. Enables the user to create and update a personal profile that is used to personalize analysis and recommendations.
UC03 – Manage Medical Parameters. Allows the user to define and update medical parameters that affect health analysis and feedback.
UC04 – Upload Blood Test. Enables the user to upload a blood test file (single or multiple images) for medical data extraction and analysis.
UC05 – Upload Images for Scan. Allows the user to upload images (e.g., grocery receipts) for text extraction and processing.
UC06 – Scan Receipt (Camera). Enables the user to scan receipts using the device camera as an alternative to image upload.
UC07 – Extract Text from Image Files. The system extracts raw text from uploaded or scanned images using OCR.
UC08 – Parse & Structure Data. The system converts extracted raw text into structured and machine-readable data.
UC09 – Analyse Medical Data. The system analyzes structured medical data to identify potential health risks or abnormalities.
UC10 – Generate and Send Feedback. The system generates personalized feedback and recommendations and presents them to the user.
UC11 – View Feedback History. Allows the user to view previously generated feedback reports and analysis results.
UC12 – Export Feedback Report. Enables the user to export feedback reports for storage or sharing.
UC13 – Manage Report. Allows the user to manage saved reports, including deletion or organization.

4.6 Activity Diagram

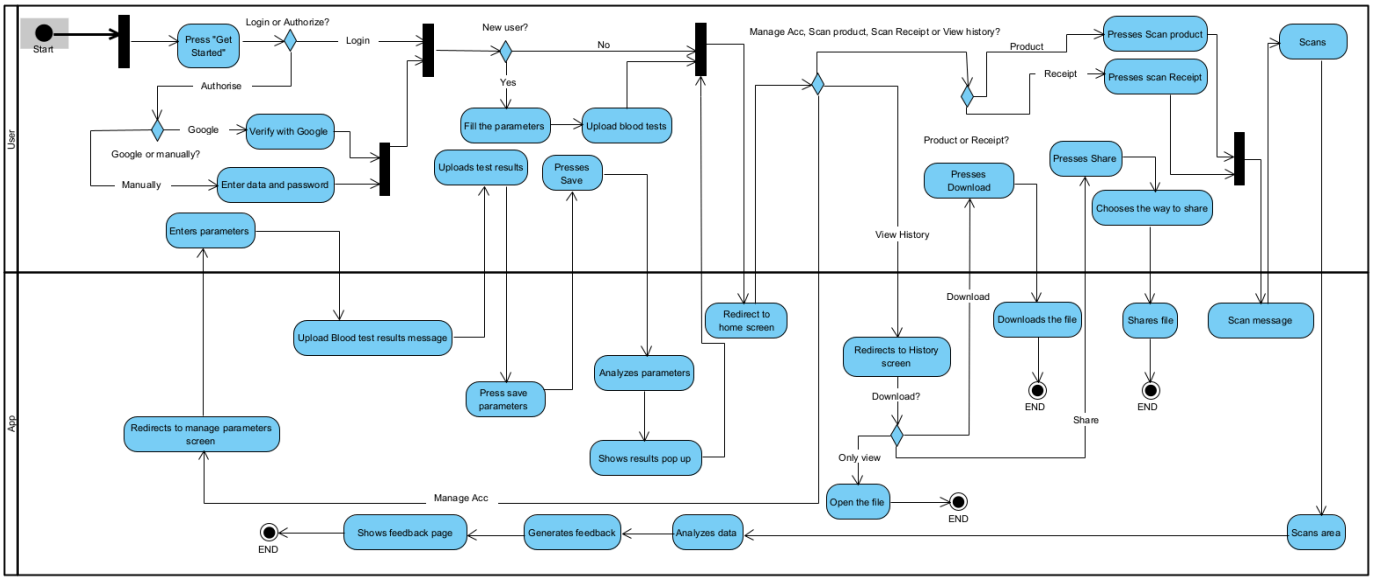

Figure 3: Activity Diagram for the end-to-end flow (login → profile → blood test → scan product/receipt → feedback → history).

Workflow:

1. Authorization
    a. The user launches the CartGenie application.
    b. The user presses “Get Started”.
    c. The system prompts the user to authenticate.
    d. The user selects one of the authentication methods:
        i. Login with Google – the system initiates Google authentication and verifies the user.
        ii. Manual Login – the user enters email and password.
    e. After successful authentication, the system proceeds to profile handling.
2. Profile Parameters Input
    a. If the user is new, the system prompts the user to fill in profile parameters (e.g., age, gender, body measurements, medical conditions).
    b. The system displays an optional message requesting the upload of blood test results.
    c. The user uploads a blood test file (single or multiple images) or skips this step.
    d. The user presses “Save Parameters”.
    e. The system validates the entered parameters.
    f. If a blood test was uploaded:
        i. the system performs OCR on the file,
        ii. parses the extracted data,

iii. analyzes biomarkers using medical thresholds.

g. The system displays a results pop-up summarizing detected health risks or abnormal indicators.
h. The user confirms by pressing “Save”.
i. The system saves the profile and (if applicable) the blood-test baseline and redirects the user to the Home screen.

