# Submission 3: Detailed Design for "Munchly"

## 1. Architecture Overview

The "Munchly" application follows the Model-View-Controller (MVC) architectural pattern, implemented within a Client-Server Architecture. The separation of concerns ensures modularity, scalability, and maintainability.

### Model:

- Resides on the server (Express.js).  
- Handles data storage and retrieval from Firebase Firestore.  
- Manages fridge inventory, user data, and recipe storage.  
- Interacts with Gemini for dynamic recipe generation.

### View:

- Resides on the client (React).  
- Provides an interactive, responsive user interface optimized for mobile.  
- Displays data such as inventory, recipes, and other features.

### Controller:

- Resides on the server (Express.js).  
- Processes client requests, communicates with the Model, and sends responses back to the View.  
- Handles logic for Google Vision API, Firebase, and Gemini AI integration.

## 2. System Components

### Frontend (View):

- React: For dynamic and responsive UI development.  
- React Native (if needed): For native mobile functionality.  
- UI Features:  
 - Camera interface for scanning fridge contents.  
 - Inventory and recipe screens.

### Backend (Controller):

- Express.js: For server-side logic and API management.  
- Firebase Admin SDK: For database and storage operations.  
- Gemini: For generating dynamic and personalized recipes.

### Database (Model):

- Firebase Firestore: Stores user inventory, preferences, and recipes.  
- Firebase Storage: Manages image uploads.

## 3. Data Flow

### Image Input:

- Users capture or upload fridge images via the client (React).  
- The image is sent to the server, where the Google Vision API processes it to identify items.

### Data Management:

- Identified items are stored and updated in Firebase Firestore.  
- Recipes are retrieved from Firebase or generated dynamically via Gemini.

## 4. API Design

### Express API Endpoints:

- POST /api/upload: Handles image uploads for Google Vision API processing.  
- GET /api/inventory: Retrieves the user’s inventory from Firebase Firestore.  
- POST /api/recipe: Generates recipes dynamically using Gemini AI.

### Firebase Firestore Collections:

- Inventory Collection:  
 - Fields: item\_id, name, category, expiration\_date.  
- Recipe Collection:  
 - Fields: recipe\_id, name, ingredients, instructions.

## 5. Testing

- Jest: For testing React components and backend logic.  
- Supertest: For API endpoint testing in Express.js.

## 6. Tools and Technologies

- Frontend: React, React Native (if needed).  
- Backend: Express.js.  
- Database: Firebase Firestore and Firebase Storage.  
- Testing: Jest and Supertest.  
- AI Integration:  
 - Google Vision API: For object detection and image recognition.  
 - Gemini: Integrated with Firebase for seamless personalized recipe generation.