## **1. Introduction**

### **1.1 Purpose**

This Software Requirements Document (SRD) describes **PantryShare**, a web‑based platform for rescuing surplus food within local school communities. It captures functional and non‑functional requirements, system architecture, and development milestones necessary for a polished hackathon deliverable.

### **1.2 Scope**

PantryShare enables high‑school students, school cafeterias, and community volunteers to:

* List surplus food items for pickup.
* Discover nearby donations via an interactive map.
* Coordinate logistics through in‑app messaging.
* Track impact metrics (meals rescued, estimated carbon saved).  
   An optional Admin Mode allows shelters to confirm receipts and manage needs.

### **1.3 Definitions, Acronyms, Abbreviations**

* **PWA** – Progressive Web App
* **CRUD** – Create, Read, Update, Delete
* **FCM** – Firebase Cloud Messaging
* **API** – Application Programming Interface
* **UI** – User Interface

## **2. Overall Description**

### **2.1 Product Perspective**

PantryShare is a standalone web application comprising:

* A mobile‑optimized frontend (React or Vue.js).
* A backend REST API (Node.js/Express or Python/Flask).
* A NoSQL document database (Firestore or MongoDB).

All modules run in the cloud; no on‑device installation is required.

### **2.2 User Classes and Characteristics**

* **Lister**: Students or cafeterias posting surplus items.
* **Volunteer**: Community members claiming pickups.
* **Shelter Admin** (optional): Confirms receipt and updates need schedules.

### **2.3 Operating Environment**

* Modern browsers (Chrome, Firefox, Safari).
* Mobile viewport support (PWA).
* Backend hosting on Heroku/Render; frontend on Vercel/Netlify.

### **2.4 Design & Implementation Constraints**

* Must use public cloud services with free tiers (Firebase, MongoDB Atlas).
* Real‑time features via WebSockets or Firebase Realtime Database.
* Map integration using Google Maps JavaScript API (API key required).

## **3. Functional Requirements**

|  |  |
| --- | --- |
| **ID** | **Requirement** |
| FR‑1 | **User Registration & Auth** |

* FR‑1.1: Users sign up/in via email & password (or OAuth).
* FR‑1.2: Session management and secure token storage. |  
   | FR‑2 | **Surplus Listing**
* FR‑2.1: Lister can create a “Surplus” with photo, description, quantity, expiry date, and pickup window.
* FR‑2.2: Edit or delete own listings before pickup. |  
   | FR‑3 | **Interactive Map**
* FR‑3.1: Display all “Available” listings as map pins with distance from user location.
* FR‑3.2: Click on pin to view details and “Claim Pickup.” |  
   | FR‑4 | **Volunteer Matching**
* FR‑4.1: Auto‑suggest nearest volunteers when a new listing appears (distance + availability).
* FR‑4.2: Volunteers receive in‑app notifications (or push via FCM). |  
   | FR‑5 | **In‑App Chat & Notifications**
* FR‑5.1: Real‑time chat between lister & volunteer for logistics.
* FR‑5.2: Push or browser notifications for new messages or claims. |  
   | FR‑6 | **Donation Tracking & Dashboard**
* FR‑6.1: Track status per listing (“Available,” “Claimed,” “Delivered”).
* FR‑6.2: Aggregate stats: total items rescued, estimated meals, CO₂ saved. |  
   | FR‑7 | **Admin Mode (Stretch)**
* FR‑7.1: Shelter Admin users confirm “Received” status.
* FR‑7.2: Admins update calendar of upcoming needs or open pickup slots. |

