***SOFTWARE PROJECT FINAL REPORT***

TastyTracker Smartphone Application

Julie White (2832202)

12/5/2024

**Table of Contents**

[Table of Figures 4](#_Toc184075832)

[List of Tables 6](#_Toc184075833)

[1. Introduction 7](#_Toc184075834)

[1.1. Purpose and Scope 7](#_Toc184075835)

[1.2. Product Overview 7](#_Toc184075836)

[1.3. Structure of the Document 8](#_Toc184075837)

[1.4. Terms, Acronyms, and Abbreviations 9](#_Toc184075838)

[2. Project Management Plan 9](#_Toc184075839)

[2.1. Project Organization 9](#_Toc184075840)

[2.2. Lifecycle Model Used 10](#_Toc184075841)

[2.3. Risk Analysis 11](#_Toc184075842)

[2.4. Hardware and Software Resource Requirements 13](#_Toc184075843)

[2.5. Deliverables and schedule 14](#_Toc184075844)

[3. Requirement Specifications 17](#_Toc184075845)

[3.1. Stakeholders for the system 17](#_Toc184075846)

[3.2. Use cases 17](#_Toc184075847)

[3.2.1. Graphic use case model 17](#_Toc184075848)

[3.2.2. Textual Description of Use Cases 22](#_Toc184075849)

[3.3. Rationale for your use case model 23](#_Toc184075850)

[3.4. Non-functional requirements 24](#_Toc184075851)

[4. Architecture 25](#_Toc184075852)

[4.1. Architectural style used 25](#_Toc184075853)

[4.2. Architectural model 26](#_Toc184075854)

[4.3. Technology, software, and hardware used 27](#_Toc184075855)

[4.4. Rationale for your architectural style and model 27](#_Toc184075856)

[5. Design 28](#_Toc184075857)

[5.1. User Interface design 28](#_Toc184075858)

[5.2. Components design (static and dynamic models of each component) 31](#_Toc184075859)

[5.3. Database design 37](#_Toc184075860)

[5.4. Rationale for your detailed design models 38](#_Toc184075861)

[5.5. Traceability from requirements to detailed design models 39](#_Toc184075862)

[6. Test Management 40](#_Toc184075863)

[6.1. A complete list of system test cases 40](#_Toc184075864)

[6.2. Traceability of test cases to use cases 45](#_Toc184075865)

[6.3. Techniques used for test case generation 45](#_Toc184075866)

[6.4. Test results and assessments 46](#_Toc184075867)

[6.5. Defects reports 46](#_Toc184075868)

[7. Conclusions 47](#_Toc184075869)

[7.1. Outcomes of the project 47](#_Toc184075870)

[7.2. Lessons learned 47](#_Toc184075871)

[7.3. Future development 48](#_Toc184075872)

[References 48](#_Toc184075873)

# **Table of Figures**

[Figure 1: Full Gantt Chart for Development 16](#_Toc184075976)

[Figure 2: Use Case Diagram of the Entire TastyTracker System 18](#_Toc184075977)

[Figure 3: Use Case Diagram for Head of Household 19](#_Toc184075978)

[Figure 4: Use Case for MWP 20](#_Toc184075979)

[Figure 5: Use Case for MWOP 21](#_Toc184075980)

[Figure 6: Component Diagram of Model-View-Controller Architecture 26](#_Toc184075981)

[Figure 7: Static Model of User Authentication Component 31](#_Toc184075982)

[Figure 8: Dynamic Model for User Authentication Component 32](#_Toc184075983)

[Figure 9: Inventory Management Static Model 33](#_Toc184075984)

[Figure 10: Dynamic Model for Inventory Management Component 34](#_Toc184075985)

[Figure 11: Static Model for Shopping List Management 35](#_Toc184075986)

[Figure 12: Dynamic Model for Shopping List Management 36](#_Toc184075987)

[Figure 13: Database Diagram 37](#_Toc184075988)

# **List of Tables**

[Table 1: Project Risk and Mitigation Strategies 12](#_Toc184076052)

[Table 2: Table Showing the Dates and Predecessors for the Gantt Chart 16](#_Toc184076053)

[Table 3: Test Case 1 40](#_Toc184076054)

[Table 4: Test Case 2 40](#_Toc184076055)

[Table 5: Test Case 3 41](#_Toc184076056)

[Table 6: Test Case 4 41](#_Toc184076057)

[Table 7: Test Case 5 41](#_Toc184076058)

[Table 8: Test Case 6 41](#_Toc184076059)

[Table 9: Test Case 7 41](#_Toc184076060)

[Table 10: Test Case 8 42](#_Toc184076061)

[Table 11: Test Case 9 42](#_Toc184076062)

[Table 12: Test Case 10 42](#_Toc184076063)

[Table 13: Test Case 11 42](#_Toc184076064)

[Table 14: Test Case 12 43](#_Toc184076065)

[Table 15: Test Case 13 43](#_Toc184076066)

[Table 16: Test Case 14 43](#_Toc184076067)

[Table 17: Test Case 15 43](#_Toc184076068)

[Table 18: Test Case 16 43](#_Toc184076069)

[Table 19: Test Case 17 44](#_Toc184076070)

[Table 20: Test Case 18 44](#_Toc184076071)

[Table 21: Test Case 19 44](#_Toc184076072)

[Table 22: Test Case 20 44](#_Toc184076073)

[Table 23: Test Case 21 45](#_Toc184076074)

# **1. Introduction**

## **1.1. Purpose and Scope**

TastyTracker is an inventory management smartphone application designed to assist households in managing their food inventory and creating shopping lists. Its primary objective is to simplify a household’s food inventory tracking and shopping list creation by enabling users to collectively add items, track their quantities, and organize grocery lists efficiently. The app is designed to improve household organization, minimize food waste, and decrease frustration by ensuring that households have the items they need, when they need them.

