# SupportHive Software Requirements Specification

## Purpose

This Software Requirements Specification (SRS) defines all functional and non-functional requirements for **SupportHive**, a web-based caregiving marketplace that connects verified caregivers with service seekers. The purpose of this document is to provide a complete description of the system’s intended behavior and quality attributes in accordance with ISO/IEC/IEEE 29148 (IEEE 830) standards [[1]](https://www.cin.ufpe.br/~in1020/docs/publicacoes/IEEE29148-srs_example.pdf#:~:text=The%20main%20purpose%20of%20this,2%20Scope). It serves as a reference for developers, testers, and stakeholders to ensure a common understanding of the system’s capabilities and constraints.

## Scope

SupportHive is a single-tenant web application built with a **Node.js**/Next.js backend, **a React frontend,** and a **MySQL** database (via XAMPP). It provides registration, authentication, profile management, service postings, bidding, booking, messaging, and admin management functions. In scope are all user-facing features (caregiver applications, service request workflows, bidding, booking, messaging, notifications) and admin console features (user and application management, dispute handling, content moderation, logging, and data export). Out of scope are external payment processing and third-party integrations beyond basic email/SMS notifications. The system is targeted at a general audience (“everyone”) and designed for compliance with GDPR, CCPA, and other relevant regulations; health data handling will follow HIPAA where applicable, and payment data handling will align with PCI-DSS.

## Product Overview

SupportHive enables **service seekers** to post caregiving requests (specifying type of care, description, location, budget, and date) and **caregivers** to find and bid on requests. Caregivers must apply and be verified by an administrator before bidding. Once a bid is accepted, a booking is created to track the scheduled service. Users can communicate via in-app messaging, and the system sends notifications (email/SMS/in-app) for events like bids, booking updates, and application status changes. Administrators have a console to approve/reject caregiver applications, manage users, moderate content, handle disputes, and export data for reporting. Audit logs record key system actions. The product must meet high standards of performance, reliability, security (OWASP ASVS-level controls), privacy (GDPR/CCPA), internationalization readiness, and accessibility (WCAG 2.2 AA). [[2]](https://owasp.org/Top10/A02_2021-Cryptographic_Failures/#:~:text=,all%20sensitive%20data%20at%20rest) [[3]](https://www.w3.org/TR/WCAG22/#:~:text=Web%20Content%20Accessibility%20Guidelines%20,laptops%2C%20kiosks%2C%20and%20mobile%20devices).

## Stakeholders and Roles

* **Service Seeker (Customer)**—An individual who requests caregiving services. Seekers can register, post service requests, view bids, accept or reject bids, manage bookings, and communicate with caregivers.
* **Caregiver (Provider)**—An individual offering caregiving services. Caregivers can register, submit an application (with documents) for verification, manage their profile, browse and search open requests, place bids with a proposed amount and message, and communicate with service seekers. Only verified caregivers may bid or be booked.
* **Administrator**—Internal operator managing the platform. Admin can review and approve/reject caregiver applications, manage user accounts (search, disable, reactivate), moderate content (flag or remove inappropriate requests/messages), resolve disputes, and export data (e.g., user list, request history, logs). Admin actions are audited.
* **System (Automated)**—Background services that send notifications (email/SMS) based on events (e.g., new bid, booking change) and enforce security (e.g., token generation, rate limiting).

## Definitions/Glossary

* **User**—Any person with a registered SupportHive account (caregiver, service seeker, or admin).
* **Service Request**—A posting by a service seeker describing needed care (type, description, location, budget, date). One seeker can create many requests.
* **Bid**—An offer by a caregiver on an open service request, including a bid amount and an optional message. Bids have statuses (pending, accepted, rejected).
* **Booking**—A confirmed engagement between a seeker and a caregiver for a specific request. On bid acceptance, one booking is created (status: scheduled, completed, or cancelled).
* **Caregiver Application** – A form submitted by a user wishing to become a verified caregiver, including identity documents and certificates. Applications have statuses (pending, approved, rejected).
* **Profile**—A user’s public information (photo, address, bio). Each user has one profile (1:1).
* **KYC (Know Your Customer)**—Identity verification process for caregivers (uploading national ID and documents) to prevent fraud [[4]](https://regulaforensics.com/blog/marketplace-identity-verification/#:~:text=KYC%20is%20not%20mandatory%20but,is%20often%20applied%20in%20ecommerce).
* **DSR (Data Subject Request)**—User requests under GDPR/CCPA (access, deletion, and correction of personal data).
* **MFA (Multi-Factor Authentication)**—Requiring two or more verification methods (e.g., password plus one-time code) for login.
* **WCAG 2.2 AA** – Web Content Accessibility Guidelines level AA (accessibility standard).
* **ASVS (Application Security Verification Standard)**—OWASP standard for security controls implementation (e.g., password hashing, CSRF protection).

## System Context

SupportHive is a cloud-hosted web application. Users access it via modern browsers (desktop or mobile) over HTTPS. The front end (React/Next.js) communicates with backend REST APIs (Node.js), which in turn query a MySQL database. The system integrates with an SMTP/email service for transactional emails (verification, notifications), an SMS gateway for optional SMS alerts, and (optionally) a third-party identity verification or document scanning service for caregiver vetting. If payments are enabled, a PCI-DSS compliant payment gateway (e.g., Stripe) will be used (card data is never stored on the platform). All network traffic is encrypted (TLS), and the database and backups are stored securely. Audit logs and backups are stored offsite. The context of external elements:  
- **User Devices:** browsers, mobile devices.  
- **External Services:** Email/SMS providers, KYC verification API, payment processor.  
- **Hosting Environment:** Node.js server, MySQL database, running on a secure host or cloud provider behind a firewall/HTTPS.

