
# Save n Bite – Software Requirements Specification (SRS)

Project Name: Save n Bite
Version: 4.0
Date: [27/09/2025]
Team: [Secure Web & Mobile Guild]
Client: Gendac

Secure Web & Mobile Guild

Team:
Sabrina-Gabriel Freeman
Marco Geral
Chisom Emekpo
Vanè Abrams
Capleton Chapfika

Contact Email: swmguild@gmail.com

# CONTENTS





Secure Web & Mobile Guild
swmguild@gmail.com

# 1. INTRODUCTION

## Business Need

Food waste is a major global concern, with tons of surplus food being discarded every day despite being safe for consumption. This not only contributes to environmental harm but also overlooks the needs of food-insecure individuals and communities. Businesses such as restaurants, hotels, and grocery stores frequently dispose of edible surplus food due to logistical and regulatory constraints.

Simultaneously, many individuals—especially students and low-income groups—face food insecurity. There is a need for a structured, secure, and accessible platform that facilitates the redistribution of surplus food in a way that is efficient, compliant with safety regulations, and scalable across communities.

## Project Scope

Save n Bite will develop a web and mobile platform that connects food suppliers with individuals and organisations in need. The system will allow verified users to list, browse, request, purchase, or request a donation of surplus food. It includes:

- Role-based dashboards
- Scheduling tools for pickups
- Innovative gamification to promote user engagement
- Review features

Exclusions:

- Delivery and logistics are not part of this project.

# 2. USER STORIES / EPICS

EPIC 1: User Profile Management

| User Stories | Acceptance Criteria |
|---|---|
| 1. As a Food Provider, I want to register a profile so that I can list surplus food items for sale or donation. | Given that a user wants to register as a Food Provider, when they select "Register as Food Provider" and enter business details, contact information, and license documents, then a confirmation of successful registration is received. |
| 2. As an Individual Consumer, I want to register an account so that I can browse and purchase food. | Given that an Individual Consumer wants to register an account, when they choose "Register as Individual" and provide their name, email, and password, then a confirmation email is sent upon successful registration. |
| 3. As an organisation, I want to register to request food donations for our cause. | Given that an organisation wants to register for food donation requests, when they select "Register as organisation" and upload proof of registration, then their status is set to "Verification pending" until approved by an admin. |
| 4. As a System Administrator, I want to verify user accounts so that only legitimate providers and organisations are approved. | Given that a System Administrator wants to verify user accounts, when they access the admin dashboard and review uploaded documents, then they can approve or reject users with comments. |

| | |
|---|---|
| 5. As any User, I want to update my profile information so that I can keep my data current. | Given that a user wants to update their profile information, when they edit fields such as email, phone, profile image, and upload updated documents, then their profile information is updated accordingly. |
| 6. As a System Administrator, I want to manage user roles and permissions so that appropriate access is granted to each user type. | Given that a System Administrator wants to manage user roles and permissions, when they access role management features, then they can assign roles and restrict access based on the selected role. |

EPIC 2: Food Listing and Management

| User Stories | Acceptance Criteria |
|---|---|
| 1. As a Food Provider, I want to create a food listing so that surplus food can be made available. | Given that a Food Provider wants to list surplus food, when they enter the food name, description, expiration date, and optionally upload images and choose a pickup/delivery method, then the food listing is created as either a donation or discount item. |
| 2. As a Food Provider, I want to receive AI-based listing suggestions so that I can optimize sales and reduce waste. | Given that a Food Provider is creating or updating a listing, when they input relevant details, then the system suggests optimal listing time and quantity based on historical data. |
| 3. As a Food Provider, I want to view and manage my own listings so that I can update or remove them as needed. | Given that a Food Provider wants to manage existing listings, when they access their dashboard, then they can view, edit, or delete current and past food items. |

EPIC 3: Food Discovery

| User Stories | Acceptance Criteria |
|---|---|
| 1. As an Individual Consumer, I want to search for food items so that I can find meals that meet my needs. | Given that an Individual Consumer is searching for food, when they use the search bar and apply filters by name, type, expiration date, or location, then relevant food items are displayed. |
| 2. As an organisation, I want to browse food listings so that I can find donations suitable for our needs. | Given that an organisation is browsing for food donations, when they filter listings by type, quantity, or expiration date, then donation-only items relevant to their needs are shown. |
| 3. As any User, I want to view food details so that I can make informed decisions. | Given that any User is viewing a food listing, when they select an item, then the full details including description, images, listing date, and provider details are visible. |
| 4. As any User, I want to receive notifications about new listings so that I can act quickly. | Given that any User has enabled notifications, when new listings that match their preferences (e.g., category or area) are posted, then they receive email or in-app alerts. |



Secure Web & Mobile Guild
swmguild@gmail.com

EPIC 4: Transactions & Donations

| User Stories | Acceptance Criteria |
|---|---|
| 1. As an Individual Consumer, I want to purchase discounted food items so that I can reduce my food cost. | Given that an Individual Consumer is ready to make a purchase, when they proceed to checkout and complete the payment, then they can review their final order before payment confirmation. |
| 2. As an organisation, I want to request food donations so that we can support those in need. | Given that an organisation wants to request a donation, when they submit a request for a specific donation listing, then they receive confirmation and available delivery options. |
| 3. As any User, I want to view my transaction history so that I can track my past activities. | Given that any User wants to review their past transactions, when they access their transaction history, then they can filter by date and type (purchase/donation) and view order details and statuses. |

EPIC 5: Pickup Coordination

| User Stories | Acceptance Criteria |
|---|---|
| 1. As an Individual Consumer, I want to schedule a pickup so that I can collect food at a convenient time. | Given that an Individual Consumer is finalizing a request or purchase, when they select a pickup time slot, then they receive pickup location, contact information, and a confirmation/reminder notification. |
| 2. As a Food Provider, I want to manage pickup schedules so that I can prepare the food ahead of collection times. | Given that a Food Provider wants to prepare for pickups, when they define pickup windows and view the upcoming schedule, then they can manage pickups and mark them as completed when done. |
| 3. As a System Administrator, I want to view and monitor all scheduled pickups so that I can ensure operations run smoothly. | Given that a System Administrator wants to monitor pickup activities, when they view the admin dashboard, then they can see all pickups, filter by user or time, and edit pickup times if needed. |

EPIC 6: Feedback & Reviews

| User Stories | Acceptance Criteria |
|---|---|
| 1. As an Individual Consumer, I want to rate my purchase experience so that I can give feedback. | Given that an Individual Consumer has completed a transaction, when they leave a review with a rating and optional comment, then the review is linked to the specific transaction. |
| 2. As a System Administrator, I want to moderate reviews so that we ensure quality content. | Given that a System Administrator wants to moderate user feedback, when they access the reviews section in the dashboard, then they can remove or flag inappropriate content. |