## **1.2. Product Overview**

TastyTracker provides the following capabilities:

* Inventory Management: Allows users to add, edit, and delete food items in a household inventory, giving the household the collective ability to track food items and their units. This gives all users the ability to have a real time understanding of their household food inventory no matter their location.
* Shopping List Management: Enables users to create and update shopping lists. The items on the shopping list can be manually added, or added via the inventory as items begin to run low. The items can then be marked as ‘purchased’, which removes the item from the shopping list and adds the item and its quantity to the household inventory, allowing for simplified management.
* Role-Based Permissions: The first user to create an account within their household is the admin account, referred to as the Head of Household. The Head of Household then determines the level of access other household members will have. Members With Out Privileges are heavily restricted in their actions, only being able to edit items already in the inventory (to help keep track of consumption) and they must request to add items to the shopping list. Whereas Members With Privileges, are able to add, edit, and delete items from both the inventory and shopping lists—only unable to manage other users permissions and review requests made by those without privileges. The requests and household management are done solely by the Head of Household.
* Synchronization: The use of databases ensures real-time updates and consistency across all household members’ devices.

## **1.3. Structure of the Document**

This document outlines the development process, including project management, requirements, architecture, design, and testing, concluding with an evaluation of the outcomes and potential future enhancements. Each section details the decisions, challenges, and achievements encountered during TastyTracker’s development.

## **1.4. Terms, Acronyms, and Abbreviations**

The following terms, acronyms, and abbreviations will be used throughout the document and are important to know to fully understand this document and the TastyTracker application.

* Head of Household (HH): The primary administrator role in the application. HH users have full permissions to manage inventory and shopping lists, approve item requests, and modify the privileges of other users.
* Member with Privileges (MWP): A secondary role that allows users to manage inventory and shopping lists, but does not have privilege modification or request approval.
* Member without Privileges (MWOP): A restricted role that enables users to request items for the shopping list, which must be approved by the HH. MWOP users can view the shopping list but cannot edit or delete items from it. MWOP users can edit and delete items in the inventory but cannot add items to the inventory.

# **2. Project Management Plan**

## **2.1. Project Organization**

The TastyTracker project was developed by a single developer, balancing the constraints of academic deadlines, resource limitations, and the project requirements. The project emphasized functionality prioritization to ensure the core features—inventory and shopping list management—were implemented effectively and thoroughly tested.

## **2.2. Lifecycle Model Used**

The development of TastyTracker adopted a hybrid plan-driven approach with elements of incremental development. While the project is not strictly agile, it incorporates flexibility in feature prioritization and feedback integration during the development process. The development of TastyTracker had the following plan-driven characteristics:

* Defined Requirements and Scope
  + The core functionalities of inventory and shopping list management were specified at the outset of the project.
  + Detailed requirements were documented in the software requirement specification to ensure a clear understanding of the goals and deliverables.
* Sequential Development Stages
  + The project followed structured stages, including requirements gathering, architectural design, implementation, and testing.
  + Each stage produced tangible outputs, such as database schemas, static and dynamic models, and test cases, which were used as inputs for the subsequent stage.
* Documentation Emphasis
  + Given this application was developed within an academic context, extensive documentation was maintained for design, testing, and project outcomes. This aligns with the plan-driven approach’s focus on formal outputs at each stage.

The following elements are how the TastyTracker development incorporated incremental elements:

* Iterative Feature Delivery
  + Features like inventory management and shopping list creation were implemented in smaller increments. Each increment was fully functional and tested before proceeding to the next.
  + Role-based permissions and request management were not integrated until testing was completed on the core functionality.
* Customer (User) Feedback
  + The developer used feedback from beta testers to refine the design and usability of key features. This added a degree of flexibility in accommodating usability adjustments.
* Emphasis on Core Functionality First
  + The plan prioritized delivering a usable, stable application with core features, deferring secondary features like recipe integration for future iterations.

## **2.3. Risk Analysis**

The development of TastyTracker faced several risks and challenges related to resource constraints, technical challenges, and the project’s timeline. A thorough risk analysis (completed at the beginning of the project) helped identify potential issues and implement mitigation strategies to ensure successful project completion. The project faced six primary risks, each varying in probability and impact. Each has a specific mitigation strategies that reduce impact through proactive measures like monitoring time management, early testing, and more. The following table gives an overview of these risks, their likelihood of occurrence at the projects onset, the risks impact on the project if realized, the severity of the impact, and the mitigation strategy implemented.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Risk | Likelihood | Impact | Severity | Mitigation Strategy |
| Delivery Deadline Tightened | Low | Medium | Critical | Constant communication with professor, prioritize essential tasks, schedule buffer time. |
| Too Large of Scope | Medium | High | Medium | Define clear scope, implement feature freeze date, review progress regularly |
| Requirement Change | Medium | High | Medium | Review requirements early, implement change control process, prioritize flexibility |
| Technology Change | High | Low | Medium | Monitor tools for updates, lock dependencies to stable versions, prepare backup options |
| Less Reuse than Planned | Medium | Low | Low | Test reusable components early, maintain a backup plan, review open-source code. |
| Inexperience | Low | Low | Low | Allocate time for learning, use familiar tools, seek external resources if needed. |

Table 1: Project Risk and Mitigation Strategies

**2.4. Hardware and Software Resource Requirements**

The TastyTracker project was developed using a combination of readily available hardware and software tools to maximize development efficiency within resource constraints. The following resources are essential for development, testing, and deployment.

Hardware Requirements:

* A laptop or desktop computer with sufficient processing power to run Android Studio efficiently
* (Optional) An Android smartphone to test real-world performance and usability.

Software Requirements

* Development Tools
  + Android Studio- Primary integrated development environment (IDE) for coding a testing.
  + Android Smartphone Emulator- To test the application on various phones of different OS and screen sizes.
  + Java Development Kit (JDK)- Required for compiling and running Java code.
* Database:
  + MySQL: Lightweight, embedded database used to manage inventory, shopping lists, and user data efficiently.
* UI and Design Tools:
  + XML- for designing user interfaces in Android Studio
  + Material Design Components- Used for creating consistent and intuitive UI elements.
* Dependencies and Libraries
  + Android libraries for UI, database management, and synchronization functionalities.
  + Third-party libraries for potential enhancements (e.g. RecyclerView for list displays).