## Assumptions, Dependencies, and Constraints

* **Tech Stack:** The system uses MySQL (via XAMPP in development), Node.js/Express (backend), and React/Next.js (frontend). Development and deployment assume familiarity with these stacks.
* **Deployment:** Single-instance deployment (single-tenant). All users share the same database (no multi-tenant isolation).
* **Authentication:** Users authenticate via email/password (hashed with a strong algorithm) and optional MFA. Session managed by secure HTTP-only cookies or JWTs.
* **Third-Party Services:**
* Email/SMS services must be configured and reliable.
* An identity verification API is used for caregiver KYC checks (e.g., automated ID scanning) if available.
* For payment processing (if in scope), a PCI-compliant gateway is required.
* **Compliance:** The system must support GDPR/CCPA (data export, deletion on request) and prepare for HIPAA compliance if any health data is collected [[2]](https://owasp.org/Top10/A02_2021-Cryptographic_Failures/#:~:text=,all%20sensitive%20data%20at%20rest) [[5]](https://cppa.ca.gov/faq.html#:~:text=,subject%20to%20some%20exceptions). SOC 2 or ISO 27001 certification is recommended for enterprise clients [[6]](https://trustnetinc.com/resources/soc-2-best-practices-by-trustnet-an-expert-compliance-guide/#:~:text=SOC%202%20compliance%20ensures%20that,negotiable).
* **Scalability:** Designed for horizontal scaling (load-balanced node servers, separate read-replica DB if needed).
* **Availability:** Target SLA 99.9% (approx. 43 min downtime/month) [[7]](https://uptime.is/#:~:text=,Yearly%3A%208h%2045m%2057s).
* **Privacy:** Users’ personal data (PHI) is encrypted at rest and in transit (SSL/TLS). Data retention complies with legal requirements: unnecessary data is purged (or tokenized) promptly [[8]](https://owasp.org/Top10/A02_2021-Cryptographic_Failures/#:~:text=,not%20retained%20cannot%20be%20stolen).
* **Constraints:** Deployment on a platform that supports TLS, compute resource limits (see NFRs). All compliance features (privacy policy, cookie consent) must be implemented.

## Functional Requirements

* **FR-1: User Registration.** The system shall allow any new user to register by providing name, email, phone number, date of birth, and password. The email must be unique and in a valid format, and the phone number must be valid. A verification email with a time-limited token will be sent; unverified users cannot perform protected actions. Passwords must meet complexity rules (min. length 8, mix of characters).
* **FR-2: User Login/Logout.** The system shall allow registered users to log in using email (or phone) and password. Valid credentials create an authenticated session/token. Users can log out, which invalidates the session. After 15 minutes of inactivity, sessions expire.
* **FR-3: Password Reset.** The system shall allow a user to initiate password recovery by email. A secure reset link/token is emailed; using it allows the user to set a new password (once). Tokens expire after a configurable time. Proper error messages are returned for invalid/expired tokens.
* **FR-4: Profile Management.** Each user can view and edit their profile. A profile includes a profile photo (URL), address (free text, required), bio (optional text up to 1000 characters), and (for caregivers) national ID (string, required for KYC). Changes are saved atomically. Only the user (or admin) can update their profile.
* **FR-5: Caregiver Application Submission.** A user registering as a caregiver shall fill out an application form, submitting required documents: profile\_photo, national ID image, and any certificates (file upload). Upon submission, the application record (status=pending, timestamped) is created in caregiver\_applications. The profile’s is\_verified flag remains false until approved.
* **FR-6: Admin Review of Applications.** Administrators can retrieve all pending caregiver applications. They can approve or reject each application; when doing so, the application’s status is updated to approved or rejected, along with the reviewed\_at timestamp and admin\_id. Upon approval, the corresponding user’s profile\_verified is set to true. The system logs these events.
* **FR-7: Service Request Posting.** A verified service seeker can create a new service request by specifying service type (enumerated category), description (text up to 1000 chars), location (text or structured address), budget (decimal), and service date (date/time). The request is saved with status=open and timestamp created\_at. The requester may edit or cancel (status=cancelled) an open request. Once a booking is scheduled, the request status becomes accepted.
* **FR-8: Search/Filter/Sort Requests.** Caregivers can browse available (status=open) requests. They can search/filter by service\_type, keyword (full-text search on description/location), date range, budget range, or geographical proximity (if location geocoded). Results can be sorted by date, budget, or distance. Pagination (limit/offset) is supported. Only requests with status=open are listed for bidding.
* **FR-9: Bid Creation.** A verified caregiver can place a bid on an open service request by specifying bid\_amount and an optional message. A new bid record is created with status=pending and created\_at. If a user has already placed a pending bid on the same request, the system returns a conflict error.
* **FR-10: Bid Management.** The service seeker can view all pending bids on their request. They can accept a bid, which sets that bid’s status=accepted and automatically marks all other bids on the request as rejected. Conversely, the seeker can explicitly reject a specific bid (status=rejected). Accepting a bid triggers booking creation (see FR-11) and updates the related request to accepted.
* **FR-11: Booking Creation and Lifecycle.** Upon bid acceptance, the system shall create a booking record linking the request, caregiver, and service seeker, with status=scheduled, booking\_date (from request), and timestamp created\_at. The booking may later transition to completed (upon service fulfilment) or cancelled (if either party cancels before completion). Only one active (scheduled) booking may exist per request. Completion or cancellation updates the booking’s status with a timestamp. All status transitions are logged.
* **FR-12: In-App Messaging.** The system shall provide a messaging feature where users (seeker or caregiver) can send private messages to each other. Each message has a sender\_id, receiver\_id, an optional request\_id (nullable, linking the context), content (up to 2000 chars), a timestamp sent\_at, and an is\_read flag. Messages can only be viewed by the participants. When a message is delivered, is\_read=false; when the receiver views it, the flag is set true. Message threads can be fetched per request\_id or conversation (user-to-user). Deleting a service request sets related messages’ request\_id to NULL (message remains for record).
* **FR-13: Notifications and Preferences.** The system shall send notifications (email/SMS/in-app) for key events: new bid received, bid accepted/rejected, booking created/updated, application status change, incoming message, etc. Users can opt in/out of email vs SMS per event type via their notification preferences. Notification sending is asynchronous but guaranteed (with retry on failure). Every notification event is recorded (no direct user-visible data exposure).
* **FR-14: Admin Console – User Management.** Administrators can view a list of all users, filter by type/status, and take actions: disable or delete a user account. Deleting a user cascades or nullifies their data as per schema rules (see Domain Model). Disabled users cannot log in until re-enabled. All admin actions are audited.
* **FR-15: Admin Console – Dispute Handling.** The system shall allow admins to view flagged issues or disputes submitted by users (e.g., through a support form). Admins can record a resolution or escalate cases. (Dispute workflow is internal; no specific API is exposed.)
* **FR-16: Admin Console – Content Moderation.** Administrators can remove or edit inappropriate content: flag and delete service requests, bids, or messages that violate terms. Upon deletion of a request, bids and booking (if any) related to it are also cancelled, and messages have request\_id nulled. Such actions are logged.
* **FR-17: Admin Console – Data Export.** Administrators can export user data, caregiver applications, service requests, bids, bookings, and logs in CSV or JSON format for reporting and compliance audits. Exports respect data privacy (e.g., anonymise or exclude sensitive fields as needed).
* **FR-18: Audit Logging.** The system shall log all critical actions (user logins, application reviews, bid acceptances, profile updates, etc.) with user ID, action type, timestamp, and relevant metadata. Logs are write-once and append-only. Admins can query logs in the console.

