Software Requirements Specification

for

CareConnect

Version 1.0 approved

Prepared by: KANDARPA SAIRAMA SUBRAHMANYA NIKHILESH, DAVIS AARON DANIEL, IGNATIUS TAN ZONG WEI, RUDOLF CHEAM YUE HAN, WONG XIAO YAO, YEO WEN HONG

SC2006- CareConnect

07 Nov 2025

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

This document outlines the software requirements for "CareConnect, Version 1.0". This SRS outlines the functionalities, features, and constraints of the platform, aiming to provide a clear and comprehensive framework for development. The scope of this SRS encompasses the entire digital platform, detailing the subsystems for user management, community assistance coordination, donation management, and volunteer services logistics. This SRS is integral for guiding the development, testing, and maintenance of the platform, ensuring alignment with the project's objectives.

1.2 Document Conventions

This SRS follows standard documentation conventions to ensure clarity and consistency. Key conventions include:

- Font Styles: 'Arial' for body text, 'Times New Roman' for headers.
- Highlighting: Bold for key terms, italics for emphasis.
- Requirement Prioritization: Higher-level requirements have inherited priority levels, cascading down to detailed requirements. Each requirement statement is explicitly labeled with a priority level (High, Medium, Low).
- Requirement Identification: Each requirement is uniquely identified for easy reference.
- Versioning: Changes in requirements through different versions are tracked for historical reference and future revisions.

1.3 Intended Audience and Reading Suggestions

This document is structured to cater to a wide array of stakeholders and should be read differently depending on the reader's role:

- Project Managers and Stakeholders: Begin with the Introduction to understand the purpose and scope, then proceed to the Overall Description for a high-level overview of the system.
- Developers: After the introductory sections, delve into the System Features for detailed descriptions of the platform's functionalities, and the External Interface Requirements for integration specifics.
- Testers: Focus on the System Features for test case development, and refer to the Nonfunctional Requirements for performance and security testing guidelines.
- User Experience Designers: The User Interface sections within the External Interface Requirements will be of particular interest, providing details on the interaction between the system and its users.
- Technical Writers: The entire document is relevant, with emphasis on User Documentation and Assumptions and Dependencies to ensure accurate and comprehensive user guides and help documentation.
- Quality Assurance: Refer to the System Features for functionality, Other Nonfunctional Requirements for performance benchmarks, and the Other Requirements for additional specifications.

All readers are encouraged to review the document in its entirety, as each section builds on the information presented in the preceding ones, providing a complete understanding of CareConnect's requirements.

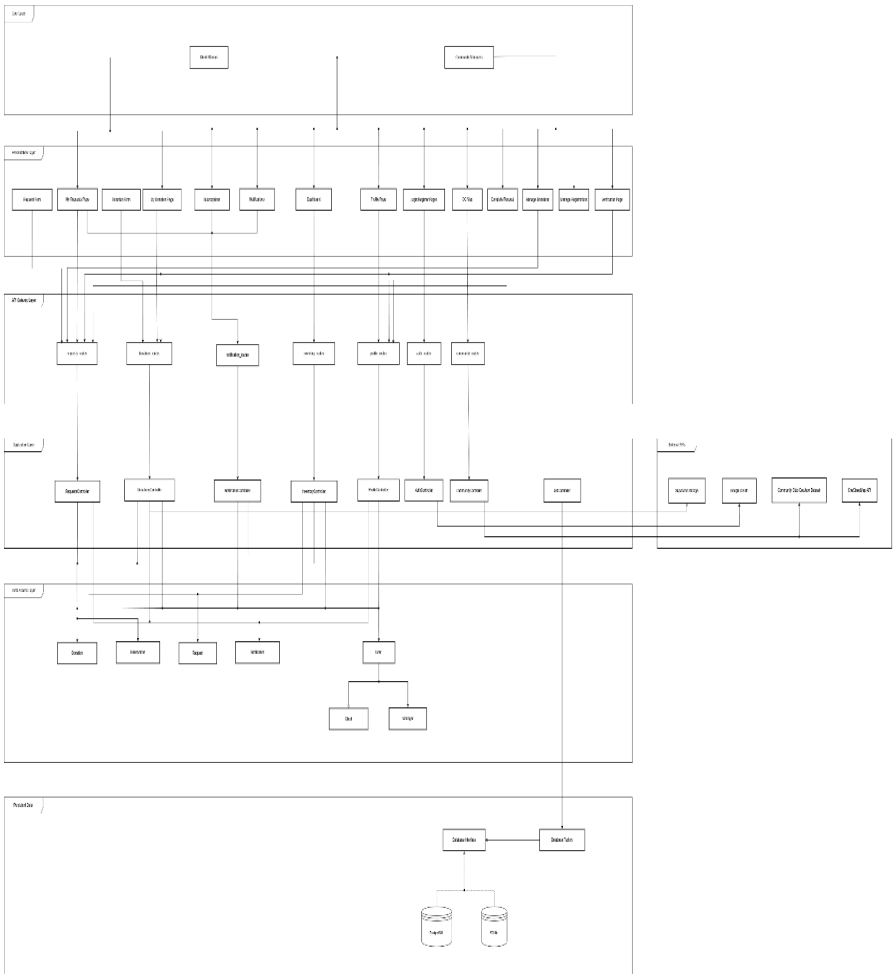
1.4 Product Scope

CareConnect is a digital solution designed to bridge the gap between community resources and individuals in need of assistance. Its primary goal is to connect community members who can offer help with those requiring support, thereby strengthening social bonds and enhancing community welfare. The platform will enable beneficiaries to post goods requests, donors to donate goods, facilitate community club coordination, and provide inventory management for resource distribution. This aligns with broader corporate objectives of social responsibility and sustainable community development through technology-enabled mutual aid networks.

2. Overall Description

2.1 Product Perspective

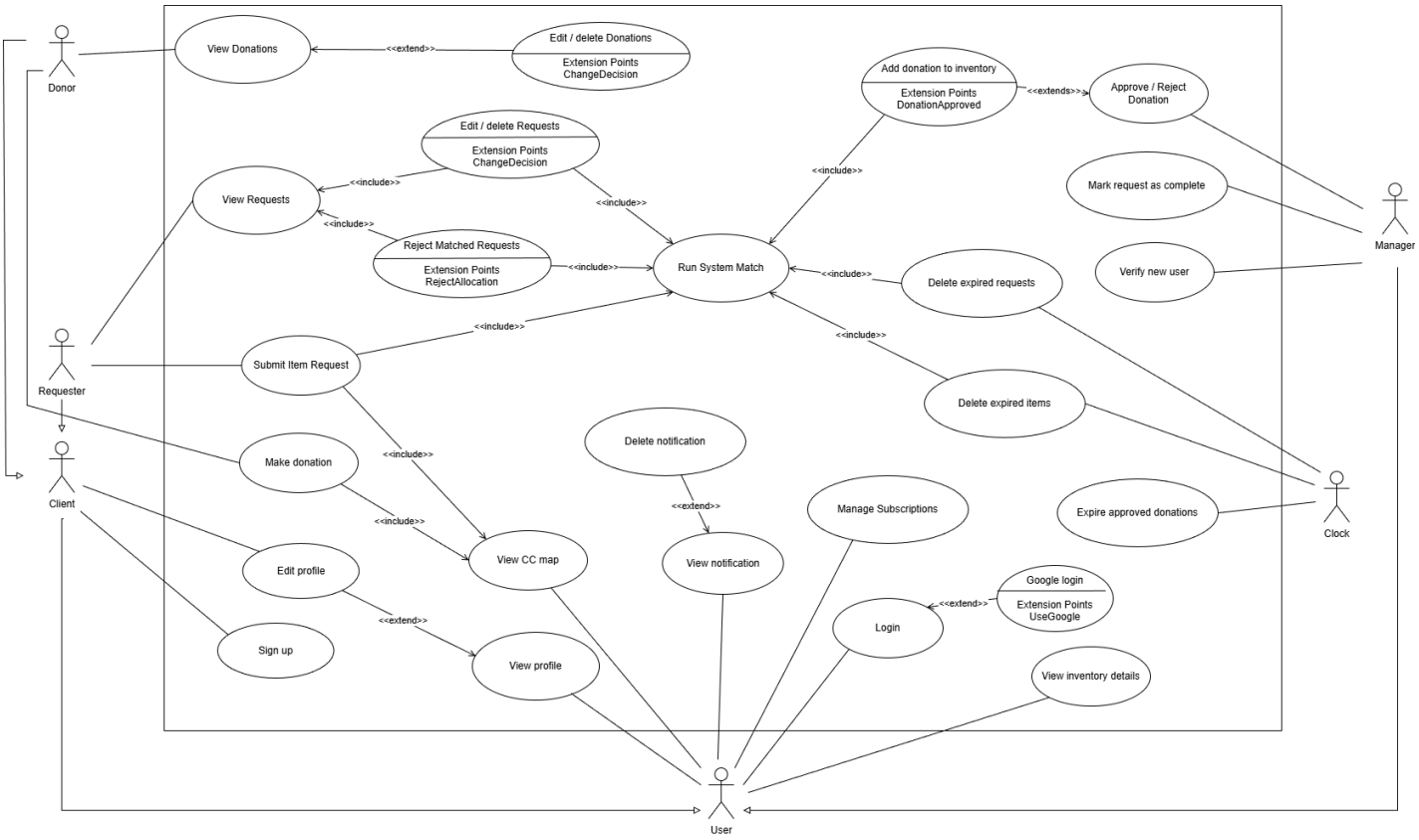
This SRS specifies the requirements for CareConnect, a new, self-contained product designed to create an ecosystem for community assistance and resource sharing. It is not a follow-on member of a product family nor a replacement for existing systems but is a pioneering effort to strengthen community bonds and facilitate mutual aid through technology. The platform functions independently and also integrates with existing technologies via APIs. A system architecture diagram (as shown below or refer to https://github.com/softwarelab3/2006-SCSG-37/blob/main/Lab 5/System Architecture Diagram.pdf if unclear) is provided to illustrate the interconnections and external interfaces.



