**Software Requirements Specification (SRS) for Workjunction**

**1. Introduction**

**1.1 Purpose**

The purpose of this document is to provide a comprehensive, unambiguous, and detailed outline of the functional and non-functional requirements of Workjunction, a modern service-on-demand digital platform.

This SRS serves as the single source of truth for all stakeholders. It specifies:

* What the system will do.
* Who will use the system.
* How the system will behave under different conditions.
* The constraints under which the system must operate.

The SRS benefits all stakeholders in the following ways:

* **Project Managers**: Define scope, allocate resources, track progress, and evaluate deliverables.
* **Developers**: Reference requirements to design and implement system features, including dependencies such as Razorpay integration and JSON Web Token (JWT) authentication.
* **Quality Assurance Testers**: Develop test plans to ensure all requirements are met.
* **Business Stakeholders**: Understand system capabilities, limitations, and expected business value.

By detailing these requirements in advance, the SRS minimizes miscommunication, scope creep, and rework during development.

**1.2 Product Scope**

Workjunction is a web-based service marketplace that connects customers with verified service providers (workers). It aims to provide transparency, trust, and convenience in booking professional and domestic services.

The scope includes the entire service lifecycle:

* **User Onboarding**: Secure registration, login, and profile management for customers, workers, service agents, and administrators.
* **Service Discovery**: Customers can browse and search for workers based on type of service, location, reviews, and pricing.
* **Booking Management**: Workflow-based booking process with statuses such as Pending, Confirmed, In Progress, Completed, or Cancelled.
* **Worker Verification**: Aadhaar, live selfie, and police verification uploads for trust building. Verified workers become visible to customers.
* **In-App Communication**: Secure chat between customers and workers.
* **Payments**: Integration with Razorpay for both pre-service and post-service payments.
* **Administrative Oversight**: Centralized admin panel for managing agents, users, services, and reports.
* **Service Agent Oversight**: Local managers assist with onboarding, verification, and worker support.

The system will use a scalable cloud-based architecture to support future growth.

**1.3 Intended Audience**

* **Project Managers**: For planning and resource allocation.
* **Developers**: For detailed technical implementation.
* **Quality Assurance Teams**: For validation and verification.
* **Business Stakeholders/Investors**: For market alignment and ROI evaluation.
* **Operations and Support Teams**: For deployment and maintenance.

**2. Overall Description**

**2.1 Product Perspective**

Standalone cloud-native system with these subsystems:

* Customer System — register, search, book, chat, pay, feedback.
* Worker System — register, verify, set availability, manage bookings, upload portfolio.
* Service Agent System — regional onboarding, verification handling, local support.
* Admin System — global governance, user/service/agent management and reports.

**2.2 User Classes and Characteristics**

* Customers: simple UI, trust & transparency.
* Workers: profile control, verification flows, earnings view.
* Service Agents: assist non-technical workers, review documents.
* Administrators: manage global settings, fraud prevention, reporting.

**2.3 Operating Environment**

* Frontend: React.js (responsive).
* Backend: Node.js + Express (REST APIs).
* Database: MongoDB.
* File Storage: Cloudinary.
* Auth: JWT.
* Payments: Razorpay.
* Hosting: Cloud (AWS/GCP/Azure).

**3. System Features and Functional Requirements**

Each requirement below preserves your **Input** and **Output** text and adds a **Process** section that clearly explains how the system converts the input into the output. I also add common validation, errors, and data persistence notes where helpful.

**3.1 User Module**

**REQ-US-001 – Registration (with Email OTP Verification)**

**Inputs:**

* Full Name
* Email Address (for OTP verification)
* Password
* Mobile Number
* OTP (entered after receiving via email)
* **Role** (Customer, Worker, Admin, Service Agent)