## Non-Functional Requirements

* **NFR-1: Performance.** Under normal conditions, 95% of API requests shall respond within **200 ms** [[9]](https://www.geeksforgeeks.org/software-engineering/non-functional-requirements-in-software-engineering/#:~:text=,growing%20workload%20or%20user%20base). The system must sustain at least **100 requests/sec** peak load (configurable threshold). Throughput and latency targets are validated by load testing.
* **NFR-2: Scalability.** The architecture must support horizontal scaling: additional server instances and database replicas can be added without downtime. The system should handle a growing user base (e.g., up to 10,000 concurrent users) without performance degradation[[9]](https://www.geeksforgeeks.org/software-engineering/non-functional-requirements-in-software-engineering/#:~:text=,growing%20workload%20or%20user%20base).
* **NFR-3: Availability/SLA.** The service shall achieve ≥**99.9% uptime** (approximately **≤44 min downtime/month** [[7]](https://uptime.is/#:~:text=,Yearly%3A%208h%2045m%2057s)). Planned maintenance downtime must be announced in advance. Automated failover and health checks should minimize outages.
* **NFR-4: Reliability.** The system shall gracefully handle failures. Transactions (e.g., bid acceptance) must be atomic. In case of errors, partial failures do not corrupt the state. The system shall automatically retry transient failures (e.g., message delivery). Example target: “The system shall have an uptime of 99.99%” [[10]](https://www.geeksforgeeks.org/software-engineering/non-functional-requirements-in-software-engineering/#:~:text=2) (four nines reliability as a goal).
* **NFR-5: Backup/Restore.** Full database backups shall occur daily, with transaction log backups hourly. Backups are encrypted and stored offsite. **Recovery Point Objective (RPO)** is 24 hours, and **Recovery Time Objective (RTO)** is 4 hours. A restore procedure must be documented and tested quarterly.
* **NFR-6: Disaster Recovery.** In case of catastrophic failure, the system can be restored in a secondary region. The Recovery Plan includes DR drills annually. Critical data (user accounts, requests, bookings) must be recoverable.
* **NFR-7: Maintainability.** The codebase shall follow standard coding conventions, be modular and well-documented, and include automated tests. Version control (e.g., Git) and CI/CD pipelines shall enforce code quality. The system should be easy to update with minimal downtime. Maintainability is aided by modular design, as described in OWASP guidelines [[11]](https://www.geeksforgeeks.org/software-engineering/non-functional-requirements-in-software-engineering/#:~:text=Maintainability%20requirements%20describe%20the%20ease,easy%20to%20test%20and%20debug).
* **NFR-8: Observability.** The system shall emit **structured logs**, metrics, and tracing for all requests and background jobs. Use a standardized logging format (JSON) and unique request IDs. Collect metrics on user activity, system load, and error rates. Distributed tracing or correlation IDs should link related events (e.g., login → profile fetch). These “three pillars” (logs, metrics, and traces) support monitoring and debugging [[12]](https://www.ibm.com/think/insights/observability-pillars#:~:text=Three%20pillars%20of%20observability%3A%20Logs%2C,easier%20to%20visualize%20and%20understand).
* **NFR-9: Internationalization/Localization.** The system shall be designed for i18n: all user-facing text must be externalized into resource files. Initially, English is supported; UI should be locale-aware for date/time, number, and currency formats. Adding new languages must be possible without code changes.
* **NFR-10: Accessibility.** The user interface shall comply with **WCAG 2.2 AA** standards [[3]](https://www.w3.org/TR/WCAG22/#:~:text=Web%20Content%20Accessibility%20Guidelines%20,laptops%2C%20kiosks%2C%20and%20mobile%20devices). For example, all images have alt text, forms have labels, color contrast meets guidelines, and the site is operable via keyboard. This ensures accessibility for users with disabilities.