2.2 Product Functions

2.2.1 Use Case Diagram

The following use case diagram (refer to https://github.com/softwarelab3/2006-SCSG-37/blob/main/Lab 5/Use Case Diagram %26 Description.pdf if unclear) depicts the key product functionalities that CareConnect includes. There are 4 primary users of CareConnect – Admin, Manager, Client, and Volunteer. Each type of user interacts with CareConnect to use their role-specific functions.



2.2.2 Major Product Functions

CareConnect will allow users to perform the following functions:

1. Authentication
    a. Login
    b. Login with Google
    c. Signup (with different roles – beneficiary and donor)

2. Manager Functions
    a. User Management (Verify Users)
    b. Complete Goods Requests
    c. Manage Donations Applications
    d. Complete Donations Applications
    e. View Community Clubs Map
    f. View Community Clubs donation statistics

3. Client Functions
    a. Create Donation application
    b. View Available Donations
    c. Edit Donation Applications
    d. View Community Clubs Map
    e. Subscription to CCs alert list

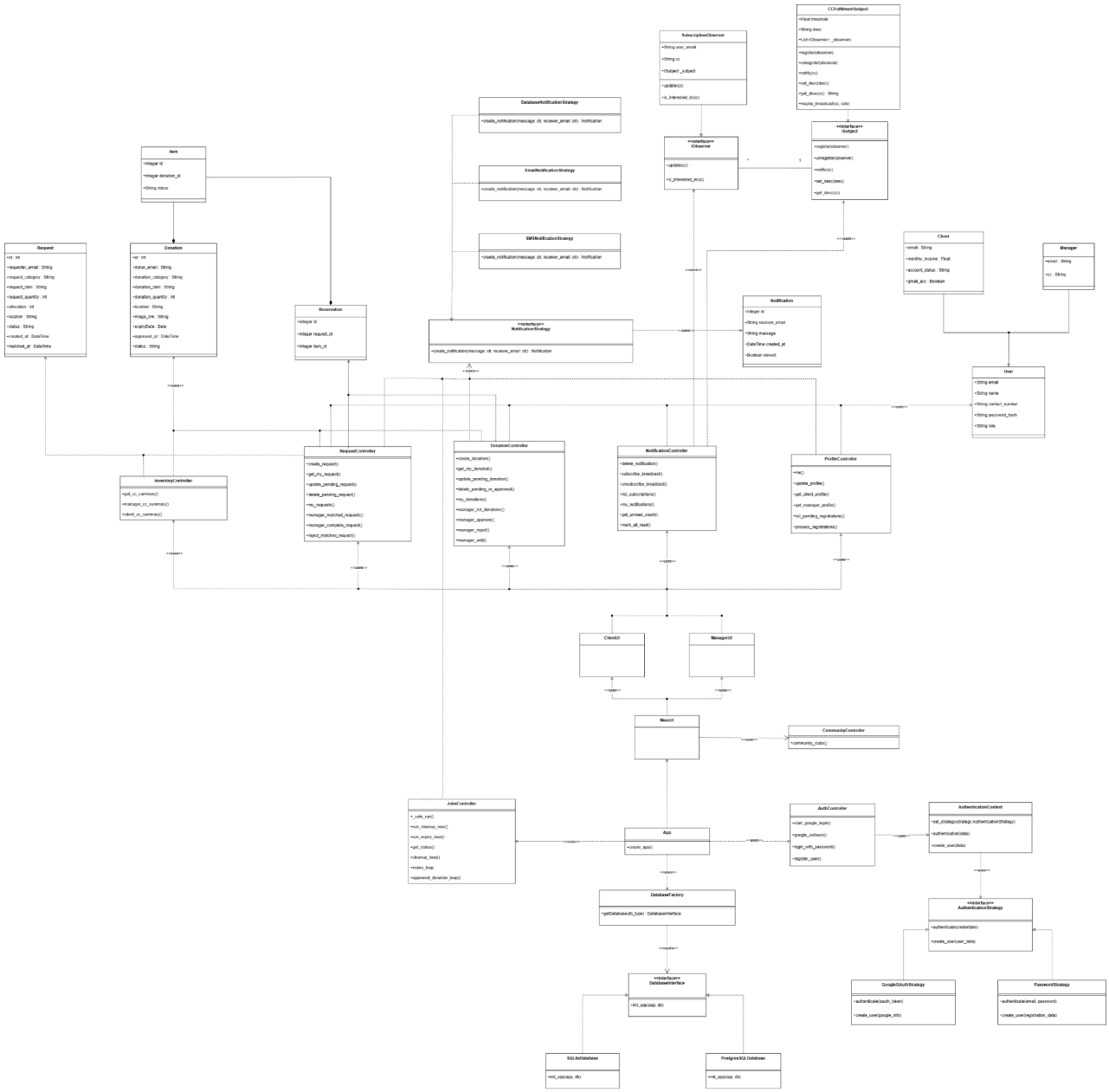
4. Volunteer Functions
    a. Create goods Requests
    b. View Request Status
    c. Edit Request Status

5. Notifications
    a. System-wide alerts and user-specific notifications
    b. Real-time updates on requests and donations

6. Other Functions
    a. Edit Profile
    b. View Activity History

2.2.3 Class Diagram

The following Class Diagram (Refer to https://github.com/softwarelab3/2006-SCSG-37/blob/main/Lab%205/Class%20Diagram.pdf if unclear) shows how different classes interact with each other, as well as the key functionalities that CareConnect needs to perform (refer to the controllers in the middle row section).



2.3 User Classes and Characteristics

Users of the platform are categorized into:

- Managers: Community organization leaders who coordinate resources and manage individual community centres. They require comprehensive management tools and dashboard functionality.
- Clients: General users of the app who will be able to donate goods to community centres for redistributions will also be alerted to shortages to try to get targeted donations.
- Beneficiaries: Clients who are of low monthly income. They may request for goods as well as do anything a client can.

All three types of users are equally important for CareConnect as they form an ecosystem that makes CareConnect functional and operational. E.g.: Beneficiaries submit goods requests, Clients donates goods to the community centres, Managers coordinate resources and verifies clients and ensures that CareConnect operates fairly and safely for all users.

2.4 Operating Environment

The platform will be web-based, optimized for desktop use, operating on the latest versions of Chrome, Firefox, Safari, and Edge. It will be locally hosted and must integrate with external APIs (Google OAuth, OneStreetMap API, CommunityClub GeoJSON API, Supabase API) and database storage (PostgreSQL).

2.5 Design and Implementation Constraints

The development of CareConnect will be subject to several constraints that will guide the design and implementation of the software:

2.5.1 Frontend Constraints

2.5.1.1 Framework (Frontend)

The user interface will be developed using React.js with Vite as the build tool and CSS for styling. (React was chosen for its component-based architecture which enhances code reusability and maintainability.)

2.5.1.2 Code Formatting

Code formatting and consistency will be enforced using standard JavaScript/React best practices so as to adhere to the best coding practices.

2.5.2 Backend Constraints

2.5.2.1 Framework (Backend)

The backend server will be developed using Flask in Python with a modular architecture including controllers, services, and routes.

2.5.2.2 Database

The database will be in SQL as our data is structured and relational. The system supports both PostgreSQL for production and SQLite for development environments. SQL will be the language for database interactions, with an emphasis on robustness and the ability to handle complex queries efficiently.

2.5.2.3 Code Formatting

Code formatting and consistency will be enforced using Python PEP 8 standards. Standard Python formatting practices will be used to maintain uniform code formatting on the backend, streamlining the development process and facilitating code reviews.

2.5.3 Language Constraints

- All software documentation, interfaces, and user interactions will be provided in English.
- The choice of English is due to its prevalence as a primary or secondary language among the anticipated user demographics and to maintain consistency across the platform.