3. Navigation Decision - The system prompts the user to choose the next action:
    a. Manage Account
    b. Scan Product
    c. Scan Receipt
    d. View History
4. Manage Account - If the user selects Manage Account:
    a. the system opens the profile management screen,
    b. the user may update profile parameters or medical conditions.
    c. The system saves changes and redirects the user back to the Home screen.
    d. End.
5. View Feedback History - If the user selects View History:
    a. the system redirects the user to the History screen.
    b. The user selects a past report.
    c. The system prompts the user to choose one of the following actions:
        - View – open the report on screen.
        - Download – download the report file to the device.
        - Share – select a sharing method and send the report
    d. After the selected action is completed, the workflow ends.
6. Scan Products - If the user selects Scan Product:
    a. the system activates the device camera.
    b. The user scans a product barcode.
    c. The system processes the scan and displays a scan message.
    d. The system analyzes the product together with the user profile.
    e. The system generates and displays personalized feedback.
    f. The result is saved to history.
    g. End.
7. Scan Receipt - If the user selects Scan Receipt:
    a. the system activates the camera or file picker.
    b. The user scans or uploads a grocery receipt.
    c. The system extracts product information using OCR.
    d. The system analyzes the full cart together with the user profile.
    e. The system generates and displays cart-level feedback.
    f. The result is saved to receipt history.
    g. End.

5. Technologies Used

CartGenie is built as a mobile-focused client–server system. The technology stack was chosen to keep development practical, the system easy to maintain, and the results explainable—while remaining appropriate for a student capstone project.

5.1 Mobile Application

React Native + Expo + TypeScript
The mobile app is developed using React Native with the Expo framework and written in TypeScript. This combination enables cross-platform development while maintaining access to native device features such as the camera, local storage, and file system. TypeScript improves code reliability through static typing and helps reduce runtime errors.
expo-router Navigation
Navigation between screens is implemented using expo-router, which provides a file-based routing structure. This simplifies navigation logic and improves code readability compared to traditional navigation stacks.
AsyncStorage
AsyncStorage is used for lightweight local persistence, such as storing authentication tokens and session state. This reduces unnecessary backend requests and improves user experience.

5.2 Authentication and Authorization

JWT-Based Authentication
User authentication is handled using JSON Web Tokens (JWT). After successful login, the backend issues a signed token that is stored on the client and included in all subsequent API requests.
Google Sign-In
Google authentication is implemented using @react-native-google-signin/google-signin. Google identity tokens are verified on the backend using official Google libraries. Due to Google's security requirements, full validation was only possible after generating an Android APK, which affected early development and testing.

5.3 Backend Services

Node.js + Express
The backend is implemented using Node.js with the Express framework. This stack was selected for its simplicity, strong ecosystem, and smooth integration with a JavaScript-based mobile client. The backend exposes REST APIs for:

- User authentication and profile management
- Blood test and receipt uploads
- OCR processing
- AI-based product and cart analysis
- History and report persistence

- File Uploads (Multer)
- Database Access (Mongoose)

5.4 Database

MongoDB Atlas
MongoDB Atlas is used as a cloud-hosted NoSQL database. Its document-oriented structure is well-suited for flexible medical data, OCR output, and historical scan records.

The database stores:

- User credentials and identities
- Profile and medical parameters
- Blood test analyses
- Product scan history
- Receipt/cart analysis history

MongoDB Atlas was chosen due to its scalability, ease of integration with Node.js, and managed infrastructure.