## **4. Non‑Functional Requirements**

|  |  |
| --- | --- |
| **Category** | **Requirement** |
| **Performance** | NFR‑1: Frontend initial load < 2 s on 4G mobile. |
| **Scalability** | NFR‑2: Support up to 1,000 concurrent users (via serverless scaling). |
| **Security** | NFR‑3: All API traffic over HTTPS; user passwords hashed (bcrypt). |
| **Usability** | NFR‑4: Mobile‑first UI; average user flow ≤ 3 taps/clicks from landing to claim. |
| **Reliability** | NFR‑5: Automatic retry on transient network failures; 99% uptime SLA on hosted services. |
| **Maintainability** | NFR‑6: Modular codebase with clear separation of frontend, backend, and data layers. |
| **Portability** | NFR‑7: PWA installable on Android and iOS; works offline to view last‑cached listings. |

## **5. System Architecture**

mathematica

CopyEdit

┌──────────────────────────┐ ┌──────────────────────────┐ ┌────────────────┐  
│ Frontend PWA │ ⟷ │ REST API Server │ ⟷ │ NoSQL DB │  
│ • React/Vue.js + PWA │ │ • Node.js/Express or │ │ Firestore / │  
│ • Google Maps Component │ │ Python/Flask │ │ MongoDB Atlas │  
│ • Socket.IO / FCM │ │ • Socket.IO (chat) │ └────────────────┘  
└──────────────────────────┘ └──────────────────────────┘

* **PWA**: Hosts UI, geolocation, map, chat.
* **REST API**: CRUD endpoints for users, listings, matches, messages, metrics.
* **NoSQL DB**: Stores user profiles, listing documents, chat threads, aggregated stats.

## **6. Detailed Component Interfaces**

### **6.1 Listing Document Schema (Firestore / MongoDB)**

json

CopyEdit

{  
 "id": "uuid",  
 "ownerId": "user\_uuid",  
 "photoUrl": "<https://…>",  
 "description": "8 slices of whole‑wheat bread",  
 "quantity": 8,  
 "expiryDate": "2025‑08‑05T18:00:00Z",  
 "pickupWindow": {  
 "start": "2025‑08‑05T15:00:00Z",  
 "end": "2025‑08‑05T17:00:00Z"  
 },  
 "location": { "lat": 40.7128, "lng": -74.0060 },  
 "status": "Available" // or "Claimed", "Delivered"  
}

### **6.2 Key REST Endpoints**

|  |  |  |
| --- | --- | --- |
| **Method** | **Path** | **Description** |
| POST | /api/auth/signup | Register new user |
| POST | /api/auth/login | Authenticate user, return JWT |
| GET | /api/listings | List all “Available” listings |
| POST | /api/listings | Create new listing |
| PUT | /api/listings/:id/claim | Claim a listing as volunteer |
| POST | /api/chat/:listingId | Send chat message |
| GET | /api/dashboard/stats | Retrieve global rescue metrics |

## **7. Development Roadmap & Milestones**

|  |  |  |
| --- | --- | --- |
| **Milestone** | **Description** | **Deadline (Days)** |
| **M1: Setup & Auth** | Project scaffold, PWA bootstrap, auth flow | 0–1 |
| **M2: Listing CRUD** | Implement surplus listing create/read/update | 1–2 |
| **M3: Map Integration** | Google Maps pins & geolocation | 2–3 |
| **M4: Volunteer Match** | Claim logic & notifications | 3–4 |
| **M5: In‑App Chat** | Real‑time messaging via Socket.IO/FCM | 4–5 |
| **M6: Dashboard** | Metrics aggregation and display | 5–6 |
| **M7: Admin Mode** | Shelter confirmations and needs calendar | 6–7 |
| **M8: Polish & Deploy** | UI refinements, bugfixes, hosting config | 7 |

## **8. Appendices**

### **A. Impact Calculations**

* **Meals Rescued** = Total quantity × standard portion size
* **CO₂ Saved** = Meals × 2.5 kg CO₂e estimate per meal

### **B. Technology References**

* Firebase Cloud Messaging: <https://firebase.google.com/docs/cloud-messaging>
* Google Maps JavaScript API: <https://developers.google.com/maps/documentation/javascript/overview>