2.6 User Documentation

CareConnect will include a comprehensive set of user documentation to ensure proper understanding and ease of use for developers. The documentation will consist of the following components:

2.6.1 README Files

2.6.1.1 Main Repository

A README file for the main repository will provide an overview of the project, including the architecture, how to set up the development environment, deployment instructions, and a high-level guide to the repository's contents.

2.6.2 Code Comments

- The source code for both frontend and backend will be well-commented, providing clarity on complex logic, functions, components, classes, and methods. Comments will also include tagging for TODOs and FIXMEs where necessary.
- Inline documentation will be present to explain the purpose and usage of various code blocks, modules, and libraries.

2.7 Assumptions and Dependencies

It is assumed that the platform will have access to the internet at all times for proper functionality of external integrations such as Google OAuth authentication, map services, and cloud-based database operations.

2.7.1 Dependencies

2.7.1.1 Front-end Libraries

The user interface of FeedItForward depends on numerous front-end libraries:

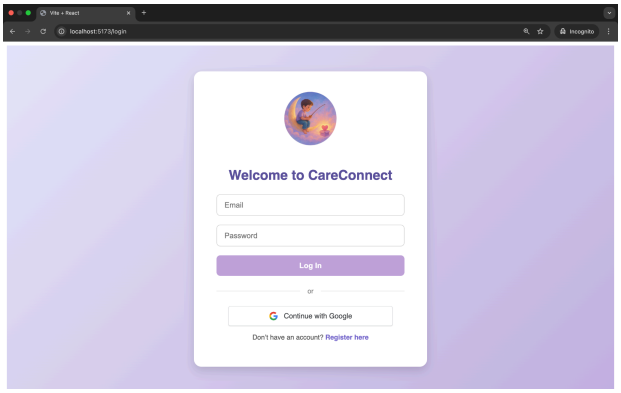
Library	Purpose
React	Core UI library for building user interfaces
React DOM	Renders React components to the DOM
React Router DOM	Client-side routing and navigation
Axios	HTTP client for API communication
Leaflet	Interactive maps for community clubs display
React Icons	Icon components for UI elements
Vite	Build tool and development server
@vitejs/plugin-react	Vite plugin for React support
Flask	Core web framework for API development

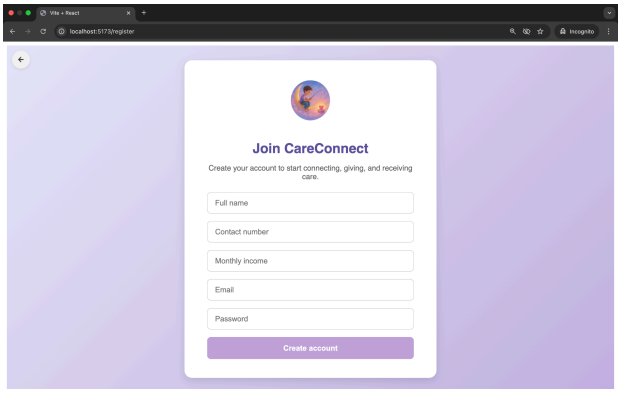
Flask-SQLAlchemy	Database ORM for data modeling
Flask-Session	User session management
Flask-CORS	Cross-origin resource sharing for frontend communication
Authlib	OAuth implementation for Google authentication
Passlib[bcrypt]	Password hashing and verification
Argon2-cffi	Advanced password hashing algorithm
psycopg2-binary	PostgreSQL database adapter
Redis	In-memory storage for session data
Supabase	Cloud storage for file uploads
Requests	HTTP requests for external API calls
Python-dotenv	Environment variable configuration

3. External Interface Requirements

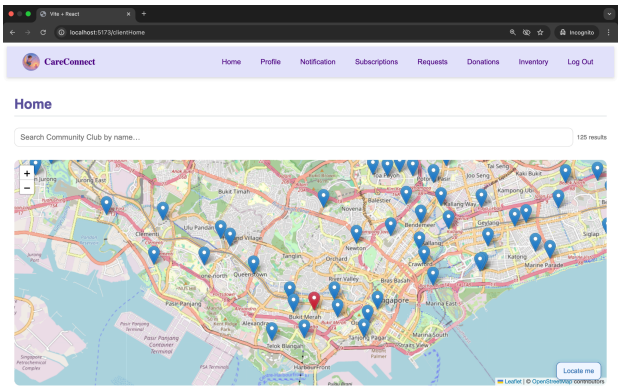
3.1 User Interfaces

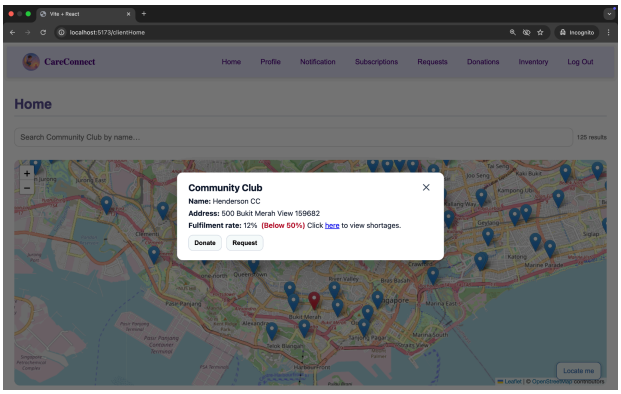
3.1.1 Login + Register Form for New Users



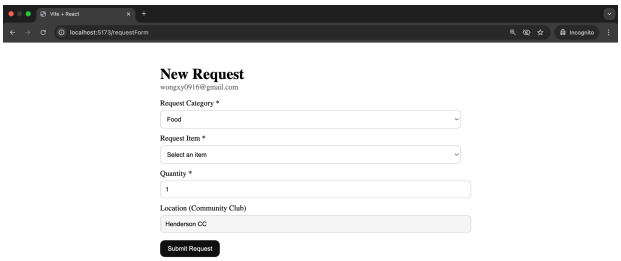


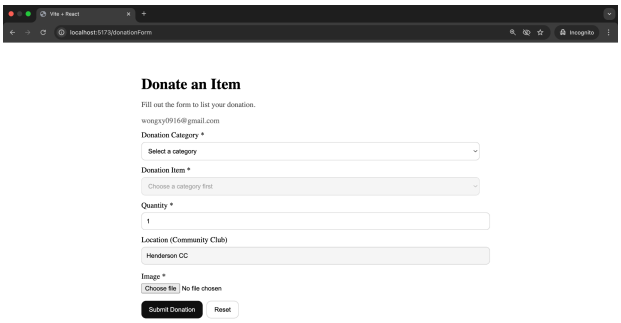
3.1.2 User Home Page + Make Request/Donation pop up