## **2.5. Deliverables and schedule**

The TastyTracker project development was structured with well-defined deliverables and a timeline to ensure timely completion of the core functionalities and features. The schedule prioritizes essential components while maintaining flexibility for iterative testing and adjustments. The deliverables for the project followed an academic schedule and ordering. The following plans were developed for the project with their given deadlines:

* Project Plan- delivered on October 3rd, 2024
* Software Requirement Specification- delivered on October 10th, 2024
* Software Design Specification- delivered on October 24th, 2024
* Test Plan- delivered on November 21st, 2024
* Project Report, User Manual, Developers Guide- delivered on December 5th, 2024

The project development timeline was divided into distinct phases to ensure systematic development and testing. Development also followed the academic calendar in that the final project was due December 5th, with an initial version being due on November 14th. The following chart and table show the initial timeline that the project abided by during the development:

Figure 1: Full Gantt Chart for DevelopmentA screenshot of a spreadsheet

Description automatically generated

A screenshot of a computer

Description automatically generated

Table 2: Table Showing the Dates and Predecessors for the Gantt Chart

# **3. Requirement Specifications**

TastyTracker’s requirement specifications focus on addressing the needs of household users to manage food inventory and shopping lists efficiently, with clearly defined use cases, rationale, and constraints.

## **3.1. Stakeholders for the system**

The stakeholders for the TastyTracker application include all household members (HH, MWP, and MWOP) as well as the developer. The household members are stakeholders as they are the users of the application. The features of the application were chosen to help a variety of households as to best serve the community of stakeholders. This application was developed by a single developer and was not created with an industry sponsor or other business in mind, given this, the single developer is the other stakeholder. The developer is invested in the project’s outcome and has responsibility for the project, making them a primary stakeholder.

## **3.2. Use cases**

The TastyTracker application has a variety of use cases, varied by the privileges of the user that is accessing the application.

## **3.2.1. Graphic use case model**

The following figures are the use cases for the TastyTracker application. I will provide one figure that is the use case model for the entire system and then use case models for each individual actor.

A diagram of a network

Description automatically generatedFigure 2: Use Case Diagram of the Entire TastyTracker System

A diagram of a person with red lines

Description automatically generated

Figure 3: Use Case Diagram for Head of Household

A diagram of a person with blue lines

Description automatically generated

Figure 4: Use Case for MWP

Figure 5: Use Case for MWOPA diagram of a person with green lines

Description automatically generated

### **3.2.2. Textual Description of Use Cases**

1. Head of Household

* Actions
  + View Inventory and Shopping List: Head of household has complete access to view all household inventory/list data
  + Add and Modify Inventory Items: HH can add, edit, and delete items in the inventory.
  + Add and Modify Shopping List items: HH can manage the shopping list, including adding items, marking them as purchased or removing them
  + Request Management
    - HH can view requests submitted by MWOPs
    - HH can approve or reject item addition requests from MWOP
  + Manage Household Permissions: HH can give/revoke permissions for a household member

2. Member with Privileges

* Actions
  + View Inventory and Shopping List: MWP has complete access to view all household inventory/list data
  + Add and Modify Inventory Items: MWP can add, edit, and delete items in the inventory.
  + Add and Modify Shopping List items: MWP can manage the shopping list, including adding items, marking them as purchased or removing them
* Key Limitations
  + MWP cannot approve requests or manage user roles

3. Member without Privileges

* Actions
  + View Inventory and Shopping List: MWOP has complete access to view all household inventory/list data
  + Modify Inventory Items: MWOP can edit and delete items in the inventory.
  + Request to Add Items to Shopping List: MWOP submits requests for adding items to shopping list, which require approval from the HH.
* Key Limitations:
  + MWOP cannot add items to the inventory, they cannot edit the shopping lists, and they must request to add items to the shopping list.

## **3.3. Rationale for your use case model**

The use case model for TastyTracker is designed to reflect the varying levels of user access and responsibilities within a household. Without taking account for the use case for each permission type, some of the features of the application (requests and permissions) would not be properly displayed. Also, the role of a use case diagram is to show the functions of the application. So, if the diagram does not show those features, the use case diagram is not sufficient in describing the system.

## **3.4. Non-functional requirements**

The TastyTracker application is designed with a set of non-functional requirements to ensure reliability, usability, and scalability. These requirements address aspects of the system that are essential for delivering a consistent and effective application. Data Consistency is a non-functional requirement for TastyTracker, as the system must maintain accurate and up-to-date data across multiple devices. Whether users are viewing the inventory, adding items to the shopping list, or approving requests, the application ensures that all data is synchronized in real time. This prevents discrepancies and ensures that all household members have access to the same information. Usability is central to the application’s design, given the diverse roles of its users. The interface has clear workflows for each role. For example, HH users are provided with buttons that other users cannot see and receive notifications when actions are needed. Performance is another key consideration, as TastyTracker must operate efficiently, even with large inventories or multiple concurrent users. Compatibility is essential to ensure the system operates as intended across different Android devices. The application is designed to function on a wide range of screen sizes and Android versions, ensuring a consistent experience for all users. Future scalability includes potential support for cross-platform compatibility with iOS devices.

# **4. Architecture**

## **4.1. Architectural style used**

TastyTracker uses a Model-View-Controller (MVC) architectural style. This design separates the application into three key layers—Model, View, and Controller—which promotes modularity, maintainability, and scalability. Supporting components, such as the database and communication modules, further enhance the architecture.

1. Model- The model layer represents the core data and logic of TastyTracker. It handles inventory shopping lists, user roles, and requests connecting to the SQLite database for data storage and retrieval.
2. View- The view layer is responsible for the user interface, designed using XML layouts in Android Studio. It displays inventory items, shopping lists, and user-specific actions based on roles. The view reflects change in real time by communicating with the controller and fetching updated data from the model.
3. Controller- The controller layer acts as the intermediary between the model and view layers. It processes user actions, validates input, and updates the model as needed. For example, it handles item additions, shopping list modifications, and request approvals. The controller ensures that validated inputs and business rules are followed before making changes to the data.