Secure Web & Mobile Guild
swmguild@gmail.com

EPIC 7: Analytics & Impact Tracking

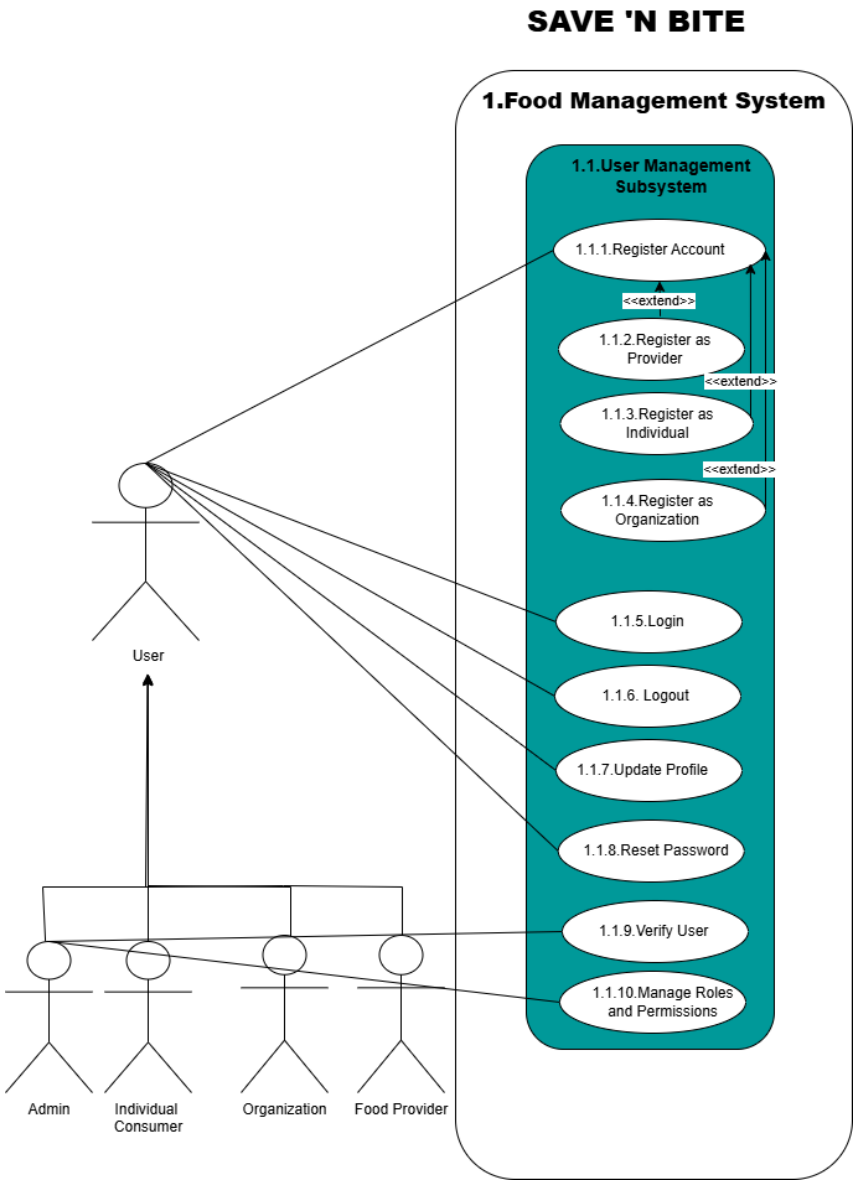
| User Stories | Acceptance Criteria |
|---|---|
| 1. As a Food Provider, I want to view waste reduction metrics so that I can measure our sustainability efforts. | Given that a Food Provider wants to assess their environmental contribution, when they view analytics, then they see metrics such as food saved and $CO_2$ reduction over time (monthly/yearly). |
| 2. As an Organisation, I want to track meals received and distributed so that we can report impact to donors. | Given that an organisation needs to track its operational impact, when they access reports, then they see visual charts of meals received and can export the data for reporting purposes. |
| 1. As a system administrator, I want to be able to monitor all performance metrics and user activities for maintenance and security. | Given that an admin wants to view and monitor the system's performance and actions of other admins, they will be able to view all system and audit logs as well as analytical data and export this data for external viewing. |

EPIC 9: Digital Garden/Badges Reward system (Wow Factor)

| User Stories | Acceptance Criteria |
|---|---|
| 1. As a Food Provider, I want to be able to visualise my progress through badges and advertise this to customers in order to encourage engagement and traffic. | Given that a Food Provider makes regular and successful use of the app, they will be rewarded with shareable badges for completing various milestones which they are able to display on their profile. |
| 3. As an individual customer, I want to have a heightened user experience through the digital garden which represents my use of the app. | Given that an individual customer uses the app regularly, they will be rewarded with plants which they can place and manage in their own garden to visualise their impact and usage of Save n Bite. |

