
**Bilkent University**

**Department of Computer Engineering**

**Senior Design Project**

*T2420*

*On the Shelf*

**Design Project Final Report**

**Team Members**

*22003354, Gülbera Tekin, gulbera.tekin@ug.bilkent.edu.tr*

*22103295, Defne Gürbüz, defne.gurbuz@ug.bilkent.edu.tr*

*22101848, İrem Hafızoğlu, irem.hafizoglu@ug.bilkent.edu.tr*

*22003049, Emirhan Büyükkonuklu, e.buyukkonuklu@ug.bilkent.edu.tr*

*22103772, Ümmügülsüm Sümer, ummugulsum.sumer@ug.bilkent.edu.tr*

**Supervisor**

*İbrahim Körpeoğlu*

**Course Instructors**

*Atakan Erdem*

*Mert Bıçakçı*

**02.05.2025**

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

# 1. Introduction

## 1.1 Overview

In today's fast-paced world, effective management of household tasks like grocery shopping has become essential, yet challenges related to planning, organization, and time management persist. Families and individuals often struggle to maintain kitchen inventory, plan meals, and create precise shopping lists tailored to their needs. While digital applications exist to address parts of this problem, there's still no comprehensive, user-friendly solution that integrates all these features into a cohesive experience. In addition, *On the Shelf*, which provides innovations in its field globally, takes its place in the application market to help Turkish users with 100% Turkish support.

*On the Shelf* introduces a grocery management app designed to simplify many tasks. The app combines receipt scanning and manual input to track kitchen inventory accurately. It also provides recipe suggestions based on available ingredients in their pantries and users' allergies, helping users maximize utility while reducing food waste.

One of the key innovations is the collaborative element: family members can work together by adding notes or updating the shopping list in real time, fostering transparent and collective planning. By merging advanced tools with practical features, the app aims to optimize resource utilization, minimize waste, and simplify grocery shopping. Ultimately, it offers a balanced, socially integrated solution for efficient and personalized household management, addressing gaps left by existing tools.

## 1.2 Competitors and Alternative Solutions

The current market for grocery management applications features several tools that offer fragmented solutions but omit critical integrated functionalities. Key competitors include:

**Whisk** [1]: Focuses on recipe curation and meal planning but lacks tools for inventory tracking or receipt scanning.

**KitchenPal** [2]: Specializes in pantry organization but lacks features like shared shopping lists or real-time inventory updates.

**Paprika** [3]: Functions as a recipe hub with basic inventory logging but does not utilize existing pantry stock to generate tailored meal recommendations.

**What's Left** [4]: Offers receipt scanning and inventory updates but excludes customization for allergy needs or eco-friendly usage tracking.

While these apps tackle isolated aspects of grocery management, none deliver a cohesive system that combines real-time inventory tracking, receipt scanning, adaptive meal planning, and collaborative list-building in a single interface. This gap highlights the need for a unified platform addressing all these areas seamlessly.

## 1.3 Why On the Shelf?

*On the Shelf* overcomes the limitations of existing solutions by addressing these four issues:

- **Automated Inventory Update**: Using OCR to automatically scan/upload receipts and update pantry contents.
- **Personalized Meal Suggestions**: Recipe recommendations based on available ingredients and allergies of users.
- **Collaborative Shopping Lists**: Shared lists and notes that multiple household members can update in real time.
- **Pantry Tracking**: All household members can see and update their pantry in real time.

Combining these features, *On the Shelf* establishes itself as a comprehensive, user-centric grocery management solution that enhances efficiency, reduces waste, and simplifies household grocery planning.

# 2. Requirements Details

## 2.1 Overview

*On the Shelf* is designed to streamline daily kitchen and household management tasks. The app allows users to scan or upload receipts to automatically update their pantry inventory, manage food allergies, select meal types, and receive personalized recipe recommendations. Users can save favorite recipes or explore new ones. The application supports a household system that enables multiple users to share a virtual kitchen, including a common pantry, shopping list, and note-taking space. All updates are synchronized in real-time to ensure consistency across household members. Users can either create a new household or join an existing one using a join code.

## 2.2 Functional Requirements

**User Registration and Authentication:** The app shall allow users to create an account, log in, and log out securely.

**Profile Management:** Users shall be able to manage allergy preferences, change their password, leave the household, and delete the account.

**Product Entry via OCR:** Users shall be able to scan receipts using OCR technology to automatically detect and store purchased items.

**Manual Product Addition:** Users shall have the option to manually add products that were not detected by OCR.

**Product Tracking:** The app shall track products added, deleted or updated to the user's shelf, including their quantities.

**Shopping List Management:** Users shall be able to create and manage shopping lists manually and write notes to each other based on their product and brand preferences.

**Recipe Suggestions:** The app shall suggest recipes based on the products available on the user's shelf and the user’s allergy preferences.

**Family Synchronization:** Users shall be able to form households, allowing shared access to shelf, shopping list and note data across multiple accounts.

**Language Support:** The app shall support Turkish.

## 2.3 Non-functional Requirements

**Performance:** The app shall respond to user inputs within 3 seconds under normal conditions.

**Scalability:** The system shall be able to accommodate increased numbers of users and data without significant degradation in performance.

**Usability:** The UI shall be intuitive and accessible, with support for Turkish users.

**Security:** User data, including authentication credentials and product data, shall be stored securely and protected from unauthorized access.

**Compatibility:** The app shall be compatible with both Android and iOS platforms.

**Maintainability:** The system shall be modular and easy to update or extend for future features.

**Availability:** The app shall be available 24/7 with minimal downtime.

**Data Accuracy:** OCR-detected products shall have a high accuracy rate so that users do not need to make too many manual changes.

**Synchronization:** Data shared between family members shall remain consistent and synchronized in real-time or with minimal delay.

## 2.4 Pseudo Requirements

- The app is developed using Flutter for cross-platform compatibility.
- Firebase is used for database, authentication, and cloud storage.
- Receipts are processed using OCR technology for text extraction.
- The initial version supports Turkish only.
- Spoonacular API is used for recipe suggestions.
- Azure Translation Service is used for recipe translation.

# 3. Final Architecture and Design Details

## 3.1 Overview

To better understand how the *OnTheShelf* system is structured and how it behaves in real world scenarios, a series of diagrams have been created to guide both design and development. These include visualizations of the core user interactions, system architecture, deployment setup, data flow, and key workflows like receipt scanning and recipe recommendation.

The diagrams cover a wide range of views. From high level system overviews to detailed activity flows helping us map out how different components connect and function together. They also illustrate how the mobile app communicates with cloud services and external APIs, including Spoonacular for recipes and Azure for translation.

By combining use cases, subsystem breakdowns, deployment architecture, hardware-software mapping, and step-by-step process diagrams, we aim to provide a clear, holistic picture of how the system works from both a technical and user centered perspective.

## 3.2 Use Case Diagram

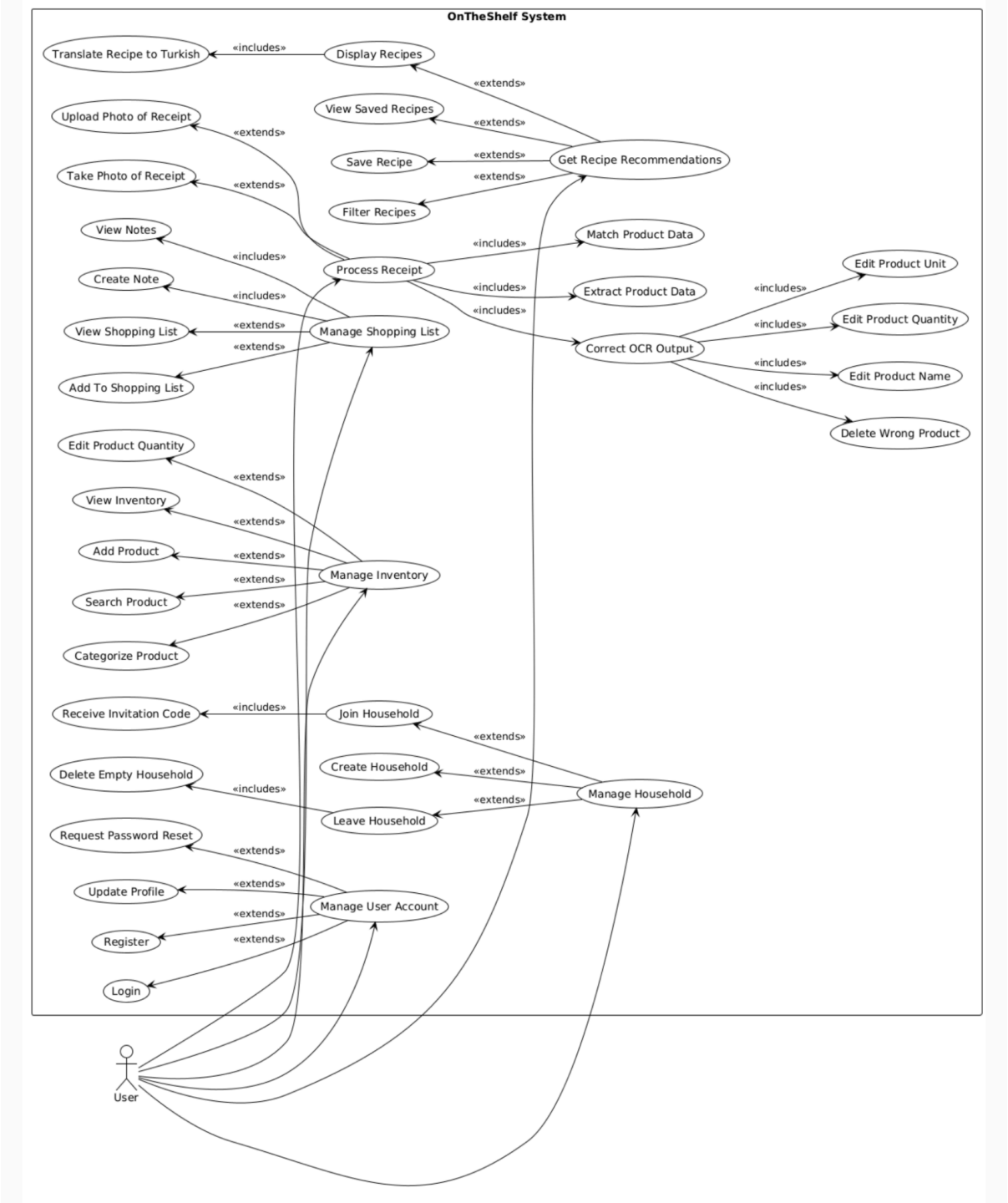



*Figure 1: Use Case Diagram*

## 3.3 Scenarios

**Scenario 1:** Registering an Account

**Actors**: User
**Preconditions**: User has installed the app.
**Steps**:

1. User opens the app and selects "Register".
2. User enters name, email, and password.
3. User confirms registration.
4. System creates the user account and logs the user in.

**Scenario 2:** Logging In

**Actors**: User
**Preconditions**: User has an existing account.
**Steps**:

1. User opens the app and selects "Login".
2. User enters email and password.
3. System authenticates the credentials and grants access.

**Scenario 3:** Updating Profile

**Actors**: User
**Preconditions**: User is authenticated.
**Steps**:

1. User navigates to profile settings.
2. User edits personal details (name, email, etc.).
3. User confirms changes.
4. System updates the user data in Firestore.

**Scenario 4:** Requesting Password Reset

**Actors**: User
**Preconditions**: User is not logged in and has forgotten the password.
**Steps**:

1. User selects "Forgot Password".

2. User enters their registered email.

3. User sets a new password.

**Scenario 5:** Creating a Household

**Actors**: User
**Preconditions**: User is not yet part of any household.
**Steps**:

1. User navigates to Household section and selects "Create Household".

2. User enters household name.

3. System creates the household and sets user as owner.

4. System generates an invitation code.

**Scenario 6:** Receiving Invitation Code

**Actors**: User
**Preconditions**: Household is created.
**Steps**:

1. System generates a unique code upon creation.

2. Code is displayed to the user to share with others.

**Scenario 7:**Joining a Household

**Actors**: User
**Preconditions**: User has received a valid invitation code.
**Steps**:

1. User opens “Join Household”.

2. User enters the invitation code.

3. System validates the code and adds user to the household.

**Scenario 8**: Leaving a Household

**Actors**: User
**Preconditions**: User is a current member of a household.
**Steps**:

1. User opens household settings and selects “Leave Household”.

2. System checks if user is the last member.

3. If yes, deletes the household; if not, simply removes the user.

**Scenario 9:** Deleting Empty Household

**Actors**: System
**Preconditions**: Last member leaves a household.
**Steps**:

1. System detects that no users remain.
2. Household is removed from Firestore.

**Scenario 10:** Viewing Inventory

**Actors**: User
**Preconditions**: User is in a household with existing inventory.
**Steps**:

1. User navigates to the Inventory tab.
2. System retrieves and displays product list with current stock info.

**Scenario 11:** Adding a Product to Inventory

**Actors**: User
**Preconditions**: User is part of a household.
**Steps**:

1. User navigates to the Inventory page.
2. User taps "Add Product".
3. User enters product name, category, unit, and quantity.
4. System adds the product to the household inventory.

**Scenario 12:** Editing Product Quantity

**Actors**: User
**Preconditions**: Inventory contains items.
**Steps**:

1. User navigates to Inventory.
2. User selects a product and chooses "Edit Quantity".
3. User updates the stock amount.
4. System saves the new quantity to the database.

**Scenario 13:** Categorizing a Product

**Actors**: User
**Preconditions**: Product is in inventory without a category.
**Steps**:

1. User selects a product.
2. User chooses a category.
3. System assigns the category and saves it.

**Scenario 14:** Searching for a Product

**Actors**: User
**Preconditions**: Inventory contains products.
**Steps**:

1. User enters a keyword in the Inventory search bar.
2. System filters and displays matching products.

**Scenario 15:** Viewing Shopping List

**Actors**: User
**Preconditions**: User is part of a household.
**Steps**:

1. User navigates to Shopping List page.
2. System displays all items currently on the list.

**Scenario 16:** Adding to Shopping List

**Actors**: User
**Preconditions**: User is part of a household.
**Steps**:

1. User selects “Add Item”.
2. User inputs product name and quantity.
3. System adds item to the shared shopping list.

**Scenario 17:** Creating a Note

**Actors**: User
**Preconditions**: User is viewing the shopping list.
**Steps**:

1. User selects “Add Note”.

2. User enters a note for the household.

3. System saves the note and makes it visible to all members.

**Scenario 18:** Viewing Notes

**Actors**: User
**Preconditions**: Notes exist in the system.
**Steps**:

1. User selects “View Notes” in the Shopping List page.

2. System displays all saved notes.

**Scenario 19:** Taking Photo of a Receipt

**Actors**: User
**Preconditions**: User has physical receipt.
**Steps**:

1. User opens the Receipt Processing feature.

2. User selects “Take Photo” and captures the image.

3. System uploads the image for OCR processing.

**Scenario 20:** Uploading a Photo of a Receipt

**Actors**:User
**Preconditions**: User has a saved image of a receipt.
**Steps**:

1. User selects “Upload Photo” from gallery.

2. Image is uploaded to the server.

3. System sends it for OCR extraction.

**Scenario 21:** Extracting Product Data from Receipt

**Actors**: User
**Preconditions**: A receipt image has been captured or uploaded.
**Steps**:

1. System receives the image.

2. System runs OCR to extract text.

3. Extracted lines are parsed into potential product data.

4. Results are sent back to the user for review.

**Scenario 22:** Matching Product Data

**Actors**: System
**Preconditions**: Extracted text exists from OCR.
**Steps**:

1. System compares OCR text to known products using fuzzy matching.
2. System suggests the closest product name, unit, and estimated quantity.
3. Matched results are prepared for user correction.