5.5 Optical Character Recognition (OCR)

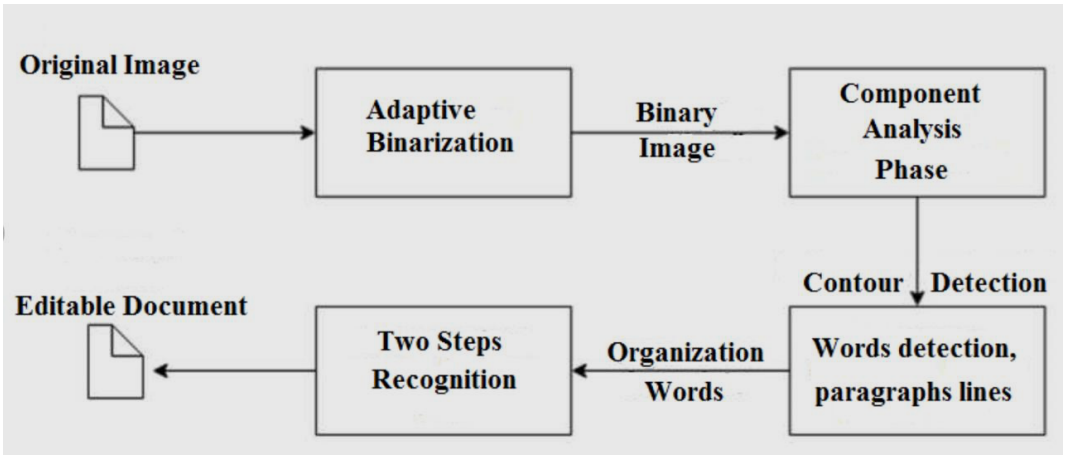

Figure 4: OCR Process Flow to build an API with Tesseract from a blog post

Tesseract OCR (tesseract.js)
Text extraction from blood test reports, grocery receipts, and nutrition labels is performed using Tesseract OCR via the JavaScript wrapper tesseract.js. Tesseract was chosen because it is open-source, free to use, supports multiple languages (including Hebrew), and performs reliably on printed medical documents and receipts.
Image Input (Blood Tests and Receipts)

Since the system processes images, blood tests and receipts are uploaded as image files only. This simplifies the pipeline and improves runtime stability in cloud deployment environments.

Image Preprocessing

To improve OCR accuracy, basic image preprocessing (such as cropping, contrast enhancement, and resizing) is applied in some cases using the sharp library, especially for receipts captured under poor lighting conditions.

Hebrew Text Handling (RTL)

When processing Hebrew documents, OCR output requires additional post-processing. The backend includes logic to correct right-to-left (RTL) text order and normalize tokens, ensuring correct interpretation of biomarker names and values.

5.6 AI Decision Component (AI Agent)

Google Gemini (Generative AI SDK)

Instead of a traditional machine-learning classifier such as XGBoost, CartGenie uses an AI agent based on Google Gemini's generative models. The agent receives a structured summary of the user's health profile, parsed blood test results, and product or cart information.

The agent returns:

- A recommendation label (SAFE / CAUTION / AVOID / UNKNOWN)
- A short, human-readable explanation
- A structured JSON response for frontend rendering

Why an AI Agent Instead of XGBoost

The original system design included an XGBoost-based classifier. During development, this approach proved to be over-engineered for the project's scope.

Key reasons for replacing XGBoost:

- Lack of reliable labeled medical datasets
- High development and maintenance overhead
- Limited explainability for non-technical users

In contrast, the AI-agent approach:

- Requires no training pipeline
- Is easier to adapt as medical rules evolve
- Produces clear, explainable outputs
- Better fits a decision-support (not diagnostic) system

As a result, CartGenie uses a hybrid decision model:

- Deterministic medical thresholds for blood test analysis
- LLM-based reasoning for product and cart evaluation

This design reduced development complexity while preserving practical value and transparency.

5.7 DevOps and Configuration

Environment Configuration
Sensitive values such as API keys, database connection strings, and secrets are managed using environment variables stored in .env files.
Local Development
Development and testing are performed locally using npm scripts for both frontend and backend services, ensuring a consistent and reproducible development environment.

6. Requirements

#	Requirement Name	Description	Type
1	User Profile Management	The system allows users to create and edit profiles	FR
1.1	User Profile Management	The parameters to edit are: age, gender, BMI, and health conditions	NFR
2	Upload Blood Test	The system allows users to upload blood tests	FR
2.1	Upload Blood Test	Blood tests are uploaded as image files only. The user may upload multiple images in a single upload session.	NFR
3	Upload Grocery Receipt	The system allows users to upload grocery receipts	FR
3.1	Upload Grocery Receipt	Grocery receipts are uploaded as image files only.	NFR
4	Text Extraction	The system extracts text from uploaded files	FR
4.1	Text Extraction	Text extracts using Tesseract OCR light.	NFR
5	Scan Receipt	The system uses the device's camera to scan a receipt.	FR
6	Analyze Blood Test	The system parses and analyzes blood tests to identify abnormal values.	FR
6.1	Analyze Blood Test	Identification by using Gemini AI agent	NFR
7	Parse & Structure Data	The system converts extracted text into structured data	FR
7.1	Parse & Structure Data	Conversion of text is done by using parsing logic.	NFR
8	Analyze Grocery Items	System matches grocery items with the user's health profile	FR
8.1	Analyze Grocery Items	Matching is done by using an AI agent to identify risks.	NFR