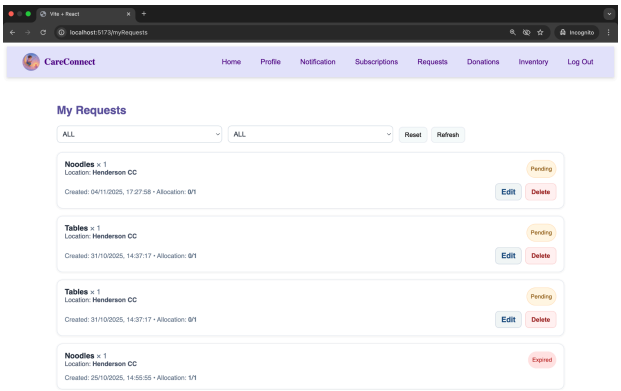


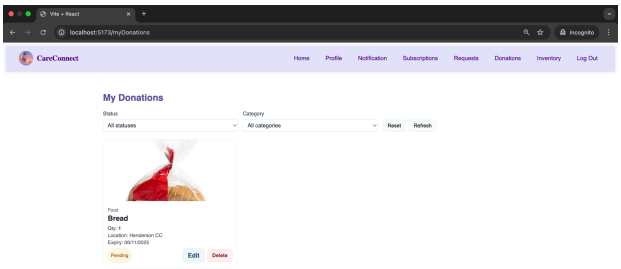
3.1.3 Request Form + Donation Form



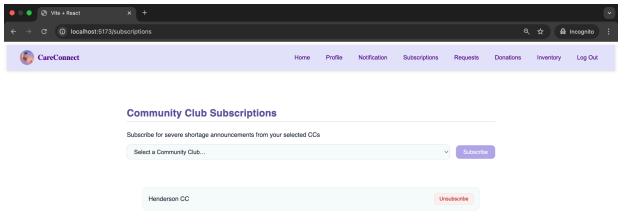


3.1.4 My Requests History + My Donations History

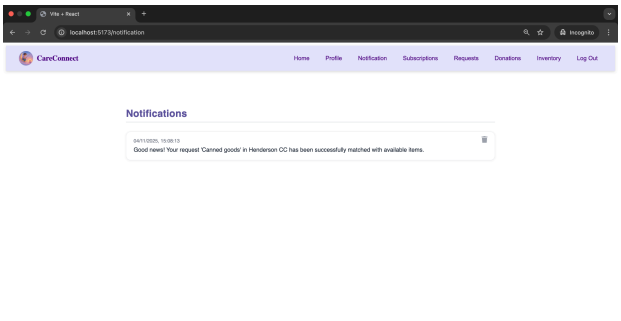


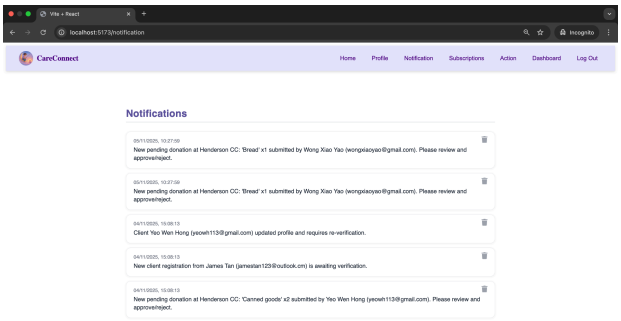


3.1.5 Subscription For CC Broadcast Notifications

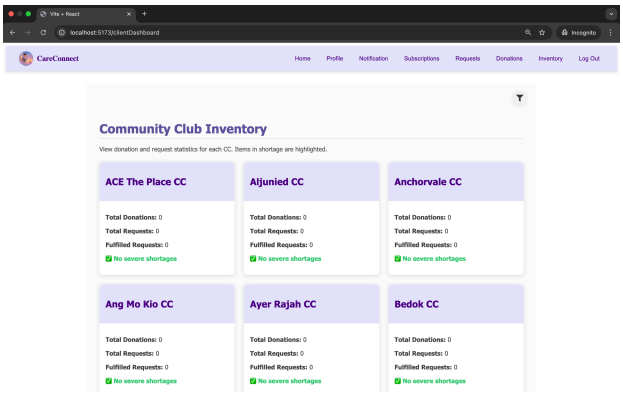


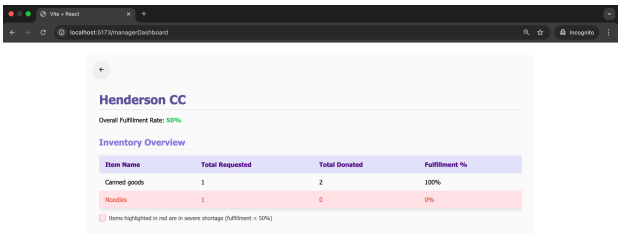
3.1.5 Notification Page for Client + Notification Page for Manager



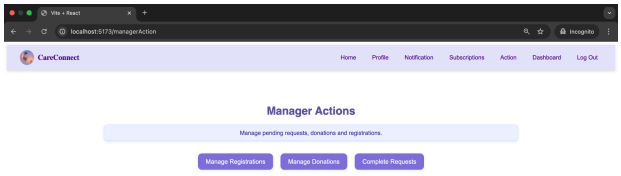


3.1.6 Inventory Page for Client + Inventory Page for Manager

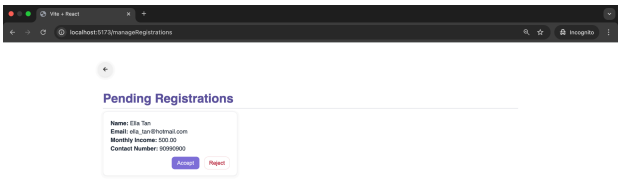


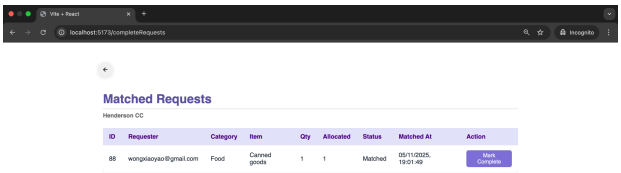


3.1.7 Action Page for Manager

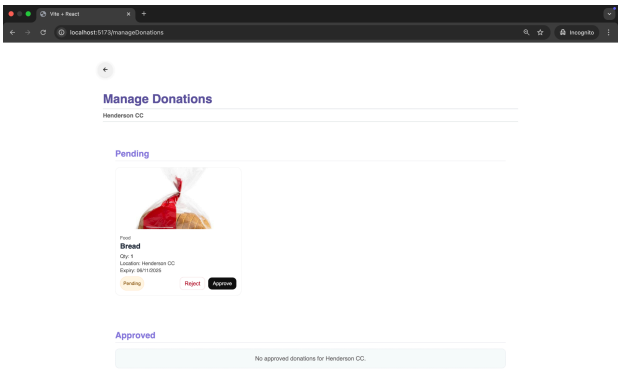


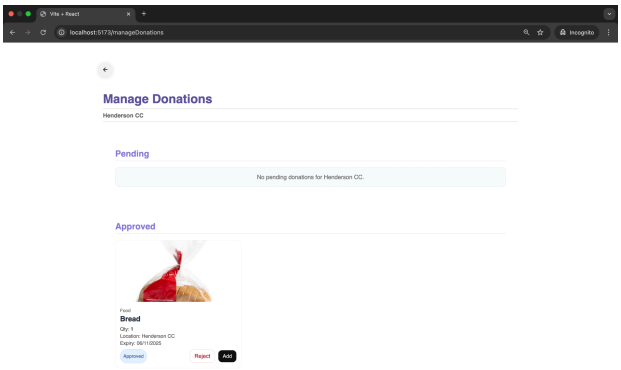
3.1.8 Manage Registrations + Complete Requests





3.1.9 Manage Donations (Approve + Add)





3.2 Hardware Interfaces

CareConnect functions mainly as a web-based platform, requiring minimal direct interaction with physical hardware. The following describes the logical and physical aspects of its hardware interfaces:

3.2.1 Supported Devices

The software is designed to be compatible with various devices, including desktop computers, laptops, tablets, and smartphones.

3.2.2 Data and Control Interactions

Users interact with the system using standard input/output hardware — such as keyboards, touchscreens, and mice for input — and displays for visual output. Internet access is facilitated through network interface cards.

3.2.3 Communication Protocols

The APIs and related services will communicate over standard web ports (80 for HTTP).

3.3 Software Interfaces

3.3.1 Backend Dependencies (Python 3.8+)

- Flask 2.3.3 - Web application framework
- SQLAlchemy 2.0.21 - Database ORM
- Flask-SQLAlchemy 3.0.5 - Flask-SQLAlchemy integration
- psycopg2-binary 2.9.7 - PostgreSQL adapter
- Flask-Session 0.5.0 - Server-side session management
- redis 4.6.0 - Session storage backend
- authlib 1.2.1 - OAuth 2.0 implementation
- argon2-cffi 23.1.0 - Password hashing
- flask-cors 4.0.0 - Cross-origin resource sharing

- Protocol: HTTP
- Authentication: Required (session token)
- Push Delivery: Observer pattern implementation

3.3.2 Frontend Dependencies (Node.js 16+)

- React 18.2.0 - UI framework
- React Router DOM 6.15.0 - Client-side routing
- Axios 1.5.0 - HTTP client
- Vite 4.4.5 - Build tool and development server

3.3.3 Database System

- PostgreSQL 13+ (Production) - Primary relational database
- SQLite 3.36+ (Development) - Local development database
- Redis 6.2+ - Session storage and caching

3.3.4 Operating System Support

- Linux (Ubuntu 20.04+, CentOS 8+)
- macOS (10.15+)
- Windows (10/11 with WSL2 recommended)

3.3.5 Data Flow and Message Interfaces

3.3.5.1 Incoming Data Items

- Authentication Data
- Donation Creation
- Request Creation

3.3.5.1 Outgoing Data Items

- User Session Data
- Notification Messages
- Analytics Data

3.3.6 Service Communication

3.3.6.1 HTTP REST API Services

- Authentication Services
    - Endpoint: /auth/*
    - Protocol: HTTP
    - Authentication: Session-based with Redis storage
    - Rate Limiting: 100 requests/minute per IP
- Donation Management Service
    - Endpoint: /donations/*
    - Protocol: HTTP
    - Authentication: Required (session token)
    - File Upload: Multipart form data for images
- Request Management Service
    - Endpoint: /requests/*
    - Protocol: HTTP
    - Authentication: Required (session token)
    - Real-time Updates: Server-sent events for status changes
- Notification Service
    - Endpoint: /notifications/*

3.3.6.2 External Service Integrations

- Google OAuth 2.0 Service
    - Authorization URL: https://accounts.google.com/o/oauth2/auth
    - Token URL: https://oauth2.googleapis.com/token
    - User Info URL: https://www.googleapis.com/oauth2/v2/userinfo
    - Scopes: openid email profile
- Database Connection Services
    - PostgreSQL: postgresql://user:pass@host:5432/careconnect
    - Redis: redis://localhost:6379/0
    - Connection Pooling: SQLAlchemy engine with pool_size=1

3.3.7 Shared Data Components

3.3.7.1 Session Data Store (Redis)

CareConnect uses Redis as a session data store to manage temporary and shared data efficiently across multiple system components and user sessions.