**Scenario 23:** Correcting OCR Output

**Actors**: User
**Preconditions**: Matched OCR output is available.
**Steps**:

1. User reviews detected products.
2. User selects products to edit, delete, or confirm.

**Scenario 24:** Deleting Wrongly Detected Product

**Actors**: User
**Preconditions**: User detected an incorrect product.
**Steps**:

1. User selects the line with an invalid product.
2. User taps "Delete".
3. System removes it from the final list.

**Scenario 25:** Editing Product Name

**Actors**: User
**Preconditions**: OCR result contains a misidentified product name.
**Steps**:

1. User selects the incorrect product.
2. User enters the correct name.
3. System saves the updated product name.

**Scenario 26:** Editing Product Quantity

**Actors**: User
**Preconditions**: Detected product has incorrect quantity.
**Steps**:

1. User selects the product.
2. User enters the correct quantity.
3. System updates the value.

**Scenario 27:** Editing Product Unit

**Actors**: User
**Preconditions**: Product unit is missing or incorrect.
**Steps**:

1. User selects the product.
2. User chooses or enters the correct unit
3. System updates the unit.

**Scenario 28:** Confirming and Updating Inventory After Receipt

**Actors**: User
**Preconditions**: OCR corrections are done.
**Steps**:

1. User taps "Confirm".
2. System adds/updates the corresponding products in inventory.
3. Inventory reflects new quantities.Ü

**Scenario 29:** Requesting Recipe Recommendations

**Actors**: User
**Preconditions**: Inventory and allergy data exist.
**Steps**:

1. User navigates to Recipes.
2. User selects cuisine preferences and household members.
3. System retrieves inventory and allergy info.
4. System sends request to Spoonacular API.

**Scenario 30:** Displaying Recipes

**Actors**: System
**Preconditions**: Recipe results are received.
**Steps**:

1. System receives recipes from Spoonacular.
2. System displays titles, images, ingredients, and preparation info.
3. User browses through the list.

**Scenario 31:** Saving a Recipe

**Actors**: User
**Preconditions**: User is viewing a recipe.
**Steps**:

1. User taps the “Save” icon or button.
2. System stores recipe in the user's saved list.

**Scenario 32:** Viewing Saved Recipes

**Actors**: User
**Preconditions**: User has saved one or more recipes.
**Steps**:

1. User navigates to “Saved Recipes” section.
2. System retrieves and displays saved recipes.

**Scenario 33:** Translating Recipe to Turkish

**Actors**: User
**Preconditions**: Recipe content is in English.
**Steps**:

1. Recipe text is sent to Azure Translation Service.
2. Translated content is received and displayed to the user.

**Scenario 34:** Managing User Account

**Actors**: User
**Preconditions**: App is installed.
**Steps**:

1. User either registers or logs in (covered in Scenarios 1 & 2).
2. User may update profile (Scenario 3) or request a password reset (Scenario 4).

**Scenario 35:** Managing Household

**Actors**: User
**Preconditions**: User has access to household features.
**Steps**:

1. User creates, joins, or leaves a household.

2. Invitation codes and cleanup of empty households are managed accordingly.

**Scenario 36:** Managing Inventory

**Actors**: User
**Preconditions**: Household exists.
**Steps**:

1. User adds, edits, searches, or categorizes inventory items.

2. Inventory is updated either manually or through receipt scanning.

**Scenario 37:** Managing Shopping List

**Actors**: User
**Preconditions**: Household exists.
**Steps**:

1. User adds items or notes to the shopping list.

2. Items and notes can be viewed and managed by any member of the household.

## 3.4 High-Level Architecture Diagram

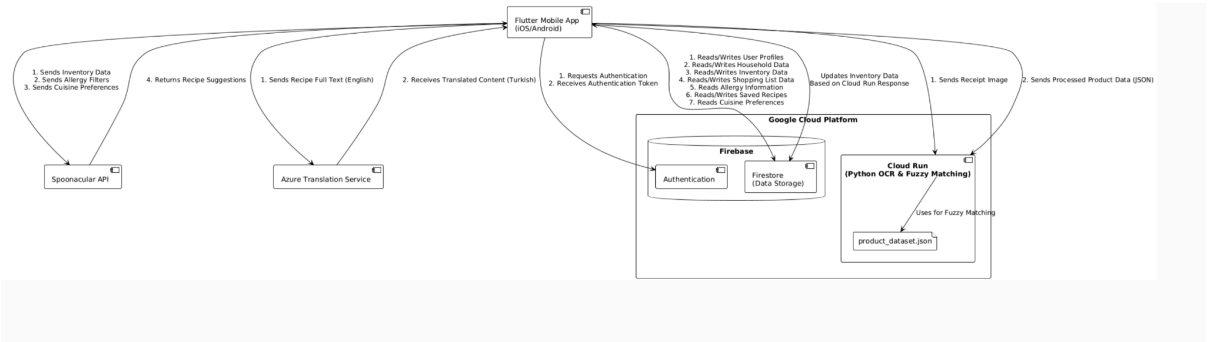



*Figure 2: High-Level Architecture Diagram*

The high-level architecture shows how different parts of the OnTheShelf system work together to support the app’s features. The mobile app connects to Firebase to handle things like login, household data, and inventory. When a user scans a receipt, the image is sent to Cloud Run, where it's processed and matched with known products. For recipe suggestions, the app uses Spoonacular, and if the recipes are in English, it sends them to Azure Translation so users can view them in Turkish.

## 3.5 Class Diagram

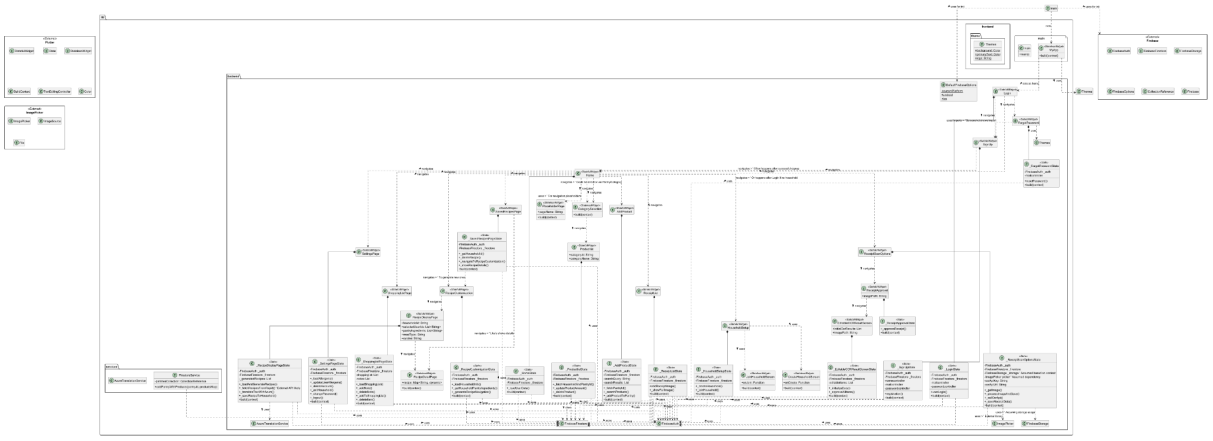


*Figure 3: Class Diagram*

Due to the high level of detail in our class diagram, it is not fully visible within the report. Therefore, we have included a link to our website where the full resolution PNG version of the diagram can be viewed clearly. Click here to go to our website.

The class diagram outlines the modular structure of *On the Shelf*, organized into main, backend, frontend, theme, and services. The main package initializes the app and configures Firebase. The backend contains key features like authentication, pantry management, receipt scanning with OCR, recipe suggestions, and shopping lists, all built using Flutter widgets and Firebase services. UI styling is handled in the frontend.theme package, while the services package includes helper classes like FirestoreService and AzureTranslationService. The diagram highlights how these components work together to support the app's goal of efficient grocery management and waste reduction.

## 3.6 Subsystem Decomposition Diagram

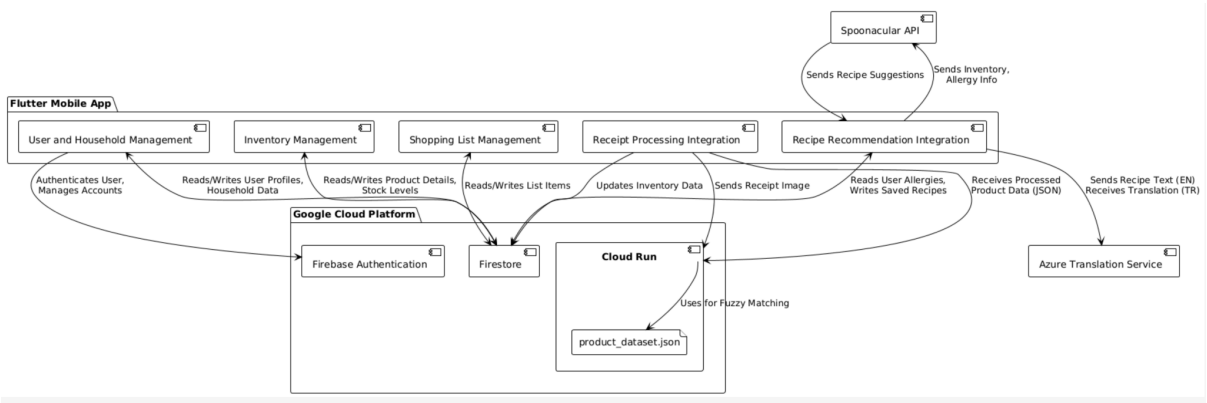



*Figure 4: Subsystem Decomposition Diagram*

The subsystem decomposition shows how the OnTheShelf mobile app is organized into focused modules, each handling a specific part of the user experience. The app separates concerns like user and household management, inventory tracking, shopping list handling, receipt scanning, and recipe recommendations. These modules connect to backend services on Google Cloud, including Firebase for authentication and data storage, and Cloud Run for processing receipt images. External APIs like Spoonacular provide recipe suggestions, while Azure Translation helps make recipe content accessible in Turkish.

## 3.7 Deployment Diagram

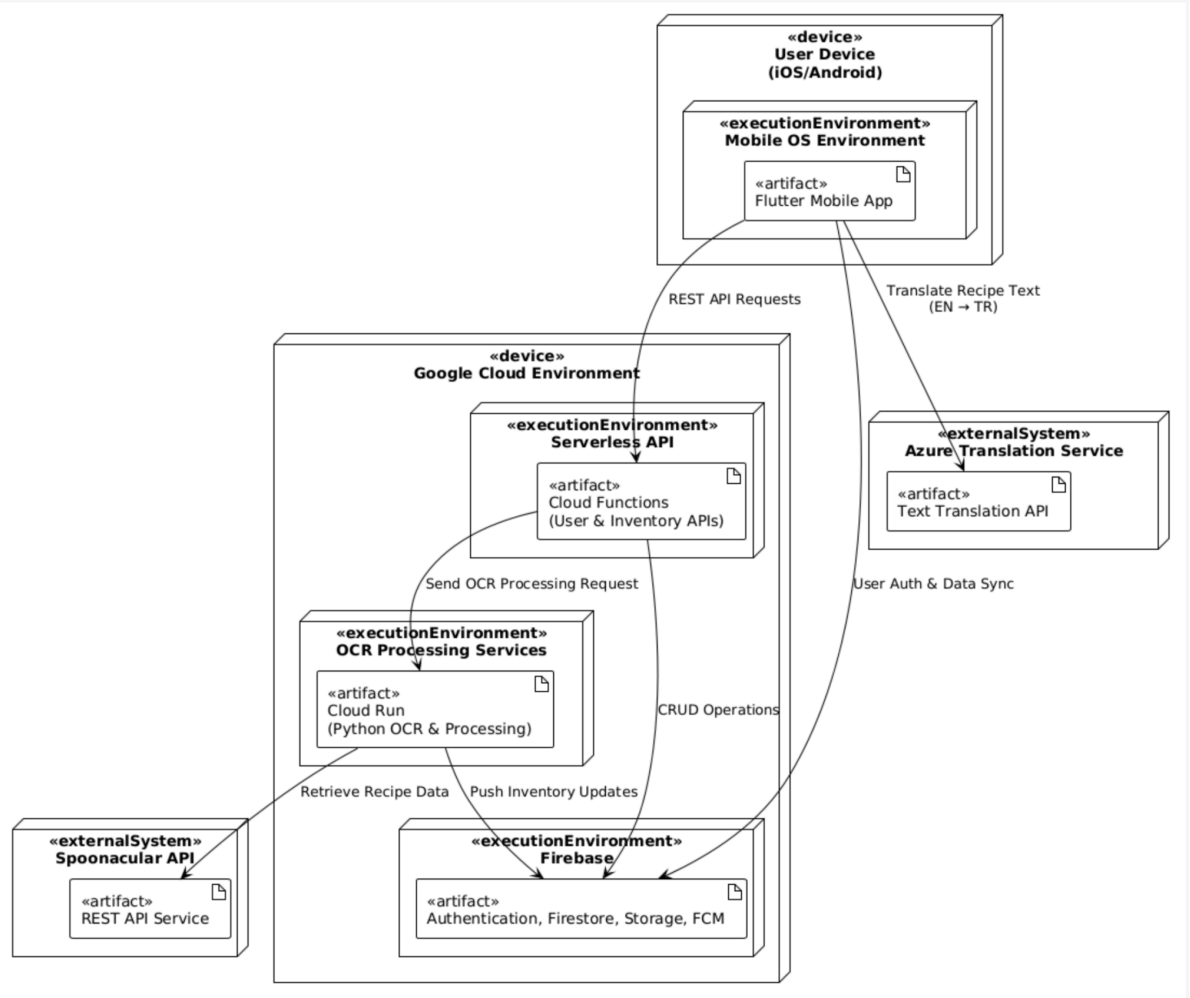



*Figure 5: Deployment Diagram*

The deployment diagram shows how OnTheShelf is distributed across mobile devices, cloud services, and external APIs. The Flutter app runs on user phones and connects to Google Cloud for backend logic, authentication, and data storage. OCR tasks are handled by a Cloud Run service, while recipe data and translations are fetched from Spoonacular and Azure.

## 3.8 Hardware/Software Mapping Diagram

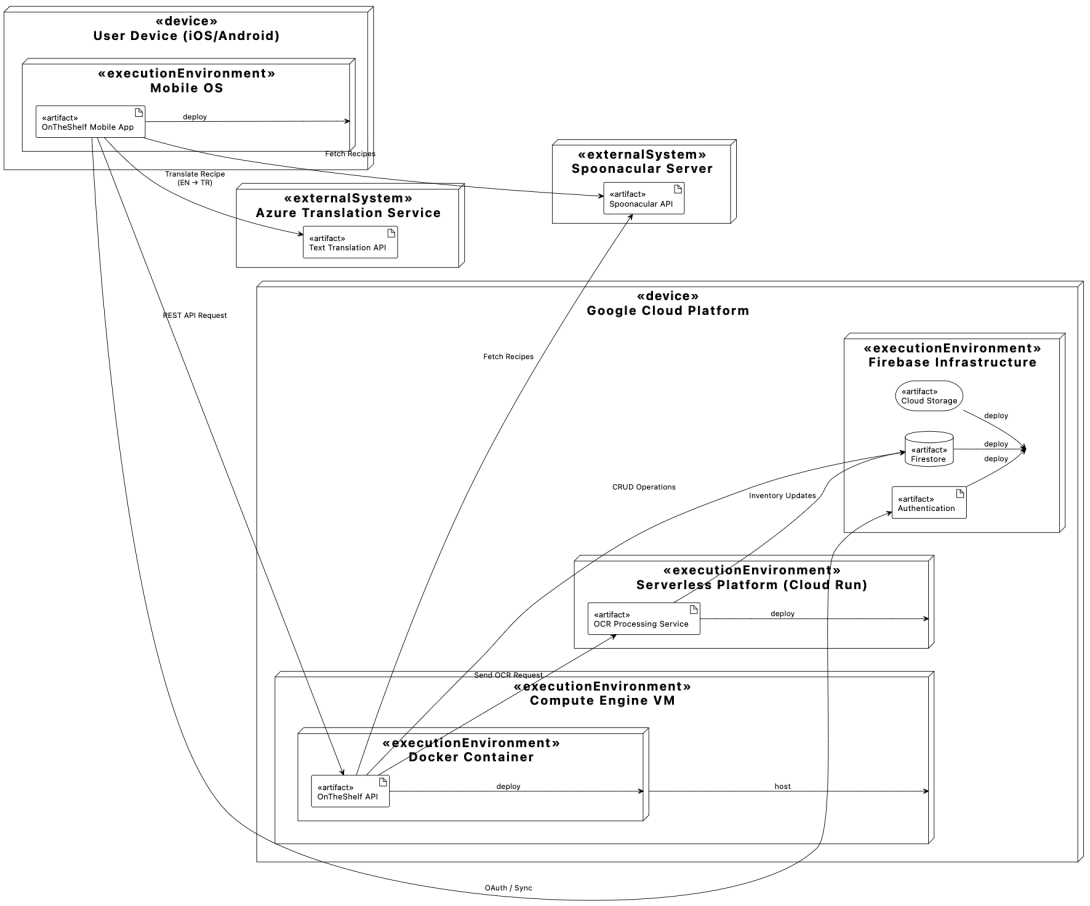



*Figure 6: Hardware Software Mapping Diagram*

The hardware software mapping diagram shows how OnTheShelf's software components are deployed across physical and virtual environments. The mobile app runs on iOS and Android devices, while backend services like OCR processing and APIs run on Google Cloud infrastructure. Firebase handles authentication and storage, and external APIs like Spoonacular and Azure Translation are accessed remotely to support recipe and language features.