* **NFR-11: Privacy and Compliance.** The system shall comply with GDPR/CCPA. Personal data is collected only as needed and retained per policy; the default is data minimization. Users can download or delete their data (Data Subject Requests) in accordance with “Right to Know” and “Right to Delete” [[5]](https://cppa.ca.gov/faq.html#:~:text=,subject%20to%20some%20exceptions). A clear privacy policy and cookie consent notice must be presented. For health-related data, HIPAA technical safeguards (encryption, audit logs) apply [[13]](https://www.hipaavault.com/resources/what-makes-a-platform-hipaa-compliant/#:~:text=HIPAA%20compliance%20applies%20when%20a,for%20radiology%20all%20handle%20PHI). For any payment features, PCI-DSS controls (encryption of card data, tokenization) are enforced [[2]](https://owasp.org/Top10/A02_2021-Cryptographic_Failures/#:~:text=,all%20sensitive%20data%20at%20rest).
* **NFR-12: Security Controls (OWASP ASVS).** The system implements OWASP-level security:
* **Authentication:** Passwords stored with strong hashing (Argon2id) [[14]](https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#:~:text=To%20sum%20up%20our%20recommendations%3A); support Multi-Factor Authentication (e.g., SMS/email OTP). No default or weak passwords.
* **Session Management:** Use secure, HttpOnly cookies with the Secure flag (HTTPS-only) [[15]](https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html#:~:text=The%20,ID%20from%20web%20browser%20traffic). Implement SameSite=strict to mitigate CSRF [[16]](https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html#:~:text=See%20also%3A%20HttpOnly). Regenerate session tokens on privilege change and logout.
* **CSRF Protection:** State-changing endpoints require CSRF tokens or equivalent (if cookies are used) [[17]](https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html#:~:text=,consider%20%2038%20using%20custom). For JSON APIs with token auth, implement same-site cookies or double-submit.
* **XSS Protection:** Sanitize/encode all user-input rendered in UI. Content Security Policy (CSP) headers will be set [[18]](https://owasp-aasvs4.readthedocs.io/en/latest/14.4.3.html#:~:text=Verify%20that%20a%20content%20security,JSON%2C%20and%20JavaScript%20injection%20vulnerabilities). Use framework escaping (React escapes by default).
* **SQL Injection:** All database queries must use parameterized queries/prepared statements [[19]](https://cheatsheetseries.owasp.org/cheatsheets/Query_Parameterization_Cheat_Sheet.html#:~:text=SQL%20Injection%20is%20best%20prevented,queries%20within%20a%20web%20application). No concatenated SQL.
* **Rate Limiting/Brute Force:** Apply request throttling on authentication endpoints to prevent brute-force attacks [[20]](http://owasp-aasvs.readthedocs.io/en/latest/requirement-2.20.html#:~:text=2,attacks%C2%B6). Account lockout after repeated failed logins (e.g., 5 attempts). Globally rate-limit the API to mitigate abuse (e.g., 100 req/min per IP).
* **Content Security Policy:** Serve a strict CSP to mitigate XSS (e.g., disallow inline scripts, allow only trusted sources) [[18]](https://owasp-aasvs4.readthedocs.io/en/latest/14.4.3.html#:~:text=Verify%20that%20a%20content%20security,JSON%2C%20and%20JavaScript%20injection%20vulnerabilities).
* **Encryption:** Encrypt sensitive data at rest (database encryption for PII/PHI) and use TLS for all network traffic [[2]](https://owasp.org/Top10/A02_2021-Cryptographic_Failures/#:~:text=,all%20sensitive%20data%20at%20rest). Use forward secrecy ciphers and HTTP Strict Transport Security (HSTS).
* **Key Management:** Encryption keys are stored securely (not in code) and rotated per policy.
* **Least-Privilege RBAC:** Database and service accounts have minimal privileges. Within the application, enforce role-based access control (admin vs. user actions).
* **NFR-13: Fraud/Abuse Prevention.** Integrate identity verification (KYC) for caregiver applications [[21]](https://regulaforensics.com/blog/marketplace-identity-verification/#:~:text=An%20identity%20verification%20process%20with,to%20prevent%20online%20marketplace%20fraud), including document and (optionally) biometric checks. Implement duplicate-account detection (e.g., block re-registration with the same SSN) [[22]](https://kycaid.com/duplicates/#:~:text=Multiple%20Account%20Detection). Monitor and rate-limit suspicious activities.
* **NFR-14: Data Retention and Deletion.** Define retention policies for user data and logs. By default, delete (or anonymize) inactive user accounts after a threshold (e.g., 2 years). Archive bookings after completion. All deletions (e.g., account closure) trigger cascade actions per schema rules. Maintain a deletion log for audit.