## **4.2. Architectural model**

A diagram of a software development

Description automatically generated with medium confidence

Figure 6: Component Diagram of Model-View-Controller Architecture

## **4.3. Technology, software, and hardware used**

The following are the technologies and software used to create TastyTracker:

* Android Studio: The IDE used for coding and testing
* Java: Programming language used to code the classes and screens of the application.
* React Native: UI software framework used to design the UI and the movement from screen to screen.
* XML: Language used to design the screens and UI for the application
* SQLite: Embedded database used for storage and management
* Android Emulator: Used to test the application

The following are the hardware requirements for building TastyTracker:

* Laptop/PC capable of running Android Studio and emulator.
* (Optional) Android Smartphone: to test the application on a real-life device.

## **4.4. Rationale for your architectural style and model**

The MVC architecture was chosen to provide clear separation of roles, making the system easier to develop, test, and maintain. The Model ensures robust data management and synchronization, while the View and Controller layers handle user interactions and logic. This modular approach simplifies adding new features, such as integrating external services, without significant changes to the core system. This value was highly important when adding in recipe API integration was a goal of the application. In the future, the MVC architecture will help recipe integration be a more seamless addition to the existing application.

# **5. Design**

## **5.1. User Interface design**

The user interface for TastyTracker is designed to provide a role-specific experience for each household member. The interface uses responsive layouts created with XML in Android Studio, ensuring compatibility across a variety of Android devices and screen sizes. Each user role (Head of Household (HH), Member with Privileges (MWP), and Member without Privileges (MWOP)) has access to different UI elements based on their permissions. The Head of Household sees buttons to settings that other users cannot see. MWOP have buttons hidden for features they are restricted from accessing (for example, MWOP cannot see the “mark items as shopped” buttons or edit button for the shopping list). Also, the interface provides users with help and hint buttons through out the application, specifically in areas where the application context is not intuitive. The UI also relies on notifications to tell the HH when there are requests for them to approve. These notifications alert the HH and are only visible to the HH. Many considerations were made to ensure that the UI is as functional as possible, including making sure that button sizes meet the minimum requirements, the font size adjusts depending on display size, and the text contrasts with the background for readability.

* Initial Screen:

When users first open TastyTracker, they are presented with this screen, which is designed to either allow users to log in to their existing account or create a new account for their household.

* Login Screen:

The login section includes fields for entering the username and password. The user must enter their credentials, and upon successful authentication, they will be directed to the Inventory Screen. If authentication fails, an error message ("Invalid credentials, please try again") will be displayed.

* Registration Screen:

If a user does not have an account, they can select the Register button, which will take them to the registration form. After successfully registering, the user is directed to the inventory screen.

* Inventory Screen:

The inventory screen displays the list of food items currently in the household, with quantity and units shown. The user can add or modify items directly from this screen.

* Shopping List Screen:

Users can view items that have been added to the shopping list, along with their quantities. After shopping, users can mark items as purchased, and the items are automatically moved to the inventory.

* Request Management Screen:

This screen is used by MWOP users to submit requests for adding new items to the inventory or shopping list. HH users can approve or deny these requests. It is accessed by the user pressing the “add item” button on either the inventory or shopping list screen.

* Household Management Screen:

Accessible only by the HH user, this screen displays the list of household members, allowing the HH to manage user privileges.