9	Generate Feedback	The system generates health-based feedback	FR
9.1	Generate Feedback	Feedback is based on user data and receipt items after AI agent analysis.	NFR
10	View History	The system allows users to view past feedback reports.	FR
11	Manage Reports	Allow users to delete or modify saved reports.	FR
12	Export Report	Allow users to export feedback	FR
12.1	Export Report	Feedback report exported as a PDF.	NFR
13	Processing Speed	The system must process documents within 5 seconds (up to 2MB).	NFR
14	Uptime Guarantee	Maintain at least 95% system uptime.	NFR
15	Mobile UI Design	UI should be intuitive and optimized for mobile devices.	NFR
16	Dark Mode Support	Allow users to toggle between light and dark themes.	NFR
17	PDF Export Formatting	Exported reports must be formatted and styled correctly.	NFR
18	Blood Test Parsing Coverage	The system must accurately extract and interpret at least 10 common biomarkers (e.g., glucose, LDL, HDL, iron, etc.).	FR
19	Receipt Item Matching	The system must match at least 80% of receipt items to known food/nutrition categories.	FR
20	Item Risk Classification	Each grocery item must be classified	FR
20.1	Item Risk Classification	Classification is divided into 'healthy', 'neutral', or 'risky' categories.	NFR
21	Explain Feedback	Feedback must include a short explanation based on health logic (e.g., 'High sugar - not suitable for elevated glucose').	FR
22	Profile Validation	The system must validate user input on the profile form	FR
22.1	Profile Validation	Validation is: age range 0-120, BMI between 10-60.	NFR
23	Data Retention Control	Users must be able to delete stored medical and receipt data at any time.	NFR
24	Battery Optimization	The app must be optimized to consume minimal battery during OCR and analysis.	NFR

7. Results and Evaluation

7.1 Achieved Features

Our implementation delivered a complete, working system that supports all core user flows defined during design.

- **Operational mobile application**
  A fully functional mobile app was developed using React Native and Expo, including onboarding, authentication, and smooth navigation.
- **Authentication and user management**
  Users can log in using email/password or Google Sign-In. Secure sessions are maintained using JWT tokens.
- **User profiles**
  Users can create and manage personal profiles with body measurements and health conditions. Profile data is stored persistently and reused for all analyses.
- **Blood test analysis**
  Blood test images can be uploaded and analyzed using OCR. The system supports uploading multiple images at once (e.g., multiple screenshots/pages) and consolidates them into a single baseline analysis.
- **Product scan and AI feedback**
  Individual products can be scanned or queried. Each product receives a clear SAFE / CAUTION / AVOID label with a short, profile-based explanation.
- **Receipt and cart analysis**
  Grocery receipts can be scanned and analyzed as a full cart, producing per-item recommendations and summary risk counts, which are saved for later review.
- **History management**
  Users can view and delete past product scans and receipt analyses, providing basic control over stored data.
- **Cross-platform development**
  Development and testing focused on Android devices. iOS builds were functionally tested, but App Store deployment was not completed due to publishing costs.

7.2 Verification Approach

Testing focused on functional correctness and system stability rather than medical validation.

- **Unit testing**
  Key helper functions (parsing, normalization, threshold checks) were tested independently.
- **Integration testing**
  Backend API routes were tested to ensure correct interaction between authentication, file uploads, OCR, AI analysis, and database storage.
- **End-to-end testing**
  Full user flows were tested on real devices, including login, profile setup, blood test upload, scanning, feedback display, and history access.

The following table summarizes representative verification tests conducted during development.

ID	Module	Test	Expected Result
T1	Authentication	Register / Login	Users can register and log in; JWT token is issued; invalid credentials are handled gracefully.
T2	Authentication	Google Sign-In	Google token is verified; user is created or linked; onboarding proceeds correctly.
T3	Profile	Input validation	Invalid ranges (age, height, weight) are rejected; valid profiles are stored successfully.
T4	Blood Test	Multi-image upload	The user uploads multiple blood test images; OCR is executed across all images; a consolidated diagnosis summary is returned.
T5	Blood Test	History overwrite	Uploading a new blood test replaces the previous baseline for the user.
T6	AI Consultation	Single product	System returns SAFE/CAUTION/AVOID label with explanation within acceptable latency.
T7	AI Consultation	Cart analysis	Per-item recommendations and cart summary are returned and stored.
T8	History	Fetch and delete	History entries are loaded correctly; deletion updates both database and UI.
T9	Performance	Typical analysis	OCR and analysis complete within a practical time on a development machine.