# 3. USE CASE DIAGRAMS

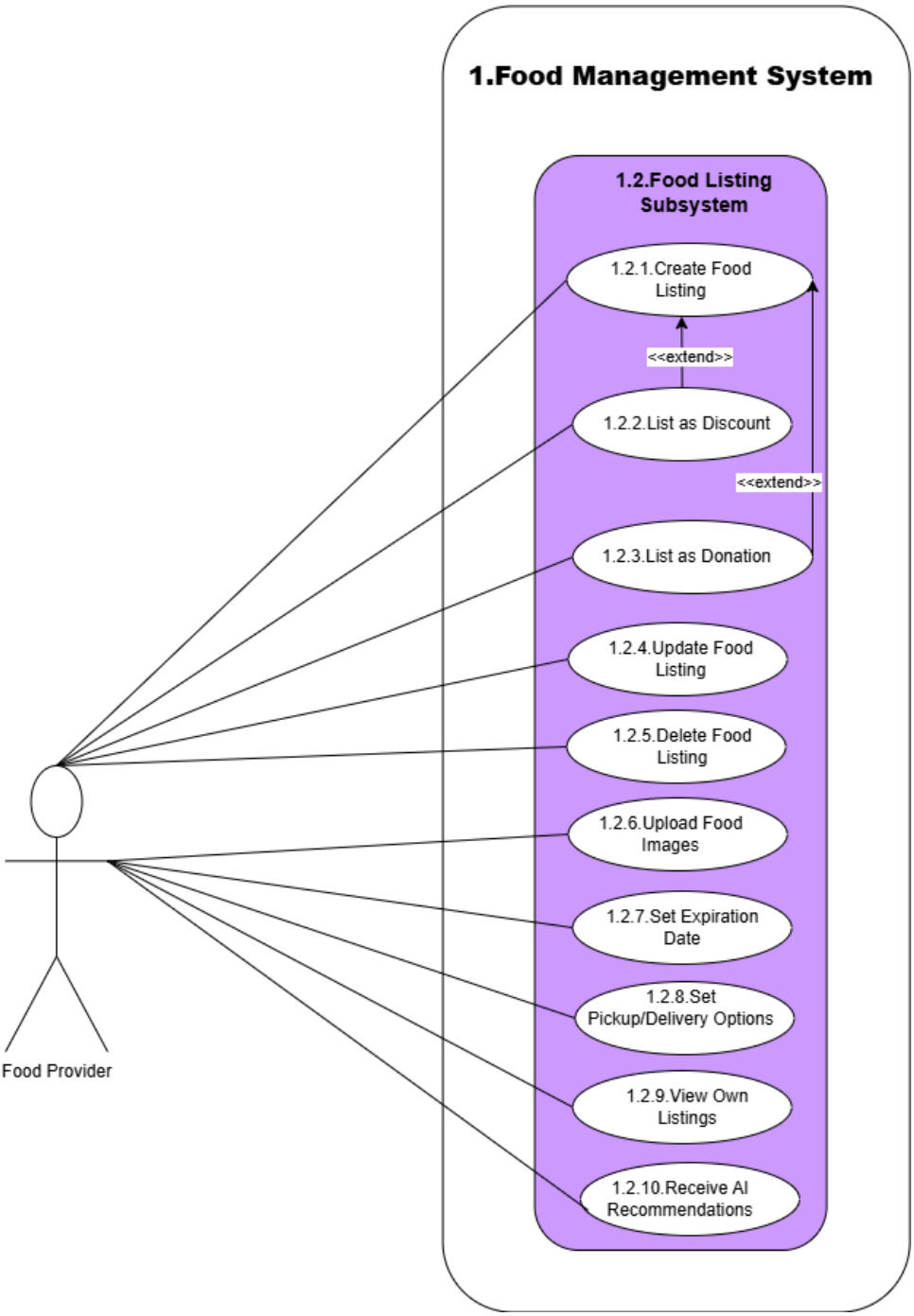
## 3.1 User Management Subsystem






Secure Web & Mobile Guild
swmguild@gmail.com

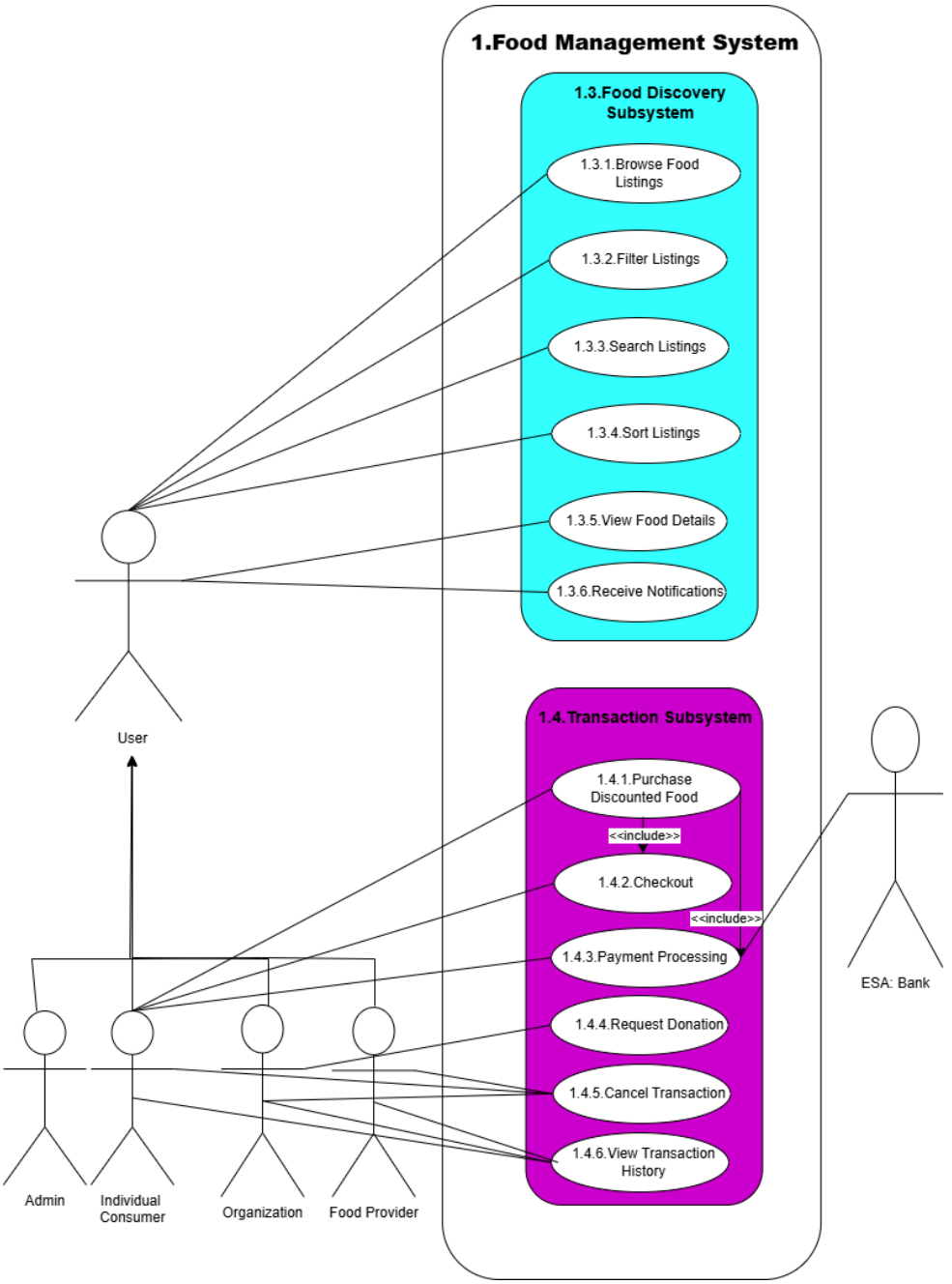
# Food Listings Subsystem






Secure Web & Mobile Guild
swmguild@gmail.com

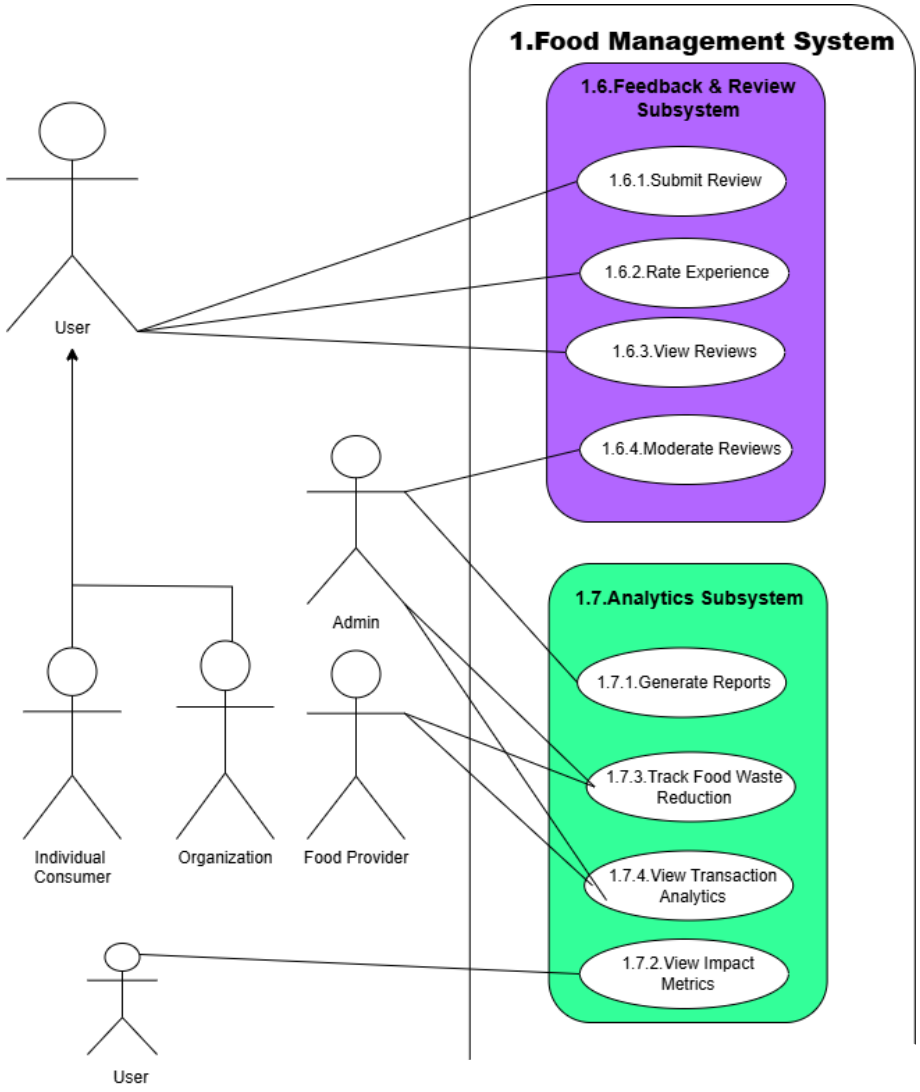
## 3.3 Food Discovery and Transactions



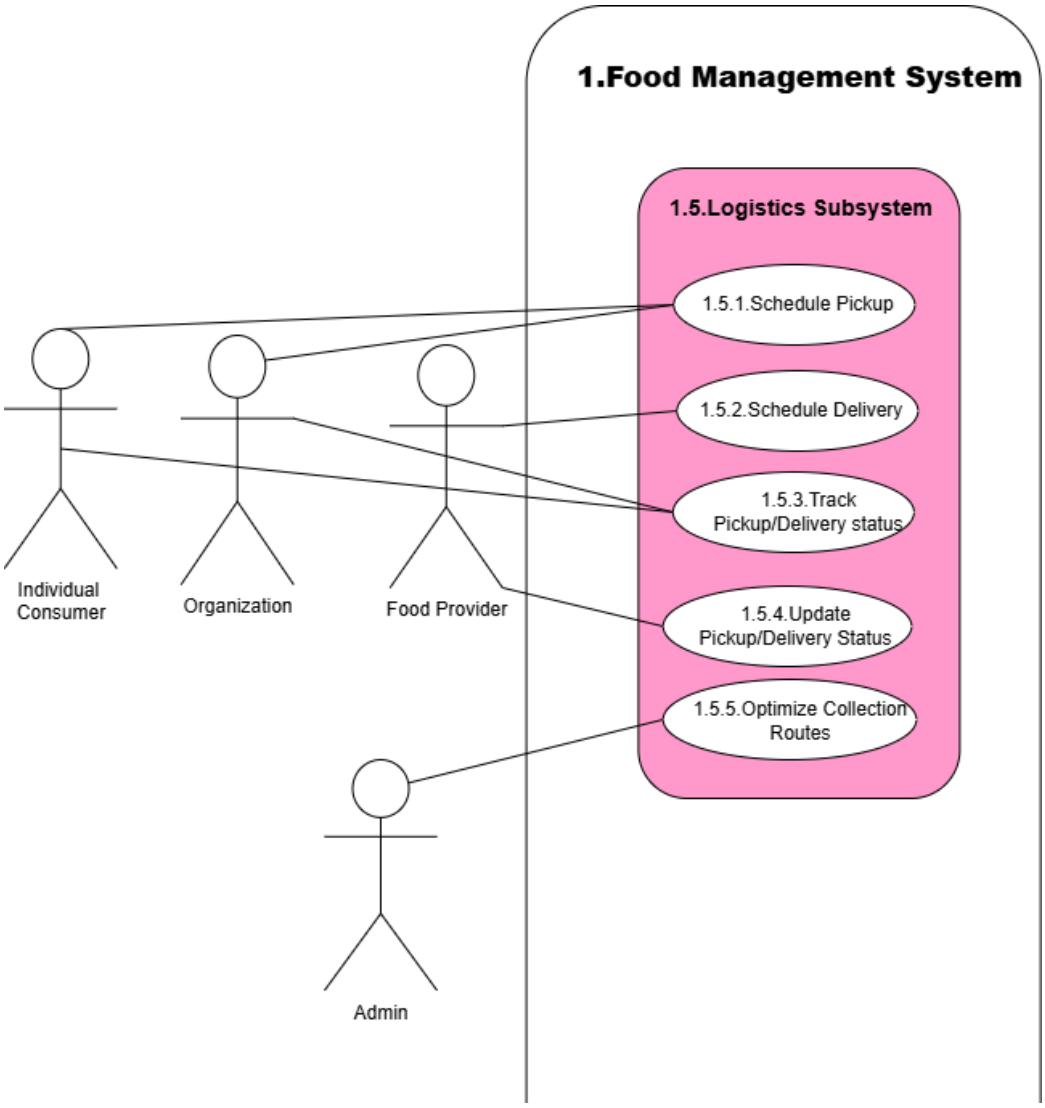



Secure Web & Mobile Guild
swmguild@gmail.com

## 3.4 Feedback/Reviews, Analytics, and AI Subsystem



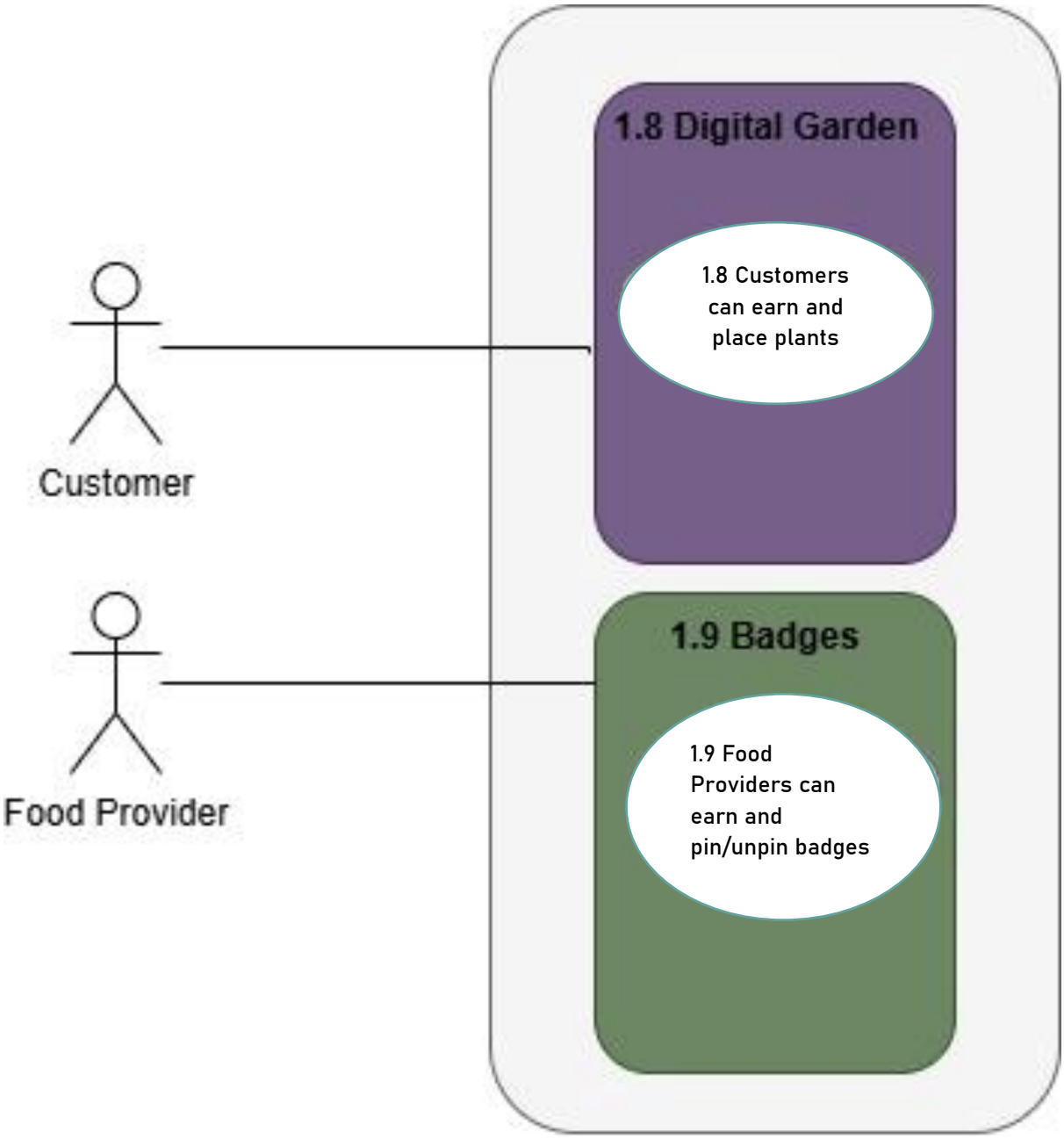
## 3.5 Logistics Subsystem






Secure Web & Mobile Guild
swmguild@gmail.com

## 3.6 Gamification (Wow Factor)






Secure Web & Mobile Guild
swmguild@gmail.com

# 4. FUNCTIONAL REQUIREMENTS

R1: User Management

R1.1 The system must support registration for individuals, organisations, and businesses.
R1.2 Users must be verified during registration.
R1.3 The system must support role-based access (e.g. individuals, organisations, businesses, admin).
R1.4 Secure login/logout functionality must be provided.
R1.5 The system must limit functionality based on user role and verification status.
R1.6 The system should allow the user to update their profile.
R1.7 The system should allow the user to reset their password.
R1.8 The admin should be able to manage roles and permissions.