## **5.2. Components design (static and dynamic models of each component)**

A diagram of a computer

Description automatically generated

Figure 7: Static Model of User Authentication Component

A diagram of a project

Description automatically generated with medium confidence

Figure 8: Dynamic Model for User Authentication Component

A diagram of a food screen

Description automatically generated

Figure 9: Inventory Management Static Model

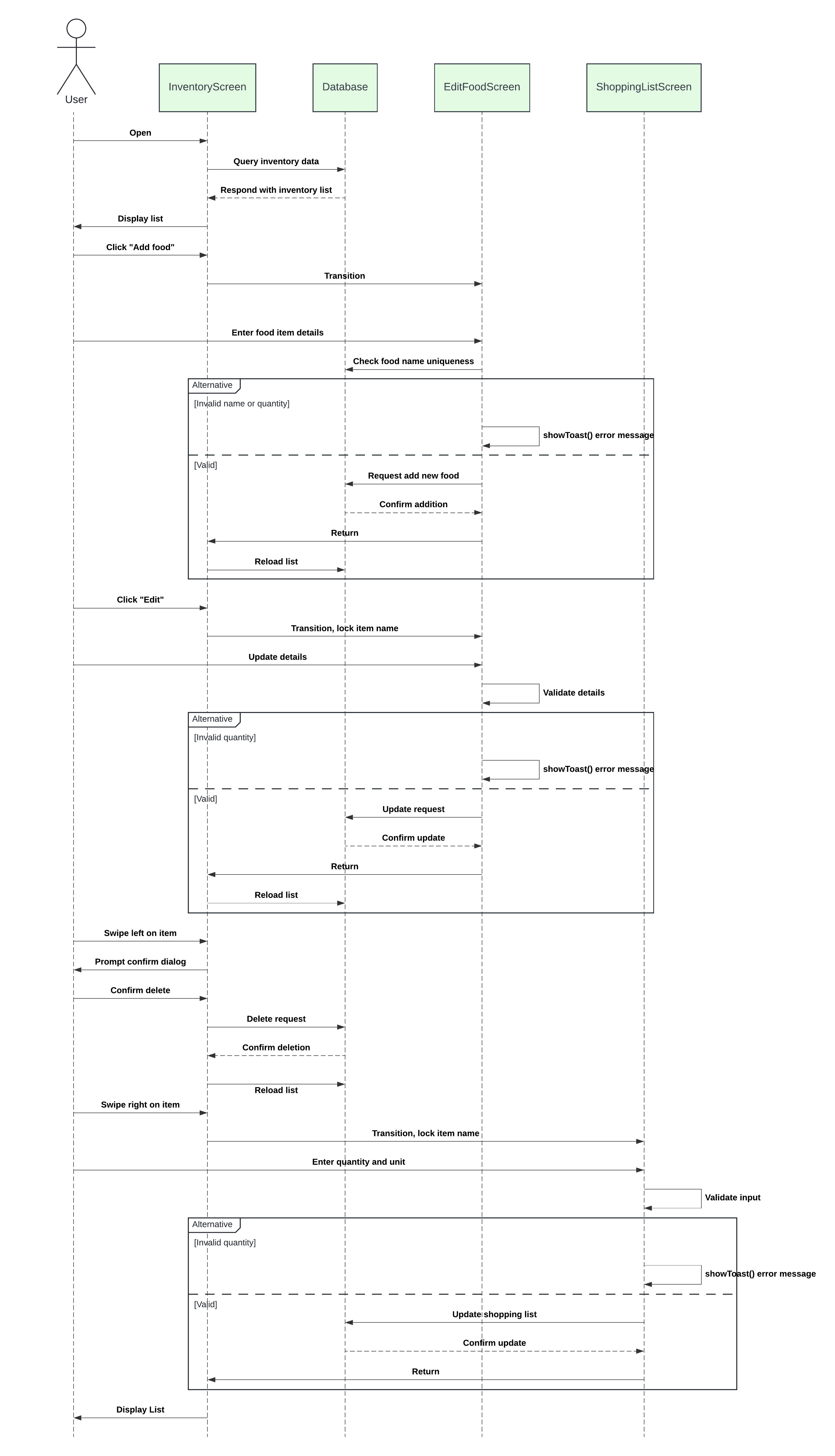


Figure 10: Dynamic Model for Inventory Management Component

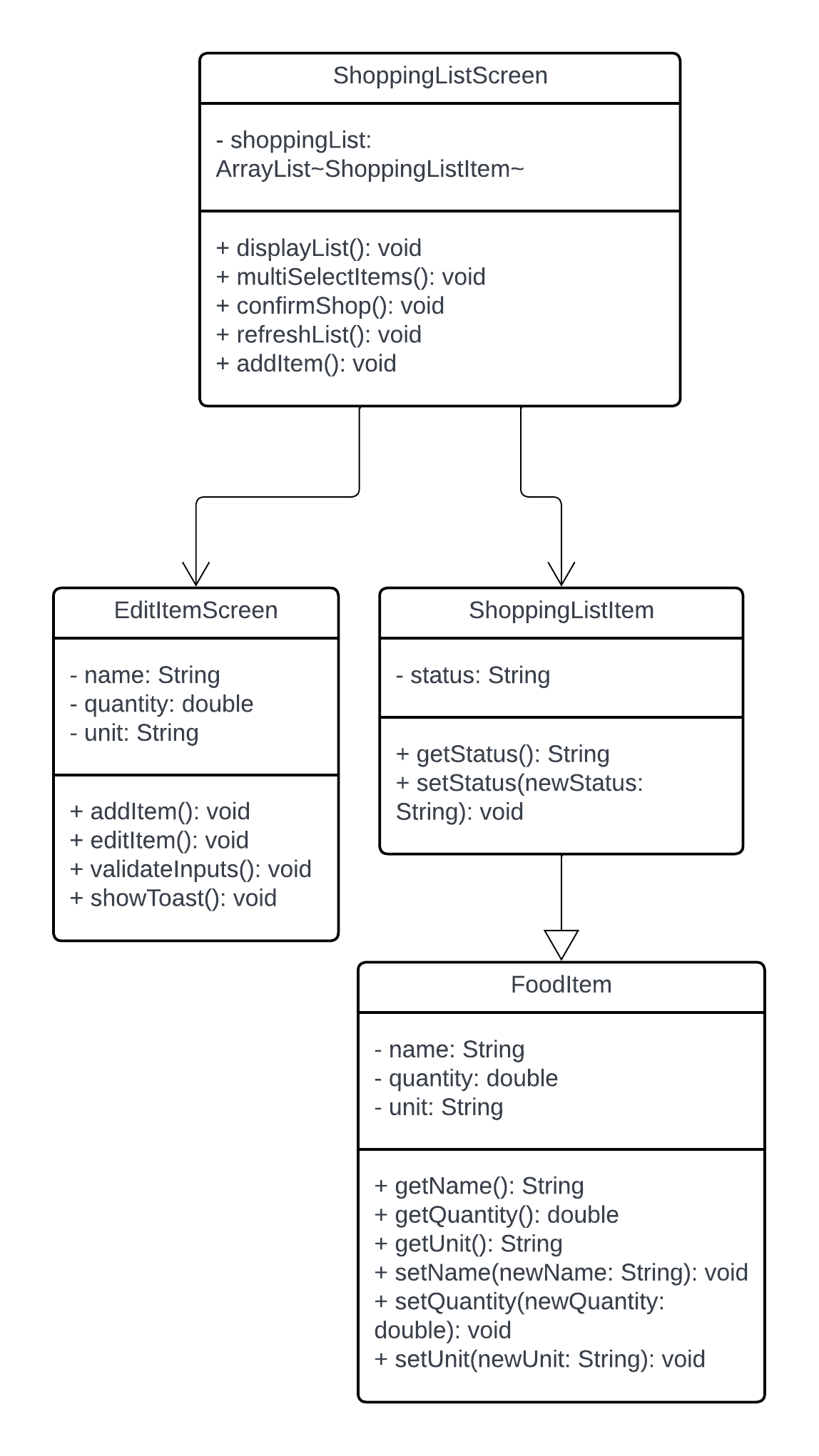


Figure 11: Static Model for Shopping List Management

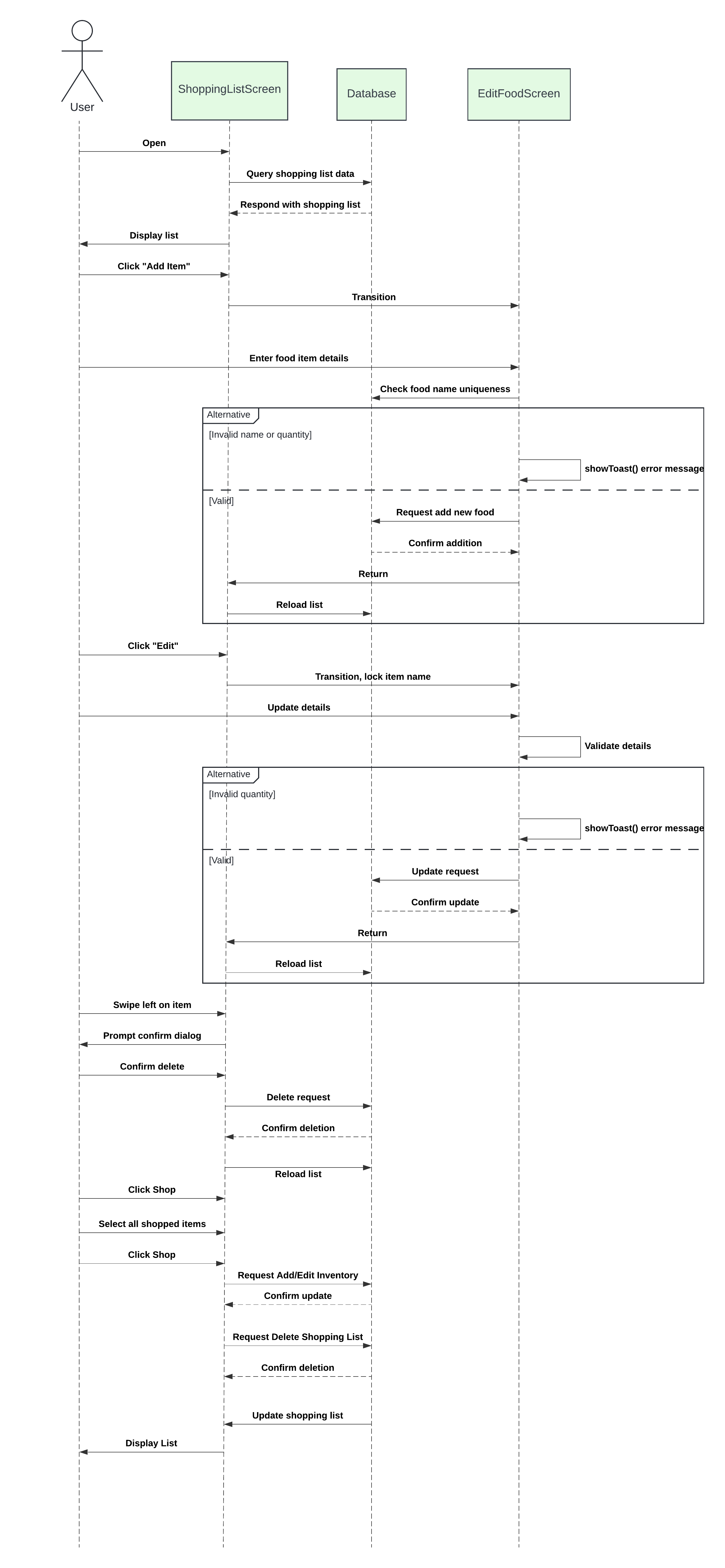


Figure 12: Dynamic Model for Shopping List Management

## **5.3. Database design**

A diagram of a computer

Description automatically generated with medium confidence

Figure 13: Database Diagram

## **5.4. Rationale for your detailed design models**

The detailed design models for TastyTracker are built to align with the role-specific requirements of the application while ensuring modularity, scalability, and user-friendliness. The UML diagrams provided illustrate the core design choices that enable efficient management of inventory, shopping lists, and user permissions. The class diagrams highlight the organization of the system. Each class focuses on a specific functionality, such as managing inventory (InventoryScreen), requests (RequestScreen), and user registration (RegisterScreen). This modular structure simplifies development, testing, and maintenance. For example, the separation of the FoodItem class ensures that inventory management operations are encapsulated, making it easier to extend or modify functionality in the future. The sequence diagrams provide a clear view of how the system handles interactions between users and components. For instance, the sequence diagram for request handling shows the interaction between MWOP users, the database, and the HH for approving or denying requests. These diagrams emphasize the smooth flow of data and actions, ensuring that each user role has a tailored workflow. This role-based interaction design minimizes