7.3 Evaluation Criteria

System evaluation was based on qualitative and functional criteria aligned with the project's objectives.

- **OCR usability**
  OCR output is sufficiently accurate to identify key blood biomarkers and product strings, even if the raw text contains noise or formatting artifacts.
- **Explainability of results**
  Each recommendation includes a short, human-readable explanation tied to the user's profile and detected health constraints, supporting transparency and user trust.
- **System stability**
  Core flows operate reliably without crashes during repeated testing on real devices.
- **User data control**
  Users are able to delete stored history items. The backend design supports future extension to full account and data deletion if required.

8. Challenges and Solutions

During the Phase B development of CartGenie we encountered several technical and organizational challenges . This section describes the most significant issues and the engineering decisions taken to address them within the scope and constraints of an academic capstone project to summarize our complicated work and making hard decisions:

8.1 Technical Challenges

- **Machine Learning Over-Engineering (XGBoost)**
  The initial design included an XGBoost-based classifier for health and food-risk prediction. In practice, this approach required labeled medical datasets, extensive feature engineering, and ongoing model maintenance, making it unsuitable for the project's scope.
- **OCR Variability and Document Diversity**
  Blood test reports and grocery receipts vary widely in layout, language, and formatting. Mixed Hebrew and English text, tables, and non-standard fonts reduced OCR consistency and introduced noise into extracted data.
- **Hebrew OCR and RTL Handling**
  Processing Hebrew documents required explicit handling of right-to-left (RTL) text order. Without additional normalization, OCR output could not be reliably parsed.

- **Google Sign-In During Development**
  Google Sign-In required precise OAuth configuration, including SHA-1 and SHA-256 fingerprints. Full validation was only possible using Android build artifacts (APK or dev-client), complicating early testing.
- **iOS Distribution Constraints**
  Although the app functioned correctly on iOS during testing, official App Store distribution requires enrollment in the Apple Developer Program with an annual fee. This limited full iOS deployment during Phase B.
- **Cross-Platform File Handling**
  Handling PDFs and image uploads across Android and iOS introduced challenges related to file URIs, permissions, and MIME type differences.
- **Nutrition Table OCR Limitations**
  The original plan relied on scanning nutrition tables directly from product packaging. Inconsistent lighting and image quality significantly reduced OCR reliability, leading to a redesign.
- **Security and Privacy of Medical Data**
  Processing medical data required minimizing logging, securing communication, and controlling data retention to reduce privacy risks.
- **PDF Blood Test Processing Constraints (Encrypted/Protected Documents + Cloud Limitations)**
  During development, we initially attempted to support blood test uploads in PDF format. In practice, many medical PDF documents are protected/encrypted or use rendering methods that cause text extraction to fail (e.g., Hebrew terms such as “cholesterol” appear as unreadable symbols). To overcome this, we built a pipeline that rendered PDF pages into images and then applied OCR on the images. While this worked reliably on a local development server, deployment to Azure (student/lean tier) introduced limitations that prevented the required conversion/rendering libraries from running successfully.

8.2 Solutions and Workarounds

- **Hybrid Decision Model**
  XGBoost was replaced with a hybrid approach combining deterministic medical thresholds for blood test analysis and an LLM-based AI agent for product and cart evaluation. This reduced complexity while maintaining explainability.
- **RTL-Aware Parsing Logic**
  Custom backend logic was implemented to normalize Hebrew OCR output and handle RTL text correctly.
- **Google Sign-In Validation Strategy**
  Authentication was validated using Android APK builds and proper OAuth configuration. Setup steps and limitations were documented in the Maintenance Guide.

- **Deferred iOS Deployment**
  iOS App Store publication was documented as future work, allowing Phase B evaluation to focus on functional correctness rather than distribution.
- **Robust Backend Design**
  Defensive error handling was added to backend APIs to manage invalid uploads, OCR failures, and missing data gracefully.
- **Privacy-Conscious Data Handling**
  Sensitive logging was minimized, data retention was limited to essential information, and all client–server communication was secured, providing a foundation for future compliance improvements.
- ***Move to Image-Only Uploads + Multi-Image Support***
  To ensure reliability in production-like cloud deployment, we removed PDF upload support for blood tests and receipts. We attempted multiple approaches before making this decision: (1) containerizing the required libraries via Docker on Azure, which did not resolve the runtime constraints; and (2) converting PDFs on the client side, which proved impractical due to outdated or unmaintained free libraries and subscription requirements in maintained alternatives. As a final and robust solution, the system now accepts **image uploads only**. For blood tests, users can upload **multiple images at once** (e.g., several screenshots/pages), and the backend consolidates OCR results across all images into a single analysis. This change significantly improved deployment stability, reduced infrastructure complexity, and kept the OCR pipeline predictable across devices and environments.