R2: Make Listings

R2.1 Only verified business users should be able to create new surplus food listings.
R2.2 Each listing must include the following details:
- Name of the item
- Description
- Food Type
- Original Price
- Discounted Price
- Quantity
- Expiry Date
- Pickup Window
- Allergens
- Dietary Information

R2.3 Business users should be able to indicate whether the listing is for sale at a discounted price or available as a donation.
R2.4 Listings should update real-time availability as items are reserved or claimed.
R2.5 Businesses should be able to update food listings.
R2.6 Businesses should be able to view their own listings.

R3: Browse Listings

R3.1 Verified users (individuals and organisations) should be able to view active food listings.
R3.2 Users should be able to search for listings by keyword (e.g., name, category).
R3.3 Users should be able to filter listings by:
- Availability (in stock)
- Type (donation or discounted sale)
- Expiry date
- Business location

R3.4 Listings should display clearly formatted information (e.g., name, expiry, image, availability).
R3.5 Users should be able to sort the listings.
R3.6 Users should be able to receive notifications whenever new listings are added for food products they're interested in.

R4: Purchase / Request Food Items

R4.1 Verified individual users should be able to purchase discounted food items.
R4.2 Verified organisation users should be able to request food items listed as donations.
R4.3 The system should enforce limits to prevent bulk purchasing or hoarding (based on user role and verification status).
R4.4 Upon purchase or request, the listing should be updated to reflect new availability.
R4.5 A confirmation screen should summarize the transaction (e.g., pickup time, location, item details).
R4.6 Users should be able to cancel purchases if they haven't paid for them yet.
R4.7 Users should be able to view their transaction history.

R5: Scheduling and Logistics
R5.1 Businesses must be able to set available pickup times for each listing.
R5.2 Organisations and Customers must be able to coordinate logistics for food pickups.
R5.3 Real-time tracking and status updates must be available for scheduled pickups.
R5.4 The system should allow users to receive notifications about pickup schedules and updates.

R6: Feedback & Review
R6.1 Verified users must be able to rate their food purchase or donation experiences.
R6.2 Businesses must be able to view feedback received.
R6.3 A moderation system must exist to prevent false or abusive reviews.
R6.4 Reviews should be linked to specific listings or transactions.

R7: Analytics
R7.1 Businesses must be able to view analytics on food waste reduction (e.g., items saved).
R7.2 The system should present user-friendly dashboards for performance metrics (e.g., total meals donated).
R7.3 Metrics may include total donations, frequent users, and overall impact.
R7.4 The system administrators must be able to view performance metrics and app engagement.

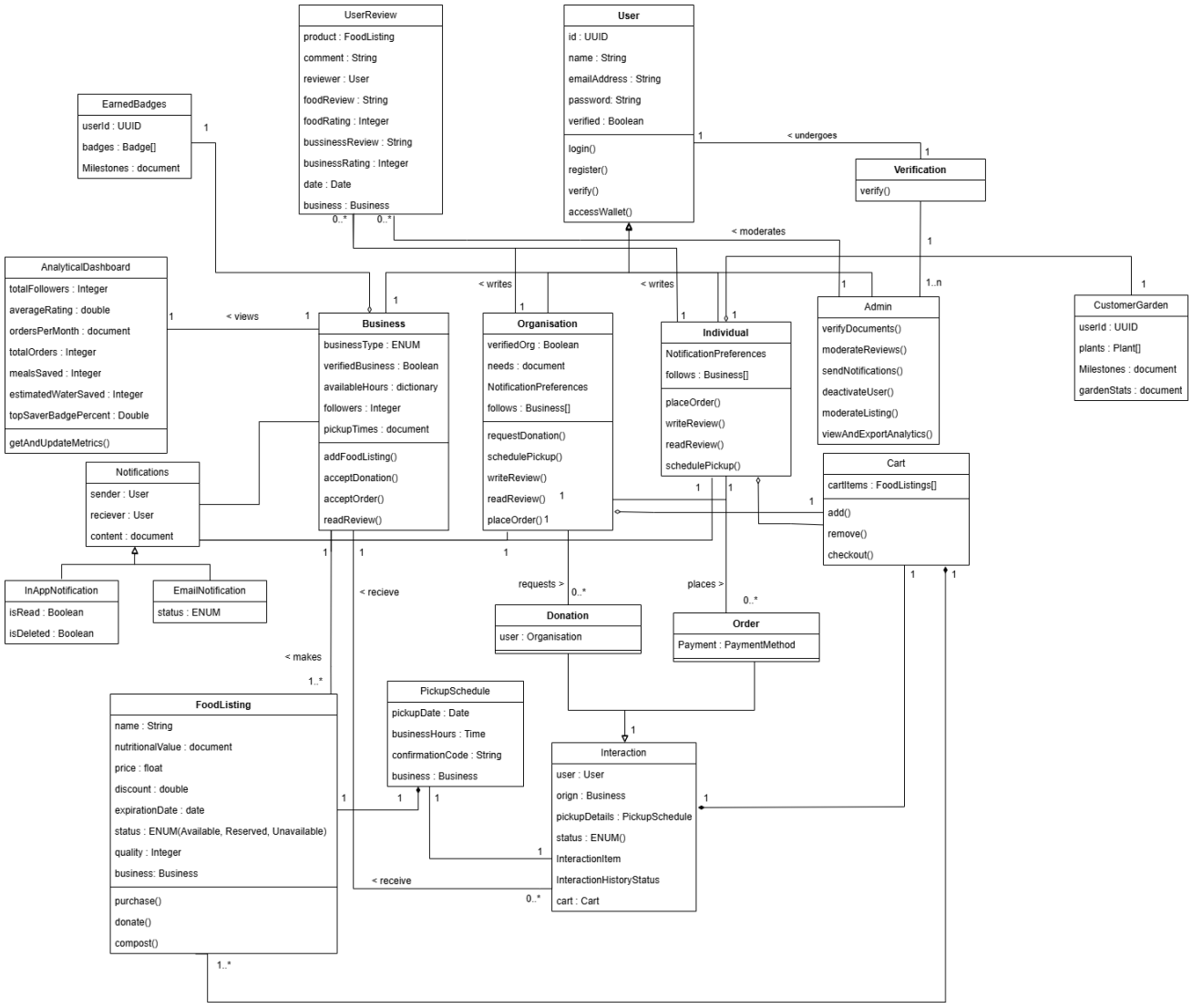
R8: Digital Garden (Wow Factor)
R8.1 Customers must be able to earn, place and manage plants in their digital garden based on their usage of the app.

R9: Badges (Wow Factor)
R9.1 Businesses must be able to earn badges based on their performance and should be able to pin/unpin these badges from their profile.