**Process:**

1. **User Submits Registration Form:**
   * User provides name, email, password, mobile number, and selects a role (Customer / Worker / Admin / Service Agent).
2. **Server Validations:**
   * Required fields (Full Name, Email, Password, mobile number, Role) must be non-empty.
   * Email format is validated.
   * Password strength is validated.
   * Mobile number is validated.
   * Ensure role value is one of the allowed roles.
3. **Uniqueness Check:**
   * Server checks if the email already exists.
   * If exists → return error (duplicate account).
4. **OTP Generation (Email):**
   * If new user, server generates a random OTP.
   * OTP is sent to the provided email via SMTP/mail service.
   * OTP (hashed + expiry time) stored temporarily along with registration data, marked as pendingVerification.
5. **OTP Verification:**
   * User enters OTP.
   * Server verifies OTP hash & expiry.
   * If OTP valid →
     + Server hashes the password using argon2.
     + Creates new user record with:
       - Unique User ID
       - Encrypted password
       - Selected role (Customer, Worker, Admin, or Service Agent)
       - Status = Active.

**Outputs:**

* **On Success:**
  + A unique user ID is generated.
  + User account created with encrypted password.
  + Role assigned as per input.
  + Success message returned.
  + Email verified successfully.
* **On Failure:**
  + Error message for invalid input, duplicate account, or OTP failure.

**REQ-US-002 – Login**

**Inputs:**

* Email Address
* Password

**Process:**

1. **User submits login form (email + password).**
2. **Server validations:**
   * Check if email and password fields are provided.
   * Validate email format.
3. **Authentication:**
   * Load user by email.
   * If not found → return "Invalid credentials".
   * Compare password with stored hash using bcrypt
4. **Successful Login:**
   * Generate **JWT** containing:
     + userId, role, issuedAt, expiry.
   * Return JWT to client.

**Outputs:**

* On success:
  + JSON Web Token (JWT) returned.
  + Success message to Customer.
* On failure:
  + Error message (invalid credentials / locked account).

**REQ-US-003 – Profile Management**

**Inputs:**

* Full Name
* Email Address
* Phone Number
* Address:
  + Street
  + House No.
  + Area
  + City
  + State
  + Pincode

**Process:**

1. **Authentication:**
   * User is authenticated using a valid JWT before accessing profile management.
2. **User submits profile update request.**
3. **Validation:**
   * Server validates all field formats:
     + Email → proper format.
     + Phone → valid number format.
     + Pincode → numeric and length constraints.
   * Reject request if invalid.
4. **Uniqueness checks (if sensitive fields changed):**
   * If **email** is changed → check uniqueness across users.
     + If new email exists → return error.
     + Otherwise, mark email as unverified and trigger email OTP verification.
   * If **phone** is changed → check uniqueness across users.
     + If new phone exists → return error.
     + Otherwise, save changes.
5. **Database Update:**
   * Update user document in DB with new details.
   * Update updatedAt timestamp.
6. **Response:**
   * Return updated profile object (excluding sensitive fields like password).
   * Return success message to client.

**Outputs:**

* Profile information updated in database.
* Success message displayed to user.
* If email/phone changed → pending re-verification until confirmed.

3.2 Customer Module

**REQ-CU-001 – Service Search and Browsing**

**Inputs:**

* **Skill** (e.g., Plumbing, Electrical, Carpentry)
* **Service** (specific service under skill, e.g., Pipe Repair, Fan Installation)
* **Price Range** (min, max)
* **Rating Range** (min, max)
* **Location** (City, Pincode, or use customer’s default address from profile)
* **Worker Name** (optional, partial match allowed)
* **Worker Phone No.** (optional, partial match allowed, rate-limited)
* **Sort By** (rating, price, distance, relevance)

**Process:**

1. **Authentication & Context Retrieval:**
   * User request includes JWT.
   * Server verifies JWT and retrieves customer’s profile (default address if location not explicitly given).
2. **Filter Selection Flow:**
   * Customer selects **Skill → Service → Additional Filters** (price, rating, location, etc.).
   * Server validates filter formats (price, rating, pincode).
   * Inputs are sanitized to prevent injection.
3. **Database Query Construction:**
   * **Skill & Service Match:** Query workers who provide the chosen service within the selected skill.
   * **Geospatial Filtering:**
     + If lat/long available from profile → run $nearSphere (MongoDB) to find nearby workers within defined radius.
     + If only city/pincode → match against indexed location fields.
   * **Filters Applied:**
     + Rating range filter.
     + Price range filter.
     + Worker name / phone (if provided).
4. **Sorting & Ranking:**
   * Sort results by relevance, rating, distance, or price (based on user selection).
   * Default: relevance + rating.
5. **Response Delivery:**
   * Return paginated list of workers.
   * If no matches → return "No results found".

