# JobTradeSasa — Software Requirements Specification (SRS), Use Cases, User Stories & UML

---

## Revision History

- **2025-10-22** — Initial SRS, use cases, user stories, data model, and UML (mermaid) diagrams for JobTradeSasa (React + Node.js + PostgreSQL, hosted on Vercel for frontend).

---

# 1. Introduction

## 1.1 Purpose

This document provides a complete Software Requirements Specification (SRS) for **JobTradeSasa**, a community-focused marketplace that connects service requesters with local service providers (artisans, technicians, suppliers, and companies). It contains functional and non-functional requirements, use cases, user stories with acceptance criteria, data model (PostgreSQL), API overview, and UML diagrams (use case, class, sequence, and component diagrams) to guide development.

## 1.2 Intended Audience

- Project owner / product manager
- Developers (frontend, backend, mobile)
- QA / Testers
- DevOps and deployment engineers
- Stakeholders and potential investors

## 1.3 Scope

JobTradeSasa aims to: - Allow requesters to post service requests (normal or emergency) and pick a provider from nearby available providers. - Allow providers to manage availability, respond, chat, and log completed jobs and earnings. - Provide an admin panel for moderation, reports, and analytics. - Support web and mobile (progressive web or hybrid app via React + React Native if later desired). - Payments are offline/agreed in-person; the system records agreed price and actual paid amount.

---

# 2. Overall Description

## 2.1 Product Perspective

A two-sided marketplace connecting requesters and providers, with features inspired by ride-hailing apps (location-based matching) and freelancing platforms (profiles, reviews). React frontend served on Vercel; Node.js backend (Express) hosting APIs; PostgreSQL DB hosted on a managed provider (e.g., Supabase, Railway, ElephantSQL).

## 2.2 Key Stakeholders

- Requesters (customers)
- Service Providers (vendors, companies)
- Admins (platform operators)
- Developers & DevOps

## 2.3 Assumptions & Dependencies

- Users have smartphones or browsers with GPS support.
- PostgreSQL managed hosting available.
- Push notifications via FCM or Web Push.
- No online payment gateway required (payments occur offline and are recorded).

---

# 3. Functional Requirements

Functional requirements are listed per role. Each requirement has an ID for traceability.

## 3.1 Common (All Authenticated Users)

- FR-COM-01: Account creation and authentication (email/phone + password, OTP phone verification).
- FR-COM-02: Edit profile (name, photo, contact info, location, languages).
- FR-COM-03: Push notifications and in-app notifications.
- FR-COM-04: Localization and language selection.

## 3.2 Requester (Customer)

- FR-R-01: Create a service request (category, title, description, photos, location, urgency flag (Emergency / Normal), preferred time, budget estimate).
- FR-R-02: See a ranked list and map of nearby providers for the requested category with filters: distance, rating, price, availability.
- FR-R-03: Select a provider and initiate chat.
- FR-R-04: View provider profile (ratings, reviews, portfolio images, badges, response time, completed jobs).
- FR-R-05: Record agreed price and later mark payment as completed (amount agreed vs paid).
- FR-R-06: Rate and review provider after job completion.
- FR-R-07: View job history, receipts, and invoices (system-recorded only).

- FR-R-08: Mark job status (cancel, complete) where applicable.

## 3.3 Service Provider (Vendor)

- FR-P-01: Provider registration and verification (ID upload, skill certificates optional).
- FR-P-02: Create and manage service offerings (categories, description, base price, images).
- FR-P-03: Set availability (online/offline), service area radius, and working hours.
- FR-P-04: Receive notifications of matching job requests and accept/decline them.
- FR-P-05: Chat with requester (text, images, voice notes).
- FR-P-06: Submit on-site estimate or mark "quote needed" and later provide quotation.
- FR-P-07: Mark job progress (accepted, en route, on site, completed).
- FR-P-08: Log actual price received and confirm payment on the app (amount agreed vs paid).
- FR-P-09: Access analytics dashboard (jobs completed, earnings, response time).