errors and ensures that users only have access to relevant functionality. The database design is well-integrated with the application's components, as shown in the sequence diagrams. The Requests table, for example, facilitates MWOP requests and ensures that HH decisions are logged accurately. The ShoppingList and Inventory tables are structured to handle dynamic updates efficiently, ensuring real-time synchronization across household devices.

The detailed design also incorporates error handling and validation at every stage, as seen in the diagrams. This ensures data consistency and improves the user experience. For instance, the validation of usernames during registration prevents duplicate accounts, while the validation of food item details ensures accurate inventory tracking.

## **5.5. Traceability from requirements to detailed design models**

The design of TastyTracker directly maps to the functional requirements identified during the planning phase, ensuring alignment between the system’s objectives and its implementation.

1. Inventory Management:
   * Requirement: Users must be able to add, edit, and delete inventory items.
   * Design: EditItemActivity enables adding and editing inventory items, while the Inventory table in the database supports storage and updates.
2. Shopping List Management:
   * Requirement: Users must manage shopping lists and track purchased items.
   * Design: The ShoppingListAdapter manages the UI for shopping lists, while the Shopping List table tracks items and their statuses.
3. Role-Based Permissions:
   * Requirement: Users have access to features based on their role.
   * Design: The database's User Roles table and the dynamic UI enforce role-specific permissions.
4. Request Management:
   * Requirement: MWOP users can submit item requests for HH approval.
   * Design: RequestApprovalActivity handles these workflows, and the Requests table logs submission and approval statuses.
5. Data Consistency:
   * Requirement: Data must be synchronized across devices.
   * Design: The synchronization module ensures updates are propagated in real-time, using the SQLite database as the central data store.