## Domain Model (Data Entities and Relationships)

The system’s data model maps to the following tables and fields, with relationships and cardinalities:

* **users**(user\_id, name, email, phone\_number, date\_of\_birth, password\_hash, user\_type, created\_at) – Main entity for all users.
* **PK:** user\_id (INT, AUTO\_INCREMENT).
* **Constraints:** email UNIQUE, phone\_number UNIQUE (nullable). user\_type is ENUM('caregiver','service\_seeker','admin'). All fields are NOT NULL except phone\_number. created\_at defaults to the current timestamp.
* **Business rules:** Only admin or system code can set user\_type=admin. Password hash stored as Argon2 hash [[14]](https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#:~:text=To%20sum%20up%20our%20recommendations%3A). On deletion (ON DELETE CASCADE), remove linked profile, applications, requests, bids, bookings, and messages.
* **Relationships:** One-to-one with **profiles** (each user has one profile). One-to-many with **service\_requests** (if seeker), **bids** (if caregiver), **bookings** (as caregiver or seeker), **messages** (as sender or receiver).
* **profiles**(profile\_id, user\_id, profile\_photo, address, bio, national\_id, is\_verified) – Extended user info.
* **PK:** profile\_id. **FK:** user\_id → users.user\_id (ON DELETE CASCADE). Unique index on user\_id (1:1 relationship).
* **Fields:** profile\_photo (VARCHAR URL), address (VARCHAR, NOT NULL), bio (TEXT, nullable), national\_id (VARCHAR, NOT NULL for caregivers), is\_verified (BOOLEAN, default FALSE).
* **Business rules:** is\_verified is set to true only when the admin approves the application (FR-6). Only caregivers have a national\_id; for seekers, it may be null.
* **caregiver\_applications**(application\_id, user\_id, profile\_photo, national\_id, certificates, status, submitted\_at, reviewed\_at, admin\_id) – Caregiver vetting records.
* **PK:** application\_id. **FKs:** user\_id → users (ON DELETE CASCADE), admin\_id → users (nullable, references admin who reviewed).
* **Fields:** profile\_photo, national\_id, certificates (e.g., URLs or BLOBs of documents), status ENUM('pending', 'approved', 'rejected') default 'pending', timestamps.
* **Business rules:** A user may have multiple applications over time, but at most one pending at a time. Status transitions: pending → approved or pending → rejected. On approval, set profiles.is\_verified.
* **service\_requests**(request\_id, user\_id, service\_type, description, location, budget, service\_date, status, created\_at) – Posts by service seekers.
* **PK:** request\_id. **FK:** user\_id → users (ON DELETE CASCADE).
* **Fields:** service\_type ENUM (e.g., 'childcare', 'eldercare', 'medical', 'disability'), description TEXT, location (VARCHAR, e.g., address or city), budget DECIMAL(10,2), service\_date DATETIME, status ENUM('open', 'accepted', 'completed', 'cancelled') default 'open'.
* **Relationships:** One request has many bids. One request can lead to one booking (when accepted). On delete request: cascade delete bids and bookings, and set request\_id in messages to NULL (retain message content).
* **Status state machine:** open → accepted (when a bid is accepted, FR-11) → (completed or cancelled). A seeker or admin can cancel an open or accepted request (resulting in cancelled). "Completed" means service was fulfilled.
* **Bids** (bid\_id, request\_id, caregiver\_id, bid\_amount, message, status, created\_at) – Offers by caregivers.
* **PK:** bid\_id. **FKs:** request\_id → service\_requests (ON DELETE CASCADE), caregiver\_id → users (ON DELETE CASCADE).
* **Fields:** bid\_amount DECIMAL, message TEXT, status ENUM('pending', 'accepted', 'rejected') default 'pending'.
* **Business rules:** Only caregivers with profiles where is\_verified=TRUE can place bids. On bid acceptance: that bid.status=accepted; all other pending bids for the request become rejected. If a bid is accepted, create a booking (FR-11).
* **State machine:** pending → accepted or rejected. Once accepted, immutable.
* **Bookings** (booking\_id, request\_id, caregiver\_id, service\_seeker\_id, status, booking\_date, created\_at) – Confirmed engagements.
* **PK:** booking\_id. **FKs:** request\_id → service\_requests (ON DELETE CASCADE), caregiver\_id → users (ON DELETE CASCADE), service\_seeker\_id → users (ON DELETE CASCADE).
* **Fields:** status ENUM ('scheduled', 'completed', 'cancelled') default 'scheduled'; booking\_date DATETIME (usually the same as the service\_date of the request); created\_at timestamp.
* **Business rules:** On creation, status=scheduled. Only one booking per request. Status transitions: scheduled → completed or cancelled. After being completed or cancelled, the booking is final. If a booking is cancelled, mark the request. canceled.
* **messages**(message\_id, sender\_id, receiver\_id, request\_id, content, sent\_at, is\_read) – In-app chat messages.
* **PK:** message\_id. **FKs:** sender\_id, receiver\_id → users (ON DELETE CASCADE), request\_id → service\_requests (ON DELETE SET NULL).
* **Fields:** content TEXT, sent\_at TIMESTAMP, is\_read BOOLEAN default FALSE.
* **Business rules:** Messages may optionally reference a request (for context). If a request is deleted, its messages’ request\_id is set to NULL [[21]](https://regulaforensics.com/blog/marketplace-identity-verification/#:~:text=An%20identity%20verification%20process%20with,to%20prevent%20online%20marketplace%20fraud). State machine: is\_read is false by default; when the receiver reads it, set it to true.

**Field Validation Constraints:**  
- Emails must match RFC format and be ≤255 chars. Phone numbers use E.164 regex (e.g., “+\d{10,15}”).  
- Names: 1–100 chars (letters, spaces).  
- Addresses: 1–200 chars.  
- Password hash: stored as a CHAR(60–100) string (hashed). Raw passwords are not stored.  
- service\_type and status enums allow only defined values.  
- budget and bid\_amount: non-negative, two decimal places, ≤10000 (configurable max).  
- Description and message text are limited to 1000–2000 chars.  
- location: if text, 1–100 chars (we may later use geocoding).  
- Timestamps (created\_at, submitted\_at, sent\_at) default to UTC now.  
- Nullability: only explicitly nullable fields (bio, phone, profile\_photo, request\_id in messages, reviewed\_at, and admin\_id) can be NULL; all others are required.