**Outputs:**

* On Success:
  + Paginated list of verified workers matching **skill, service, and filters**.
  + Worker details: name, skill, service, rating, pricing, distance, availability.
* On Failure:
  + Error for invalid filters or authentication failure.
  + "No results found" if no workers match.

**REQ-CU-002 – Booking History**

**Inputs:**

* **User ID** (retrieved implicitly from JWT authentication)
* **Optional Filters:**
  + Date Range (from–to)

**Process:**

1. **Authentication:**
   * User sends an authenticated request with JWT.
   * Server validates JWT and extracts userId.
2. **Database Query:**
   * Query the bookings collection for records where customerId = userId.
   * Apply optional filters (date range).
3. **Data Enrichment:**
   * Join booking records with worker profile to fetch:
     + Worker Name
     + Worker Phone No.
     + Worker Service details
   * Mask sensitive info if needed (e.g., hide phone number until booking is confirmed).
4. **Sorting:**
   * Sort bookings by bookingDate (most recent first).
5. **Response Delivery:**
   * Return structured booking history data.
   * If no bookings exist → return "No booking history available".

**Outputs:**

* On Success:
  + A paginated list of bookings for the authenticated user.
  + Each booking record includes:
    - Service Name
    - Worker Details (Name, Phone No., Service Type)
    - Booking Date & Time
    - Booking Status (Pending, Confirmed, Completed, Cancelled)
    - Payment Status (if applicable)
* On Failure:
  + Error message for invalid token or DB query failure.
  + "No booking history available" if no records found.

**REQ-CU-003 – Ratings and Reviews**

**Inputs:**

* **Booking ID**
* **Rating** (1–5 stars)
* **Review Text** (optional)

**Process:**

1. **Authentication & Booking Validation:**
   * Verify JWT and extract userId.
   * Confirm booking exists and customerId = userId.
2. **Eligibility Check:**
   * Ensure booking status = Completed.
3. **Review Storage:**
   * Create or update review record containing:
     + bookingId
     + rating
     + reviewText
     + timestamp
4. **Aggregate & Update Worker Profile:**
   * Recalculate worker’s average rating and total rating count:
     + avgRating = sum(ratings) / count(ratings)
     + ratingCount = count(ratings)
   * Update worker document with new rating values.
   * Use database transaction or background worker for consistency.
5. **Notification:**
   * Notify worker (email) about the new review.

**Outputs:**

* On Success:
  + Review is stored and linked to the booking & worker profile.
  + Worker’s avgRating and ratingCount updated.
  + Success message returned to customer.
* On Failure:
  + Error for invalid booking, duplicate review, expired review window, or invalid rating.
  + Message if review moderation blocks submission.

**REQ-CU-004 – In-App Chat**

**Inputs:**

* SenderUserID
* ReceiverUserID
* MessageContent

**Process:**

1. **Authentication:**
   * Sender authenticated via JWT.
2. **Validation:**
   * Ensure sender and receiver exist in the system.
   * Ensure message content is not empty.
3. **Persist Message:**
   * Save message in messages collection with fields:
     + messageId (unique)
     + senderId
     + receiverId
     + content (text or file URL)
     + timestamp
4. **Notification:**
   * Push notification is sent to the receiver.

**Outputs:**

* Success response confirming message is sent and stored.
* Message visible in chat history for both sender & receiver.

**REQ-CU-005 – Multi-Language Support**

**Inputs:** Language preference (e.g., 'en' for English, 'hi' for Hindi).

**Process :**

1. User sets a language preference in their profile or browser settings.
2. Server (or client) loads localized resource bundles for the selected language.
3. API responses that include UI strings are provided in the chosen language (or keys are returned and frontend renders text).
4. Save preference in user profile for subsequent sessions.
5. Fallback to English for missing translations.

