Technical Documentation for the Expert Belongings Marketplace Platform

May 24, 2025

Abstract

This document outlines the technical specifications for a web-based platform designed to facilitate the sale of belongings by experts (e.g., United Nations or embassy staff) leav ing Tanzania. The platform uses PostgreSQL for data storage, Spring Boot for backend logic, and TypeScript for the frontend. Key features include OTP-based login, item list ings with category-specific image requirements, restricted buying for verified users, admin management of listings, and notification systems. The system does not handle payments, focusing instead on connecting buyers and sellers.

1 Introduction

1.1 Overview

The Expert Belongings Marketplace Platform enables experts, defined as individuals working in United Nations organizations or embassies in Tanzania with specific email domains (e.g., @undp.com, @un.org), to sell their belongings to other experts. Currently, these experts advertise via email, which is inefficient. This platform provides a structured, user-friendly solution for listing, browsing, and purchasing items.

1.2 Purpose and Goals

The platform aims to:

- Simplify the process of selling belongings for

  experts leaving Tanzania. • Restrict purchasing to

  verified experts based on email domains.

- Provide administrative oversight to manage listings.

- Facilitate communication between buyers and sellers without handling payments.

1.3 Key Features

- User Authentication: Login with any email using OTP verification, no signup required.
- Item Listings: Users can list items with titles, descriptions, categories (e.g., electronics, furniture), and images, with specific standards (e.g., cars require front, left, right views).

- Buying Process: Only users with verified email domains can contact sellers to buy items.
- Admin Role: Admins monitor items listed for over two weeks, send up to three notifi cations, and archive unresponsive listings.
- Notifications: Alerts for buyer-seller communication and admin updates.
- No Payment Handling: Transactions occur offline.
- Tech Stack: PostgreSQL, Spring Boot, TypeScript, with email and image storage ser vices.

2 System Architecture

2.1 High-Level Architecture

The platform follows a three-tier architecture:

- Frontend: Built with TypeScript (e.g., React or Angular), providing the user interface. • Backend: Built with Spring Boot (Java), handling API requests and business logic. • Database: PostgreSQL, storing all data.

Additional services include:

- Email Service: For OTPs and notifications (e.g., SendGrid, https://sendgrid. com).
- Image Storage: For item images (e.g., AWS S3, https://aws.amazon.com/s3). • Caching: Optional, using Redis for performance (https://redis.io).

2.2 Component Diagram

The system components interact as follows:

[Frontend (TypeScript)] <--> [Backend (Spring Boot)] <--> [Database (Post| | | |---
[Email Service] |--- [Image Storage] | | |--- [Caching (Redis)] |

2.3 Workflow

1. Login: Users enter an email, receive an OTP, and verify it. New users are created auto matically.
2. Item Listing: Users create listings with required fields and images.
3. Browsing/Buying: All users can browse; only verified buyers can contact sellers.
4. Admin Actions: Admins monitor and manage listings, sending notifications and archiv ing as needed.
5. Notifications: Sent for buyer-seller messages and admin updates.

3 Database Schema

3.1 Tables and Columns

The database schema includes the following tables:

Table 1: Users Table

Column	Type	Description
user$_id$	serial	Primary key
email	varchar (unique)	User's email address
role	enum ('user', 'admin')	User role, default 'user'
is$_verified_buyer$	boolean	True if email domain is allowed, default false
created$_at$	timestamp	Creation timestamp
updated$_at$	timestamp	Last update timestamp

Table 2: AllowedDomains Table

Column	Type	Description
domain$_id$	serial	Primary key
domain	varchar (unique)	Allowed email domain (e.g., 'undp.com')

Table 3: Items Table

Column	Type	Description
item$_id$ title description category$_id$ seller$_id$ listing$_date$ status created$_at$ updated$_at$	serial varchar text integer (FK to Categories) integer (FK to Users) timestamp enum ('active', 'archived') timestamp timestamp	Primary key Item title Item description Category reference Seller reference Listing date Item status, default 'active' Creation timestamp Last update timestamp

3.2 Relationships

- Users: One-to-many with Items (seller), Messages (sender/receiver), Notifications (re cipient).
- Items: Many-to-one with Users (seller), Categories; one-to-many with Images, Mes sages.
- Categories: One-to-many with Items.
- Images: Many-to-one with Items.
- Messages: Many-to-one with Users

(sender/receiver), Items (optional). • Notifications:

Many-to-one with Users.

- AdminActions: Many-to-one with Items.