3.3.7.2 Database Schema Sharing

- Cross-table relationships: Foreign key constraints
- Shared enums: Status types, categories, urgency levels
- Audit fields: created_at, updated_at across all entities

3.3.7.3 Application State Management

- Frontend: React Context API for user authentication state
- Backend: Flask session management with Redis persistence
- Database: Transaction isolation level READ_COMMITTEDs

3.3.8 API Protocol Documentation

3.3.8.1 RESTful Endpoint Patterns

```
GET    /api/{resource}          # List resources
POST   /api/{resource}          # Create resource
GET    /api/{resource}/{id}     # Get specific resource
PUT    /api/{resource}/{id}     # Update resource
DELETE /api/{resource}/{id}     # Delete resource
```

3.3.8.2 Response Format Standard

```
{
  "success": boolean,
  "data": object|array,
  "message": string,
  "errors": array
}
```

3.3.8.3 Error Response Format

```
{
  "success": false,
  "error": {
    "code": "ERROR_CODE",
    "message": "Human readable message",
    "details": object
  }
}
```

3.4 Communications Interfaces

3.4.1 Web Browser Communication

3.4.1.1 HTTP Protocol Requirements

- Primary Protocol: HTTP for all communications
- Port Configuration:
    - Backend: 5000 (development), 80 (production)
    - Frontend: 5173 (development), 80 (production)

3.4.1.2 Browser Compatibility

- Minimum Requirements: Chrome 90+, Firefox 88+, Safari 14+, Edge 90+
- JavaScript: ES2020 support required

3.4.2 Network Server Communication

3.4.2.1 RESTful API Communication

Content-Type: application/json

Accept: application/json

Authorization: Session-based (cookies)

Access-Control-Allow-Credentials: true

3.4.2.2 Session Management Protocol

Set-Cookie: session=<session_id>; HttpOnly; SameSite=Lax
X-CSRF-Token: <csrf_token>

3.4.3 OAuth 2.0 Communications

3.4.3.1 Google OAuth Flow

- Authorization Request
- Token Exchange

3.4.4 Message Formatting Standards

3.4.4.1 API Request Format

```
{
  "method": "POST|GET|PUT|DELETE",
  "headers": {
    "Content-Type": "application/json",
    "X-Requested-With": "XMLHttpRequest"
  },
  "body": {
    "data": "object",
    "timestamp": "ISO8601",
    "request_id": "uuid"
  }
}
```

3.4.4.2 API Response Format

```
{
  "success": true|false,
  "data": "object|array|null",
  "message": "string",
  "timestamp": "ISO8601",
```

```
  "request_id": "uuid",
  "errors": [
    {
      "field": "string",
      "code": "ERROR_CODE",
      "message": "string"
    }
  ]
}
```


3.4.4.3 File Upload Format


```
POST /api/donations
Content-Type: multipart/form-data

--boundary
Content-Disposition: form-data; name="donation_data"
Content-Type: application/json

{
  "item_name": "string",
  "category": "string",
  "quantity": 1
}

--boundary
Content-Disposition: form-data; name="image"; filename="image.jpg"
Content-Type: image/jpeg

<binary_image_data>
--boundary--
```

3.4.5 Communication Standards

3.4.5.1 HTTP Methods Usage

- GET: Retrieve resources (idempotent)
- POST: Create new resources
- PUT: Update existing resources (idempotent)
- DELETE: Remove resources (idempotent)
- OPTIONS: CORS preflight requests

3.4.5.2 Status Code Standards

200 OK # Successful GET, PUT
201 Created # Successful POST
204 No Content # Successful DELETE
400 Bad Request # Invalid request data
401 Unauthorized # Authentication required
403 Forbidden # Insufficient permissions
404 Not Found # Resource not found
409 Conflict # Resource conflict
422 Unprocessable # Validation errors
500 Internal Error # Server error

3.4.5.3 CORS Configuration

Access-Control-Allow-Origin: http://localhost:5173
Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS
Access-Control-Allow-Headers: Content-Type, Authorization, X-Requested-With
Access-Control-Allow-Credentials: true
Access-Control-Max-Age: 86400

3.4.6 Security and Encryption

3.4.6.1 Transport Layer Security

- Protocol: HTTP (no encryption at transport layer)

- Security Note: Data transmitted in plain text over HTTP

3.4.6.2 Authentication Security

```
# Session configuration
SESSION_COOKIE_SECURE = False     # HTTP allowed
SESSION_COOKIE_HTTPONLY = True    # No JavaScript access
SESSION_COOKIE_SAMESITE = 'Lax'   # CSRF protection
SESSION_PERMANENT = False         # Session expires on browser close
```

3.4.6.3 Password Security

```
# Argon2 Configuration
argon2 = Argon2(
    time_cost=2,        # 2 iterations
    memory_cost=65536,  # 64 MB memory
    parallelism=1,      # Single thread
    hash_len=32,        # 32 byte hash
    salt_len=16         # 16 byte salt
)
```

3.4.7 Data Transfer Specifications

3.4.7.1 Transfer Rate Requirements

- API Response Time: < 200ms for 95th percentile
- File Upload: Maximum 5MB per request
- Concurrent Connections: Support 100 simultaneous users
- Bandwidth: Minimum 1 Mbps per user

3.4.7.2 Connection Pooling

```
# Database Connection Pool
engine = create_engine(
    DATABASE_URL,
    pool_size=10,           # Base connections
    max_overflow=20,        # Additional connections
    pool_timeout=30,        # Connection timeout
    pool_recycle=3600,      # Recycle after 1 hour
    pool_pre_ping=True      # Validate connections
)
```

3.4.7.3 Redis Connection Configuration

```
redis_client = redis.Redis(
    host='localhost',
    port=6379,
    db=0,
    decode_responses=True,
    socket_connect_timeout=5,
    socket_timeout=5,
    retry_on_timeout=True,
    health_check_interval=30
)
```

3.4.8 Electrical Forms Communication

3.4.8.1 Form Validation Protocol

```
   const locationToUse = editMode ? form.location : ccFromState;
    if (!locationToUse) return setError("Missing location (CC). Please select a Community
Club first.");


    if (!form.request_category) return setError("Select a request category.");
```

```
    if (!form.request_item) return setError("Select a request item.");
    if (!form.request_quantity || form.request_quantity < 1)
      return setError("Quantity must be at least 1.");
```

3.4.8.2 Form Submission Format

```
POST /api/donations HTTP/1.1
Content-Type: application/json
X-Requested-With: XMLHttpRequest
Cookie: session=<session_id>

{
  "item_name": "Canned Food",
  "category": "Food",
  "quantity": 10,
  "expiry_date": "2024-12-31",
  "community_club": "Toa Payoh Community Club"
}
```

4. System Features

4.1 Map & CC Anchor Points

4.1.1 Description and Priority

The system provides an interactive map UI that displays all Community Clubs (CCs) as anchor points. This serves as the central navigation and logistics hub for all donation activities. Priority: High.

4.1.2 Stimulus/Response Sequences

The user opens the app and the system displays a map with CC markers.
The user selects a CC and the system shows a pop up with the name, address and fulfillment rate of the selected CC.

4.1.3 Functional Requirements

REQ-1: The system shall fetch CC locations from the CC dataset API.
REQ-2: The system shall display CCs on an interactive map.
REQ-3: The system shall allow users to select a CC and view associated data.
- REQ-3.1: All users should be able to see the cc's name and address.
- REQ-3.2: All users should be able to see the request fulfillment rate.
- REQ-3.3: Only clients with monthly income < 800 and account_status = "Confirmed" shall be able to see the option to request for goods.
- REQ-3.4: All clients with account_status = "Confirmed" shall be able to see the option to donate goods.

REQ-4: The system shall indicate CCs with severe shortages on the map using a red mark.
REQ-5: Severe shortage is defined as fulfillment rate < 50% based on current inventory vs pending requests.
REQ-6: The system shall provide search functionality to filter community clubs by name.
REQ-7: The system shall allow users to locate themselves on the map using GPS.

4.2 Request Posting (Requesters)

4.2.1 Description and Priority

Requesters can submit requests for items (food, household goods). Requests include details such as request category, request item, quantity and location. Priority: High.

4.2.2 Stimulus/Response Sequences

Requester opens the app and selects nearest CC.
Requester fills in the request form with item details.
System checks CC inventory for the requested item:
If stock is available, the system creates a match and notifies the requester to either go down to the CC to collect the goods or the requester rejects the offer in-app.
If stock is unavailable, the request is marked Pending, awaiting donor contributions.
The system publishes the request to the CC's request list.
The system archives fulfilled requests for reporting.
If inventory exists, the system creates a match.