9. Conclusions and Future Work

9.1 Conclusions

In the CartGenie project we successfully demonstrate an end-to-end decision-support system that translates personal medical data into practical, item-level guidance for grocery shopping. Development delivered a fully functional mobile application integrated with backend services, OCR-based document processing, AI assistance, persistent data storage, and an explainable recommendation workflow.

A key engineering achievement was the shift from nutrition-table OCR to barcode-based product identification, which significantly improved system reliability and robustness in real-world conditions. Another major design decision was replacing the originally planned XGBoost-based machine learning model with a hybrid approach combining deterministic medical rules and an AI-agent-based reasoning component. This change reduced system complexity, removed the need for labeled medical datasets, and improved explainability, while still meeting project goals and time constraints.

Overall, the project shows that a lightweight, well-structured decision-support system can effectively bridge the gap between clinical health information and everyday grocery shopping decisions without over-engineering.

9.2 Future Work

Several enhancements could further improve the system:

- **Improving OCR robustness**
  Apply advanced preprocessing techniques (e.g., deskewing, denoising, layout normalization) and introduce domain-specific templates for common Israeli blood test formats.
- **Advanced data privacy and compliance**
  Implement full GDPR-style data management, including complete account deletion, data export, and fine-grained retention controls.
- **Production deployment and hardening**
  Complete official deployment to Google Play and the Apple App Store, and add production-level features such as monitoring, rate limiting, and alerting.
- **Clinician-oriented mode**
  Introduce an optional clinician or caregiver mode that generates structured reports suitable for medical consultation, without providing medical diagnoses.

10. References

Google Developers. (n.d.). *ML Kit text recognition*. Google. Retrieved June 9, 2025, from https://developers.google.com/ml-kit/vision/text-recognition

Tesseract OCR. (n.d.). *Tesseract open source OCR engine*. GitHub. Retrieved June 9, 2025, from https://github.com/tesseract-ocr/tesseract

MongoDB Inc. (n.d.). *MongoDB manual*. MongoDB. Retrieved June 9, 2025, from https://www.mongodb.com/docs/

U.S. Department of Agriculture. (n.d.). *FoodData Central*. USDA. Retrieved June 9, 2025, from https://fdc.nal.usda.gov/

World Health Organization. (2023). *Healthy diet*. Retrieved June 9, 2025, from https://www.who.int/news-room/fact-sheets/detail/healthy-diet

Android Developers. (n.d.). *PdfRenderer | Android developers*. Google. Retrieved June 9, 2025, from https://developer.android.com/reference/android/graphics/pdf/PdfRenderer

Zeevi, D., Korem, T., Zmora, N., Israeli, D., Rothschild, D., Weinberger, A., Ben-Yacov, O., Lador, D., Avnit-Sagi, T., Lotan-Pompan, M., Suez, J., Mahdi, J. A., Matot, E., Malka, G., Kosower, N., Rein, M., Zilberman-Schapira, G., Dohnalová, L., Pevsner-Fischer, M., ... Segal, E. (2015). Personalized nutrition by prediction of glycemic responses. *Cell, 163*(5), 1079–1094. https://doi.org/10.1016/j.cell.2015.11.001

Wang, Y., Xue, H., Huang, Y., & Huang, L. (2019). Effectiveness of mobile health interventions on diabetes and obesity treatment and management: Systematic review of systematic reviews. *JMIR mHealth and uHealth, 7*(4), e10299. https://doi.org/10.2196/10299

Sinha, A. (2020, April). OCR with Tesseract, OpenCV and Python. *Nanonets Blog*. https://nanonets.com/blog/ocr-with-tesseract/

11. User Guide

11.1 Purpose

This User Guide describes how to operate the CartGenie mobile application. The guide is intended for end users and explains the main functionalities, workflows, and limitations of the system.

11.2 System Overview

CartGenie is a mobile application designed to assist users in making healthier grocery shopping decisions. The system combines user health profiles, product and receipt scanning, and AI-based feedback to provide personalized recommendations.

The application supports:

- User profile management
- Product barcode scanning
- Grocery receipt scanning using OCR
- Health-based feedback and recommendations
- History tracking of scanned products and receipts

11.3 Supported Platforms and Requirements

Supported Platform

- Android (recommended version: Android 10 or higher)

Device Requirements

- Camera (for barcode scanning and for blood test documents)
- Internet connection
- Storage access (for uploading receipts and blood tests)

11.4 User Registration and Login

Installation:

1. Enter the following folder: CartGenie APK
2. Download file ‘CG1.0.7_Ready.apk’
3. Enter local storage on the mobile phone.
4. Run the installation.
5. The app will appear at home screen.