Table 4: Categories Table

Column	Type	Description
category$_id$ name description	serial varchar text timestamp	Primary key Category name (e.g., 'electronics') Category description Creation timestamp

created$_at$ updated$_at$	timestamp	Last update timestamp

Table 5: Images Table

Column	Type	Description
image$_id$	serial	Primary key
item$_id$	integer (FK to Items)	Item reference
url	varchar	Image URL (e.g., AWS S3)
type	varchar (nullable)	Image type (e.g., 'front' for cars)
created$_at$	timestamp	Creation timestamp
updated$_at$	timestamp	Last update timestamp

3.3 Entity-Relationship Diagram

The ER diagram includes:

• Entities: Users, AllowedDomains, Items, Categories, Images, Messages, Notifications, AdminActions.

• Relationships: As described above, with foreign keys linking tables.

(Developers can use tools like pgAdmin or DBeaver to visualize the schema.)

4 User Authentication and Authorization

4.1 Login Process

1. User enters email.

2. Backend sends OTP via email service.

3. User verifies OTP.

4. If email is new, create user; else, log in existing user.

5. Check email domain against AllowedDomains; set is$_v$*erif ied*$_b$*uyer.IssueJW T tokenforsessionmanagem*

4.2 Authorization

6.• Sellers: Any user can list items.

• Buyers: Only users with is*verif iedbuyer* = *truecansendmessagestobuy.***Admins** :

*Userswith*role = 4

Table 6: Messages Table

Column	Type	Description
message*id*	serial	Primary key
sender*id*	integer (FK to Users)	Sender reference
receiver*id*	integer (FK to Users)	Receiver reference
item*id*	integer (FK to Items, nullable)	Item reference
message*text*	text	Message content
timestamp	timestamp	Message timestamp
is*read*	boolean	True if read, default false

Table 7: Notifications Table

Column	Type	Description
notification*id*	serial	Primary key
user*id*	integer (FK to Users)	Recipient reference
message	text	Notification content
type	enum (’item*update*’, ’*admin**message*’)	Notification type
timestamp	timestamp	Notification timestamp
is*read*	boolean	True if read, default false

5 Item Management

5.1 Listing Items

Users provide:

- Title, description, category.

• Images, with validation for categories like cars (front, left, right).

Images are stored in cloud storage (e.g., AWS S3).

5.2 Browsing Items

All logged-in users can view active items (status = 'active').

5.3 Buying Items

Only verified buyers can send messages to sellers. Transactions occur offline.

5.4 Admin Management

• Monitor items with listing$_d$*ateover*2*weeks.Sendnotif ications*(*uptothree*). • Archive items (status = 'archived') if no response.

Table 8: AdminActions Table

Column	Type	Description
action$_i$*d* item$_i$*d* action$_t$*ype* timesta mp	serial integer (FK to Items) enum ('update', 'archive') timestamp	Primary key Item reference Action type Action timestamp

6 Communication and Notifications

6.1 Messages

Buyers send messages to sellers about items, stored in the Messages table. Sellers receive notifications.

6.2 Notifications

System notifications (e.g., item updates) are stored in the Notifications table.

7 Technical Stack

• Backend: Spring Boot (https://spring.io/projects/spring-boot). • Frontend: TypeScript (e.g., React, https://reactjs.org).

• Database: PostgreSQL (https://www.postgresql.org).

• Email Service: SendGrid or AWS SES.

• Image Storage: AWS S3.

• Caching: Redis (optional).

8 API Endpoints

• User:

  – POST /login: Login with email and OTP.
  – GET /user/profile: Get user profile.

• Items:

  – GET /items: List active items.
  – POST /items: Create item.
  – GET /items/{id}: Get item details.
  – PUT /items/{id}: Update item (seller).
  – DELETE /items/{id}: Delete item (seller).

• Categories:

  – GET /categories: List categories.

• Messages:

  – POST /messages: Send message.
  – GET /messages: Get messages.

• Notifications:

  – GET /notifications: Get notifications.
  – PUT /notifications/{id}/read: Mark as read.

• Admin:

– GET /admin/items/overdue: Get overdue items.
– POST /admin/items/{id}/update: Send update.
– POST /admin/items/{id}/archive: Archive item.

9 Security Considerations

• Authentication: OTP-based, no passwords.

• Authorization: Role-based access control.

• Data Validation: Validate inputs to prevent injection.

• Protection: Use HTTPS, secure Spring Boot and TypeScript practices.

10 Performance and Scalability

• Indexing: On item.status, item.listing$_{d}$*ate,* user.email.**Caching** : *Redisforcategori*• Scalability: Single instance initially; add load balancing if needed.

11 Deployment

• Backend: Deploy on AWS EC2 or Heroku.

• Frontend: Deploy on Netlify or Vercel.

• Database: Use AWS RDS or Heroku Postgres.

• Configuration: Environment variables for credentials.

12 Future Enhancements

• Add payment handling.

• Enhance search with filters (e.g., category,

location). • Add user profiles.

• Integrate social media sharing.