# **6. Test Management**

## **6.1. A complete list of system test cases**

This section gives a complete list of test cases for the software.

|  |  |
| --- | --- |
| ID | 1 |
| Test Input | Add 'Milk, 1 gallon' to inventory |
| Expected Output | 'Milk' is added to inventory list with correct quantity and unit. |
| Description | Validates that users can add items to the inventory with accurate details. |

Table 3: Test Case 1

|  |  |
| --- | --- |
| ID | 2 |
| Test Input | Update 'Milk' quantity to 0.5 |
| Expected Output | Inventory displays ‘Milk’ with updated quantity of 0.5. |
| Description | Validates updating the quantity of an existing inventory item. |

Table 4: Test Case 2

|  |  |
| --- | --- |
| ID | 3 |
| Test Input | Delete ‘Milk’ from inventory |
| Expected Output | ‘Milk’ is removed from the inventory list. |
| Description | Validates that users can delete items from inventory. |

Table 5: Test Case 3

|  |  |
| --- | --- |
| ID | 4 |
| Test Input | Add 'Bread, 2 loaves' to shopping list |
| Expected Output | 'Bread' is added to shopping list with correct quantity. |
| Description | Validates adding items to the shopping list manually. |

Table 6: Test Case 4

|  |  |
| --- | --- |
| ID | 5 |
| Test Input | Add ‘Milk’ from inventory to shopping list with quantity ‘1 gallon' |
| Expected Output | ‘Milk’ is added to shopping list with correct quantity. |
| Description | Validates adding items to the shopping list via the inventory. |

Table 7: Test Case 5

|  |  |
| --- | --- |
| ID | 6 |
| Test Input | Mark 'Bread' as purchased |
| Expected Output | 'Bread' moves to inventory with correct quantity and unit. ‘Bread’ is removed from the shopping list. |
| Description | Verifies 'Mark as Purchased' transfers items from shopping list to inventory correctly. |

Table 8: Test Case 6

|  |  |
| --- | --- |
| ID | 7 |
| Test Input | Log in as Head of Household (HH) and attempt to access inventory, shopping list, recipe search, request management, and privilege management features. |
| Expected Output | The HH can access all inventory management, shopping list management, recipe search, request management, and privilege management functions without restrictions. |
| Description | Validates that the HH has full access to all functionalities. |

Table 9: Test Case 7

|  |  |
| --- | --- |
| ID | 8 |
| Test Input | Log in as Member with Privileges (MWP) and attempt to add to inventory, add to shopping list, and access recipe search features, request management, and privilege management |
| Expected Output | MWP can add to inventory/shopping list and access recipe features but cannot access privilege management or request management. |
| Description | Validates the MWP functionalities |

Table 10: Test Case 8

|  |  |
| --- | --- |
| ID | 9 |
| Test Input | Log in as Member without Privileges (MWOP) and attempt to add to inventory/shopping, and access recipe search features, request management, and privilege management. |
| Expected Output | MWOP can view the inventory/shopping list, but when tries to add to either is made to submit a request. MWOP can access recipe features but cannot access privilege management or request management. |
| Description | Validates the MWOP functionalities |

Table 11: Test Case 9

|  |  |
| --- | --- |
| ID | 10 |
| Test Input | Log in as HH and change a MWOP to MWP privileges while they are not in the app and then log in to the new MWP account |
| Expected Output | The account should now allow full MWP functionality (can be tested with 8) |
| Description | Verifies the permissions change when the user is not in the application |

Table 12: Test Case 10

|  |  |
| --- | --- |
| ID | 11 |
| Test Input | Log in as HH and change a MWOP to MWP privileges while they are not in the app and then log in to the new MWP account |
| Expected Output | The account should now allow full MWP functionality (can be tested with 8) |
| Description | Verifies the permissions change (MWOP to MWP) when the user is not in the application |

Table 13: Test Case 11

|  |  |
| --- | --- |
| ID | 12 |
| Test Input | Log in as HH and change a MWP to MWOP privileges while they are not in the app and then log in to the new MWOP account |
| Expected Output | The account should now allow only MWOP functionality (can be tested with 9) |
| Description | Verifies the permissions change (MWP to MWOP) when the user is not in the application |

Table 14: Test Case 12

|  |  |
| --- | --- |
| ID | 13 |
| Test Input | Log in as HH and change a MWOP to MWP privileges while they are in the app, on refresh or screen change the new MWP should have new functionality |
| Expected Output | The account should now allow full MWP functionality (can be tested with 8) |
| Description | Verifies the permissions change (MWOP to MWP) when the user is in the application |

Table 15: Test Case 13

|  |  |
| --- | --- |
| ID | 14 |
| Test Input | Log in as HH and change a MWP to MWOP privileges while they are in the app, on refresh or screen change the new MWOP should have restricted functionality |
| Expected Output | The account should now allow only MWOP functionality (can be tested with 9) |
| Description | Verifies the permissions change (MWP to MWOP) when the user is in the application |

Table 16: Test Case 14

|  |  |
| --- | --- |
| ID | 15 |
| Test Input | MWOP requests 'Add Apples, 5' for inventory |
| Expected Output | Apples is not added to the list and MWOP gets text notification that a request has been sent |
| Description | Validates request submission to inventory for MWOP. |

Table 17: Test Case 15

|  |  |
| --- | --- |
| ID | 16 |
| Test Input | HH approves request for 'Apples, 5' |
| Expected Output | 'Apples' added to the inventory; request status updates to 'Approved' and item is removed from requests list. |
| Description | Validates request approval process by Head of Household (HH). |

Table 18: Test Case 16

|  |  |
| --- | --- |
| ID | 17 |
| Test Input | HH rejects request for ‘Apples, 5’ |
| Expected Output | Request status updates to 'Rejected'; no inventory change and item is removed from requests list. |
| Description | Validates request rejection functionality by Head of Household (HH). |