**Outputs:** The user interface text and labels are rendered in the selected language.

**3.3 Worker Module**

**REQ-WO-001a – Add Skills to Worker Profile**

**Inputs:**

* Worker ID
* Skills (selected from existing skills in the system, e.g., Plumbing, Carpentry)

**Process:**

1. **Authentication:**
   * Worker authenticates via JWT.
2. **Skill Selection:**
   * Worker selects one or more skills from the predefined list in the system.
   * System validates that the selected skills exist in the master table.
3. **Database Update:**
   * Selected skills are linked to the worker profile in the database.

**Outputs:**

* Worker’s selected skills are updated in the database.
* Success message displayed.
* Invalid selections are rejected with an error message.

**REQ-WO-001b – Add Services to Worker Profile**

**Inputs:**

* Worker ID
* Skill ID (under which the service falls)
* Service ID (selected from existing services linked to the skill)
* Service Details:
  + Description (short detail of service)
  + Price (base cost or expected pricing/hourly rate)
  + Pricing type (fixed/hourly)
  + EstimatedDuration (expected time to complete the service)

**Process:**

1. **Authentication:**
   * Worker authenticates via JWT.
2. **Service Selection:**
   * Worker selects one or more services under their chosen skills.
   * System validates that selected services exist in the master table and match the selected skill.
3. **Service Details Update:**
   * Worker can update pricing and estimated duration.
   * Optional details can be added if applicable.
4. **Database Update:**
   * Selected services and their details are linked to the worker profile in the database.

**Outputs:**

* Worker’s selected services (with full details) are updated in the database.
* Success message displayed.
* Invalid selections or mismatched skill-service pairs are rejected with an error message.

**REQ-WO-002 – Availability Management**

**Inputs:**

* **Availability Status: Available, Busy, Off Duty**
* **Availability Schedule:**
  + **Recurring Weekly Slots (for Available status: day + start/end time)**
  + **One-time Custom Slots (specific dates/times, for Available or Off Duty)**
* **Optional: Special Exceptions (holidays, leave days, personal time)**

**Process:**

1. **Authenticated Update Request:**
   * **Worker is authenticated via JWT.**
   * **Request payload includes availability status and corresponding schedule/slots.**
2. **Validation:**
   * **Validate status against allowed set (Available, Busy, Off Duty).**
   * **For Available slots:**
     + **Check start < end, valid date/time, timezone-aware.**
     + **Ensure no overlap with existing bookings (Busy slots) or Off Duty periods.**
   * **For Busy slots:**
     + **Automatically derived from confirmed bookings; cannot be manually modified.**
   * **For Off Duty slots:**
     + **Check start < end, valid date/time, no conflicts with confirmed bookings (system may auto-reschedule or notify affected customers).**
3. **Database Update:**
   * **Store availability schedule in structured format:**
     + **Recurring Weekly Slots (Available)**
     + **Custom One-time Slots (Available / Off Duty)**
     + **Busy Slots (from confirmed bookings)**
   * **Update updatedAt timestamp.**
   * **Record audit logs for tracking all changes.**
4. **System Updates:**
   * **Search Visibility:**
     + **Available → worker visible in customer search, eligible for bookings.**
     + **Busy → worker temporarily hidden for booked slots.**
     + **Off Duty → worker hidden from search.**
   * **Booking Rules:**
     + **Customers can only book workers during Available slots.**
     + **System prevents bookings during Busy or Off Duty slots.**
5. **Optional Integrations:**
   * **Sync availability with external calendars (Google, Outlook) using OAuth.**
   * **Automatically block slots if external events conflict with worker’s schedule.**

**Outputs:**

* **Worker’s availability schedule is updated in the DB.**
* **Worker’s visibility in search and booking system reflects current status (Available, Busy, Off Duty).**
* **Success message displayed to the worker.**
* **Optional notification/summary sent to worker about updated availability.**