# 3.9 Data Flow Diagram

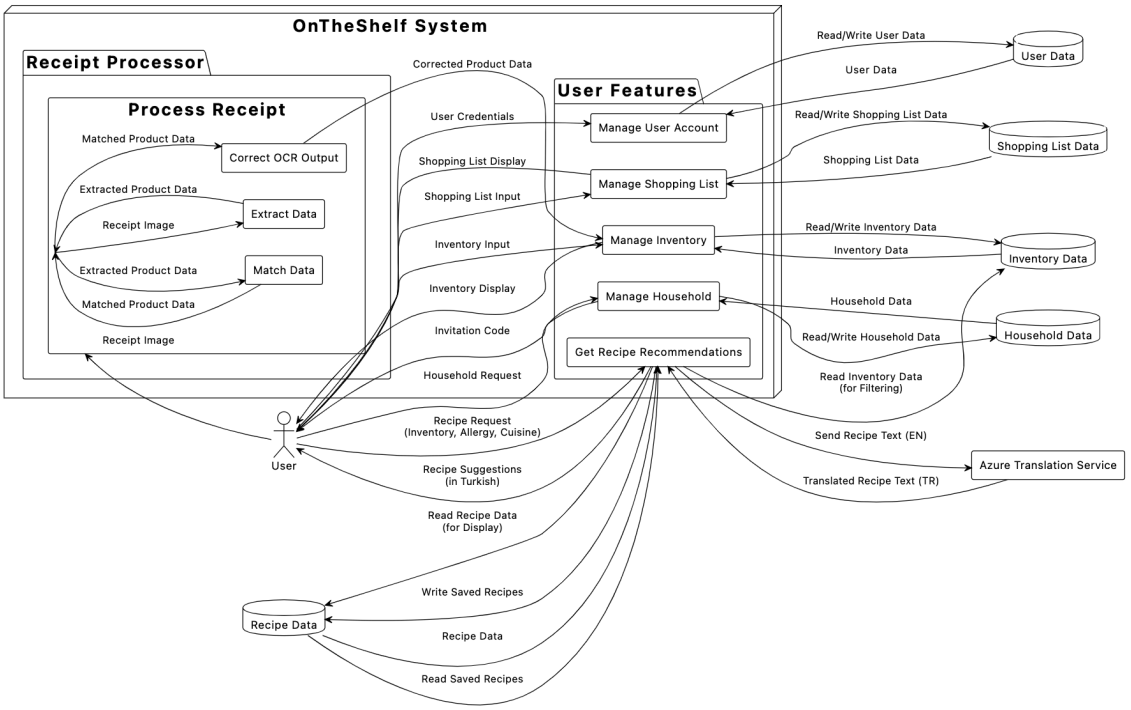



*Figure 7: Data Flow Diagram*

The data flow diagram illustrates how information moves through the OnTheShelf system. From user inputs to backend components and external services. It shows how user actions trigger updates to various data stores, how receipt images are processed through OCR and matched, and how recipe data is retrieved and translated before being displayed. The integration with Azure for translation and Spoonacular for recipe suggestions completes a clear, end-to-end data journey from input to intelligent output.

## 3.10 Activity Diagrams

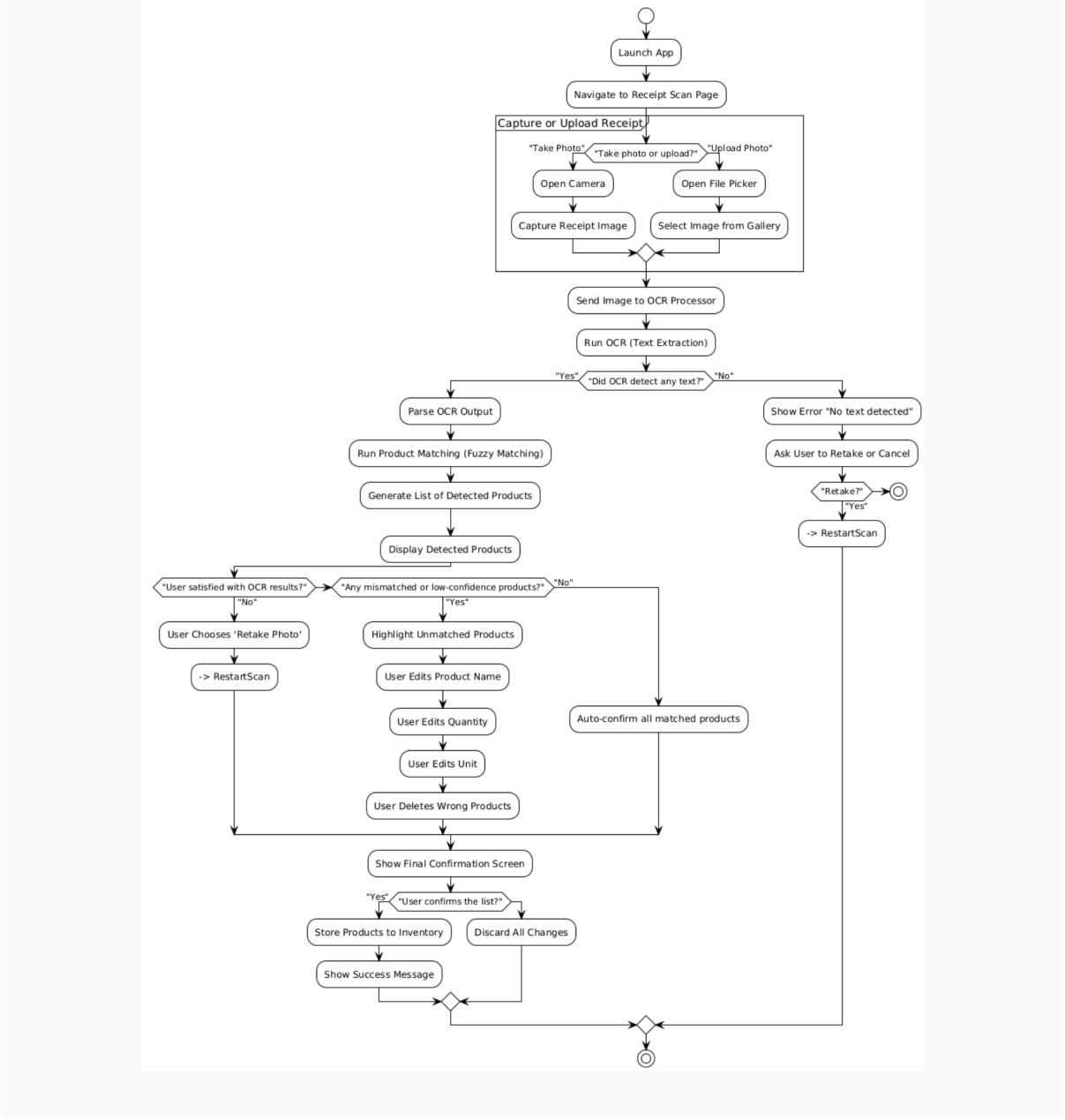



*Figure 8: Receipt Scanning Activity Diagram*

This activity diagram shows the full process of scanning or uploading a receipt, extracting and correcting product data, and updating the inventory based on user confirmation.

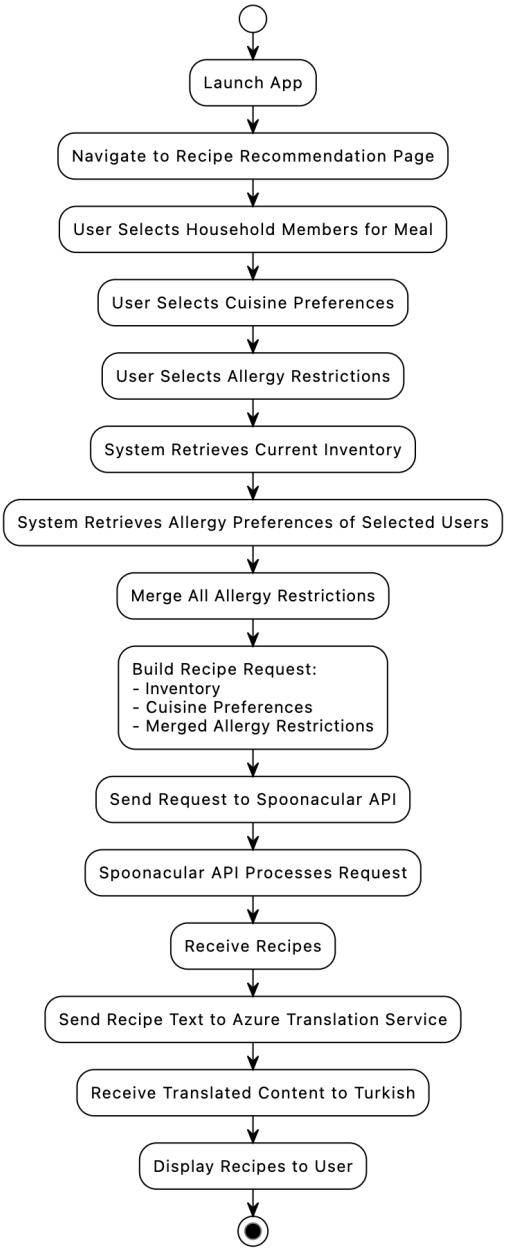



*Figure 9: Recipe Activity Diagram*

This activity diagram outlines the process of generating recipe recommendations based on inventory and preferences, translating them into Turkish, and displaying the results to the user.

# 4. Development/Implementation Details

## 4.1 Database

The application uses Google Firestore, a NoSQL document database, to store and manage its structured data. The Firestore database is organized into multiple collections such as users, products, categories, allergens, households, and pantries, each serving a distinct role in the application's data architecture.

The households collection, for example, is designed to support collaborative features within the app. Each document under this collection represents a household and contains fields such as householdPantryID, householdName, invitationCode, invitationCodeTimestamp, and an array of members, each member being identified by their user ID. Each household document can also contain subcollections like recipes, notes, receipts and shoppingList, enabling shared access and synchronization between members of the same household.

Firestore was selected for its real-time synchronization features, seamless integration with Flutter, and flexible schema structure. Its ability to handle nested data (subcollections), automatic scalability. With Firestore, any update to pantry items or shopping lists is reflected instantly across all user devices in the same household.

To ensure data quality, validation is implemented in the Flutter frontend before data is written to Firestore. Security rules are also defined to restrict unauthorized access and enforce user-level and household-level permissions. Firestore's managed infrastructure reduces the overhead of backend maintenance. This allows us to focus on feature development and improving the user experience.

## 4.2 Backend

The backend architecture of the application is based entirely on Firebase services, eliminating the need for traditional server-side code. We used Firebase Authentication to manage user registration and login, providing secure access control across users and households.

Most of the backend logic is handled directly through Firestore operations embedded within the Flutter frontend, which performs basic CRUD (Create, Read, Update, Delete) operations. No custom HTTP requests (except for translation service) or JavaScript code were written by us. The FirestoreService centralized common Firestore operations, such as saving and updating product data across pantry collections, ensuring efficient data management. We relied on cloud-integrated services to extend functionality.

This serverless approach allowed us to focus on product features rather than infrastructure, leveraging Dart and Flutter throughout for both frontend and data-handling logic. Firebase’s tight integration with Flutter helped streamline development and maintain a consistent codebase.

The system incorporated some services and utilities to enhance its functionality. The Spoonacular API is integrated to provide personalized recipe recommendations based on user allergies, meal types, and cuisine preferences, offering tailored suggestions aligned with pantry contents and dietary restrictions. Additionally, the AzureTranslationService is used to translate recipe data from Spoonacular into Turkish, utilizing Azure's translation API to handle the translation of recipe content via HTTP calls, ensuring accessibility for Turkish-speaking users.

## 4.3 Frontend

The frontend of *On the Shelf* was developed using the Flutter framework, which enabled the team to build a single, unified codebase that runs on both Android and iOS platforms. The primary responsibility of the frontend was to present data to users in a clear, visually appealing, and responsive manner, while also handling user interactions smoothly.

Flutter SDK was chosen for its robust cross-platform capabilities and expressive UI components. Dart, the programming language used with Flutter, was utilized to construct the UI and manage state efficiently. The architecture was based on a widget hierarchy, leveraging both stateless and stateful widgets to promote code reusability and maintain consistency throughout the interface.

The application featured several key screens and components. The Home Page displayed a greeting message alongside the app logo and the user's name, with navigation buttons leading to core features such as settings, receipts, recipes, the shopping list, and the pantry. The Shopping List Page allowed users to manage items they needed to purchase, including their quantity, note-taking, and real-time synchronization with household members. The Receipt Scanning Page integrated OCR functionality to automatically extract product names from scanned or uploaded receipts, streamlining pantry updates. The Recipe Suggestions Page offered meal ideas based on available pantry items and the selected meal type, with filtering that excluded allergen-containing recipes. The Pantry Page categorized and listed all available items along with their quantities. Finally, the Settings Page allowed users to edit their password, manage allergies, toggle between light and dark themes, generate or join a household code, and perform critical actions such as deleting their account, leaving the household, or clearing all pantry data.

Design-wise, the frontend adhered to Material Design principles to ensure a familiar and user-friendly interface. Responsive design was implemented to guarantee that the app rendered correctly on various screen sizes and resolutions. Color themes were carefully defined and applied consistently across components, such as buttons, to maintain a coherent visual identity. Theme support was also included, offering both light and dark modes. Users could toggle between them

based on their preferences, and their selection was stored locally to ensure persistence between sessions.

## 4.4 Cloud Run

This project integrates a Python based OCR and fuzzy product matching module into a backend infrastructure using Google Cloud Run. The primary role of this system is to receive receipt images from a Flutter mobile application, extract meaningful product and quantity data from the image using Tesseract OCR, and return structured information in JSON format. The entire pipeline is deployed serverlessly, ensuring scalability, zero infrastructure management.

The backend is developed in Python and utilizes the Flask web framework to expose an HTTP endpoint. The OCR is handled using the Tesseract engine via the pytesseract wrapper, configured with Turkish language support. Image preprocessing is applied using the Pillow library, which includes grayscale conversion and automatic contrast enhancement. Fuzzy string matching is implemented with fuzzywuzzy. The backend logic is containerized using Docker with a base image and deployed to Google Cloud Run using Artifact Registry for storing and managing container images.

The backend service is built and pushed to Google Artifact Registry using the following commands:

```
docker build -t ocr-server .
docker tag ocr-server us-central1-docker.pkg.dev/on-the-shelf-2025/ocr1/ocr-server
docker push us-central1-docker.pkg.dev/on-the-shelf-2025/ocr1/ocr-server
```

Once pushed, the container is deployed to Cloud Run using:

```
gcloud run deploy ocr-server \
 --image=us-central1-docker.pkg.dev/on-the-shelf-2025/ocr1/ocr-server \
 --platform=managed \
 --region=us-central1 \
 --allow-unauthenticated
```

The service is then publicly accessible via HTTPS. The endpoint accepts multipart/form-data POST requests and returns a structured JSON response. Temporary image files used in processing are deleted immediately after use.