**Business Rules (Cross-field/Invariants):**  
- Only users with profiles.is\_verified = TRUE (and an approved application) can place bids or accept jobs.  
- Accepting a bid creates exactly one booking and sets the request.status=accepted. Attempting to accept multiple bids for the same request is prevented by a database constraint/transaction.  
- If a user (caregiver or seeker) is deleted, ON DELETE rules apply: their profile, applications, bids, bookings, and messages are also deleted or nullified per schema. For example, deleting a service request nullifies the request\_id in messages.  
- **Concurrency:** Bid acceptance must be atomic: if two caregivers attempt to accept at the same time, transactions and locks ensure only one booking is created, and the others fail with conflict. Booking creation on bid acceptance is idempotent: if the same request is already accepted, additional accept calls have no effect (error or ignored).

**Error Handling:** All API errors return JSON with a consistent structure { code: <numeric>, message: <string>, details?: <object> }. Common error codes:  
- 400 Bad Request—invalid input (e.g., malformed JSON, validation error).  
- 401 Unauthorized – missing/invalid auth.  
- 403 Forbidden – valid auth but insufficient permissions.  
- 404 Not Found – requested resource or endpoint does not exist.  
- 409 Conflict – conflicting operation (e.g., duplicate email, bid on closed request).  
- 422 Unprocessable Entity—semantic errors (e.g., bid on non-open request).  
- 500 Internal Error – unexpected server error (should not reveal details).

## API Specification

Version the API as v1. All endpoints use JSON over HTTPS. Use Bearer tokens (JWT) for authentication (except registration/login). Example endpoints:

* **User Authentication & Profile:**
* POST /v1/users – Register a new user.
  + *Request:* { "name":"Alice","email":"alice@example.com","phone\_number":"+15551234567","date\_of\_birth":"1980-05-20","password":"Secret123!" }
  + *Response:* 201 Created { "user\_id":123, "message":"Registration successful. Verify your email." }.
* POST /v1/auth/login – Log in.
  + *Request:* { "email":"alice@example.com","password":"Secret123!" }
  + *Response:* 200 OK { "token":"<JWT>", "user":{"id":123,"name":"Alice"} }. (Token sent as Authorization: Bearer <token> on future calls.)
* POST /v1/auth/logout – Log out (invalidate token).
* POST /v1/auth/password-reset – Request password reset email. *Request:* { "email":"alice@example.com" }.
* POST /v1/auth/password-reset/confirm – Confirm reset. *Request:* { "token":"<resetToken>","new\_password":"NewPass456!" }.
* GET /v1/profile – Get own profile. *Response:* { "user\_id":123,"name":"Alice","email":"alice@example.com","profile\_photo":"/img/a.png","address":"123 Main St","bio":"Hello","is\_verified":false, ... }.
* PUT /v1/profile – Update profile. *Request:* { "address":"456 Elm St","bio":"Experienced caregiver","phone\_number":"+15557654321" }.
* **Caregiver Applications:**
* POST /v1/caregiver-applications – Submit application. *Request:* { "profile\_photo":"<file>","national\_id":"<file>","certificates":"<file>" }.  
  *Response:* 201 Created { "application\_id":456, "status":"pending" }.
* GET /v1/admin/caregiver-applications?status=pending – (Admin) List pending applications.
* POST /v1/admin/caregiver-applications/{id}/approve – (Admin) Approve application.
* POST /v1/admin/caregiver-applications/{id}/reject – (Admin) Reject application.
* **Service Requests:**
* POST /v1/requests – Create service request.
  + *Request:*
  + {   
     "service\_type":"eldercare",   
     "description":"Help with daily living activities",   
     "location":"Boston, MA",   
     "budget":25.00,   
     "service\_date":"2025-09-15T10:00:00Z"   
    }
  + *Response:* 201 Created { "request\_id":789, "status":"open" }.
* GET /v1/requests?status=open&service\_type=eldercare&min\_budget=20&location=Boston&page=2&limit=20 – Search/filter open requests (supports pagination, filters, sorting).
* GET /v1/requests/{id} – Get details of a request.
* PUT /v1/requests/{id} – Update an open request (seeker only).
* DELETE /v1/requests/{id} – Cancel a request (sets status=cancelled).
* **Bidding:**
* POST /v1/requests/{id}/bids – Place a bid on request {id}.
  + *Request:* { "bid\_amount":30.00, "message":"I have 10 years of experience." }.
  + *Response:* 201 Created { "bid\_id":101, "status":"pending" }.
* GET /v1/requests/{id}/bids – List bids on a request (seeker only).
* POST /v1/bids/{bid\_id}/accept – (Seeker) Accept the bid. *Response:* 200 OK { "booking\_id":202 }.
* POST /v1/bids/{bid\_id}/reject – (Seeker) Reject a bid.
* **Bookings:**
* GET /v1/bookings – List user’s bookings (caregiver or seeker). Supports filter by status/date.
* GET /v1/bookings/{id} – Booking details.
* POST /v1/bookings/{id}/complete – Mark a scheduled booking as completed (caregiver).
* POST /v1/bookings/{id}/cancel – Cancel a scheduled booking (seeker or caregiver).
* **Messaging:**
* POST /v1/messages – Send a message. *Request:* { "receiver\_id":234, "request\_id":789, "content":"When should I arrive?" }.
* GET /v1/messages?conversation\_with=234&request\_id=789 – Fetch thread of messages between current user and user 234 for request 789.
* POST /v1/messages/{id}/read – Mark message as read (receiver).
* **Notifications:**
* GET /v1/notifications – List current user’s notifications.
* POST /v1/notifications/{id}/read – Mark notification as read.
* **Admin Actions:**
* Endpoints as above (e.g. /admin/...). Additionally:
  + GET /v1/admin/users – List/filter all users (search by name, type).
  + POST /v1/admin/users/{id}/disable – Disable a user login.
  + POST /v1/admin/users/{id}/delete – Delete a user account (cascades data).
  + GET /v1/admin/flags – List flagged requests or messages for moderation.
  + GET /v1/admin/logs – Search audit logs.