4.2.3 Functional Requirements

REQ-1: The system shall only allow clients with monthly income < 800 to submit item requests.
REQ-2: The system shall store for each request:

- Name of good
- Category of good
- Quantity Requested
- Requester ID
- Status (Pending | Matched | Expired | Completed)

REQ-3: The system shall store and display all requests in the CC's request list for cc managers.
REQ-4: The system shall store and display all of the user's requests in their own request list.

REQ-4.1: Users can filter the request based on its status
REQ-4.2: Users can filter the request based on its category

REQ-5: The system shall match available items automatically when a request is submitted.
REQ-6: The system shall, when stock is available, create a match and notify the requester to come down to the CC to collect the goods or Reject the offer in-app.
REQ-7: The system shall mark requests as pending if the requested item is unavailable.
REQ-8: The system shall create Reservation records linking requests to items and update Item status to "Unavailable".
REQ-9: Upon full allocation, the system shall mark the request matched and alert the requester with collection instructions.
REQ-10: Upon Reject, the system will delete the request, and release the reserved items. Donated Item status will change from unavailable to available.
REQ-11: The system shall allow requesters to edit a request while its Status $\in$ {Pending}.
REQ-12: Editable fields: Category of good, Name of good, Quantity Requested. Requester ID and CC name are immutable.

REQ-13: If a request in Pending is edited, re-run the matching logic.
REQ-14: The system shall update the timestamps of requests that they edited, such that it is inserted at the back of the queue for the matching algorithm to work again.
REQ-15: The system shall allow requesters to delete a request any time before matched; if a partial match exists, release the reservation and delete the request.
REQ-16: The system shall display the current allocated quantity for each request.
REQ-17: Once a request is fulfilled and items are collected, the system shall mark the request as Completed and decrement the inventory.
REQ-18: The system shall archive fulfilled requests for reporting.

4.3 Donor Pre-Approval (Manual Intake)

4.3.1 Description and Priority

Donors must submit a donation application for review before bringing goods to the CC. CC managers verify condition/expiry from images, then approve or reject the donation before inviting the donor down. Priority: High.

4.3.2 Stimulus/Response Sequences

Donor selects CC
Donor fills donation form (category, good name, quantity, images incl, expiry if applicable)
System records application as Submitted
CC manager reviews (Approve / Reject)
If Approved, the donor is invited to hand over goods
Manager confirms outcome
If the physical item is verified, goods are added to inventory.

4.3.3 Functional Requirements

REQ-1: The system shall require donors to submit a donation application before drop-off.
REQ-2: The system shall store for each donation: donor_email, donation_category, donation_item, donation_quantity, location, image_link, expiryDate, status, approved_at.
REQ-3: The system shall validate that an image is provided.
REQ-4: The system shall create the donation with status = "Pending".
REQ-5: The system shall allow CC managers to approve or reject donation applications based on submitted details and photo.
REQ-6: The system shall notify donors to come down to the CC only after their application is approved.
REQ-7: The system shall allow CC managers to confirm physical handover of goods when donors arrive (Approved -> Added).
REQ-8: The system shall allow CC managers to mark the outcome of the handover as added or rejected.

4.4 CC Manager

4.4.1 Description and Priority

CC managers can review donation and new user applications, approve or reject them, and view donation and request records. Priority: High.

4.4.2 Stimulus/Response Sequences

CC manager logs into the system and toggles to the action page.
The system displays pending user and donation applications awaiting approval.
The manager approves or rejects users' applications.
Manager approves or rejects donation applications to their cc.
The system automatically generates shortage alerts by comparing requests and inventory.
Manager views donation and request counts and request fulfilment rates of the CC.

4.4.3 Functional Requirements

REQ-1: The system shall provide CC managers with authentication of new users and dashboard access.

- REQ-1.1 Dashboard access allows CC mangers to look at:
- REQ-1.1.1: Specific items and its amount requested and donated
- REQ-1.1.2: Specific items fulfillment rate
- REQ-1.1.3: CC overall fulfillment rate

REQ-2: The system shall display pending user applications requiring approval.
REQ-3: The system shall display pending donation applications requiring approval, including uploaded item images where applicable.
REQ-4: The system shall allow managers to approve or reject donation applications.
REQ-5: The system shall notify donors once their donation application is approved and they are invited to come down.
REQ-6: The system shall allow managers add the items to the cc inventory or reject it
REQ-7: The system shall automatically generate severe shortage alerts by comparing request lists with current inventory.
REQ-8: The system shall allow managers to mark matched requests as "Completed".

4.5 Matching & Notifications

4.5.1 Description and Priority

The system ensures requesters are matched to donated resources at their CCs. Users receive timely notifications of approvals, rejections, and request fulfilments. CC managers receive notifications for donation approval tasks and new user sign-ups. Priority: Medium-High.

4.5.2 Stimulus/Response Sequences

Requester posts a request.
Donor submits donation and system notifies CC manager for approval.
CC manager approves donation and system notifies donors to handover goods physically
CC manager confirms collected items and updates inventory.
System creates a provisional match when matching stock becomes available and notifies the requester to Accept/Reject in-app.
System sends a collection alert to the requester if matched.
If requester rejects the match, the request is deleted
If timed out (goods are not collected within 2 days), the request expires.

4.5.3 Functional Requirements

REQ-1: matching logic will be run in the following situations:

1. When manager adds a donation to the inventory
2. When requesters submit a new request
3. When requesters rejects a matched request
4. When items in the inventory are expired
5. When a matched request expires
6. When requesters delete a pending request

REQ-2: The system shall notify donors when their donation applications are approved or rejected.
REQ-3: The system shall notify requesters when their requests are fulfilled, either immediately or after new donations are added to inventory.
REQ-4: The system shall send an alert to requesters for collection appointments.
REQ-5: The system shall update request and donation statuses in real time.
REQ-6: The system shall notify CC managers of new donation applications requiring approval.
REQ-7: The system shall notify CC managers of new user sign-ups requiring approval.
REQ-8: The system shall notify donors when to come down for handover of goods after their donation application is approved.
REQ-9: The system shall notify requesters of matches and prompt collection alert and Reject in-app.
REQ-10: The system shall notify requesters if a request expires (matched goods are not collected within 2 days)
REQ-11: Clients will be able to subscribe to a CC to receive notifications when there is a shortage at the CC

4.6 Authentication

4.6.1 Description and Priority

The authentication subsystem enables users to securely access the CareConnect platform using either their registered email and password or their Google account. It ensures identity verification, controlled access based on user roles (Client or Manager), and secure session management throughout the platform. Priority: High.

4.6.2 Stimulus/Response Sequences

1. The user navigates to the login page.
2. The user enters their email and password, or selects “Continue with Google.”
3. The system validates the provided credentials or initiates the Google login process.
    a. If authentication succeeds, the system creates a user session.
        i. The system redirects the user to their appropriate home page based on their role.
    b. If authentication fails, the system displays an error message describing the issue.
4. For new users, the user accesses the registration page, submits their information, and receives confirmation of successful registration.
5. The system validates registration inputs, checks for duplicate accounts, and creates a new user and associated profile upon success.
6. Managers are notified of new client registrations pending verification.
7. Users may log out at any time, upon which their session is cleared and they are returned to the login screen.

4.6.3 Functional Requirements

REQ-1: The system shall allow users to log in using their registered email and password.

> REQ-1.1: The system shall validate the email format and password complexity before processing the login request.
>
> REQ-1.2: The system shall verify the provided credentials against stored user data.
>
> REQ-1.3: The system shall authenticate users with valid credentials and initiate a secure session.
>
> REQ-1.4: The system shall display an error message for invalid or missing credentials.

REQ-2: The system shall allow users to register for an account using their name, contact number, monthly income, email, and password.

> REQ-2.1: The system shall validate that all required fields are provided and properly formatted.

REQ-2.2: The system shall reject registration attempts that use duplicate email addresses.

REQ-2.3: The system shall create a new user and a linked client profile upon successful registration.

REQ-2.4: The system shall send a confirmation message to the newly registered user.

REQ-2.5: The system shall notify managers of new client registrations awaiting approval.

REQ-3: The system shall allow users to log in using their Google accounts.

REQ-3.1: The system shall redirect users to a Google sign-in page for authentication.

REQ-3.2: The system shall verify the user's Google account information and obtain a verified email address.

REQ-3.3: The system shall authenticate existing users or automatically create a new account using verified Google information.

REQ-3.4: The system shall display a clear error message if Google login fails.

REQ-4: The system shall direct authenticated users to their respective dashboards based on role (Client or Manager).

REQ-5: The system shall allow users to log out from any authenticated page.

REQ-5.1: Upon logout, the system shall terminate the active session and clear all session data.

REQ-5.2: The system shall confirm successful logout and return the user to the login screen.

REQ-6: The system shall ensure consistent session handling and data protection for all authentication methods.

REQ-7: The system shall protect user credentials and personal information during transmission and storage.