**REQ-WO-003 – Service Booking Requests**

**Inputs:**

* Booking ID (from pending booking requests)
* Action (Accept | Decline)
* Optional: Decline Reason (text)

**Process:**

1. **View Queue**
   * Worker (authenticated via JWT) retrieves pending booking requests assigned to them.
   * Results are paginated and sorted by booking time/requested date.
2. **Action Handling**
   * Worker selects a booking and performs an action:
     + **Accept**:
       - System validates booking is still in Pending state.
       - Updates booking status to Confirmed.
       - Sends confirmation notification to customer.
     + **Decline**:
       - System updates booking status to Declined.
       - Records optional decline reason.
       - Sends decline notification to customer.

**Outputs:**

* Booking status is updated (Confirmed or Declined).
* Worker’s availability schedule is updated accordingly (if accepted).
* Customer receives notification of the booking outcome.

**REQ-WO-004 – Service Portfolio / Gallery**

**Inputs:**

* Service ID
* Image Files
* Optional: Caption/Description for each image

**Process:**

1. **Upload Request**
   * Worker (authenticated via JWT) selects a **service** and uploads one or more images via UI (multipart form).
2. **Validation**
   * Validate selected service exists in worker’s profile.
   * Validate file type (allow JPG, PNG, WebP).
   * Enforce file size limit (e.g., max 5MB per image).
   * Validate image dimensions (e.g., min 400x400).
3. **Cloud Storage**
   * Images uploaded to Cloudinary using secure, signed requests.
   * Generate secure URLs.
4. **Portfolio Management**
   * Worker can:
     + Add new images under a specific service
     + Update captions
     + Delete images (server removes DB record + Cloudinary asset)
5. **Customer Display**
   * When customers view a service in the worker’s profile, service-specific images are shown.

**Outputs:**

* Images are uploaded to Cloudinary and linked to the corresponding service.
* Metadata saved in DB under the worker’s **service portfolio**.
* Customers see **service-specific galleries** when browsing worker services.
* Success message displayed after upload.

**REQ-W-005 – Multi-Language Support**

**Inputs:** Language preference.

**Process :** Same pattern as customer multi-language; worker UI text changes and preference saved in profile.

**Outputs:** The worker interface is rendered in the chosen language.

**3.4 Worker Verification Module**

**REQ-WV-001 – Document Upload & Verification**

**Inputs:**

* Aadhaar Card (image/PDF)
* Live Selfie (image, real-time capture or upload)
* Police Verification Certificate (image/PDF)

**Process:**

1. **Document Requirements**
   * **Aadhaar Card**: Clear front image or PDF showing name & Aadhaar number.
   * **Live Selfie**: Real-time selfie capture via camera or recent high-quality photo.
   * **Police Verification Certificate**: Clear scan of certificate issued by local police authority.
     + Serves as background clearance proof.
2. **Upload Flow**
   * Worker (authenticated via JWT) uploads.
   * Server performs validation:
     + Allowed file types: JPG, PNG, PDF
     + Enforce file size limits (e.g., max 10MB)
   * Upload documents to **Cloudinary.**
   * Update worker profile → verificationStatus = Pending.
   * Trigger notification to assigned **Service Agent** for review.

**Outputs:**

* Documents securely uploaded to Cloudinary with restricted access.
* Worker’s verification status set to **Pending**.
* Notification sent to assigned Service Agent/Admin for review.
* Success message displayed to worker.

**REQ-WV-002 – Verification Status**

**Inputs:** N/A (this is a system state).

**Process :**

1. System tracks verificationStatus field for each worker (Unverified, Pending, Verified, Rejected).
2. Status transitions:
   * Unverified → on document upload → Pending.
   * Pending → after Agent/Admin review → Verified or Rejected.
3. When Agent approves, server records approverId, approvalTimestamp and sets verifiedAt timestamp.
4. When rejected, server stores rejectionReason and notifies worker with steps for resubmission.

**Outputs:** The worker's profile status changes based on the review process. The possible outputs are: 'Unverified', 'Pending', 'Verified', or 'Rejected'.