Launch:

1. Launch the CartGenie application.
2. Select **Sign Up** (new user) or **Login** (existing user).
3. Authenticate using email and password or Google Sign-In (if available).
4. After successful authentication, the user is redirected to the Home screen.

11.5 Profile Management

1. From the Home screen, navigate to the profile management section.
2. Enter or update the following information:
    - Personal details (age, gender)
    - Body measurements (height, weight)
    - Medical conditions
3. Save the profile.

Profile data is stored in the backend database and used to personalize all analyses and recommendations.

11.6 Blood Test Upload (Optional)

Uploading a blood test improves personalization but is not mandatory.

1. Navigate to the **Upload Blood Test** screen.
2. Select one or more blood test images (image files only).

3. Upload the file and wait for processing.

The system performs OCR and analysis and displays identified health indicators. If applicable, suggested conditions may be added to the user profile.

11.7 Product Scan

1. Select **Scan Product** from the Home screen.
2. Grant camera permission if prompted.
3. Scan the product barcode.

The system retrieves product information, analyzes it together with the user profile, and displays a recommendation (SAFE / CAUTION / AVOID) with a short explanation. The result is saved to history.

11.8 Receipt Scan

1. Select **Scan Receipt**.
2. Scan or upload a receipt image.
3. Wait for OCR and analysis to complete.

The system analyzes all detected items as a shopping cart and displays per-item recommendations and a cart-level summary. Results are stored in receipt history.

11.9 History Management

1. Navigate to the **History** screen.
2. View past product scans and receipt analyses.
3. Select an entry to view details.
4. Delete history entries if required.

11.10 Permissions

The application requires the following permissions:

- Camera access (barcode scanning)
- File access (uploading receipts and blood tests)

For best OCR results, users should provide clear images with good lighting.

11.11 Data Handling and Privacy (High-Level)

The system stores:

- User account and profile data
- Product and receipt scan history
- Blood test analysis results

All feedback is informational only and does not replace professional medical advice.

11.12 Known Limitations

- OCR accuracy depends on image quality and document format.
- Product identification depends on database coverage.
- AI recommendations rely on correct user profile data.

11.13 Frequently Asked Questions

Q: Is uploading a blood test mandatory?
A: No. Uploading a blood test is optional but improves personalization.

Q: Why was a product not identified?
A: The barcode may not exist in the current database.

Q: Why is profile information required?
A: Recommendations are generated based on user health data and selected conditions.

12. Maintenance Guide

12.1 System Overview

CartGenie is implemented as a client–server system with clear separation between the mobile application and backend services.

The system consists of:

- **Mobile Client** – React Native + Expo (TypeScript)
- **Backend API** – Node.js + Express
- **Database** – MongoDB Atlas (accessed via Mongoose ODM)
- **OCR Engine** – Tesseract OCR (tesseract.js)
- **AI Feedback Engine** – Google Gemini API (product and cart consultation)

This architecture enables independent development and maintenance of the mobile and backend components while centralizing OCR and AI logic on the server.

12.2 Operating Environment

Backend Environment

- Node.js version **18+** (recommended)
- MongoDB Atlas cloud database
- Environment variables stored in `.env` (Does not exist in repository due to privacy)
- Local storage for temporary uploads (receipts, blood tests)

Mobile Environment

- Expo toolchain (Expo CLI / EAS)
- Android Studio (for Android builds)
- Xcode only if native iOS development is required

12.3 Installation Instructions

Option A: Local Development (Manual)

Clone the repository:

- `git clone https://github.com/CoderxX22/CartGenie.git`

Backend Setup

- `cd CartGenie_Backend`
- `npm install`

Create a `.env` file inside `CartGenie_Backend` and configure:

- `MONGO_URI=<CartGenie_mongo_atlas_connection_string> //(Contact with Nir Froiman to get the URI)`
- `GEMINI_API_KEY=<your_api_key>`
- `PORT=4000`

Start the backend server:

- `npm run dev`

Mobile Setup

- `cd ../CartGenie`
- `sudo npm install -g eas-cli`
- `eas login`
- `eas build:configure`
- `eas build --profile development --platform android`
- `npm install`
- `npx expo start`

Mobile Preparation

- Wait for the build to finish in the terminal.
- Scan the QR code provided in the terminal output using your phone's camera.
- Download and Install the APK file (Allow installation from unknown sources if asked).

Connecting

- Open the newly installed app on your phone.
- Scan the QR code shown in the terminal after 'npx expo start'.