# 5. SERVICE CONTRACTS

- Authentication Service Contract
- Food Listing Service Contract
- Interactions Service Contract
- Notifications Service Contract
- Scheduling Service Contract
- Reviews and Feedback Service Contract
- Analytics Service Contract
- System Admin Service Contract
- Digital Garden Service Contract
- Badges Service Contract

# 6. DOMAIN MODEL






Secure Web & Mobile Guild
swmguild@gmail.com

# 7. ARCHITECTURAL REQUIREMENTS

## 7.1 Architectural Design Strategy

Our architecture design was based on our quality requirements which are identified in section 7.3. Each aspect of our architecture corresponds to one or more quality requirements. We identified these requirements by determining what the most important necessities of a heavily front-facing, user-centred app would be. These necessities were used to find architectural strategies and patterns that would best suit our application's needs.

Design Process:

1. Identify critical quality requirements from food redistribution business needs
2. Decompose system into presentation, business, and data layers
3. Apply repository pattern for clean data access across domains and security
4. Integrate message broker for responsive, asynchronous operations
5. Validate architecture decisions against quantified quality metrics

This strategy ensures every architectural decision directly supports our five core quality requirements while maintaining clear separation of concerns.

## 7.2 Architectural Strategies

Our system employs a Three-Layer Architecture with Presentation, Business and Data Layers.

### Three-Layered Architecture Explained:

Components:

- Presentation Layer (View and ViewModel components)
- Business Layer (Model and Message Broker)
- Data Layer (Repository Services and Data Stores)

Connectors:

- Data Binding between View and ViewModel
- Call & Receive operations between ViewModel and Business Layer
- API calls between Message Broker and Repository Services
- Direct data access between Repository Services and Data Stores

Constraints:

- Views cannot directly access Business or Data layers
- Repository Services can only be accessed through Message Broker
- Each Repository Service manages only its own domain data
- Data Stores are only accessible through their corresponding Repository Service

## 7.3 Architectural Quality Requirements

Priority 1: Performance - Database Efficiency

Description: Optimized database operations and query performance
Justification: Ensures responsive user experience and system scalability

Architectural Pattern: Microservices
Quantification Metrics:

- API response times under 500ms for 100% of requests
- Database query optimization with proper indexing
- ACID compliance for all transactions
- Efficient connection pooling and resource management

Implementation Evidence:

- Secure database with ACID transaction support
- ORM with optimized queries and select_related/prefetch_related
- Database indexes on frequently queried fields
- Connection pooling for efficient resource utilization

Performance Optimizations:

- Indexed fields: user roles, listing status, creation dates, ratings
- Efficient foreign key relationships with proper joins
- Cached user sessions and authentication tokens
- Optimized serializers for API responses

Testing Strategy:

- API Response Time Testing: Automated tests verify all API endpoints respond within 500ms requirement
- Database Query Optimization Testing: Tests confirm optimized queries using only 5 database calls (69% improvement from initial 16 queries)
- Bulk Operations Performance: Tests measure performance of bulk data operations and large dataset handling under 5 seconds
- Resource Utilization Testing: Memory usage and connection pooling efficiency validated during concurrent operation.

---

Priority 2: Usability - Responsive User Experience

Description: Intuitive, accessible interface across all device types
Justification: Maximizes user adoption and ensures accessibility compliance
Architectural Pattern: MVVM
Quantification Metrics:

- 100% responsive design across device breakpoints
- Page load times under 3 seconds on standard connections
- Maximum 3 clicks to reach any core functionality
- Cross-browser compatibility
- Accessible help menu, support centre and FAQs

Implementation Evidence:

- Progressive loading strategies for optimal performance
- Mobile-first design approach
- Accessible UI components with proper ARIA labels

Device Support:

- Mobile: < 768px (Optimized for smartphone usage)
- Tablet: 768px - 1024px (Enhanced navigation and content)
- Desktop: > 1024px (Full feature access and dashboard views)

Testing Strategy:

- API Response Format Consistency Testing: Automated tests verify consistent pagination and response formats across all endpoints
- Error Message Clarity Testing: Tests validate that error messages are informative and indicate specific field requirements
- Cross-device Testing: Manual testing across mobile (<768px), tablet (768px-1024px), and desktop (>1024px) breakpoints
- User Journey Efficiency Testing: Manual verification that core functionality is reachable within 3 clicks maximum

---

Priority 3: Modularity - Component Independence

Description: Loosely coupled system architecture with clear separation of concerns
Justification: Enables parallel development, easier maintenance, and system scalability
Architectural Pattern: MVVM
Quantification Metrics:

- 7 independent sub-systems with distinct responsibilities
- Minimal circular dependencies between modules
- Each module independently testable and deployable
- Clear API contracts between components
- High code cohesion within modules

Implementation Evidence:

System Architecture:

```
├──save-n-bite-backend
  ├── admin_system/
  ├── authentication/
  ├── food_listings/
  ├── interactions/
  ├── notifications/
  ├── analytics/
  ├── scheduling/
  └── reviews/
```

Module Responsibilities:

- Admin System: User verification and management, review moderation, content management
- Authentication: User registration, login, JWT management, role-based permissions
- Food Listings: Food item management, search, filtering, availability tracking

- Interactions: Shopping cart, purchase flows, order management
- Notifications: User alerts, email notifications, preference management
- Analytics: Business metrics, waste reduction tracking, impact reporting
- Scheduling: Pickup coordination, time slot management, logistics
- Reviews: User feedback, rating system, content moderation

Testing Strategy:

- Django App Independence Testing: Automated tests verify all 10 Django apps (admin_system, analytics, authentication, badges, digital_garden, food_listings, interactions, notifications, scheduling, reviews) are properly installed and have core components
- Circular Dependency Testing: Tests verify models can be imported independently without circular import errors
- Module Integration Testing: Tests verify clean API contracts between components through successful cross-module operations
- Component Isolation Testing: Each Django app tested independently to ensure standalone functionality

---

Priority 4: Responsiveness – Real-time System Reactivity

Description: Immediate responses to user actions and real-time updates.
Justification: Food redistribution is time-sensitive due to expiration constraints and competitive demands.
Architectural Pattern: Message Broker
Quantification Metrics:

- Target 99.5% system uptime
- Automated test coverage >80% across all modules
- Comprehensive error handling with appropriate HTTP status codes
- Zero data loss during normal operations
- Automatic failover for critical system components

Implementation Evidence:

- Message Broker Coordination: Central Message Broker handles asynchronous communication between all Repository Services
- Cache Integration: Strategic cache placement between Business Layer and Message Broker for immediate data access
- Real-time Data Binding: MVVM pattern supports immediate UI updates when data changes

Testing Strategy:

- System Uptime Simulation Testing: Automated tests perform 100 consecutive API calls to verify 99.5% success rate target
- Concurrent User Load Testing: Tests simulate 20+ concurrent users to verify response times remain under 2 seconds
- Error Handling Consistency Testing: Automated tests verify appropriate HTTP status codes and comprehensive error handling
- Transaction Reliability Testing: Tests verify zero data loss through atomic transaction rollback testing

---

Priority 5: Security - Authentication & Authorization

Description: Secure user authentication and role-based access control system
Justification: Critical for food safety compliance, fraud prevention, and user trust
Architectural Pattern: Repository
Quantification Metrics:

- 100% of protected API endpoints require authentication
- 4 distinct user roles with specific permissions (customer, provider, organisation, admin)
- JWT token-based authentication with configurable expiration
- Password validation with security validators
- Zero unauthorized access to protected resources

Implementation Evidence:

- REST Framework with JWT authentication
- Role-based permissions implemented across all 7 Django applications
- Custom user model with user_type field for role differentiation

Testing Strategy:

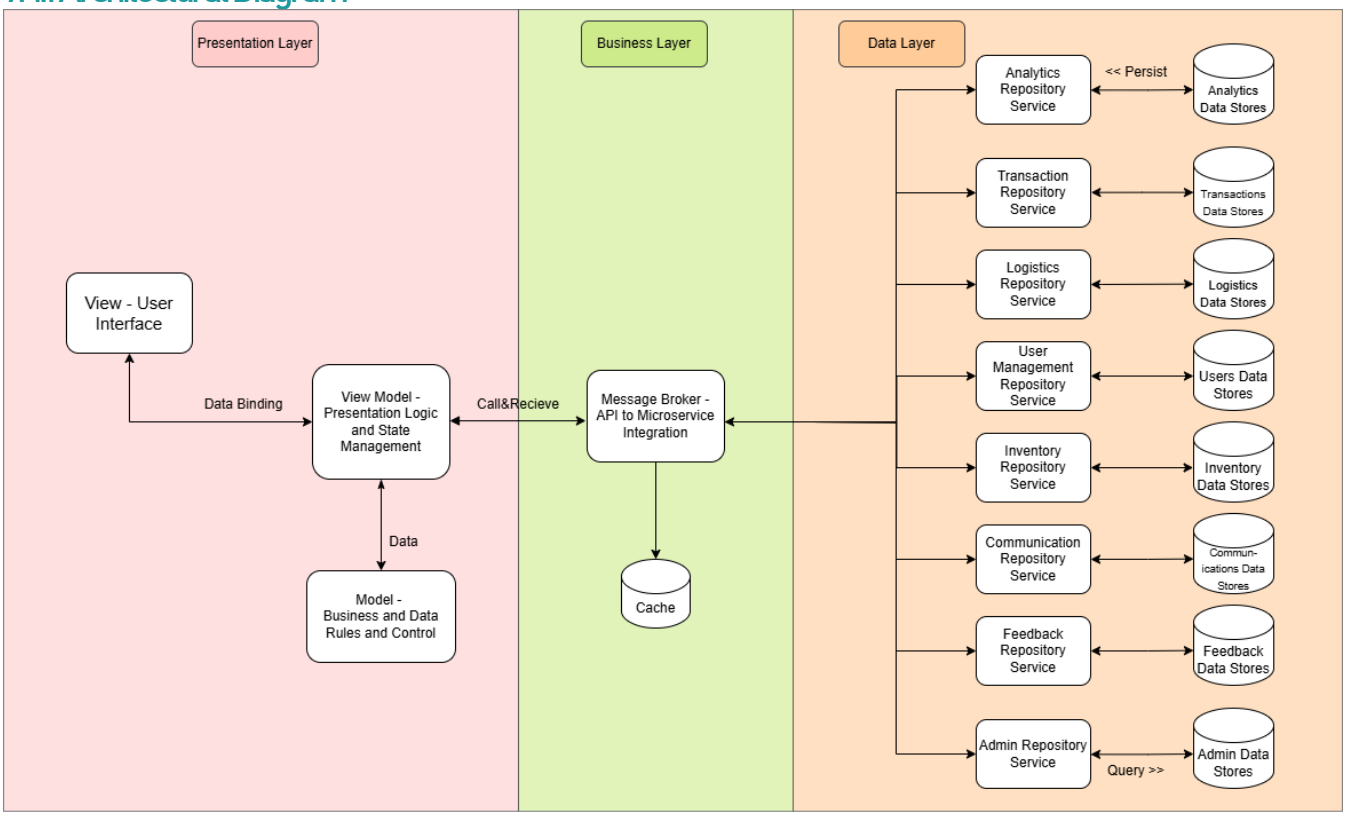
- JWT Authentication Testing: Automated tests verify 100% of protected endpoints require valid JWT tokens and return 401/403 for unauthenticated access
- Role-based Authorization Testing: Tests verify 4 distinct user roles (customer, provider, organisation, admin) have appropriate access permissions and cannot access unauthorized endpoints
- Password Security Testing: Automated tests verify weak passwords are rejected by security validators
- Input Validation Testing: Tests verify protection against malicious inputs
- Security Configuration Testing: Tests verify JWT configuration, security middleware, and password validation are properly implemented

## 7.4 Architectural Design and Pattern

| Pattern | Definition/Description: | Reason for Implementation: | System Constraints: |
|---|---|---|---|
| Microservices | Architecture style implementing eight independent Repository Services (Analytics, Transaction, Logistics, User Management, Inventory, Communication, Feedback, Admin), each managing a specific business domain with its own data store and operating independently through the central Message Broker | Enables independent scaling and deployment of different food redistribution domains. Supports diverse data requirements across user management, food inventory, and transaction processing. Allows specialized optimization for each service's unique | Repository Services must communicate only through Message Broker. Each service maintains data consistency within its domain boundaries. Service interfaces must remain stable for system integration |
| Repository | Data access abstraction layer providing uniform interfaces for all eight domain-specific Repository Services, encapsulating data storage logic and providing consistent CRUD operations across Analytics, | Provides clean separation between business logic in the Business Layer and data persistence in Data Layer. Enables independent testing and optimization of data | Repository Services cannot directly access other repositories' data stores. All data operations must follow uniform interface contracts. Data access |

| | | | |
|---|---|---|---|
| | Transaction, Logistics, User Management, Inventory, Communication, Feedback, and Admin domains | access patterns. Supports multiple data storage strategies per domain without affecting business logic | must maintain transactional consistency within each repository's domain |
| Security | Comprehensive authentication and authorization implementation across all three layers, with role-based access control enforced at the Message Broker level and data protection mechanisms implemented within each Repository Service | Critical for food safety compliance, user verification requirements, and fraud prevention in food redistribution. Ensures only verified providers can list food and authorized users can access sensitive operations and personal data | All communication between layers must be authenticated. Role-based permissions must be enforced at Message Broker before accessing Repository Services. Sensitive data must be encrypted within Data Stores |
| Mode-View-ViewModel | Three-layer separation with View (User Interface), ViewModel (Presentation Logic and State Management), and Model (Business and Data Rules and Control), connected through data binding and call-receive mechanisms as shown in the architecture diagram | Enables independent development of user interface and business logic components. Supports multiple client interfaces sharing common business logic. Facilitates automated testing of presentation logic and responsive user experience development | View layer cannot directly access Model layer or Repository Services. ViewModel must not contain business rules or direct data access logic. All data flow must follow the prescribed binding and call-receive patterns shown in the diagram |

### 7.4.1 Architectural Diagram




## 7.5 Architectural Constrains

Technical Constraints:

- Database: PostgreSQL required for ACID compliance and complex food safety queries
- Authentication: JWT tokens for stateless, scalable session management
- API Standards: RESTful design following OpenAPI specifications
- Browser Support: Modern browsers with JavaScript ES6+ support

Business Constraints:

- Food Safety Compliance: Must track expiration dates, allergen information, and safety metadata
- Privacy Regulations: GDPR/POPIA compliance for user data handling and verification documents
- Verification Requirements: All users must be verified before accessing core platform features
- Academic Timeline: 6-month development window with incremental demo deliveries

Performance Constraints:

- Response Time: Core API operations must complete within 500ms
- Data Retention: Transaction and audit history must be maintained for compliance
- Mobile Compatibility: Must function on devices with limited processing power and bandwidth

Security Constraints:

- Authentication: All protected endpoints require valid JWT tokens
- Authorization: Role-based access strictly enforced

