**Fooditude: A Community-Driven Mobile Application for Hyper-Local Food Waste Reduction**

**From Web to Mobile: Implementing a Cross-Platform React Native Solution**

**Course:** CSIS 4495 - Applied Research Project, Section 001  
**Student:** Frederick K. Okornoe (300373397)  
**Date:** October 25, 2025

**Table of Contents**

1. Introduction
2. Proposed Research Project
3. Project Planning and Timeline
4. Implemented Feature 1: Cross-Platform Mobile Application Core
   1. Introduction and Pivot to Mobile-First
   2. React Native Architecture and Component Library
5. Implemented Feature 2: Integrated User and Data Management System
   1. Introduction to System Integration
   2. Authentication and State Management
   3. Data Flow and API Integration
6. Lessons Learned and Future Work
7. Concluding Remarks
8. Appendix
   1. Appendix A: Installation Guide
   2. Appendix B: User Guide
   3. Appendix C: API and Backend Services
   4. Appendix D: Technology Stack & Architecture
   5. Appendix E: Code Explanation

**Introduction**

This report details the midterm progress of the applied research project, "Fooditude." The project's core mission remains to tackle the significant problem of household food waste by creating a digital platform that facilitates the sharing of surplus food within local communities. As outlined in the original proposal, the goal is to bridge the gap between individual food surplus and local need, fostering community engagement and promoting sustainable practices. This midterm report demonstrates the transition from the proposal phase into active, tangible development, resulting in a functional, cross-platform mobile application prototype.

**Proposed Research Project**

The proposed project aims to design, develop, and validate a functional prototype of the "Fooditude" platform. The core objectives are:

1. **Design and Develop:** Create an intuitive, mobile-first application with modules for user profiles, food listing, discovery, claiming, and communication.
2. **Pilot Study:** Deploy the prototype in a targeted community to test usability and functionality.
3. **Data Collection and Analysis:** Gather quantitative and qualitative data to assess the platform's effectiveness in changing food-sharing behaviors and reducing waste.

**Project Planning and Timeline**

The project has adhered closely to the Agile methodology and timeline outlined in the proposal. The "Phase 2: Development" milestones have been successfully executed, with a significant strategic pivot noted below. The following Gantt chart summarizes the completed and upcoming phases:

A screenshot of a computer

AI-generated content may be incorrect.

**Implemented Feature 1: Cross-Platform Mobile Application Core**

**Introduction and Pivot to Mobile-First**

During the initial development phase, a critical strategic decision was made to pivot from a **mobile-responsive web application** (React.js) to a **true cross-platform mobile application** using **React Native and Expo**. This decision was driven by user-centric design principles and research into successful sharing economy platforms, which are predominantly native mobile applications. A mobile app offers superior access to device features like the camera for instant photo uploads, push notifications for instant updates on claims, and a more immersive, performant user experience—all critical factors for a platform reliant on timely, location-based exchanges.

**React Native Architecture and Component Library**

The core of the Fooditude application has been built as a structured collection of reusable React Native components, forming a solid foundation for all user interactions.

**Details of Implementation:**

1. **Unified Design System (**theme.ts**):** A central theme file was created to enforce design consistency across the entire application. This exports all colors, spacing, font sizes, and border radii, ensuring the UI is cohesive and maintainable.

A screenshot of a computer program

AI-generated content may be incorrect. A screen shot of a computer program

AI-generated content may be incorrect. A screen shot of a computer program

AI-generated content may be incorrect.

*Screenshot 1: Excerpt from*theme.ts*showing the color palette and spacing constants.*

1. **Core UI Components:** The application's user interface is constructed from a set of purpose-built components:
   * ItemCard**:** Displays a preview of a food listing, complete with an image, title, description, dietary tags, and key details like location and expiry date. It features intelligent "Urgent" and "Reserved" badges driven by a formatExpiryDate function.
   * Header**:** Contains the app logo, a "Share Food" button, and a user menu dropdown for accessing the profile and logging out.
   * ItemFilters**:** Provides a search input and dropdown filters for "Category" and "Dietary" preferences, allowing users to efficiently browse listings.

A close up of a tomato

AI-generated content may be incorrect. A screenshot of a food recipe

AI-generated content may be incorrect.

*Screenshot 2: The main Browse screen showing a list of ItemCard components and the Header.*

1. **Modal-Based User Flows:** Key flows are handled through full-screen modals for a focused user experience.
   * AddItemModal**:** A comprehensive form for sharing food, featuring field validation, a custom category picker, dietary tag selection, and integrated image upload from the camera or gallery.
   * ItemDetailModal**:** Provides an in-depth view of a selected food item, displaying all information and action buttons to "Reserve This Food" or "Contact Donor."
   * A screenshot of a phone

     AI-generated content may be incorrect.  
     *Screenshot 3: The AddItemModal open, showing the form and category picker.*
   * A screenshot of a computer

     AI-generated content may be incorrect. *A screenshot of a phone

     AI-generated content may be incorrect.*  
     *Screenshot 4: The ItemDetailModal showing a food item's full details.*

**Implemented Feature 2: Integrated User and Data Management System**

**Introduction to System Integration**

A functional prototype requires more than a static interface; it needs a dynamic data layer and user system. This feature encompasses the full-stack integration that brings the Fooditude UI to life, enabling user authentication, real-time data flow, and state management.