The server exposes a single POST endpoint: /process_receipt. The mobile application sends a .jpg file using a multipart request. The image is saved temporarily, processed for OCR and product matching, and then deleted. The result is a JSON list of detected products with associated quantities and units, ready to be displayed in the Flutter UI.

Each response object includes three fields: product (string), quantity (float), and unit (string). If the system cannot determine a quantity or unit, it falls back to 0 and "unknown", respectively.

The core of the backend is the matching logic that attempts to identify grocery products from noisy OCR output. After OCR processing, each line of text is normalized by lowercasing and removing all non-alphabetical characters except Turkish letters and spaces. The cleaned line is then evaluated using a two step matching strategy.

The system first attempts to match the entire cleaned line against a list of 342 canonical Turkish grocery product names loaded from dataset.json. If a match is found with a similarity score of at least 85, it is accepted. However, stricter rules are applied to shorter matches: if the candidate product name has fewer than 5 characters, a minimum score of 95 is required to reduce false positives. If this condition is met, the product is added to the result set and sliding matching is skipped.

If the full line match fails to meet the threshold, the system generates sliding substrings of length 5, 6, and 7 from the cleaned line. Each substring is independently compared to the dataset entries. The best scoring match is retained. For sliding matches, a slightly relaxed threshold of 80 is applied. However, if the best match is shorter than 5 characters, the stricter 95 threshold still applies. This step allows the system to recover from partial OCR errors or formatting inconsistencies like splitting of words or missing letters.

Both matching stages are designed to be tolerant of common OCR issues like, merged words, or noise characters. The fuzzywuzzy algorithm ensures that edit distance, rather than exact spelling, drives the matching decision.

In parallel with product detection, the system attempts to extract numeric quantity and measurement units from each original OCR line. Two regular expressions are used to capture both space-separated and joined patterns like "300 g", "500ml", "6x1L". Units supported include grams (g, gr, kg), volume (ml, lt, l), and count-based identifiers (adet, paket, kutu, x). If a valid pattern is found, the numeric quantity is parsed as a float and normalized. If parsing fails or no unit is found, the defaults of quantity 0 and unit of “unknown" are applied.

Once deployed, the Cloud Run service runs continuously and autoscaling is handled by Google Cloud. Cold starts are minimal, and each request spins up an isolated container that runs the Flask server. The endpoint is publicly accessible to simplify mobile integration without the need for user authentication.

The Flutter mobile application captures a receipt image and encodes it as multipart/form-data. The image is sent as a file field named receipt_image. The backend responds with a JSON array of matched products and their quantities, which the app uses to populate the inventory update screen for user confirmation and editing.

The Tesseract OCR engine, despite being trained for Turkish, often misinterprets characters under real-world conditions. Frequent failure modes observed during testing are as follows. OCR tends to misread S as 5, G as 6, O as 0, B as 8, and vice versa. For example, “sosis” may appear as “5o5is” in OCR output. Turkish characters like ğ, ç, ı, and ş are sometimes converted to their ASCII equivalents or omitted entirely, e.g., “yoğurt” being scanned as “yogurt” or “yourt”. Words may be joined without spaces or split arbitrarily across multiple lines, such as “meyveliyogurt” instead of “meyveli yoğurt”. Borders, logos, and formatting characters (x, *, -, :) may be introduced into the OCR lines and affect parsing and matching. Also, the unstandardized receipts from across several companies make it difficult for OCR to achieve 100% accuracy.

# 5. Test Cases and Results

| **Test ID** | TC01 | **Category** | Functional | **Severity** | High |
|---|---|---|---|---|---|
| **Objective** | Verify that users can add an item to the pantry | | | | |
| **Steps** | 1. Open the app<br>2. Navigate to the “Mutfağımdakiler”<br>3. Select a category<br>4. Click on the add button on the top right of the page<br>5. Enter item details and save | | | | |
| **Expected** | The item is successfully added to the pantry. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC02 | **Category** | Functional | **Severity** | High |
|---|---|---|---|---|---|
| **Objective** | Verify that users can delete an item from the pantry | | | | |
| **Steps** | 1. Open the app<br>2. Navigate to the “Mutfağımdakiler”<br>3. Select a category<br>4. Select an item<br>5. Click "Delete" | | | | |
| **Expected** | The item is successfully removed from the pantry. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC03 | **Category** | Functional | **Severity** | Medium |
|---|---|---|---|---|---|
| **Objective** | Verify that users can edit an item in the pantry | | | | |
| **Steps** | 1. Open the app<br>2. Navigate to the “Mutfağımdakiler”<br>3. Select a category<br>4. Select an item<br>5. Edit details and save | | | | |
| **Expected** | The item details are updated successfully . | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC04 | **Category** | Functional | **Severity** | High |
|---|---|---|---|---|---|
| **Objective** | Verify that users can filter items by category | | | | |
| **Steps** | 1. Open the app<br>2. Navigate to the “Mutfağımdakiler”<br>3. Select a category | | | | |
| **Expected** | Only items matching the selected category are displayed. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC05 | **Category** | Functional | **Severity** | Low |
|---|---|---|---|---|---|
| **Objective** | Verify that users can search for items in the pantry | | | | |
| **Steps** | 1. Open the app<br>2. Navigate to the “Mutfağımdakiler”<br>3. Select a category<br>4. Enter an item name in the search bar<br>5. View results | | | | |
| **Expected** | Only relevant items appear in search results. | | | | |
| **Result/Notes** | Planned to be added in the upcoming version. | | | | |

| **Test ID** | TC06 | **Category** | Functional | **Severity** | High |
|---|---|---|---|---|---|
| **Objective** | Verify that users can input their allergies | | | | |
| **Steps** | 1. Open the app<br>2. Navigate to the settings<br>3. Click manage allergens<br>4. Add allergies<br>5. Save | | | | |
| **Expected** | Allergies are successfully saved. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC07 | **Category** | Functional | **Severity** | High |
|---|---|---|---|---|---|
| **Objective** | Verify that generated recipes exclude allergic ingredients | | | | |
| **Steps** | 1. Set allergies<br>2. Navigate to “Tarifler”<br>3. Generate a new recipe | | | | |
| **Expected** | Generated recipes do not include allergens. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC08 | **Category** | Functional | **Severity** | Hightu |
|---|---|---|---|---|---|
| **Objective** | Verify that users can save recipes | | | | |
| **Steps** | 1. Open recipes section<br>2. Click "Tarifi Kaydet"<br>3. Enter details and save | | | | |
| **Expected** | The recipe is saved successfully. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC09 | **Category** | Functional | **Severity** | High |
|---|---|---|---|---|---|
| **Objective** | Verify that the app syncs data correctly | | | | |
| **Steps** | 1. Add an item to the pantry<br>2. Restart the app | | | | |
| **Expected** | The item remains in the pantry after restart. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC10 | **Category** | Non-Functional | **Severity** | High |
|---|---|---|---|---|---|
| **Objective** | Verify app loading time is under 3 seconds | | | | |
| **Steps** | 1. Open the app | | | | |
| **Expected** | The app loads within acceptable time. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC11 | **Category** | Non-Functional | **Severity** | Medium |
|---|---|---|---|---|---|
| **Objective** | Verify app response time for adding an item | | | | |
| **Steps** | 1. Add an item to the pantry<br>2. Measure response time | | | | |
| **Expected** | The item is added within acceptable response time. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC12 | **Category** | Functional | **Severity** | High |
|---|---|---|---|---|---|
| **Objective** | Verify that users can log in using Firebase authentication | | | | |
| **Steps** | 1. Open the app<br>2. Enter valid credentials<br>3. Click "Giriş Yap" | | | | |
| **Expected** | User successfully logs in. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC13 | **Category** | Functional | **Severity** | Medium |
|---|---|---|---|---|---|
| **Objective** | Verify that users cannot log in with incorrect credentials | | | | |
| **Steps** | 1. Open the app<br>2. Enter invalid credentials<br>3. Click "Giriş Yap" | | | | |
| **Expected** | Error message is displayed. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC14 | **Category** | Functional | **Severity** | High |
|---|---|---|---|---|---|
| **Objective** | Verify that users can reset their password | | | | |
| **Steps** | 1. Click "Şifremi Unuttum"<br>2. Enter email<br>3. Check email for reset link | | | | |
| **Expected** | Password reset email is received. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC15 | **Category** | Functional | **Severity** | Medium |
|---|---|---|---|---|---|
| **Objective** | Verify that users can log out | | | | |
| **Steps** | 1. Open settings<br>2. Click "Çıkış Yap" | | | | |
| **Expected** | User is logged out and returned to login screen. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC16 | **Category** | Functional | **Severity** | High |
|---|---|---|---|---|---|
| **Objective** | Verify that users can update their profile information | | | | |
| **Steps** | 1. Open profile settings<br>2. Edit details and save | | | | |
| **Expected** | Profile updates successfully. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC17 | **Category** | Functional | **Severity** | High |
|---|---|---|---|---|---|
| **Objective** | Verify that the database updates in real-time | | | | |
| **Steps** | 1. Open the pantry on two devices<br>2. Add an item on Device 1<br>3. Check Device 2 | | | | |
| **Expected** | The new item appears instantly on Device 2. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC18 | **Category** | Functional | **Severity** | Low |
|---|---|---|---|---|---|
| **Objective** | Verify that users can sort pantry items by name | | | | |
| **Steps** | 1. Open the pantry<br>2. Click on the sort button<br>3. Select "İsme Göre Sırala" | | | | |
| **Expected** | Items are sorted alphabetically. | | | | |
| **Result/Notes** | Planned to be added in the upcoming version. | | | | |

| **Test ID** | TC19 | **Category** | Functional | **Severity** | Medium |
|---|---|---|---|---|---|
| **Objective** | Verify that users can view a history of consumed items | | | | |
| **Steps** | 1. Open the app<br>2. Navigate to "Tüketim Geçmişi" | | | | |
| **Expected** | A list of previously used items is displayed. | | | | |
| **Result/Notes** | Planned to be added in the upcoming version. | | | | |

| **Test ID** | TC20 | **Category** | Non-Functional | **Severity** | Medium |
|---|---|---|---|---|---|
| **Objective** | Verify that the app does not drain battery excessively | | | | |
| **Steps** | 1. Use the app continuously for an hour<br>2. Check battery consumption | | | | |
| **Expected** | Battery usage remains within acceptable limits. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC21 | **Category** | Non-Functional | **Severity** | High |
|---|---|---|---|---|---|
| **Objective** | Verify that the app works on different screen sizes | | | | |
| **Steps** | 1. Open the app on multiple devices | | | | |
| **Expected** | The UI adjusts correctly for all screen sizes. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC22 | **Category** | Non-Functional | **Severity** | Medium |
|---|---|---|---|---|---|
| **Objective** | Verify that the app functions properly in dark mode | | | | |
| **Steps** | 1. Open settings<br>2. Enable dark mode | | | | |
| **Expected** | The app appears correctly in dark mode. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC23 | **Category** | Functional | **Severity** | Medium |
|---|---|---|---|---|---|
| **Objective** | Verify that pantry data only visible for users in same household | | | | |
| **Steps** | 1. Log in with User u1 that is part of Household h1<br>2. Add an item to the pantry<br>3. Log out and log in with User u1 that is not a part of Household h1 | | | | |
| **Expected** | Pantry data from User u1 is not visible in User u2. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC24 | **Category** | Functional | **Severity** | Medium |
|---|---|---|---|---|---|
| **Objective** | Verify that users can clear all pantry data | | | | |
| **Steps** | 1. Open settings<br>2. Click "Mutfağımdakileri sil" | | | | |
| **Expected** | All pantry items are removed. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC25 | **Category** | Non-Functional | **Severity** | High |
|---|---|---|---|---|---|
| **Objective** | Verify that the app handles large amounts of data efficiently | | | | |
| **Steps** | 1. Add all the items in the database to the pantry | | | | |
| **Expected** | The app remains responsive and does not crash. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC26 | **Category** | Functional | **Severity** | High |
|---|---|---|---|---|---|
| **Objective** | Verify that the app extracts product details accurately from a clear receipt image | | | | |
| **Steps** | 1. Open the app and<br>2. Select the "Fiş Okut" option<br>3. System processes the receipt and extracts product details<br>4. System displays extracted data for user confirmation<br>5. User confirms the data | | | | |
| **Expected** | The inventory is updated. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC27 | **Category** | Functional | **Severity** | Medium |
|---|---|---|---|---|---|
| **Objective** | Verify that users receive an error for an invalid receipt scan | | | | |
| **Steps** | 1. Open the app<br>2. Click "Fiş Okut"<br>3. Scan an invalid receipt | | | | |
| **Expected** | An error message is displayed. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC28 | **Category** | Functional | **Severity** | Medium |
|---|---|---|---|---|---|
| **Objective** | Verify that the system correctly identifies and adds the deleted items to the shopping list | | | | |
| **Steps** | 1.Ensure some products in the inventory are deleted<br>2.Open the app and navigate to the "Alışveriş Listesi" section<br>3.Verify if deleted items appear automatically in the list | | | | |
| **Expected** | The shopping list includes low-stock items with the correct quantities. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC29 | **Category** | Functional | **Severity** | Medium |
|---|---|---|---|---|---|
| **Objective** | Verify that users can remove incorrectly suggested items | | | | |
| **Steps** | 1.Open the "Alışveriş Listesi" section<br>2.Select an item from the list<br>3.Click on the remove option | | | | |
| **Expected** | The item is successfully removed from the list. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC30 | **Category** | Functional | **Severity** | High |
|---|---|---|---|---|---|
| **Objective** | Verify that the OCR function works with receipts of different layouts and font styles | | | | |
| **Steps** | 1.Collect receipts from different stores with varying formats<br>2.Scan each receipt using the app | | | | |
| **Expected** | The OCR extracts and interprets text accurately across different receipt formats. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC31 | **Category** | Functional | **Severity** | High |
|---|---|---|---|---|---|
| **Objective** | Verify the OCR can handle lengthy receipts without truncation | | | | |
| **Steps** | 1.Scan a long grocery receipt with more than 20 items | | | | |
| **Expected** | The system extracts all items correctly without missing entries. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC32 | **Category** | Functional | **Severity** | High |
|---|---|---|---|---|---|
| **Objective** | Verify that the system correctly distinguishes between kitchen-related and non-kitchen items | | | | |
| **Steps** | 1.Scan a receipt containing both kitchen and non-kitchen products | | | | |
| **Expected** | The system filters and only adds kitchen-related items to inventory. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC33 | **Category** | Functional | **Severity** | High |
|---|---|---|---|---|---|
| **Objective** | Verify that the system updates quantities instead of duplicating item | | | | |
| **Steps** | 1.Scan a receipt containing items already in the inventory | | | | |
| **Expected** | The system updates existing product quantities instead of adding duplicates. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC34 | **Category** | Functional | **Severity** | Medium |
|---|---|---|---|---|---|
| **Objective** | Verify OCR performance on faded or low-contrast receipts | | | | |
| **Steps** | 1.Scan a faded receipt with light ink | | | | |
| **Expected** | The system accurately extracts text or prompts the user for manual verification. | | | | |
| **Result/Notes** | Failed - Tested on 01.05.2025. | | | | |