## 3.4 Admin

- FR-A-01: Admin login and role-based access control.
- FR-A-02: Approve/Reject provider registrations (verification).
- FR-A-03: Manage categories, featured promotions, and materials specials section.
- FR-A-04: View all jobs, conversations, disputes, and reports.
- FR-A-05: Resolve disputes and adjust records if required (refunds are offline but admin can annotate).
- FR-A-06: Analytics and reporting (daily/weekly/monthly jobs, top providers, average response time).
- FR-A-07: Send global announcements and promotional push notifications.

---

# 4. Non-Functional Requirements

- NFR-SEC-01: All API traffic over HTTPS. Passwords stored with salted hashing (bcrypt).
- NFR-SEC-02: Chat messages stored encrypted at rest if possible; at minimum, access controlled.
- NFR-PERF-01: Typical API response time < 500ms for core endpoints (search, provider list) under normal load.
- NFR-SCAL-01: System must be horizontally scalable: stateless API servers, shared DB, caching layer (Redis).
- NFR-AV-01: 99.5% uptime for API and web frontend (hosted on Vercel / managed provider).
- NFR-INT-01: Mobile-first responsive UI, accessibility standards (WCAG AA basic).
- NFR-MTBF-01: Backups: daily DB snapshots and point-in-time where supported; ability to restore within 4 hours.
- NFR-PRIV-01: GDPR-like privacy compliance for personal data; users can request data deletion.

---

# 5. Data Model (PostgreSQL)

## 5.1 Core Tables (suggested schema)

### users

- id (uuid, PK)
- role (enum: requester, provider, admin)
- name
- email (unique)
- phone
- password_hash
- profile_photo_url
- bio
- created_at
- updated_at
- is_verified (bool)

### providers (extended profile)

- user_id (uuid, PK, FK -> users.id)
- company_name
- service_categories (jsonb) -- list of category ids
- base_price_info (jsonb)
- service_area_radius_meters (integer)
- average_response_time_seconds (integer)
- rating_average (numeric)
- completed_jobs_count (integer)
- verification_documents (jsonb)

### categories

- id (serial, PK)
- name
- parent_id (nullable)
- description

### jobs

- id (uuid, PK)
- requester_id (uuid, FK -> users.id)
- provider_id (uuid, FK -> users.id, nullable until accepted)
- category_id (int, FK -> categories.id)
- title
- description (text)
- photos (jsonb)
- location (geography point) -- PostGIS recommended or store lat/lng
- urgency (enum: normal, emergency)
- preferred_time (timestamp)
- status (enum: open, offered, accepted, enroute, onsite, completed, cancelled)
- price_agreed (numeric, nullable)

- price_paid (numeric, nullable)
- created_at
- updated_at

### messages

- id (uuid, PK)
- job_id (uuid, FK)
- sender_id (uuid, FK -> users.id)
- message_text (text, nullable)
- attachments (jsonb)
- voice_note_url (text, nullable)
- created_at

### ratings

- id (uuid, PK)
- job_id (uuid, FK)
- from_user_id (uuid, FK)
- to_user_id (uuid, FK)
- rating (int: 1..5)
- comment (text)
- created_at

### promotions

- id (serial)
- provider_id (uuid)
- title
- description
- start_date
- end_date
- media (jsonb)

### provider_analytics (materialized or computed)

- provider_id
- period_start
- jobs_completed
- total_earnings
- avg_response_time

## 5.2 Indexing & Extensions

- Use PostGIS for efficient geo-queries (nearest providers). If PostGIS is not available, create GIN indices on lat/lng and use manual bounding-box queries for distance.
- Index jobs(status), jobs(category_id), users(email) UNIQUE.
- GIN index on providers.service_categories (jsonb).