**Authentication:** All endpoints (except registration/login/password-reset) require a valid JWT in Authorization: Bearer <token>. Admin endpoints require the user\_type='admin'.

**JSON Schemas:** (Representative examples)  
- *User Registration:* {"name": "string","email": "string","phone\_number": "string","date\_of\_birth": "YYYY-MM-DD","password": "string"}.  
- *Create Request:* fields as shown above.  
- *Pagination:* GET /v1/requests?page=1&limit=20 returns { "data": [...], "page":1, "total\_pages":5 }.

## Data Model DDL (MySQL Schema)

Representative SQL table definitions (with sensible indexes/constraints):

CREATE TABLE users (  
 user\_id INT PRIMARY KEY AUTO\_INCREMENT,  
 name VARCHAR(100) NOT NULL,  
 email VARCHAR(255) NOT NULL UNIQUE,  
 phone\_number VARCHAR(15) UNIQUE,  
 date\_of\_birth DATE NOT NULL,  
 password\_hash CHAR(100) NOT NULL,  
 user\_type ENUM('caregiver','service\_seeker','admin') NOT NULL DEFAULT 'service\_seeker',  
 created\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP  
);  
CREATE INDEX idx\_users\_email ON users(email);  
  
CREATE TABLE profiles (  
 profile\_id INT PRIMARY KEY AUTO\_INCREMENT,  
 user\_id INT NOT NULL UNIQUE,  
 profile\_photo VARCHAR(255),  
 address VARCHAR(200) NOT NULL,  
 bio TEXT,  
 national\_id VARCHAR(50),  
 is\_verified BOOLEAN NOT NULL DEFAULT FALSE,  
 FOREIGN KEY (user\_id) REFERENCES users(user\_id) ON DELETE CASCADE  
);  
  
CREATE TABLE caregiver\_applications (  
 application\_id INT PRIMARY KEY AUTO\_INCREMENT,  
 user\_id INT NOT NULL,  
 profile\_photo VARCHAR(255),  
 national\_id VARCHAR(50),  
 certificates TEXT,  
 status ENUM('pending','approved','rejected') NOT NULL DEFAULT 'pending',  
 submitted\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,  
 reviewed\_at TIMESTAMP NULL,  
 admin\_id INT,  
 FOREIGN KEY (user\_id) REFERENCES users(user\_id) ON DELETE CASCADE,  
 FOREIGN KEY (admin\_id) REFERENCES users(user\_id) -- no cascade  
);  
  
CREATE TABLE service\_requests (  
 request\_id INT PRIMARY KEY AUTO\_INCREMENT,  
 user\_id INT NOT NULL,  
 service\_type ENUM('childcare','eldercare','disability','medical','companion') NOT NULL,  
 description TEXT NOT NULL,  
 location VARCHAR(100) NOT NULL,  
 budget DECIMAL(10,2) NOT NULL,  
 service\_date DATETIME NOT NULL,  
 status ENUM('open','accepted','completed','cancelled') NOT NULL DEFAULT 'open',  
 created\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,  
 FOREIGN KEY (user\_id) REFERENCES users(user\_id) ON DELETE CASCADE,  
 INDEX idx\_requests\_status (status),  
 FULLTEXT KEY idx\_requests\_search (description, location)  
);  
  
CREATE TABLE bids (  
 bid\_id INT PRIMARY KEY AUTO\_INCREMENT,  
 request\_id INT NOT NULL,  
 caregiver\_id INT NOT NULL,  
 bid\_amount DECIMAL(10,2) NOT NULL,  
 message TEXT,  
 status ENUM('pending','accepted','rejected') NOT NULL DEFAULT 'pending',  
 created\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,  
 FOREIGN KEY (request\_id) REFERENCES service\_requests(request\_id) ON DELETE CASCADE,  
 FOREIGN KEY (caregiver\_id) REFERENCES users(user\_id) ON DELETE CASCADE  
);  
  
CREATE TABLE bookings (  
 booking\_id INT PRIMARY KEY AUTO\_INCREMENT,  
 request\_id INT NOT NULL UNIQUE,  
 caregiver\_id INT NOT NULL,  
 service\_seeker\_id INT NOT NULL,  
 status ENUM('scheduled','completed','cancelled') NOT NULL DEFAULT 'scheduled',  
 booking\_date DATETIME NOT NULL,  
 created\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,  
 FOREIGN KEY (request\_id) REFERENCES service\_requests(request\_id) ON DELETE CASCADE,  
 FOREIGN KEY (caregiver\_id) REFERENCES users(user\_id) ON DELETE CASCADE,  
 FOREIGN KEY (service\_seeker\_id) REFERENCES users(user\_id) ON DELETE CASCADE  
);  
  