REQ-8: The system shall provide clear and consistent error messages for all authentication and registration failures.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

1. The system shall display the community clubs map and listings within ≤5 seconds under normal load.
2. The system shall store new requests and donations within ≤5 seconds.
3. The system shall deliver notifications to users within ≤10 seconds of an event.
4. Users will be able to login using Google OAuth within ≤3 seconds.
5. Users will be able to locate themselves on the map using GPS integration.
6. The system shall handle file uploads (donation images) within ≤10 seconds.
7. The system shall process job assignments and volunteer matching within ≤5 seconds.

5.2 Safety Requirements

1. The system shall validate all user inputs (e.g., prevent negative quantities, invalid dates).

5.3 Security Requirements

1. The system shall require all users to register with a verified email and password.
2. Passwords shall be stored securely using a hashing algorithm (Argon2 Library).
3. The system shall implement role-based access control (RBAC) for Clients, Managers, Volunteers, and Admins.
4. The system shall implement session management using Redis for secure user sessions.

5.4 Software Quality Attributes

1. Usability: The UI shall allow a user to complete a basic task (make a request or donation) within 3 taps/clicks.

2. Reliability: The system shall maintain at least 95% uptime during normal operating hours (excluding planned maintenance).
3. Maintainability: The system shall use a modular code structure so that adding or modifying small features does not affect core functions.
4. Scalability: The system shall support growth from 100 to 500 registered users without major redesign.
5. Availability: The system shall be available for at least 24 hours per day.
6. Portability: The system shall be compatible with major web browsers (Chrome, Firefox, Safari, Edge).

6. Other Requirements

6.1 Internationalisation Requirements

1. Initially, CareConnect will support only the English language. Future releases will consider multi-language support to cater to a broader audience in Singapore, especially senior citizens who may prefer local dialects.
2. Date and time representations should be in the Singapore timezone (GMT+8).

6.2 Legal Requirements

1. Compliance with PDPA when handling user's sensitive data.
2. Adherence to local regulations for community assistance and volunteer coordination activities.

6.3 Reuse Objectives

1. Components developed for this platform should be modular and reusable within the project and for potential future projects.
2. APIs should be designed with industry standards (REST API) to facilitate external usage of our platform through APIs.

6.4 Accessibility Requirements

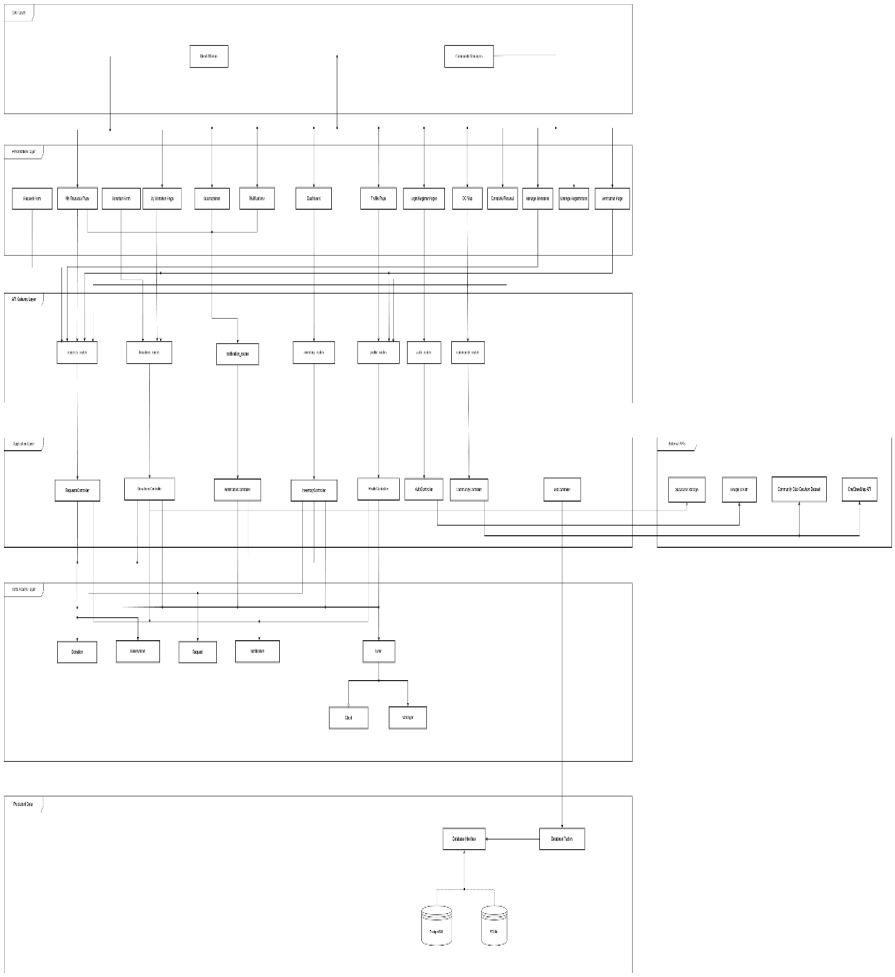
1. Must meet WCAG 2.1 Level AA accessibility standards to ensure it is usable by people with disabilities.

Appendix A: Glossary

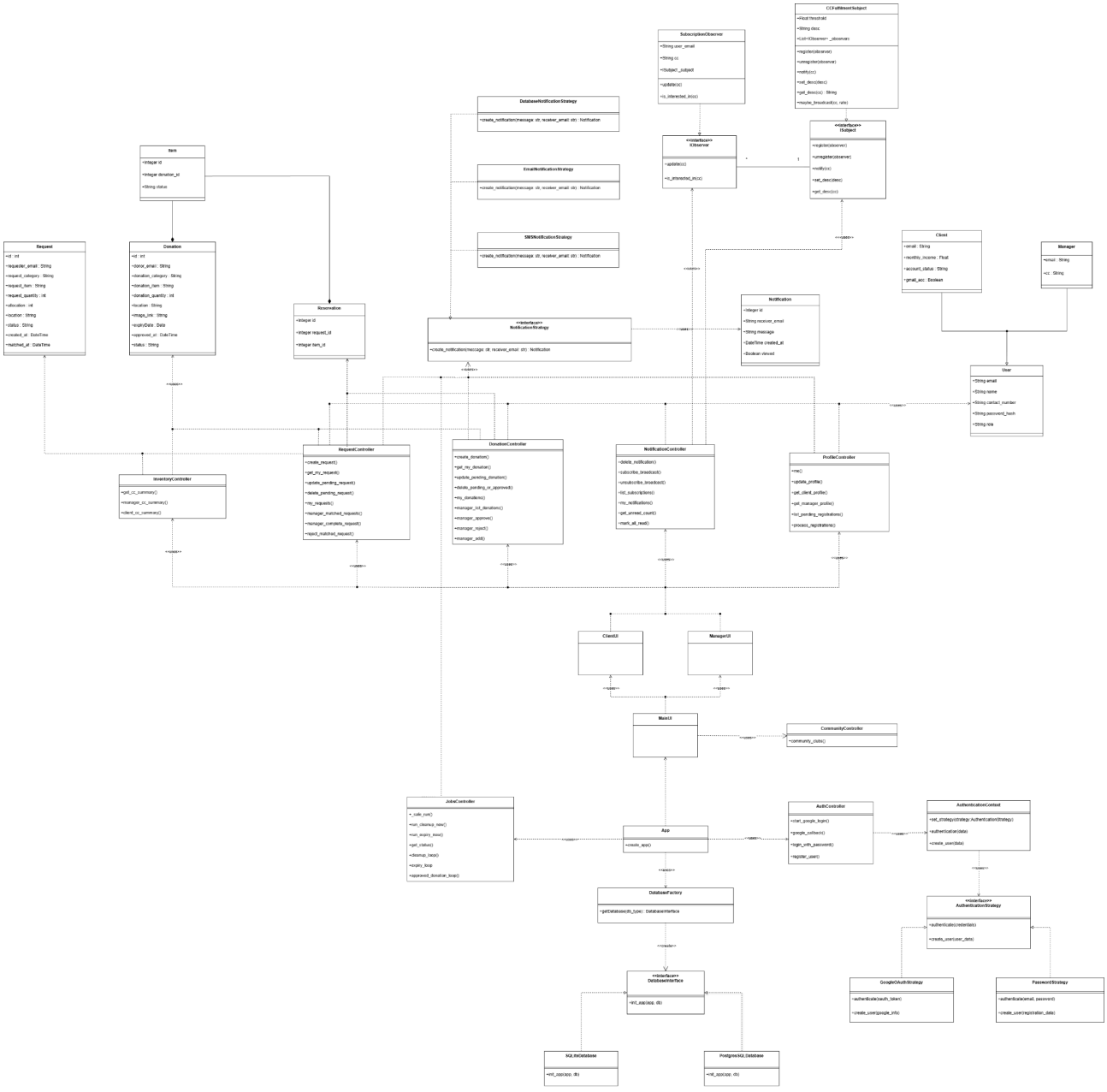
Term	Definition
CareConnect	A community-driven digital platform connecting donors, beneficiaries, and community clubs for resource sharing and assistance.
Community Club (CC)	Local hub where donations are received, stored, and distributed to beneficiaries.
User Roles	The three main user types Client (general donor), Beneficiary (low-income client), and Manager (CC coordinator).
Inventory	The collection of available donated goods at each CC; updated after donation handover or request fulfillment.
Request	A beneficiary's formal item request stored in the system until matched with available donations.
Request Status	The state of a request — Pending, Matched, Completed, Expired, or Cancelled.
Donation	Goods contributed by donors, subject to manager approval and physical handover before entering inventory.
Matching Logic	The automated process that pairs pending

	requests with available donations based on item type and quantity.
Notification	System alert informing users about approvals, matches, expirations, or shortages
Redemption Flow	The process where matched items are collected by requesters and inventory is automatically updated.
RBAC (Role-Based Access Control)	Security model that restricts system actions according to user roles.
OAuth 2.0	Secure login method allowing users to sign in via Google accounts.
API (Application Programming Interface)	The communication layer between the frontend (React) and backend (Flask) using RESTful HTTP requests.
Frontend / Backend	Frontend is the user interface (React + Vite); Backend is the server logic and database layer (Flask + PostgreSQL).
PDPA	Singapore's **Personal Data Protection Act**, ensuring user data privacy and compliance in all system operations.