| **Test ID** | TC35 | **Category** | Functional | **Severity** | High |
|---|---|---|---|---|---|
| **Objective** | Verify that extracted product names are matched correctly with known kitchen product datasets | | | | |
| **Steps** | 1.Scan a receipt and allow the system to process it | | | | |
| **Expected** | The system matches product names accurately with its kitchen product database. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC36 | **Category** | Functional | **Severity** | High |
|---|---|---|---|---|---|
| **Objective** | Verify that the system recommends recipes using current inventory items | | | | |
| **Steps** | 1.Navigate to the “Tarifler” section<br>2.Check suggested meals | | | | |
| **Expected** | Recipes use available ingredients. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC37 | **Category** | Functional | **Severity** | Medium |
|---|---|---|---|---|---|
| **Objective** | Verify users can join a household using the household ID | | | | |
| **Steps** | 1.Navigate to "Eve Katıl"<br>2.Enter the household ID | | | | |
| **Expected** | The user joins the household. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC38 | **Category** | Functional | **Severity** | High |
|---|---|---|---|---|---|
| **Objective** | Verify household creation works correctly | | | | |
| **Steps** | 1.Navigate to "Household Management"<br>2.Click "Create Household"<br>3.Enter household name | | | | |
| **Expected** | A new household is created. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC39 | **Category** | Functional | **Severity** | High |
|---|---|---|---|---|---|
| **Objective** | Verify that users in the same household have a shared pantry | | | | |
| **Steps** | 1. User A logs into the app and adds an item to the pantry<br>2. User B logs into the same household account on a different device<br>3. User B checks the pantry | | | | |
| **Expected** | The pantry data is identical for both users, and the item added by User A is visible to User B | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC40 | **Category** | Functional | **Severity** | High |
|---|---|---|---|---|---|
| **Objective** | Verify that users can delete their accounts successfully | | | | |
| **Steps** | 1. User logs into the app<br>2. User navigates to the account settings page<br>3. User selects the "Hesabı Sil" option<br>4. User confirms account deletion<br>5. User attempts to log in again using the deleted account credentials | | | | |
| **Expected** | The account is deleted, and the user is unable to log in with the deleted credentials. All associated data is removed as per the app's policy | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC41 | **Category** | Functional | **Severity** | High |
|---|---|---|---|---|---|
| **Objective** | Verify that users in the same household can add items to the shopping list and that it is visible to all household members | | | | |
| **Steps** | 1. User A logs into the app<br>2. User A navigates to the shopping list section<br>3. User A adds an item to the shopping list<br>4. User B (who belongs to the same household) logs into their account<br>5. User B navigates to the shopping list section | | | | |
| **Expected** | The item added by User A is visible in the shopping list for User B. All household members can see and update the shared list. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC42 | **Category** | Functional | **Severity** | Medium |
|---|---|---|---|---|---|
| **Objective** | Verify that users in the same household can add notes to the shopping list and that other household members can see them | | | | |
| **Steps** | 1. User A logs into the app<br>2. User A navigates to the notes section<br>4. User A adds a note (e.g., "Get whole wheat")<br>5. User B (who belongs to the same household) logs into their account<br>6. User B navigates to the notes section | | | | |
| **Expected** | The note ("Get whole wheat") are visible to User B and other household members. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC43 | **Category** | Functional | **Severity** | Medium |
|---|---|---|---|---|---|
| **Objective** | Verify that items in the pantry are placed in the correct category based on their type | | | | |
| **Steps** | 1. User logs into the app<br>2. User adds various items to the pantry<br>3. User navigates to the pantry section and checks the category labels | | | | |
| **Expected** | Items are automatically categorized correctly. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC44 | **Category** | Functional | **Severity** | Medium |
|---|---|---|---|---|---|
| **Objective** | Verify that users can leave a household successfully. | | | | |
| **Steps** | 1.Navigate to settings and select "Aile Hesabından Ayrıl”<br>2.Confirm the action when prompted<br>3.System removes the user from the household | | | | |
| **Expected** | User successfully leaves the household and is redirected to the household management screen. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC45 | **Category** | Functional | **Severity** | Medium |
|---|---|---|---|---|---|
| **Objective** | Verify that duplicate email registration is not allowed | | | | |
| **Steps** | 1.Enter an email already used for another account<br>2.Fill in the remaining fields and attempt to sign up | | | | |
| **Expected** | The app displays an error message for duplicate email | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC46 | **Category** | Functional | **Severity** | Low |
|---|---|---|---|---|---|
| **Objective** | Verify that invalid email formats are rejected | | | | |
| **Steps** | 1.Enter an incorrectly formatted email (e.g., "user@com" or "user#email.com")<br>2.Click "Sign Up" | | | | |
| **Expected** | The app displays an error message for user to enter a valid email | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC47 | **Category** | Functional | **Severity** | Medium |
|---|---|---|---|---|---|
| **Objective** | Verify that duplicate products are prevented in the inventory | | | | |
| **Steps** | 1.Attempt to add a product with an identical name | | | | |
| **Expected** | System prompts to update quantity instead. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC48 | **Category** | Functional | **Severity** | Medium |
|---|---|---|---|---|---|
| **Objective** | Verify that users can edit products in the shopping list | | | | |
| **Steps** | 1.Navigate to the "Alışveriş Listem" section<br>2.Select an item and modify its name, quantity, or category<br>3.Save the changes | | | | |
| **Expected** | The edited product details are updated in the shopping list successfully. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC49 | **Category** | Functional | **Severity** | Medium |
|---|---|---|---|---|---|
| **Objective** | Verify that users can delete products from the shopping list | | | | |
| **Steps** | 1.Navigate to the "Alışveriş Listem" section<br>2.Select an item and choose the delete option<br>3.Confirm deletion | | | | |
| **Expected** | The selected item is removed from the shopping list. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC50 | **Category** | Functional | **Severity** | High |
|---|---|---|---|---|---|
| **Objective** | Verify that the system recommends recipes based on the meal type | | | | |
| **Steps** | 1.Navigate to the “Tarifler” section<br>2.Choose a meal type<br>3.Check suggested meals | | | | |
| **Expected** | Recipes are based on the chosen cuisine. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC51 | **Category** | Functional | **Severity** | Medium |
|---|---|---|---|---|---|
| **Objective** | Verify the confirmation and deletion process for "Delete All Notes" in the Shopping List | | | | |
| **Steps** | 1.Navigate to "Alışveriş Listesi"<br>2.Ensure there are multiple notes present<br>3.Find and tap the "Tüm Notları Sil" button<br>4.A confirmation dialog appears | | | | |
| **Expected** | All the notes get deleted | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC52 | **Category** | Functional | **Severity** | High |
|---|---|---|---|---|---|
| **Objective** | Verify that changing the password fails if the current password entered is incorrect | | | | |
| **Steps** | 1.Navigate to “Ayarlar” page<br>2.Tap”Şifremi Değiştir"<br>3.Enter an incorrect current password<br>4.Enter a valid new password and confirm it<br>5.Tap "Şifremi Değiştir" | | | | |
| **Expected** | An error message is displayed indicating the current password was wrong. The password is not changed in Firebase Authentication. The user remains on the change password screen/dialog . | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC53 | **Category** | Functional | **Severity** | Medium |
|---|---|---|---|---|---|
| **Objective** | Verify error handling when attempting to join a non-existent household | | | | |
| **Steps** | 1.Sign as a user<br>2.Navigate to Household Setup<br>3.Tap "Eve Katıl"<br>4.Enter a Household ID that does not exist in Firestore<br>5.Tap the "Eve Katıl" button | | | | |
| **Expected** | An error message is displayed indicating the current password was wrong. The password is not changed in Firebase Authentication. The user remains on the change password screen/dialog . | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC54 | **Category** | Non-Functional | **Severity** | Low |
|---|---|---|---|---|---|
| **Objective** | Verify consistent application of theme colors across major screens | | | | |
| **Steps** | 1.Navigate through key screens: Giriş, Ana Sayfa, Mutfağımdakiler, Tarifler, Ayarlar<br>2.Visually inspect background colors, primary text colors, button colors, input field backgrounds | | | | |
| **Expected** | Colors should match the theme. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC55 | **Category** | Functional | **Severity** | Low |
|---|---|---|---|---|---|
| **Objective** | Verify that the old household invitation code expires when a new household invitation code gets generated | | | | |
| **Steps** | 1.Navigate to the “Ayarlar” page and generate a new family invitation code<br>2.Try to enter to the household from another device with the old family invitation code | | | | |
| **Expected** | User should not be able to enter the household | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC56 | **Category** | Functional | **Severity** | Medium |
|---|---|---|---|---|---|
| **Objective** | Verify that the app does not allow users to add an item to their pantry if the amount is zero | | | | |
| **Steps** | 1.Navigate to pantry click on the plus button<br>2.Choose an item and enter the amount as zero | | | | |
| **Expected** | The screen should display a clear message indicating that amount cannot be zero (e.g. Miktarı 0 olan ürünler eklenemez.). | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC57 | **Category** | Functional | **Severity** | Low |
|---|---|---|---|---|---|
| **Objective** | Verify the display of the product list when a category contains zero items | | | | |
| **Steps** | 1.Ensure a specific pantry category has no items associated with it<br>2.Navigate to "Mutfağımdakiler"<br>3.Select the empty category<br>4.View the product list screen | | | | |
| **Expected** | The product list screen (Productlist) should display a clear message indicating that there are no items in this category (e.g., "Ürün Yok.") instead of just an empty space. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC58 | **Category** | Functional | **Severity** | Medium |
|---|---|---|---|---|---|
| **Objective** | Verify editing item quantity directly from the Shopping List view | | | | |
| **Steps** | 1.Navigate to the “Alışveriş Listesi” page<br>2.Locate an item with a quantity display<br>3.Tap on the quantity or specific +/- buttons next to it<br>4.Modify the quantity | | | | |
| **Expected** | The quantity is updated in the UI for that item and saved to Firestore without needing to navigate to a separate edit screen. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC59 | **Category** | Functional | **Severity** | Medium |
|---|---|---|---|---|---|
| **Objective** | Verify the behavior of the "Add Product" search when no results are found | | | | |
| **Steps** | 1.Navigate to "Mutfağımdakiler"page<br>2.Enter a search query in the product search bar that is guaranteed not to match any products in the database/search source | | | | |
| **Expected** | The search results list remains empty or is cleared. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC60 | **Category** | Functional | **Severity** | Medium |
|---|---|---|---|---|---|
| **Objective** | Verify that the user is prevented from approving all OCR results if any single item is still in an active editing state, and that appropriate feedback is provided | | | | |
| **Steps** | 1.Open the app and successfully scan a receipt image<br>2.Locate an item in the list<br>3.Tap to the editing icon associated with that specific item<br>4.Do not tap the confirmation icon for the item being edited. Leave it in the active editing state<br>5.Attempt to finalize the entire list by tapping the main "Hepsini Mutfağa Ekle" button | | | | |
| **Expected** | The main approval action is not executed. No data is saved to the pantry at this point. An error message appears at the bottom of the screen displaying the message "Lütfen önce tüm düzenlemeleri tamamlayın (onay ikonuna basın)." | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC61 | **Category** | Functional | **Severity** | Low |
|---|---|---|---|---|---|
| **Objective** | Verify handling of special characters and emojis when adding a note to the shopping list | | | | |
| **Steps** | 1.Navigate to "Alışveriş Listesi"<br>2.Tap the "Notlar" option<br>3.In the note input field, enter text containing special characters and emojis<br>4.Save the note.<br>5.View the notes list. | | | | |
| **Expected** | The note is saved successfully. The note text, including special characters and emojis, is displayed correctly in the notes list without corruption or errors. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

| **Test ID** | TC62 | **Category** | Functional | **Severity** | Medium |
|---|---|---|---|---|---|
| **Objective** | Verify behavior when canceling the "Var Olan Aileye Katıl" process midway. | | | | |
| **Steps** | 1.Sign up as a user and tap to "Yeni Aile Oluştur”<br>2.Enter a household name<br>3.Instead of tapping "Yeni Aile Oluştur", tap to the back icon<br>4.Attempt to tap "Yeni Aile Oluştur” again | | | | |
| **Expected** | The process reset successfully. The previously entered household name was not retained. The user could reinitiate the family creation process without issues. | | | | |
| **Result/Notes** | Satisfactory - Tested on 01.05.2025. | | | | |

# 6. Maintenance Plan and Details

To ensure long-term reliability and user satisfaction, *On the Shelf* will follow a structured maintenance plan post-deployment. Regular updates will be scheduled to address bug fixes, performance optimizations, and compatibility with new versions of iOS and Android. The app's codebase, written in Flutter, allows for efficient cross-platform maintenance from a single code repository. Firebase services, including Firestore and Authentication, will be monitored to ensure backend stability and security. In addition, automated testing and continuous integration pipelines will be employed to catch regressions early during updates. As new features are introduced, scalability and backward compatibility will be prioritized to maintain a seamless experience for existing users.

# 7. Other Project Elements

## 7.1. Consideration of Various Factors in Engineering Design

### 7.1.1 Constraints

During the development of *On the Shelf*, several constraints shaped design, implementation, and deployment decisions:

**Technical Constraints**

- Cross-Platform Constructability: The project was developed using a single Flutter codebase to ensure seamless deployment across both iOS and Android devices, but this limited the use of platform-specific customizations.
- Performance Limitations: The integration of OCR technology for receipt scanning required careful optimization to ensure smooth performance on devices with varying hardware capabilities.
- Energy Efficiency: In alignment with sustainability goals, the app had to be designed with energy-efficient principles, placing limits on resource usage and background processes.

**User-Centered Constraints**

- Accessibility and Usability: The UI was required to follow universal design principles, balancing visual aesthetics with intuitive experience. This limited the use of overly complex UI elements.
- User Effort Minimization: Features such as automatic OCR and household synchronization were prioritized to reduce manual input, impacting the structure and timing of data flows within the app.

**Security and Legal Constraints**

- Data Privacy Compliance: The app was developed in accordance with KVKK and other relevant data privacy regulations, imposing constraints on how user data could be stored, accessed, and shared.
- Secure Coding Practices: The project adhered to secure mobile development standards, which imposed requirements on password handling, authentication mechanisms, and database access.

**Scalability & Maintainability Constraints**

- Extensibility: The app architecture needed to support future features like advanced recipe filters or sorting products. This required a modular structure while maintaining simplicity to meet short-term development deadlines.
- Code Standards and Industry Practices: The team was expected to follow modern mobile development standards, limiting the use of outdated libraries or shortcuts in implementation.

**Time and Resource Constraints**

- Development Timeline: The app was developed under a 2 semester timeline, including phases for implementation, testing, and final documentation.
- Team Size: With five developers, task allocation and prioritization became critical, constraining the complexity of individual features and the overall scope.

**Environmental and Social Constraints**

- Sustainability Alignment: The app needed to promote sustainable practices, such as reducing food waste, while itself being environmentally friendly in terms of resource usage.
- Market and Policy Alignment: The app's functionality had to align with responsible consumption goals and comply with app store guidelines for deployment.

### 7.1.2 Standards

- UML 2.5.1 for modeling
- IEEE 830 for requirements documentation
- IEEE for referencing style

## 7.2. Ethics and Professional Responsibilities

The team prioritized ethical considerations and professional responsibilities throughout the development:

- **User Privacy:** Sensitive data such as allergies, household membership, and inventory details are stored securely in Firebase. No data is shared with third parties.
- **Security Measures:** Passwords are securely stored, and functions like password change and account deletion require user confirmation to avoid misuse.
- **Transparency:** Users are informed before performing critical actions like leaving a household or deleting their account.
- **Accessibility and Usability:** The app is designed with a clear interface and dark/light mode toggle to accommodate user preferences and needs.
- **Data Integrity:** OCR-generated data is editable by users to prevent incorrect entries from impacting shelf or recipe results.
- **Inclusive Design:** The focus on allergies (instead of broad dietary preferences) helps ensure features are grounded in medically relevant and actionable needs.

## 7.3. Teamwork Details

### 7.3.1. Contributing and functioning effectively on the team to establish goals, plan tasks, and meet objectives

Each team member contributed actively throughout the development of the project. Gülbera focused primarily on UI design and user experience improvements, ensuring the app interface remained intuitive and user-friendly. Emirhan worked on OCR, image processing features, and product dataset preparation conducting meetings with executives at Migros for dataset retrieval. Also conducted potential funding meetings which later failed. Defne led backend development efforts, managing the integration with Firebase and optimizing database interactions also, contributed in backend functionalities. Ümmügülsüm contributed significantly to Flutter development and implemented many of the user-facing features such as the recipe suggestion and pantry. İrem worked on backend features like shopping list modules and coordinated documentation efforts and also contributed to bug fixing of the pages. Through weekly meetings and shared task boards, everyone stayed aligned with the development timeline and completed their tasks effectively.