**3.5 Booking Module**

**REQ-SB-001 – Worker Matching**

**Inputs:** Customer's Location, Requested Service Type, Customer's Booking Date and Time.

**Process :**

1. Validate requested time slot and service type.
2. Query available workers who:
   * Offer requested service type.
   * Are verified=true.
   * Have availability for requested date/time.
   * Are within acceptable distance from customer's location.
3. Rank results by relevance: proximity, rating, response time, price.

**Outputs:** A list of available and verified workers who are located near the customer and offer the requested service.

**REQ-SB-002 – Booking Workflow**

**Inputs:**

* **Customer**: Selected Service, Worker, Timeslot, Location (from profile or entered manually)
* **Worker**: Action (Accept | Decline)
* **System**: Booking ID (auto-generated)

**Process:**

1. **Booking Creation (Customer Action)**
   * Customer selects service, worker, location, and timeslot.
   * System validates input (service availability, timeslot format, location coverage).
   * Create booking with:
     + status = Pending
     + Auto-generated bookingId
   * Notify worker with booking details (service, customer location, contact).
2. **Worker Response**
   * **Accept**:
     + System re-validates worker availability for the timeslot.
     + Updates booking status = Confirmed.
     + Hard-locks the worker’s calendar for that timeslot.
     + Sends confirmation notification to customer.
   * **Decline**:
     + System updates booking status = Declined.
     + Records optional decline reason.
     + Notifies customer of decline.
3. **Service Execution**
   * **In Progress**:
     + Worker marks booking as In Progress when service starts.
   * **Completion**:
     + Capture actualAmount if different from estimate.
     + Customer confirmation required before closing booking.
4. **Cancellation & Refunds**
   * If customer/worker cancels **before service starts**:
     + Updates status = Cancelled.
   * If cancellation happens **after service start**:
     + Partial refund or penalty may apply.

**Outputs:**

* **Booking Creation**: New booking record with Pending status. Worker notified.
* **Worker Acceptance**: Booking updated to Confirmed. Customer notified.
* **Worker Decline**: Booking updated to Declined. Customer notified + optional alternate matches.
* **Completion**: Booking marked Completed, amount recorded, service closed.
* **Cancellation**: Booking updated to Cancelled, refunds/penalties processed if applicable.

**REQ-PM-001 – Payment Integration (Cash / Razorpay with OTP for Cash)**

**Inputs:**

* Payment Amount
* Customer Payment Details (e.g., credit card info, UPI ID) *(if Razorpay chosen)*
* Booking ID
* Payment Mode (Cash / Razorpay)

**Process:**

1. **Client Selects Payment Mode at Checkout:**
   * **Cash:** Customer chooses to pay directly in person.
   * **Razorpay:** Client creates payment request for bookingId and amount.
2. **If Razorpay:**
   * Server creates Razorpay order through Razorpay API (server-to-server), storing RazorpayOrderId and amount in DB.
   * Client uses Razorpay checkout or UPI flow to complete payment (tokens handled by Razorpay).
   * Razorpay sends webhook/callback to server on payment success/failure. Server validates webhook signature and updates payment record accordingly.
3. **If Cash:**
   * System generates a **random OTP** and sends it to the worker via SMS/email.
   * Payment record is created in DB with status = Pending (Cash).
   * Customer hands over cash to worker.
   * Worker provides OTP to customer.
   * Customer enters OTP in the app or system interface.
   * Server verifies OTP:
     + If valid → payment is marked **Paid (Cash)**, booking status updated, and transaction entries (worker payout, platform commission) are created.
     + If invalid → system rejects OTP entry, payment remains pending.
   * OTP expires after a defined time window for security.
4. **On Success (both modes):**
   * Booking payment status updated.
   * Transaction entries created (worker payout, platform commission).
   * Invoices and receipts triggered.

**Outputs:**

* **Razorpay:** A secure payment session is initiated.
* **Cash:** Pending payment record created; payment confirmed via OTP verification.
* **Completion:** Payment success/failure status returned to the Workjunction system.
* Notifications sent to customer and worker on successful payment.