---

# 6. Use Cases (textual) + Mermaid Use Case Diagram

## 6.1 Top Use Cases (text)

- UC1: Post job request (Requester posts a job; system notifies nearby providers)
- UC2: View and filter providers (Requester views provider list & map)
- UC3: Accept/Decline job (Provider accepts a job and updates status)
- UC4: In-app chat (Requester & Provider communicate with text/images/voice)
- UC5: Record payment (Either party records amount paid and marks job paid)
- UC6: Rate provider (Requester rates provider after completion)
- UC7: Provider analytics (Provider views performance metrics)
- UC8: Admin verification (Admin approves/rejects provider documents)

## 6.2 Mermaid Use Case Diagram

```
usecaseDiagram
  actor Requester
  actor Provider
  actor Admin

  Requester --> (Post Job Request)
  Requester --> (View Providers)
  Requester --> (Chat with Provider)
  Requester --> (Rate Provider)
  Requester --> (Record Payment)

  Provider --> (Receive Job Notification)
  Provider --> (Accept/Decline Job)
  Provider --> (Chat with Requester)
  Provider --> (Update Job Status)
  Provider --> (View Analytics)

  Admin --> (Verify Provider)
  Admin --> (Manage Categories)
  Admin --> (Resolve Disputes)
```

---

# 7. User Stories & Acceptance Criteria

### Epic A — Job Posting & Matching

- **US-A1**: As a *Requester*, I want to post a job with location and urgency so that nearby providers can respond.

- *Acceptance*: Job appears in requester history, at least 3 providers within 10km are notified (if available).

- **US-A2**: As a *Requester*, I want to filter providers by distance, rating, and price so I can pick the best one.
- *Acceptance*: Filters apply in under 1 second and results update on UI.

### Epic B — Communication & Confirmation

- **US-B1**: As a *Requester*, I want to chat with providers (text, images, voice) so we can clarify the job.
- *Acceptance*: Messages deliver in near real-time; attachments upload successfully; voice notes playable.
- **US-B2**: As a *Provider*, I want to accept or decline a job notification so I can manage workload.
- *Acceptance*: Accepting updates job status and notifies the requester.

### Epic C — Payments & Ratings

- **US-C1**: As a *Requester*, I want to record the agreed price and mark when I paid so records are preserved.
- *Acceptance*: Job record shows agreed price and payment status (paid/unpaid) with timestamp.
- **US-C2**: As a *Requester*, I want to rate a provider after job completion so future users can trust good providers.
- *Acceptance*: Rating updates provider average and appears in provider profile.

### Epic D — Admin & Analytics

- **US-D1**: As an *Admin*, I want to verify provider credentials so that platform trust remains high.
- *Acceptance*: Admin can mark provider verified; verified badge appears on the provider profile.
- **US-D2**: As an *Admin*, I want to view weekly reports of jobs and top providers so I can promote deserving vendors.
- *Acceptance*: Admin dashboard shows jobs count, top 10 providers, and average response time for selected period.

---

# 8. API Overview (High-level)

Base: `https://api.jobtradesasa.example.com`

### Authentication

- `POST /auth/signup` — create account (role in payload)
- `POST /auth/login` — login (returns JWT)
- `POST /auth/verify-phone` — OTP verify

**Requester / Jobs**

- `POST /jobs` — create job
- `GET /jobs/:id` — job details
- `GET /jobs?lat=&lng=&category=&radius=` — search jobs / nearby providers logic
- `PATCH /jobs/:id` — update job (cancel, mark completed)

**Providers**

- `GET /providers?category=&lat=&lng=&radius=&sort=` — find providers
- `POST /providers/:id/accept` — accept job
- `PATCH /providers/:id/status` — set availability

**Chat**

- `GET /jobs/:id/messages` — get chat messages
- `POST /jobs/:id/messages` — post message (text, images, voice note)

**Admin**

- `GET /admin/providers?status=pending` — list pending verifications
- `PATCH /admin/providers/:id/verify` — verify provider

Authentication: JWT in `Authorization: Bearer <token>` header.