### 7.3.2. Helping creating a collaborative and inclusive environment

The team fostered a respectful and open environment where every member felt comfortable sharing ideas and feedback. Decisions were made through consensus, and all voices were heard, regardless of the task domain. Constructive criticism and support were encouraged, promoting mutual growth and understanding. Tools such as shared documents, messaging platforms, and version control systems further supported collaboration.

### 7.3.3. Taking lead role and sharing leadership on the team

Leadership responsibilities were shared at different stages. Gülbera took the lead for implementing frontend and merging backend and frontend features. Defne led the database structure design and backend implementation, guiding the integration with Firebase services. Emirhan coordinated the OCR module development and dataset preparation, proposing algorithms and testing workflows. Ümmügülsüm led during backend implementation of functionalities. İrem took initiative during the bug fixes and backend enhancement, ensuring quality standards and completeness. This rotation of leadership allowed everyone to develop project management skills and take ownership of critical components.

### 7.3.4. Meeting objectives

According to our initial project plan, we had set key milestones such as completing the requirements analysis, designing the UI, implementing core features (OCR-based product scanning, shared shoppinglist, inventory management, and allergy-aware recipe recommendations), and conducting final testing before release. All major objectives were met on schedule. And now, before the demo our all core features are ready and we will have the chance to enhance our features in our next versions. We successfully adhered to our project plan while maintaining quality and performance benchmarks.

## 7.4 New Knowledge Acquired and Applied

Throughout the project, the team gained and applied new knowledge in the following areas:

- **Flutter App Development:** Team members learned how to build a mobile application using Flutter, including widget management, state handling, and responsive UI design.
- **Firebase Integration:** Skills were gained in using Firebase Authentication, Firestore, and real-time database synchronization for user and household data.
- **OCR Integration:** The team explored and integrated OCR tools for scanning receipts, learning how to preprocess and extract text data efficiently.
- **UI/UX Design Principles:** Best practices for mobile UX design were studied and applied, such as clear settings layouts, confirmation dialogs, and theme switching.
- **Team Collaboration:** Version control with Git and the issue tracker via Jira improved workflow efficiency and collaboration.
- **Requirement Engineering:** The team revised functional and non-functional requirements based on evolving goals and technical feasibility.

# 8. Conclusion and Future Work

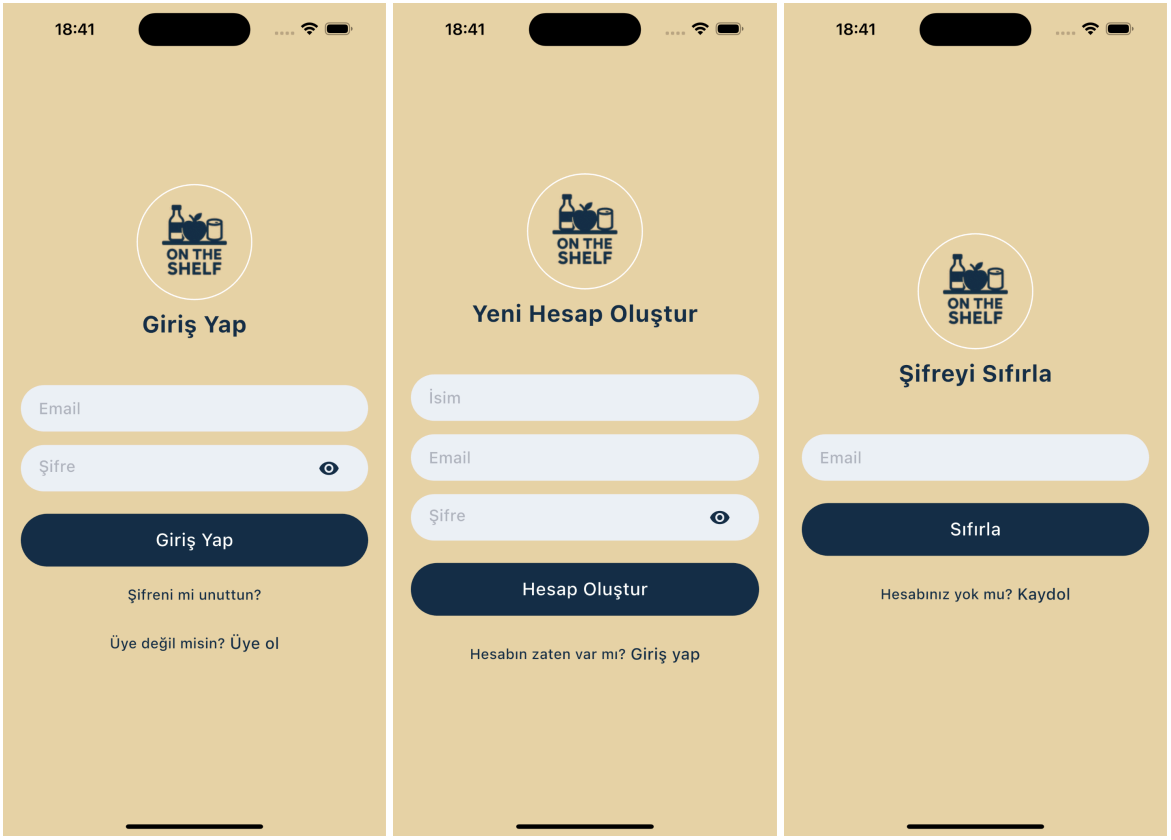
Throughout the development of *On the Shelf*, we have accomplished several key milestones that transformed our initial idea into a functional mobile application. We began by conducting market research and user surveys to identify common pain points in grocery management. Using those insights, we designed wireframes and UIs in Figma, then implemented core features such as inventory tracking, shopping list generation, and receipt scanning using OCR technology. We developed the app using Flutter for cross-platform compatibility and integrated Firebase as the backend. Agile development practices allowed us to work iteratively, test frequently, and adapt to feedback, ultimately leading to a stable and user-friendly prototype.

This project has also been a significant learning experience for our entire team. As our first time developing a cross-platform mobile app, we were introduced to new technologies and frameworks, and we strengthened our ability to collaborate effectively in a team environment. We learned how to manage a full software development lifecycle from ideas and design to implementation and testing. Beyond the technical aspects, we also deepened our skills in user research, product thinking, and sustainable design. Working on a real-world problem and seeing our solution come to life was both educational and fulfilling.

Looking ahead, we have several plans to further improve and expand *On the Shelf*. One of our primary goals is to improve the OCR functionality by integrating advanced machine learning techniques to handle a wider range of receipt formats with greater accuracy. We also plan to include personalized features such as consumption analytics and smart reminders. In addition, we aim to integrate third-party grocery APIs for real-time price comparisons and item suggestions. Most importantly, we plan to officially publish the app on both the App Store and Google Play Store to make it widely available. With further development and continuous user feedback, we believe *On the Shelf* can grow into a practical and sustainable tool for households around Türkiye.

# 9. User Manual

## 9.1. Login, Sign Up and Forgot Password




**Login Screen:**

- Enter your registered email and password, then tap "Giriş yap" to access your account.
- If you don’t have an account, tap "Üye ol" below the login button.

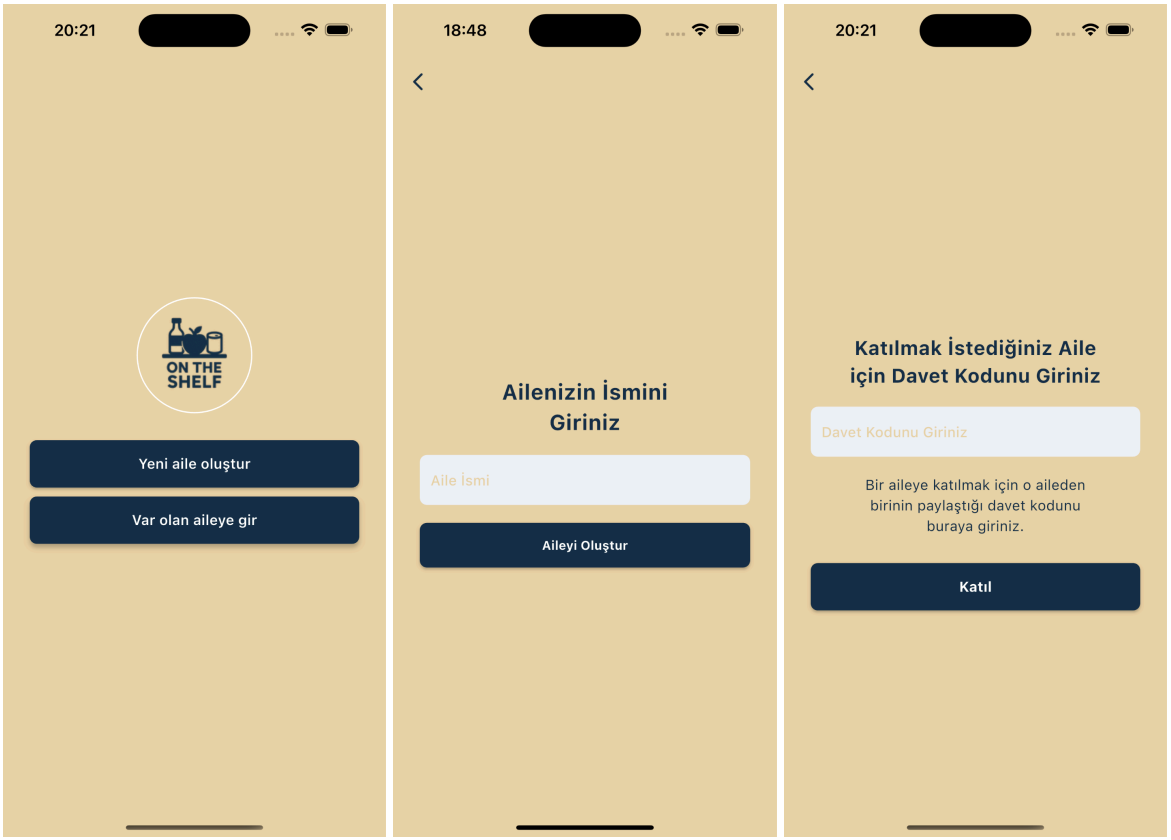
**Sign Up Screen:**

- Fill in your name, email address, and a secure password. Tap "Üye ol" to create your account.
- If you already have an account, tap "Giriş Yap" to return to the previous screen.

**Forgot Password Screen:**

- If you've forgotten your password, tap "Forgot Password" on the Login screen.
- Enter your email address and follow the instructions sent to your inbox to reset your password.

## 9.2. Household Setup – Create or Join




After signing up or logging in for the first time, you'll be asked to set up your household.

You can choose to either:

**Create a New Household:**

- Enter a household name and tap "Aileyi oluştur".

**Join an Existing Household:**

- Enter the Household Invitation Code generated by an existing member and tap "Katıl".
- Once you create or join a household, you'll be taken to the Home Page created for that household.

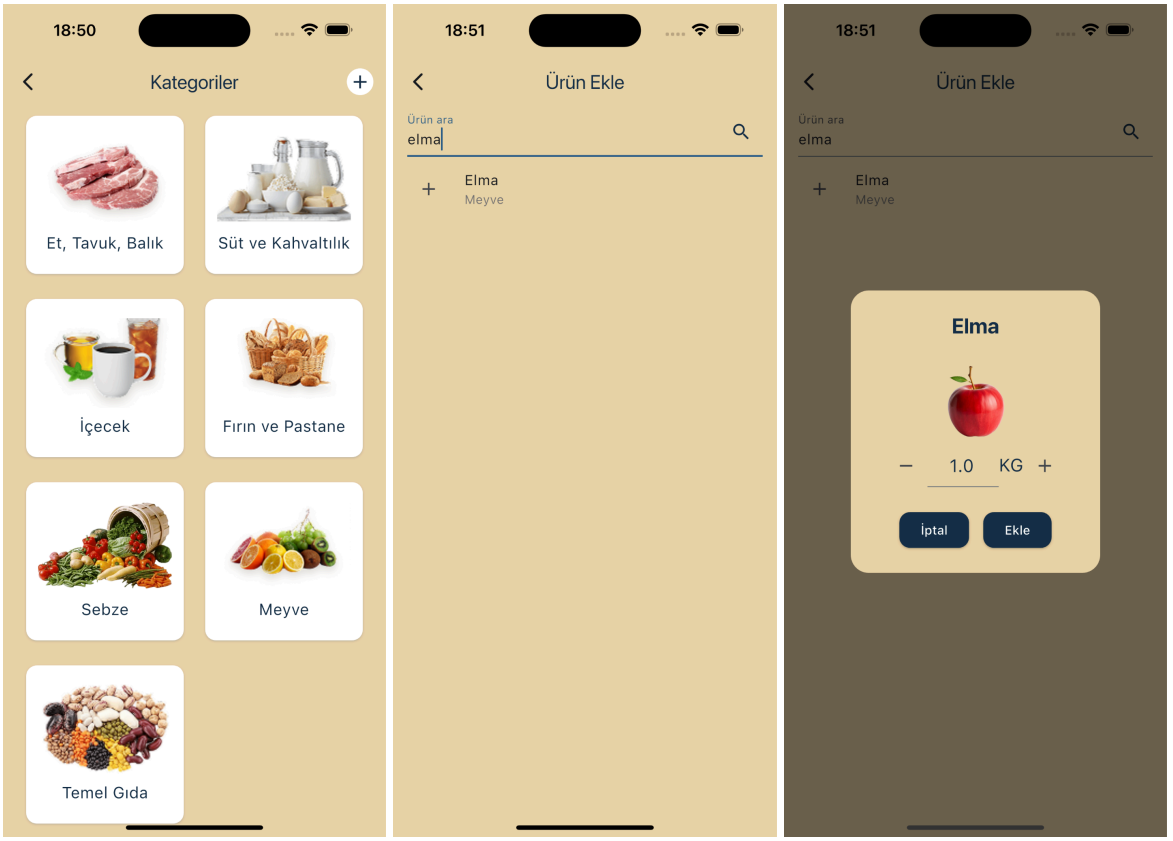
## 9.3. Home Page Overview


The Home Page includes four main sections that are accessible via the buttons.

- **Fişlerim (My Receipts)**: View and manage your scanned or uploaded grocery receipts.
- **Tariflerim (My Recipes)**: Browse and save recipes based on your allergies and available ingredients.
- **Alışveriş Listem (My Shopping List):** View and edit your shopping list.

- **Mutfağımdakiler (My Pantry)**: View and manage the items currently in your kitchen.

## 9.4 Managing Your Pantry (Mutfağımdakiler)

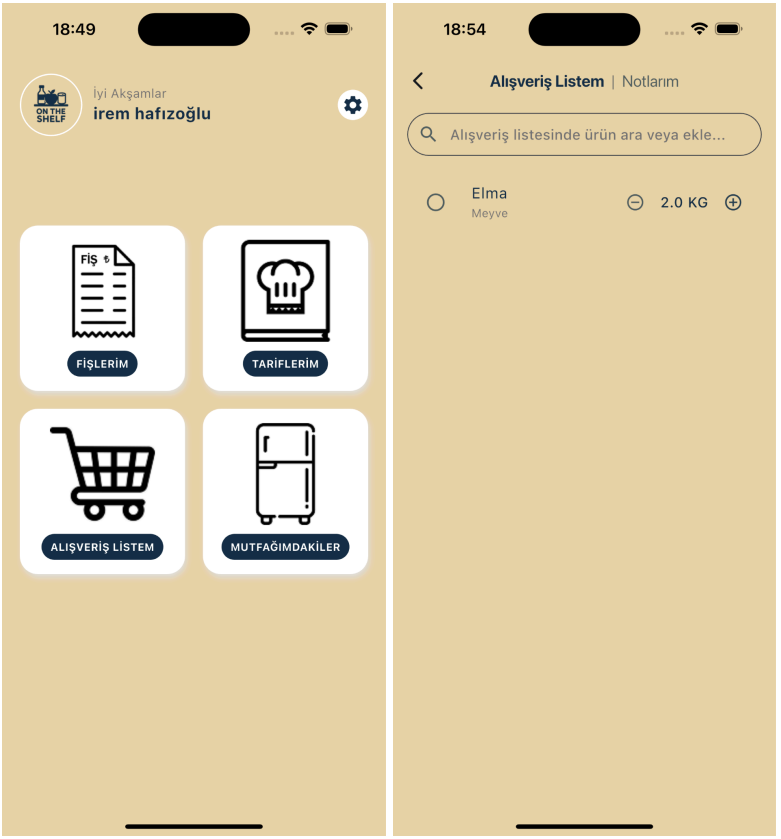



- Tap "Mutfağımdakiler" to access your pantry.
- This page displays your pantry items, organized into 7 categories for easy browsing.
- To add a new item, tap the "+" (plus) button at the top-right corner of the screen.
    - In the search bar, type the name of the item you want to add.
    - Once you find the item, tap the "+" button next to its name.
    - Enter the quantity of the item in the pop-up window.
    - Tap "Ekle" (Add) to confirm.
- You can then go back and open the relevant category to see the item listed with the quantity you entered.