Important networking note:
The mobile client and backend must be reachable on the same network or via a tunnel (e.g., ngrok).
Ensure the mobile app points to the correct backend base URL.

Important note:
If a new NATIVE library (like camera, location, etc.) is installed, you MUST create a new build (run 'eas build...' again)!

For simple JS/UI changes, just restarting the server is enough.

12.4 How to Extend or Modify the System

12.4.1 Update AI Recommendations (Gemini Logic)

- Modify prompt templates and parsing logic inside the backend AI agent module.
- Keep the output JSON schema stable to avoid breaking the mobile UI.

12.4.2 Update Blood Test Interpretation Logic

- Adjust threshold rules and regex-based parsing for biomarkers.
- When adding a new illness:
    - Implement detection logic in the backend analyzer.
    - Map diagnosis → illness flag on the mobile side.

12.4.3 Expand Product Database Coverage

- Add new products to MongoDB with barcode and nutritional fields.
- Ensure barcode values are consistent (string format, no spaces).

12.4.4 Improve Receipt Extraction

- Add image preprocessing (contrast enhancement, denoising).
- Expand barcode regex patterns.
- Implement fallback parsing (store name + line items) when barcodes are missing.

12.5 Project File System Structure

12.5.1 Repository Root Structure

- `CartGenie/`
- `├── CartGenie/ # Mobile application (React Native + Expo)`
- `├── CartGenie_Backend/ # Backend API (Node.js + Express)`
- `├── README.md`
- `├── package.json`
- `├── tsconfig.json`
- `└── Project Book documents`
- **CartGenie/** – Mobile client source code
- **CartGenie_Backend/** – Backend server source code
- **README.md** – Project overview and usage notes

12.5.2 Mobile Client Structure (CartGenie/)

- `CartGenie/`
- `├── app/ # Screens and navigation (Expo Router)`

- `|   ├── (auth)/`
- `|   ├── (tabs)/`
- `|   ├── home.tsx`
- `|   ├── scanProduct.tsx`
- `|   ├── scanReceipt.tsx`
- `|   ├── history.tsx`
- `|   ├── profile.tsx`
- `|   └── _layout.tsx`
- `├── components/                    # Reusable UI components`
- `├── services/                      # API communication logic`
- `├── storage/                       # AsyncStorage helpers`
- `├── assets/                        # Images, icons, fonts`
- `├── app.json`
- `├── package.json`
- `└── tsconfig.json`

Notes:

- File-based routing is implemented using `expo-router`.
- `services/` encapsulates backend API calls.
- `storage/` handles local persistence (tokens, session state).

12.5.3 Backend Server Structure (CartGenie_Backend/)

- `CartGenie_Backend/`
- `├── src/`
- `|   ├── controllers/              # API request handlers`
- `|   ├── routes/                   # Express route definitions`
- `|   ├── models/                   # Mongoose schemas`
- `|   ├── services/                 # OCR, AI agent, analysis logic`

- `|   ├── middleware/              # Authentication and error handling`
- `|   ├── utils/                   # Parsing helpers (including RTL logic)`
- `|   └── app.js                   # Express app setup`
- `├── uploads/                     # Temporary uploaded files`
- `├── .env                         # Environment variables (not committed)`
- `├── package.json`
- `└── server.js                    # Backend entry point`

12.5.4 Configuration Files

- `.env` (backend only) – stores secrets such as:
  - MongoDB URI
  - Gemini API key
  - JWT secrets
- `app.json` – Expo configuration
- `package.json` – dependency management for both projects

12.6 Package and Architecture Overview

Backend Packages (High-Level)

- express – REST API framework
- mongoose – MongoDB ODM
- multer – file uploads
- tesseract.js – OCR engine
- sharp – image preprocessing
- @google/generative-ai – Gemini AI integration

Mobile Packages (High-Level)

- expo, expo-router

- expo-camera – barcode scanning
- expo-document-picker – file uploads
- axios / fetch – API communication
- AsyncStorage – local state persistence

12.7 Maintenance Checklist

- Rotate API keys (Gemini, MongoDB) periodically
- Monitor MongoDB Atlas usage (storage and throughput)
- Clean temporary upload folders
- Validate API contracts after backend changes
- Perform regression testing:
    - Login / signup
    - Product scan
    - Receipt upload
    - History loading
    - Blood test upload

12.8 Troubleshooting (Maintenance)

- **Mobile cannot reach backend:** verify base URL, network/tunnel, CORS, firewall
- **OCR returns empty:** check image quality, Tesseract language packs
- **Gemini errors:** validate API key, rate limits, model availability
- **MongoDB errors:** verify IP whitelist, connection string, and network access