- Data Protection: Sensitive information encrypted at rest and in transit

## 7.6 Technology Choices

Backend Technology Stack

| Component | Technology | Alternatives Considered | Justification |
|---|---|---|---|
| Web Framework | Django + Django REST Framework | Node.js/Express, Spring Boot, FastAPI | Built-in security features, rapid development, excellent ORM for complex relationships |
| Database | PostgreSQL | MySQL, MongoDB | ACID compliance for financial transactions, complex query support, JSON field support |
| Authentication | JWT + Django Simple JWT | Session-based auth, Auth0, Firebase Auth | Stateless authentication, scalability, cross-domain support |
| API Design | RESTful APIs | GraphQL, gRPC | Industry standard, excellent tooling, team familiarity |

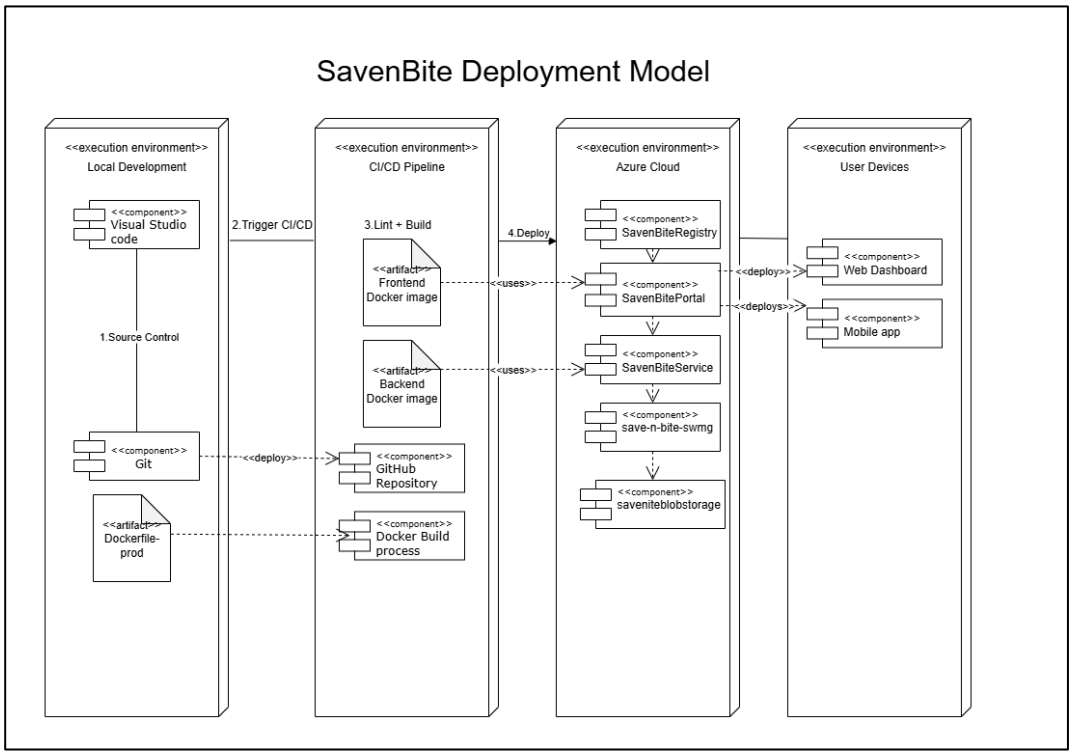
Frontend Technology Stack

| Component | Technology | Alternatives Considered | Justification |
|---|---|---|---|
| Frontend Framework | React | Vue.js, Angular, Svelte | Large ecosystem, component reusability, team expertise |
| State Management | React Hooks + Context | Redux, MobX | Simplified state management, reduced complexity |
| Styling | CSS Modules + Bootstrap | Styled Components, Tailwind | Responsive design, component isolation |
| Build Tool | Create React App | Vite, Webpack | Zero configuration, production-ready setup |

Development & DevOps

| Component | Technology | Alternatives Considered | Justification |
|---|---|---|---|
| Testing | pytest + Jest | unittest, Mocha/Chai | Comprehensive testing features, excellent Django integration |
| Version Control | Git + GitHub | GitLab, Bitbucket | Team familiarity, excellent CI/CD integration |
| Dependency Management | Poetry (Python) + npm | pip + requirements.txt, Yarn | Deterministic builds, lock file support |
| Code Quality | ESLint + Black | Prettier, flake8 | Consistent code formatting, error prevention |

# 8. DEPLOYMENT MODEL




The Save-N-Bite system is deployed in a cloud-based environment using Microsoft Azure. The deployment follows a containerized microservices topology with distinct application and data tiers, ensuring scalability, portability, security, and maintainability.

## Target Environment

The system is hosted entirely on the Azure Cloud, leveraging managed services for application hosting, relational database storage, and file storage. This cloud-native approach removes the overhead of managing on-premises infrastructure, while enabling elasticity, high availability, and global reach.

## Deployment Flow

1. Local Development

Development takes place locally in Visual Studio Code, where the React frontend and Django backend are containerized using Docker.

2. Containerization and Registry

- A deployment.yml file orchestrates the build of Dockerfile-prod for both frontend and backend services.
- Once built, the Docker images are stored in Azure Container Registry (ACR) (savenbiteregistry), which acts as a secure and versioned container repository.

3. Azure Services Deployment

- Frontend Service: The container image for the React frontend is deployed as an Azure App Service (SavenBiteService).
- Backend Service: The Django backend API is deployed as another Azure App Service (SavenBitePortal).
- Database: Application data is stored in Azure Database for PostgreSQL, which ensures reliability and full ACID (Atomicity, Consistency, Isolation, Durability) compliance.

- Blob Storage: Files such as images and media are stored in Azure Blob Storage, providing scalable and cost-efficient object storage.

4. User Access

End-users interact with the system through the frontend service. The frontend communicates with the backend API, which in turn integrates with both the database and blob storage

## Tools and Platforms

- Docker – containerization of frontend and backend services.
- Azure Container Registry (ACR) – secure storage and versioning of container images.
- Azure App Service – managed hosting for frontend and backend services.
- Azure Database for PostgreSQL – fully managed relational database.
- Azure Blob Storage – cost-efficient, scalable object storage.

## Support for Quality Requirements

- Scalability:
  Azure App Services automatically scale horizontally based on demand, while Blob Storage seamlessly scales with file uploads.
- Reliability and Availability:
  Azure provides built-in redundancy and automatic failover for PostgreSQL Database and Blob Storage. Services are distributed across isolated containers, reducing single points of failure.
- Maintainability:
  Docker images and a centralized registry simplify updates and rollbacks. The microservices design ensures that frontend, backend, and storage components can evolve independently.
- Security:
  All communications are encrypted (SSL/TLS), and Azure provides role-based access control (RBAC), managed identities, and secure database connections. Container isolation reduces the attack surface.
- Portability:
  Since all services are containerized, the deployment can be replicated on other cloud platforms (AWS, GCP) or on-premises Kubernetes clusters with minimal configuration changes.
- Performance:
  Azure App Services allocate compute resources dynamically, ensuring responsive application performance during peak loads. Blob Storage is optimized for high-throughput file access.
- Monitoring and Observability:
  Azure Monitor and Application Insights provide real-time logging, performance tracking, and automated alerts, enabling proactive issue detection and resolution.
- Cost Efficiency:
  Azure's pay-as-you-go model ensures efficient resource utilization, scaling services up or down based on workload demands.
- Testability:
  Containerized services allow replication of the production environment in local or staging setups. Azure App Service supports deployment slots for testing new versions before release.