## 9.5. Viewing and Editing Items in a Category

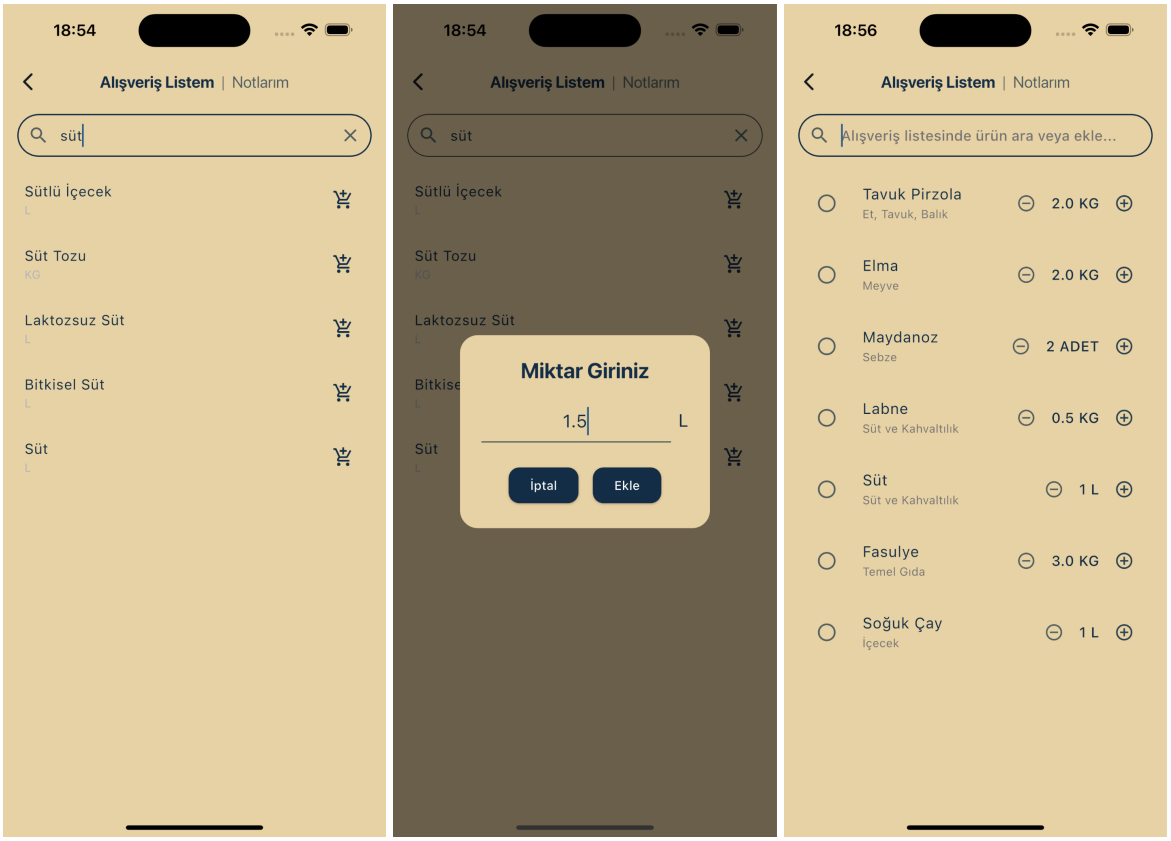

- When you tap on a specific category in your pantry, you'll see a list of all items stored under that category.
- From this screen, you can update item quantities in two ways:
    - Tap the "+" or "–" buttons below each item to increase or decrease the quantity.
    - Tap the edit icon at the top-right corner to enter a new quantity manually.
- To remove an item from your pantry, tap the trash/delete icon located at the top-right corner of the item card.
- A prompt will appear asking if you'd like to add the deleted item to your shopping list:
    - Tap "Evet" (Yes) to send the item to your Shopping List before it's deleted from the pantry.
    - Tap "İptal" (Cancel) if you do not want to add it to the Shopping List the item will still be removed from your pantry.

## 9.6. Using the Shopping List (Alışveriş Listem)

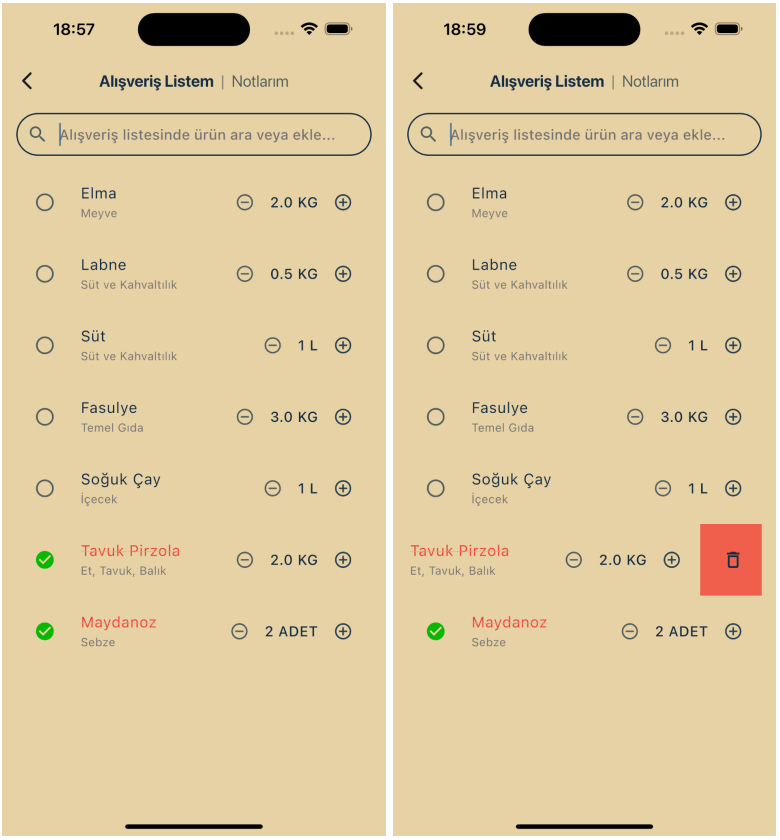



- When you return to the Home Page and tap "Alışveriş Listem", you'll see your current shopping list.
    - If you previously chose "Yes" when deleting a pantry item, it will now appear here.

## 9.7. Adding and Managing Products

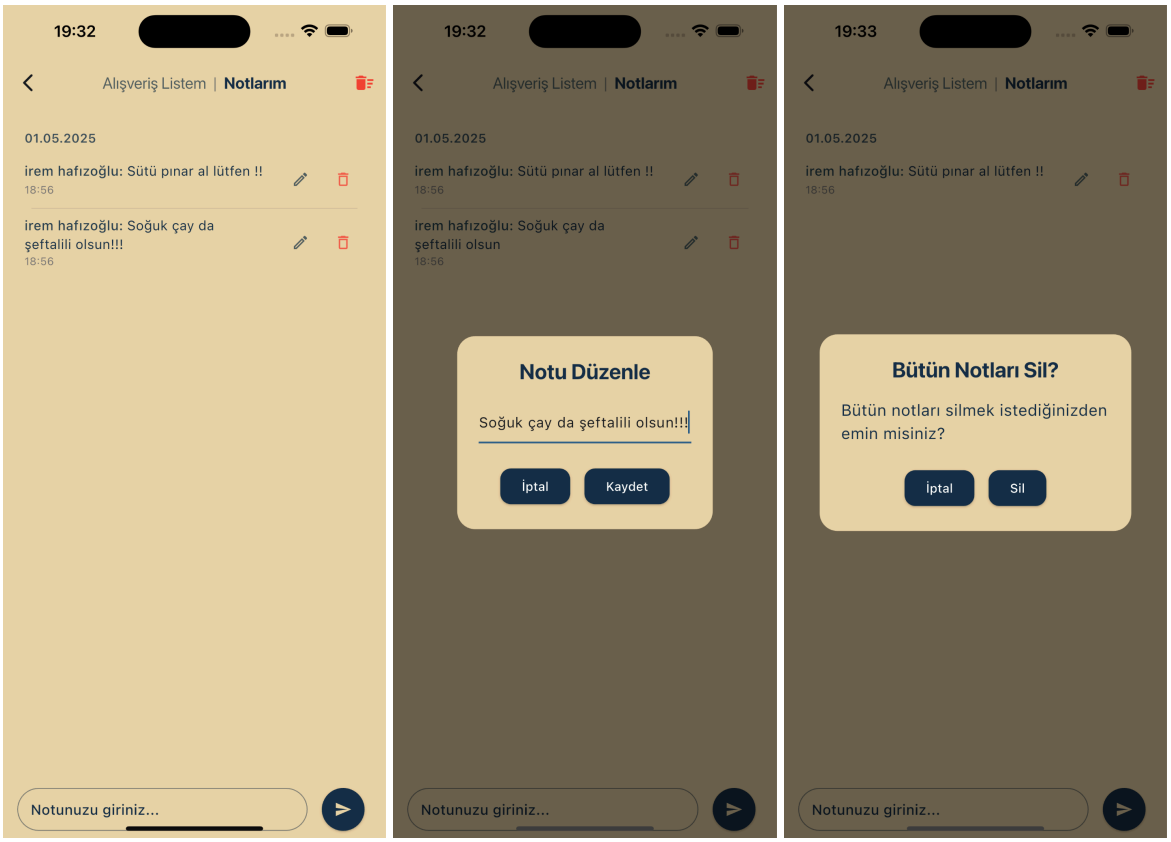



- To add a new product, tap on the search bar at the top of the screen and type the name of the item.
- When it appears in the results, tap the shopping basket icon on the right side.
- Enter the quantity and tap to confirm.
- You can update quantities in two ways:
    - Use the “+” and “–” buttons below each item.
    - Tap on the item itself to manually enter the desired quantity.



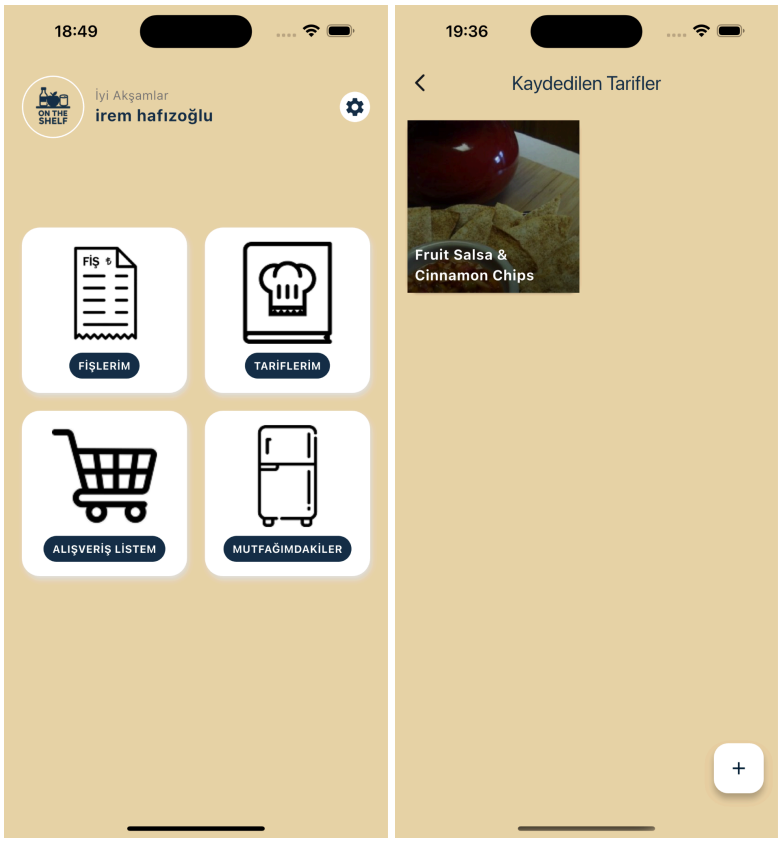

- To mark an item as purchased, tap the empty circle to the left of the product.
    - The item will be crossed out and moved to the bottom of the list.
    - This helps prioritize items that still need to be bought.
- To permanently delete an item from the list, simply swipe left on the item.

## 9.8. Notes Section (Notlarım)



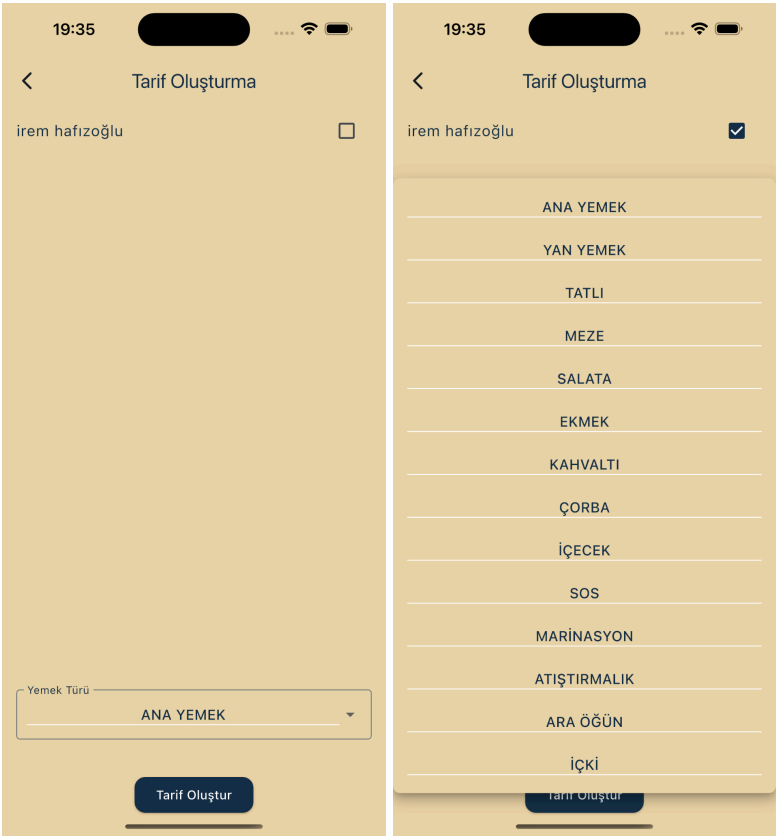

- At the top of the Alışveriş Listem page, you'll see two tabs: "Alışveriş Listem" | "Notlarım"
- Tap "Notlarım" to switch to the notes section, where you and other household members can leave notes for everyone in the household to see.
- To add a note, type your message in the input field and submit it.
  You can edit your own notes after posting them by:
  - Tapping the edit icon on the left side of the note.
  - A pop-up will appear where you can update your text.
  - Confirm the changes to save your edited note.
- To delete your own note, tap the trash icon on the right side of the note.
  This will remove the note from the shared list.
- If you've completed your shopping and want to clear all notes at once, tap the trash icon at the top-right corner of the page.
- A confirmation will appear before deleting to prevent accidental removal of all notes.