Table 19: Test Case 17

|  |  |
| --- | --- |
| ID | 18 |
| Test Input | MWOP requests 'Add Apples, 5' for shopping list |
| Expected Output | Apples is not added to the list and MWOP gets text notification that a request has been sent |
| Description | Validates request submission to shopping list for MWOP. |

Table 20: Test Case 18

|  |  |
| --- | --- |
| ID | 19 |
| Test Input | User A and User B both add “Apples, 5 each” to list at the same time (A presses save first). |
| Expected Output | List displays Apples with quantity 5, user B gets notification that “Apples was added to inventory recently, double check inventory and retry if necessary” |
| Description | Verifies that concurrent additions are on purpose and not a matter of users accidentally adding the same set of items at the same time |

Table 21: Test Case 19

|  |  |
| --- | --- |
| ID | 20 |
| Test Input | User A and User B both edit “Apples” at the same time (A enters edit first). (This also works for delete as deleting option is within the edit screen). |
| Expected Output | A is allowed to edit Apples. B is rejected from entering edit screen and notification displays that Apples is already being edited by another user. |
| Description | Verifies that concurrent editing of an item is not allowed to ensure data integrity. |

Table 22: Test Case 20

|  |  |
| --- | --- |
| ID | 21 |
| Test Input | User A marks “Bread” as shopped in the shopping list while User B attempts to delete “Bread” from the shopping list. |
| Expected Output | B receives a notification that Bread cannot be deleted as it is marked as shopped, either the “Mark all as shopped” button must be pressed to process the purchasing or Bread must be unchecked. |
| Description | Verifies that an item cannot be deleted when it is marked for shopped. |

Table 23: Test Case 21

## **6.2. Traceability of test cases to use cases**

Each test case is directly tied to one or more use cases, ensuring comprehensive validation of system functionality. For example:

* Use Case: Add Inventory Item
  + Test Case 1: Adding an item to the inventory.
  + Test Case 2: Updating an inventory item's quantity.
  + Test Case 3: Deleting an inventory item.
* Use Case: Request Management
  + Test Case 15: Submitting a request as MWOP.
  + Test Case 16: HH approving a request.
  + Test Case 17: HH rejecting a request.
* Use Case: Role Management
  + Test Cases 10–14: Validating permission changes across roles.

## **6.3. Techniques used for test case generation**

In order to generate test cases, a variety of methods were used. Firstly requirement-based testing. Test cases were derived directly from the functional requirements and use cases identified during the design phase. These test cases are to make sure that the application does what it is intended to do. Next, I tried to think of not just if the user did everything correctly—what if they did everything incorrectly. Both valid and invalid inputs were tested to ensure the system behaves correctly in all scenarios. For example, adding a valid inventory item was tested alongside attempts to add items with invalid names or negative quantities. Lastly, I tried to make sure the entire sequence of functions worked as intended. For example, I wanted to make sure that when a MWOP user submits a request, the HH can review and approve it, and then the item is added to the inventory.

## **6.4. Test results and assessments**

The majority of test cases for inventory management, shopping list operations, and role-based permissions passed successfully. There were a few issues found with delays in real-time synchronization when two users are in the application at one time. Specifically, the application was not refreshing for the second user when the first updated either the inventory or the shopping list. Also, I found in using the app outside of the test cases, there were issues that I had not accounted for. For example, I had not accounted for users checking an item as shopped and then moving away from the shopping list screen without using the “mark as shopped” button first. This caused items to be incorrectly marked as shopped and messed up their display in the shopping list. I think that the test cases I developed were good for confirming in depth quality of the core functions. However, they were not robust and did not cover all of the errors that the application may actually have. Some of those issues, however, may best be found through use of the application.

## **6.5. Defects reports**

Defects identified (and resolved) during testing included:

* 1. Real-time synchronization lag when multiple user were looking at the inventory and another user changed the inventory
  2. User can inadvertently delete items off of the shopping list be checking an item in the shopping list and then moving away from the shopping list screen without pressing “mark as shopped”
  3. Incorrect UI scaling on devices of different screen sizes.

# **7. Conclusions**

## **7.1. Outcomes of the project**

The TastyTracker application achieved its primary objective of providing a household food inventory management system. The application implements all the core functionalities such as role-based permissions, inventory tracking, and shopping list management, ensuring users can collaborate effectively. Additionally, the incorporation of role-based permissions and the requests system allows users more control over their inventory and household management. The system also incorporates data synchronization, enabling consistent updates across multiple devices in real-time. While the initial plan included integrating recipe management features via an external API, this functionality was not implemented due to time constraints. Overall, I feel the application is a success and I am very proud of the app I have built.

## **7.2. Lessons learned**

I feel significantly more confident in my app development skills. I learned a lot and gained a lot of insight in terms of coding and specifically building UI’s. However, one of the biggest things I learned is how much writing goes into software development. I have been shocked by the amount of writing I have had to do in the development of this application. While it was very time consuming, I did find a lot of value in this work. Normally when I develop a program, I don’t follow a strict process and i figure out the application “on the fly”. By being forced to consider the application before development, I really ironed out a lot of the details ahead of time. This means I didn’t code something just to realize it wouldn’t work because of a detail I didn’t foresee, I already thought it through. It also made coding simpler following the many diagrams that I made. I built the classes, databases, etc. quickly because I already knew what they needed to look like.

## **7.3. Future development**

The only goal I didn’t achieve was implementing the recipe integration. In the future, it would be great to see that goal realized in the TastyTracker application. As mentioned previously, I already have the idea and structure for this figured out, so it would just be a matter of implementing it. Also, I would like to see the application usable on iOS devices so that I could see the application on my own phone. I think seeing my application on my own phone would be an immense point of pride. While there is space for TastyTracker to grow, I am still very pleased with the app in its current form, and I am proud of the work I have done.

# **References**

This paper did not use any external resources, but did use LucidSpark to create all figures.