**REQ-PM-003 – Payment Status Tracking**

**Inputs:** Payment ID from Razorpay.

**Process :**

1. On Razorpay webhook, server validates signature and maps Razorpay paymentId to local Payment record.
2. Update status to Unpaid, Processing, Paid, or Failed. Attach metadata (paymentMethod, payer details, timestamps).
3. Trigger downstream actions: release payout to worker, email receipt to customer, update booking status to Paid if payment required pre-service.

**Outputs:** The payment status in the database is updated to 'Unpaid', 'Processing', 'Paid', or 'Failed' based on the Razorpay callback.

**3.7 Admin and Service Agent Modules**

**Admin Module**

**REQ-AD-001 – Manage Service Agents**

**Inputs:**

* Service Agent Name (from registered agents)
* Assigned Area
* Status (Verified, Active, Suspended, Inactive)

**Process:**

1. **Fetch Registered Agent:**
   * Admin searches for an existing service agent by Service Agent Name.
   * System retrieves profile details (name, email, registration date, etc.).
2. **Assign Area (Admin Action):**
   * Admin assigns or reassigns a registered agent to a specific service area.
   * System validates the assignment rules (e.g., one area per agent, multiple allowed if policy permits).
3. **Verify Agent (Admin Action):**
   * Admin reviews submitted documents and registration details.
   * If approved, agent status changes to *Verified / Active*.
   * System notifies the agent of successful verification.
4. **Suspend / Reactivate Agent (Admin Action):**
   * Admin can suspend an agent (temporarily disabling their access and job assignments).
   * Reactivation restores access with the same assigned area and permissions.

**Outputs:**

* A registered Service Agent can be **verified, assigned to an area, suspended, or reactivated** by the Admin using their Service Agent ID.
* Agents are notified of verification, suspension, or reactivation.

**REQ-AD-003a – Add New Skill**

**Inputs:**

* Skill Name

**Process:**

1. Admin enters a new skill name.
2. System validates uniqueness of the skill name.
3. Database is updated with the new skill record.

**Outputs:**

* A new skill is created and added to the system.
* Customers can see the new skill in search filters and menus.
* Confirmation message is sent to Admin on successful skill creation.

**REQ-AD-003b – Add New Service**

**Inputs:**

* Name (Service Title)
* RequiredSkills (linked SkillId)

**Process:**

1. Admin selects a skill (via SkillId) under which the service will be created.
2. System validates uniqueness of the service name under the selected skill.
3. Database is updated with the new service record.

**Outputs:**

* A new service is created and linked to an existing skill.
* Updated search filters and menus reflect the new service.
* Confirmation message is sent to Admin on successful service creation.

**Service Agent Module**

**REQ-SA-001 – Worker Verification Handling**

**Inputs:**

* Worker ID
* Document URLs (Aadhaar, selfie, police certificate)
* Action (Approve / Reject)
* Rejection Reason (if applicable)

**Process:**

1. **Document Review:**
   * Service Agent reviews uploaded worker documents.
2. **Approval Flow:**
   * If Approve:
     + Set verificationStatus = Verified.
     + Update worker’s visibility in the system (eligible for assignments).
3. **Rejection Flow:**
   * If Reject:
     + Set verificationStatus = Rejected.
     + Store rejectionReason.
     + Notify worker with rejection reason and resubmission instructions.
4. **Periodic Re-Verification:**
   * System automatically triggers a **selfie recheck every 1 month**.
   * If the new selfie does not match or is not uploaded, worker status may revert to *Pending Verification* until reviewed again.

**Outputs:**

* Worker’s verification status is updated to **Verified** or **Rejected**.
* Worker receives a notification (approve/reject/recheck required).

**REQ-SA-002 – Worker Request Handling**

**Inputs:**

* Worker ID
* Booking ID
* Action (Accept / Decline)

**Process:**

1. **Authority Check:**
   * System verifies that the service agent is authorized to act on behalf of the worker for the given area.
2. **Request Handling:**
   * If Accept: Update booking status to *Accepted* and allocate the job to the worker.
   * If Decline: Update booking status to *Declined* and notify the customer.
3. **Notifications:**
   * Customer is notified of the worker’s decision (email).
   * Worker also receives a record of the action taken by the agent on their behalf.