## 9.9. Recipes Section (Tariflerim)

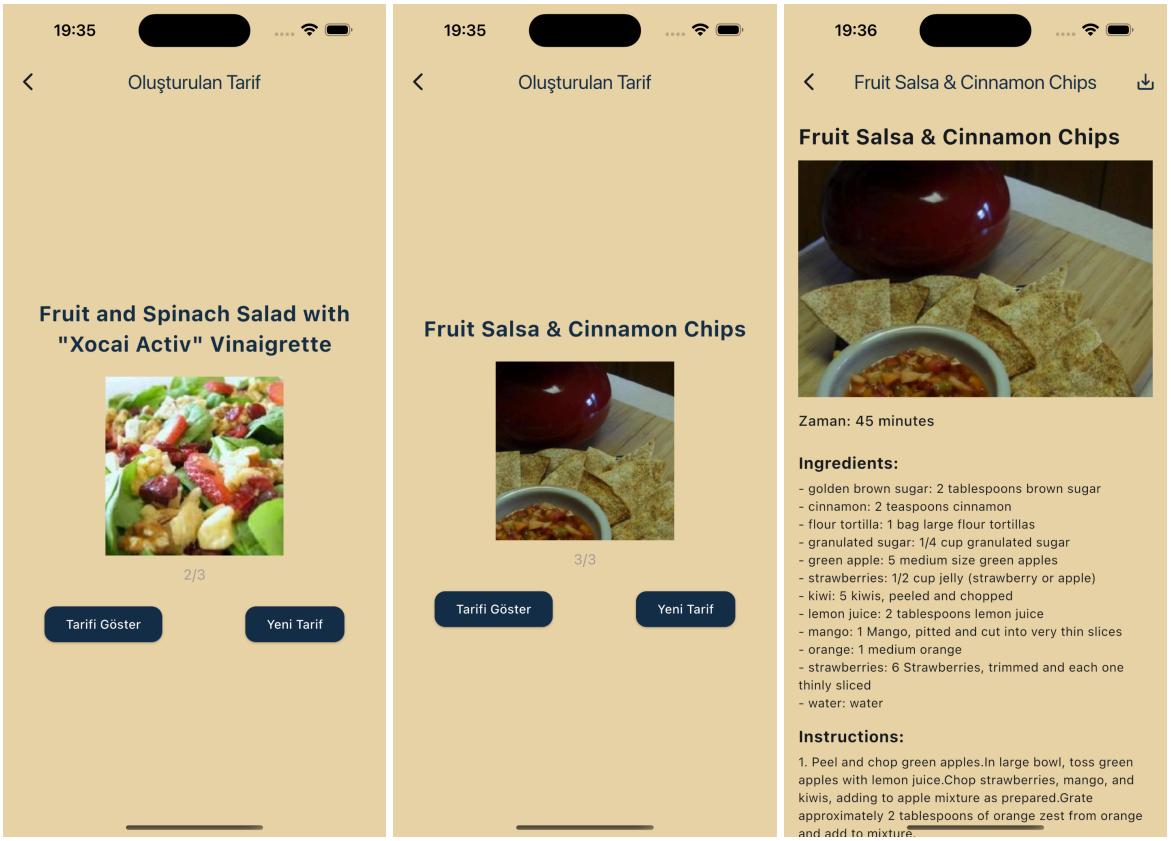



- From the Home Page, tap "Tariflerim" to access your saved recipes.
- Here, you can browse through the recipes you've previously saved and view their instructions at any time.

## 9.10. Generating New Recipes

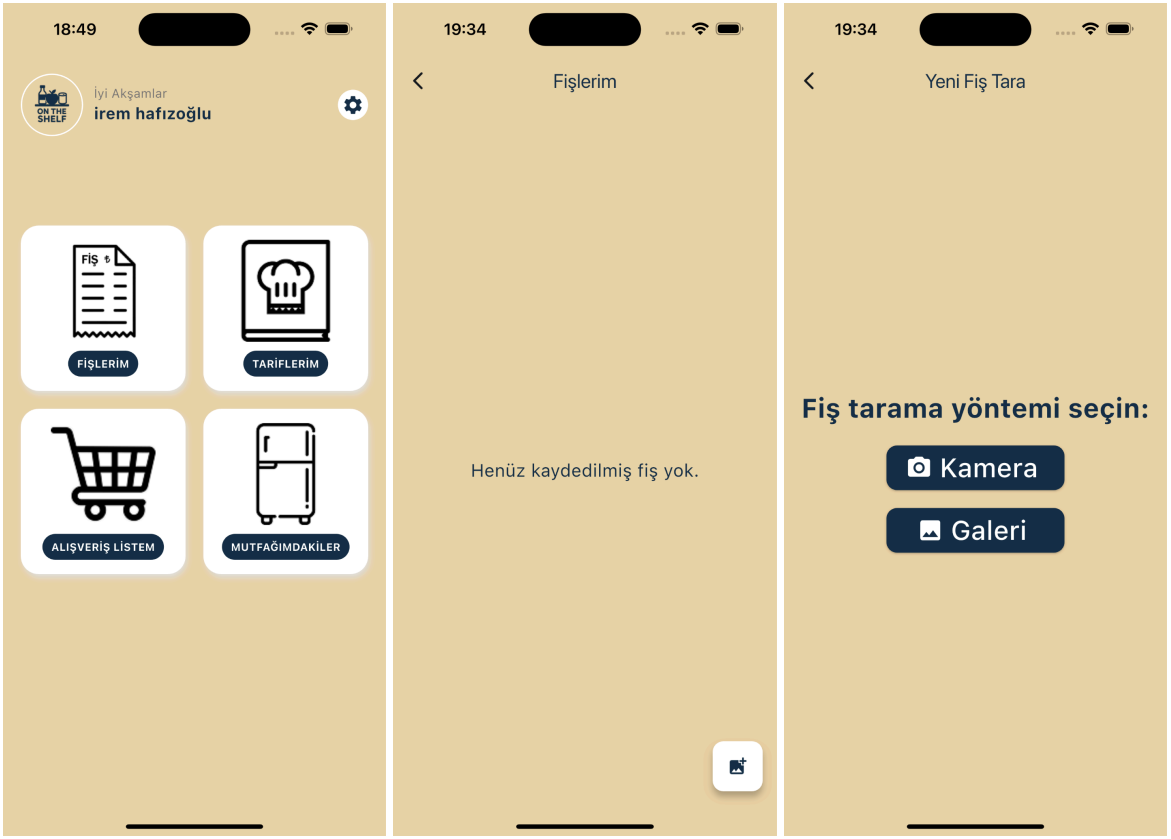



- To create new recipe suggestions, tap the "+" (plus) button at the bottom-right corner of the screen.
- You'll be asked to select the household members you're cooking for.
    - The app will take each selected member's allergies into account while generating recipe suggestions.
- Then, choose the meal type, such as "Ana Yemek" (Main Course) or "Tatlı" (Dessert).




- The app will generate 3 recipe suggestions based on your selections.
- Tap on any recipe to view its detailed instructions and ingredients.
- If you like a recipe, tap the save icon at the top-right corner to add it to your saved list in Tariflerim.

## 9.11. Receipts Section (Fişlerim)

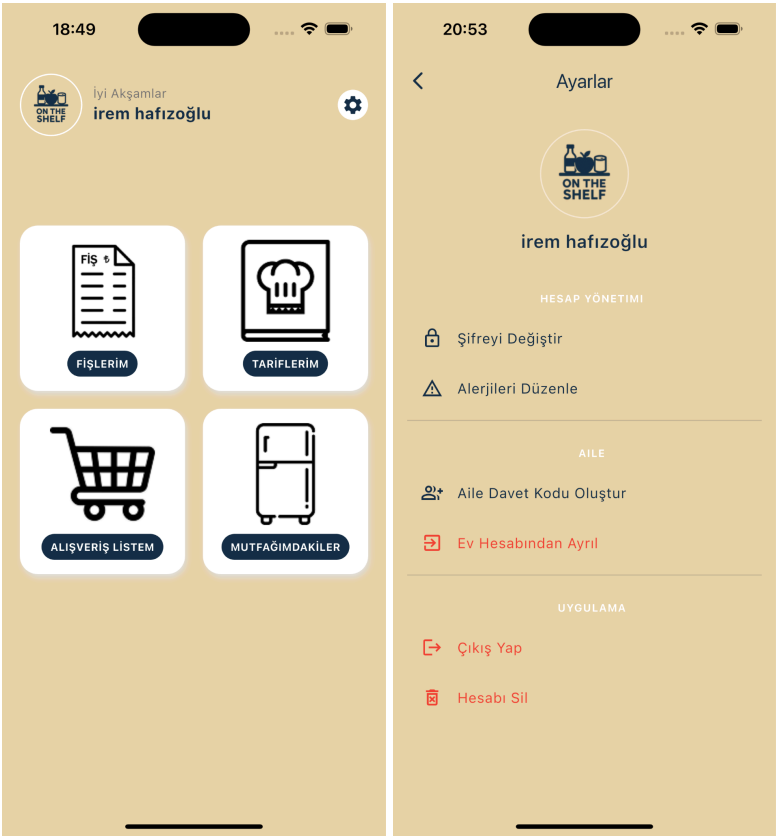



- From the Home Page, tap "Fişlerim" to view your saved receipts.
- To scan a new receipt, tap the "+" (plus) button at the bottom-right corner of the screen.
- You will be given two options:
    - Camera : Use your device’s camera to scan a physical receipt.
    - Gallery : Choose an existing receipt photo from your gallery.
- Once you select a receipt, the app will use OCR to extract product information.

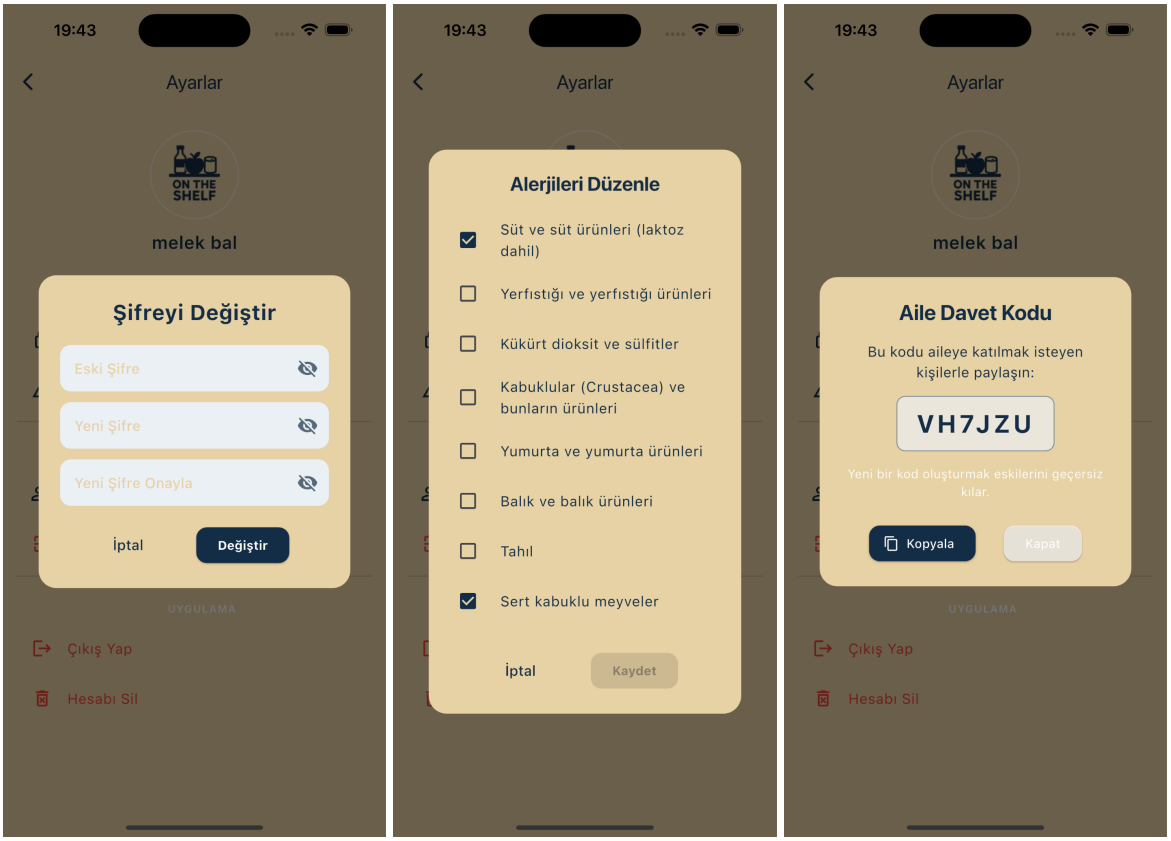
## 9.12. Reviewing and Editing OCR Results


- The OCR results will appear as a list of products with their; Name, quantity, unit.
- If any information was scanned incorrectly, you can edit these fields directly before saving the receipt.

## 9.13. Settings Page (Ayarlar)

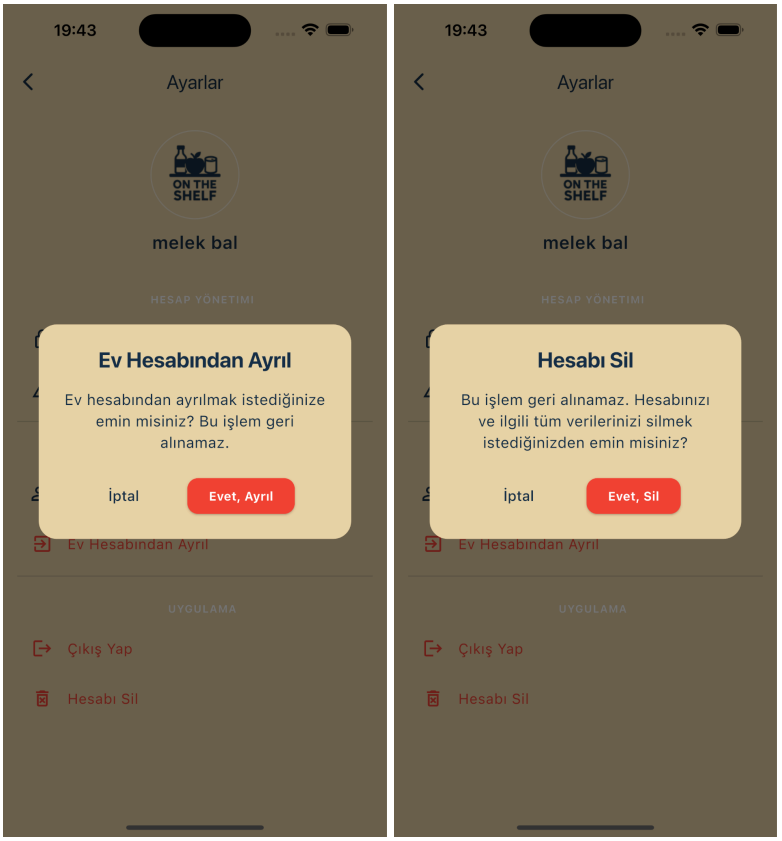



- From the Home Page, tap the settings icon (⚙️) at the top-right corner of the screen to access the Settings page.




## Available Settings Options

- **Change Password**
  - Enter your current password and your new password, then confirm the change.
- **Edit Allergies**
  - Check or uncheck items from the allergy list to update your preferences.
  - These settings help personalize recipe suggestions and ensure safety.
- **Generate Family Invitation Code**
  - Tap this option to generate a unique code that others can use to join your household.




- **Leave Household**
    - If you no longer wish to be part of your current household, select this option to leave the household group.
- **Delete Account**
    - Permanently delete your account and all associated data.
- **Log Out**
    - Sign out of the app and return to the login screen.

# 10. Glossary

- **OCR:** Optical Character Recognition
- **AI:** Artificial Intelligence
- **UML:** Unified Modelling Language
- **IEEE:** Institute of Electrical and Electronics Engineers
- **UI:** User Interface
- **KVKK:** Turkish Data Protection Law
- **API:** Application Programming Interface

# 11. References

[1] "Whisk" [Online]. Available: https://www.whiskapp.net/. [Accessed 01 May 2025].

[2] "KitchenPal" [Online]. Available: http://kitchenpalapp.com/en/. [Accessed 01 May 2025].

[3] "Paprika" [Online]. Available: https://www.paprikaapp.com/. [Accessed 01 May 2025].

[4] "What’s Left" [Online]. Available: https://tellmewhatsleft.de/index_en.html. [Accessed 01 May 2025].