**Authentication and State Management**

A secure and user-friendly authentication system has been implemented.

**Details of Implementation:**

1. **Login & Signup Flow (**LoginPage.tsx**):** A tab-based interface allows users to either log in or create a new account. The form includes validation and secure password handling.
2. **API Service Layer (**api.ts**):** A dedicated service layer handles all communication with the backend. It manages authentication tokens, securely stored using React Native's AsyncStorage, and includes robust error handling.
3. **Application State:** The application state, including the authenticated user object and the global list of food items, is managed using React's Context API or component state, ensuring a reactive UI.

**Data Flow and API Integration**

The frontend components are fully integrated with backend services via a defined API.

**Details of Implementation:**

1. **Data Models:** TypeScript interfaces (like FoodItem and User) are used throughout the application to ensure data consistency and type safety.
2. **CRUD Operations:** The api.ts module exposes methods for all necessary operations:
   * **Auth:** authAPI.signup, authAPI.signin
   * **Items:** itemsAPI.getAll, itemsAPI.create
   * **Profile & Reviews:** profileAPI.get, reviewsAPI.getByUser
   * **Messaging:** messagesAPI.send
3. **Image Upload Service:** A mobile-optimized uploadImageMobile function handles image uploads from the device's storage or camera, returning a URL to be stored with the food listing.

A screen shot of a computer program

AI-generated content may be incorrect. A computer screen with text

AI-generated content may be incorrect.

*Screenshot 5: Code excerpt from*api.ts*showing the*makeRequest*function and*authAPI*methods.*

A screenshot of a login form

AI-generated content may be incorrect. *A screenshot of a login form

AI-generated content may be incorrect.*  
*Screenshot 6: The LoginPage screen, showing the login and signup tabs.*

**Lessons Learned and Future Work**

**Lessons Learned:**

* **Pivot Validation:** The decision to switch to React Native was crucial. The development process highlighted the importance of choosing a technology stack that aligns perfectly with the target user's behavior (i.e., using a mobile device for on-the-go, location-based tasks).
* **TypeScript is Invaluable:** Using TypeScript from the outset prevented numerous potential runtime errors and served as excellent documentation, making the development process more efficient.
* **Component-Driven Development:** Building a library of reusable components accelerated development and ensured UI consistency.

**Future Work:**

* **Complete Backend Integration:** The immediate next step is to finalize the connection between the frontend and the Supabase backend for all API endpoints.
* **Pilot Testing:** Conduct the planned pilot study to gather real-world usage data and user feedback.
* **Enhanced Features:** Implement push notifications, refine the messaging system, and add more advanced features like mapping for pickup locations based on the data and feedback from the pilot.

**Concluding Remarks**

The midterm milestone for the Fooditude project has been successfully achieved. The core hypothesis—that a user-friendly, community-driven digital platform can facilitate food sharing—is now testable through a fully functional, cross-platform mobile application prototype. All primary modules from the proposal (User Profile, Food Listing, Discovery & Search, Claiming & Reservation, and In-App Messaging) have been implemented. The project is on schedule to proceed to Phase 3: Testing & Reporting. I would like to acknowledge Professor Bambang A.B. Sarif for his guidance throughout the project's development.

**Appendix**

**Appendix A: Installation Guide**

1. **Prerequisites:** Ensure you have Node.js, npm/yarn, and the Expo CLI installed.
2. **Clone the Repository:**

bash

1. git clone <https://github.com/freddouglas/F2025_4495_071_Fok397>
2. **Install Dependencies:**

bash

npm install

1. **Start the Development Server:**

bash

npm start

1. **Run on Device:** Scan the QR code with the Expo Go app (Android) or the Camera app (iOS).

**Appendix B: User Guide**

1. **Onboarding:** Open the app and create an account or log in.
2. **Browsing Food:** View available food items on the main Browse screen. Use the search bar and filters to find specific items.
3. **Sharing Food:** Tap the "+ Share" button in the header. Fill out the form, take a picture of your food, and submit.
4. **Claiming Food:** Tap on a food item to view its details. If available, tap "Reserve This Food" to claim it.
5. **Viewing Profile:** Tap your avatar in the header and select "View Profile" to see your stats and reviews.

**Appendix C: API and Backend Services**

The application uses a custom backend API built with Node.js and Supabase, hosted on Google Cloud Platform. The API endpoints are structured as follows:

* POST /auth/signup - User registration
* POST /auth/signin - User login
* GET /items - Fetch all food listings
* POST /items - Create a new food listing
* POST /upload-image - Handle image uploads

**Appendix D: Technology Stack & Architecture**

* **Frontend/Mobile:** React Native, Expo, TypeScript
* **Backend:** Node.js, Express.js
* **Database & Auth:** Supabase (PostgreSQL, Authentication)
* **Cloud Platform:** Google Cloud Platform (GCP)
* **State Management:** React Context API / Component State
* **Storage:** AsyncStorage for local data, Supabase Storage for images

**Appendix E: Code Explanation**

**Key Code Snippet: The**ItemCard**Component**  
The ItemCard is a central component. It uses the formatExpiryDate function to dynamically calculate and display the expiry status. It conditionally renders badges based on the item's status and expiry urgency.

typescript

// Example function from ItemCard.tsx

A computer screen shot of code

AI-generated content may be incorrect.

*This function takes an ISO date string and returns a user-friendly relative date string, which is crucial for conveying urgency to the user.*