**Outputs:**

* Booking request status is updated on behalf of the worker.
* Notifications are sent to customer and worker.

**REQ-SA-003 – Area Statistics Dashboard**

**Inputs:**

* N/A (system-generated from internal data)

**Process:**

1. **Data Aggregation:**
   * System aggregates metrics for the agent’s assigned area, including:
     + Number of verified workers
     + Number of completed bookings
     + Number of pending verifications
     + Active vs. inactive workers
2. **Dashboard Population:**
   * Data is displayed on the agent’s dashboard as:
     + Charts (bar, line, pie)
     + Lists (pending verifications, workers by status)
     + Alerts (threshold-based, e.g., “Verification backlog exceeded 50”).

**Outputs:**

* Agent dashboard displays key metrics for the assigned area.

**5. Worker Verification Workflow — expanded & process-oriented**

**High-level Steps:**

1. **Registration:** Worker registers → verificationStatus = Unverified.
2. **Document Upload:** Worker uploads Aadhaar, Live Selfie, Police Verification Certificate. (See REQ-WV-001 for details & data stored.)
3. **Notification:** System notifies assigned Service Agent and Admin (push + email).
4. **Review:**
   * **Agent Review:** Agent inspects documents, checks face-match, and confirms authenticity.
   * **Admin Oversight:** Admin may spot-check or review flagged cases (random or high-risk).
5. **Decision:**
   * **Approve →** Verified → worker becomes visible to customers and gets verification badge.
   * **Reject →** Rejected with rejectionReason. Worker gets message with steps for re-upload.
6. **Post-Verification:** On Verified, worker can appear in search; platform schedules payouts and rating history starts collecting.

**Document descriptions (more descriptive as requested):**

* **Aadhaar Card (Identity Document):** Must be a clear front-side image or PDF. System requires readable Aadhaar number and name. Used for identity verification and must be masked when displayed publicly.
* **Live Selfie (Liveness & Face Match):** Recent selfie taken within the app or uploaded with liveness proof. Used to compare against Aadhaar photo using face-match services to ensure the person is genuine. High face-match threshold required for auto-approval.
* **Police Verification Certificate (Background Check):** Issued by local police authority or authorized background-check partner. Must include issuing authority, validity date, and worker name. Used to confirm no criminal record.

**Resubmission & Appeals:**

* If documents rejected, worker receives clear rejectionReason and allowed to resubmit. For disputes, worker can appeal to Admin with extra documents or in-person verification.

**Audit & Retention:**

* All decisions logged with agentId/adminId, timestamps, and versioned document history. Retention policy states documents stored for X years (configurable). Access controlled and encrypted

**5. Non-Functional Requirements**

**5.1 Performance**

* 10,000 concurrent users, API < 2s response time.
* Search < 3s for 95% of cases.
* Payments < 5s excluding gateway delays.

**5.2 Scalability**

* Horizontal scaling on cloud infra.
* MongoDB sharding & replication.
* Load balancing across backend instances.

**5.3 Availability & Reliability**

* SLA: 99.9% uptime.
* Auto-recovery with Kubernetes.
* Daily backups (RTO = 2h, RPO = 15m).

**5.4 Security**

* TLS 1.2+, JWT-based auth.
* Password hashing with bcrypt/argon2.
* Document encryption at rest.
* RBAC enforced for all roles.
* Audit logs maintained.

**5.5 Usability**

* Responsive UI, multi-language support.
* Accessibility: WCAG 2.1 AA compliance.

**5.6 Maintainability**

* Modular code, 12-Factor principles.
* Automated CI/CD with ≥ 80% test coverage.

**5.7 Compliance & Privacy**

* Indian IT Act, Aadhaar handling guidelines.
* GDPR-style consent & data deletion policies.

**5.8 Disaster Recovery**

* Multi-zone deployment.
* Incident response plan with escalation SLAs.