---

# 9. UML Diagrams (Mermaid source)

## 9.1 Class Diagram

```
classDiagram
  class User {
    +UUID id
    +String name
    +String email
    +String phone
    +String role
    +Boolean is_verified
  }
  class Provider {
    +UUID user_id
    +JSON service_categories
    +Integer completed_jobs
    +Float rating_average
  }
  class Job {
    +UUID id
    +UUID requester_id
    +UUID provider_id
    +String title
    +String description
    +Enum status
    +Numeric price_agreed
  }
  class Message {
    +UUID id
    +UUID job_id
    +UUID sender_id
    +String text
    +String voice_note_url
  }
  User <|-- Provider
  User "1" o-- "*" Job : requests
  Provider "1" o-- "*" Job : performs
  Job "1" o-- "*" Message : has
```

## 9.2 Sequence Diagram (Requester posts job -> Provider accepts -> Chat -> Complete)

```
sequenceDiagram
  participant R as Requester
  participant API as Backend
  participant P as Provider
  R->>API: POST /jobs (title, location, urgency)
  API-->>P: Push notification (matching providers)
  P->>API: POST /providers/:id/accept (job_id)
  API->>R: Notify accepted
  R->>API: POST /jobs/:id/messages ("where are you?")
  P->>API: POST /jobs/:id/messages ("arriving in 10min")
  P->>API: PATCH /jobs/:id (status=completed, price_paid)
  API->>R: Notify job completed
```

## 9.3 Component Diagram

```
flowchart LR
  subgraph Frontend
    FE[React App / React Native]
  end
  subgraph Backend
    API[Node.js/Express API]
    Auth[Auth Service (JWT)]
    Chat[Chat Service / WebSocket]
  end
  subgraph Data
    DB[(PostgreSQL + PostGIS)]
    Cache[(Redis)]
  end
  FE --> API
  API --> DB
  API --> Cache
  FE --> Chat
  Chat --> API
  API --> Auth
```

---

## 10. Acceptance Testing & QA Notes

- Unit tests for core services: auth, job creation, provider matching.
- Integration tests for chat, job lifecycle, and recording payments.
- End-to-end tests (Playwright or Cypress) for major user flows (post job → accept → chat → complete → rate).
- Load tests for search and push-notification endpoints.

---

## 11. Deployment & Hosting Recommendations

- **Frontend**: React SPA hosted on Vercel.
- **Backend**: Node.js/Express on a provider that supports persistent server processes or serverless with connection pooling (Railway, Render, Fly.io, or Vercel serverless functions with a connection pooler).
- **Database**: Managed PostgreSQL with PostGIS support (Supabase, Railway, ElephantSQL, or AWS RDS).
- **Real-time chat**: WebSocket server (Socket.IO) hosted on the same backend or a dedicated service (Pusher / Supabase Realtime).
- **Push notifications**: Firebase Cloud Messaging (for Android and web push).
- **Monitoring**: Sentry for errors, Prometheus + Grafana for metrics or provider equivalent.

---

## 12. Roadmap for an MVP (Suggested scope for first release)

- User accounts (requester/provider) + basic profiles
- Post job, provider discovery by distance, accept/decline
- Real-time chat (text + images + voice note uploads)
- Job lifecycle (open → accepted → completed) and offline payment recording
- Ratings & reviews
- Admin dashboard (basic provider verification and reports)

---

## 13. Next Steps

1. Review and approve this SRS document.

2. Break down into prioritized backlog (sprints) and estimate effort.
3. Start implementing the MVP (frontend skeleton, backend auth, jobs CRUD, provider discovery).
4. Build chat and provider analytics in subsequent sprints.

---

## 14. Appendix

- Suggested libraries and tools:
- Frontend: React, Redux or Zustand, React Router, Leaflet or Google Maps React.
- Mobile: React Native (if creating native mobile later) or PWA approach.
- Backend: Node.js, Express, Socket.IO, JWT, Joi (validation).
- Database: PostgreSQL, PostGIS, Prisma ORM or Sequelize.
- DevOps: Vercel (frontend), Supabase/Railway (DB), Docker for local dev.

---

*End of document.*