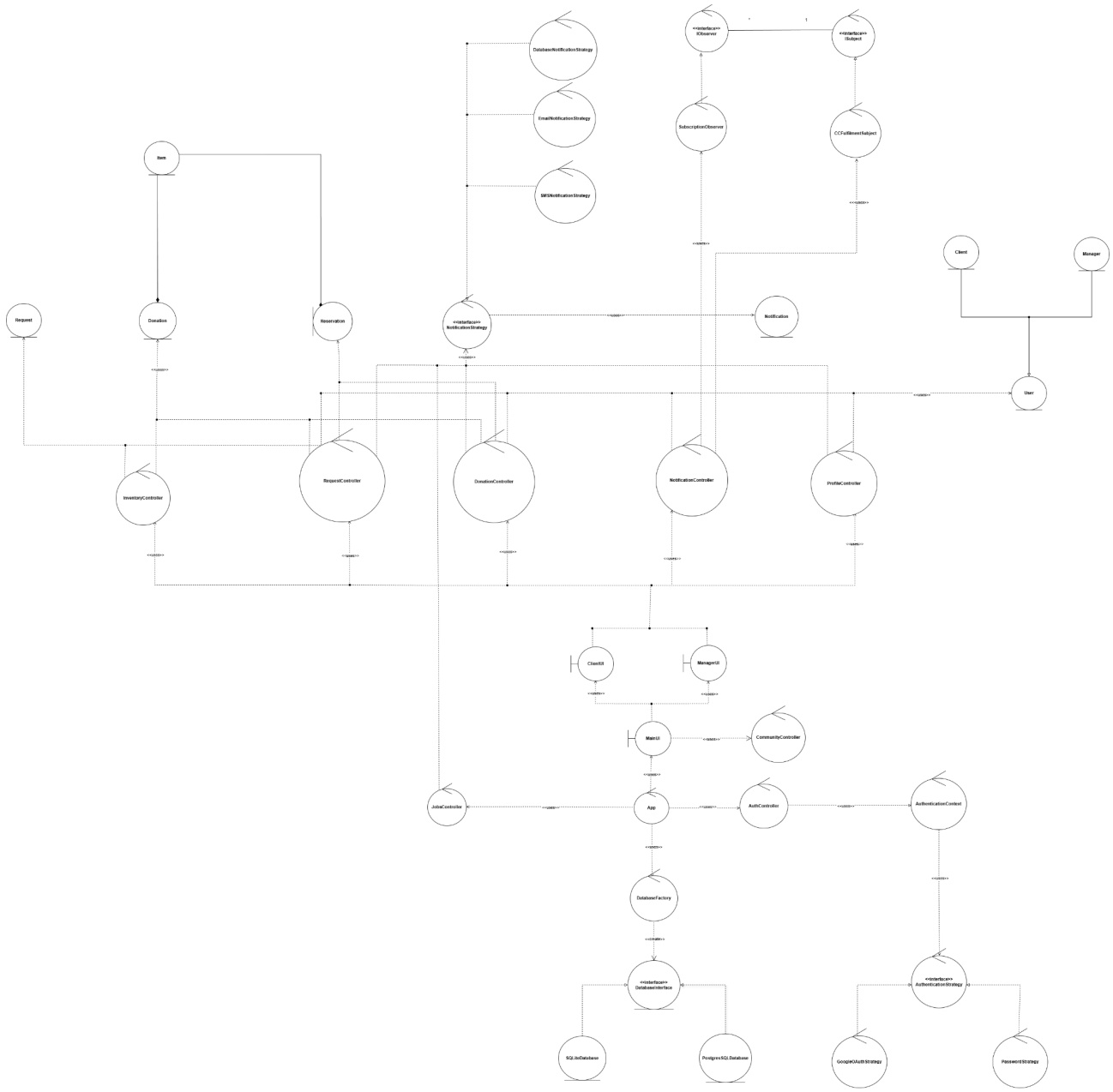
Appendix B: Analysis Models



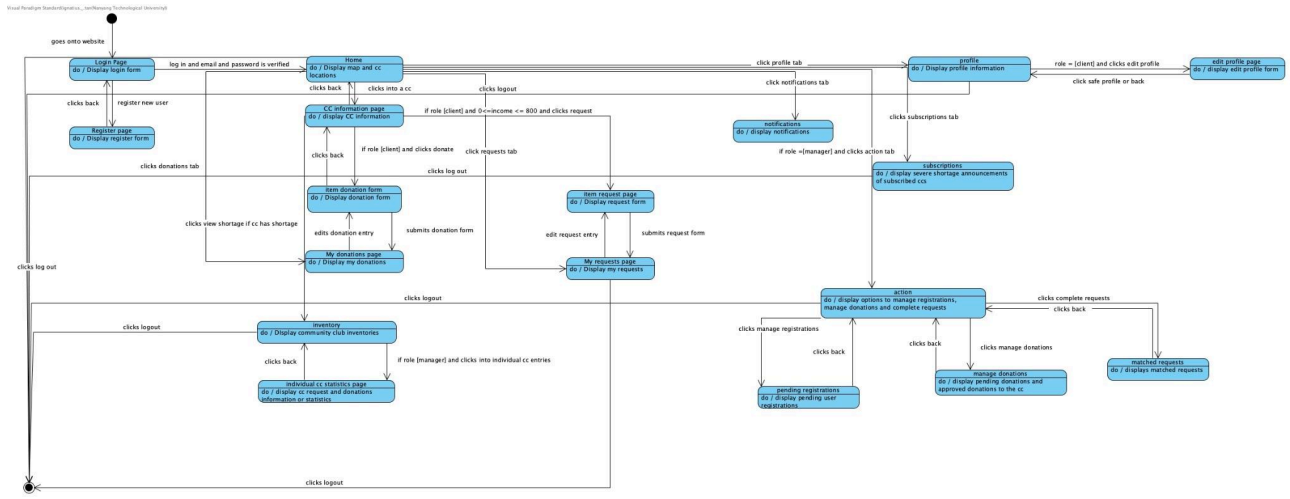
Architecture Diagram



Class Diagram



Stereotype Class Diagram


goes onto website
Login Page
do / Display login form
log in and email and password is verified
Home
do / Display map and cc locations
clicks back
register new user
Register page
do / Display register form
clicks into a cc
clicks logout
click profile tab
profile
do / Display profile information
role = [client] and clicks edit profile
edit profile page
do / display edit profile form
click safe profile or back
clicks back
CC information page
do / display CC information
if role [client] and 0<=income <= 800 and clicks request
clicks back
if role [client] and clicks donate
click requests tab
clicks log out
clicks donations tab
item donation form
do / Display donation form
item request page
do / Display request form
clicks view shortage if cc has shortage
edits donation entry
submits donation form
edit request entry
submits request form
My donations page
do / Display my donations
My requests page
do / Display my requests
clicks log out
clicks logout
inventory
do / Display community club inventories
clicks logout
clicks back
if role [manager] and clicks into individual cc entries
individual cc statistics page
do / display cc request and donations information or statistics
click notifications tab
notifications
do / display notifications
clicks subscriptions tab
if role =[manager] and clicks action tab
subscriptions
do / display severe shortage announcements of subscribed ccs
action
do / display options to manage registrations, manage donations and complete requests
clicks complete requests
clicks back
clicks manage registrations
clicks back
clicks back
clicks manage donations
pending registrations
do / display pending user registrations
manage donations
do / display pending donations and approved donations to the cc
matched requests
do / displays matched requests
clicks logout

Dialog Map

Appendix C: To Be Determined List

There are no TBD items for this version of Software Requirement Specification.

Source: http://www.frontiernet.net/~kwiegers/process_assets/srs_template.doc