CREATE TABLE messages (  
 message\_id INT PRIMARY KEY AUTO\_INCREMENT,  
 sender\_id INT NOT NULL,  
 receiver\_id INT NOT NULL,  
 request\_id INT NULL,  
 content TEXT NOT NULL,  
 sent\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,  
 is\_read BOOLEAN NOT NULL DEFAULT FALSE,  
 FOREIGN KEY (sender\_id) REFERENCES users(user\_id) ON DELETE CASCADE,  
 FOREIGN KEY (receiver\_id) REFERENCES users(user\_id) ON DELETE CASCADE,  
 FOREIGN KEY (request\_id) REFERENCES service\_requests(request\_id) ON DELETE SET NULL  
);

**Notes:** Added indexes (e.g., on emails, status, and a FULLTEXT index on requests for search). Unique constraints: users.email, profiles.user\_id, bookings.request\_id. Foreign keys enforce integrity; messages.request\_id is nullable with ON DELETE SET NULL as required (deleted requests don’t delete messages, just drop the request link).

**Seed Data Examples:**

INSERT INTO users (name,email,phone\_number,date\_of\_birth,password\_hash,user\_type) VALUES  
('Alice','alice@example.com','+15551230001','1985-04-12','<argon2-hash>','service\_seeker'),  
('Bob','bob@example.com','+15551230002','1990-08-22','<argon2-hash>','caregiver'),  
('Carol','carol@example.com','+15551230003','1975-11-30','<argon2-hash>','admin');  
  
INSERT INTO profiles (user\_id,address,bio,national\_id,is\_verified) VALUES  
(1,'123 Maple St, Boston','Looking for a caregiver for my mother',NULL,false),  
(2,'456 Oak Ave, Boston','Licensed RN, 10 years experience','ID987654',true);  
  
INSERT INTO caregiver\_applications (user\_id,profile\_photo,national\_id,certificates,status,submitted\_at) VALUES  
(2,'/uploads/bob.jpg','ID987654','/uploads/bob\_cert.pdf','approved','2025-06-01 09:00:00');  
  
INSERT INTO service\_requests (user\_id,service\_type,description,location,budget,service\_date,status) VALUES  
(1,'eldercare','Daily home visits for medication and meals','Boston',35.00,'2025-09-10 10:00:00','open');  
  
INSERT INTO bids (request\_id,caregiver\_id,bid\_amount,message,status) VALUES  
(1,2,30.00,'I am available weekdays morning.', 'pending');  
  
-- Assume request\_id=1, bid\_id=1  
INSERT INTO bookings (request\_id,caregiver\_id,service\_seeker\_id,status,booking\_date) VALUES  
(1,2,1,'scheduled','2025-09-10 10:00:00');  
  
INSERT INTO messages (sender\_id,receiver\_id,request\_id,content) VALUES  
(1,2,1,'What time shall I arrive?'),  
(2,1,1,'Please come at 9am.');

**Sample Queries:**  
- *Open requests near a location:*

SELECT \* FROM service\_requests   
WHERE status='open'   
 AND MATCH(description,location) AGAINST('+Boston' IN BOOLEAN MODE)  
ORDER BY service\_date ASC  
LIMIT 20;

(Alternatively, use GIS if lat/long were stored).  
- *Pending bids for a request:*

SELECT b.bid\_id,b.bid\_amount,b.message,u.name AS caregiver\_name   
FROM bids b JOIN users u ON b.caregiver\_id=u.user\_id  
WHERE b.request\_id=1 AND b.status='pending';

- *Unread message count per conversation:*

SELECT sender\_id, COUNT(\*) AS unread\_count  
FROM messages   
WHERE receiver\_id=? AND is\_read=FALSE  
GROUP BY sender\_id;

## User Stories (Gherkin-style Acceptance Criteria)

Feature: User Registration  
 As a new visitor,  
 I want to register with my email and password  
 So that I can log in to the SupportHive  
  
 Scenario: Successful registration  
 Given I am on the registration page  
 When I submit valid user details (name, email, phone, DOB, password)  
 Then a new user account is created with status "unverified"  
 And I receive a verification email  
  
Feature: Caregiver Application  
 As a logged-in user who wants to be a caregiver  
 I want to submit an application with my ID and certificates  
 So that an admin can review my eligibility  
  
 Scenario: Submit caregiver application  
 Given that I am a verified user,  
 When I POST /v1/caregiver-applications with the required documents  
 Then a caregiver\_application record is created with status "pending"  
 And I see "Application submitted" response.  
  
Feature: Service Request Posting  
 As a service seeker  
 I want to create a service request  
 So that caregivers can view and bid on it  
  
 Scenario: Post a new service request  
 Given I am logged in as a verified service seeker  
 When I POST /v1/requests with valid request data (type, description, etc.)  
 Then a service\_request is created with status "open"  
 And I see the request\_id in response  
  
Feature: Bid Placement  
 As a caregiver  
 I want to bid on an open request  
 So that the service seeker can hire me  
  
 Scenario: Place a bid  
 Given there is an open request with id 100  
 And I am logged in as a verified caregiver  
 When I POST /v1/requests/100/bids with bid amount and message  
 Then a new bid is created with status "pending"  
 And I see the bid\_id in response  
  
Feature: Bid Acceptance and Booking  
 As a service seeker  
 I want to accept a caregiver’s bid  
 So that a booking is created  
  
 Scenario: Accept a bid  
 Given a request has a pending bid with id 200  
 When I POST /v1/bids/200/accept as the request owner  
 Then that bid’s status changes to "accepted"  
 And all other pending bids are "rejected"  
 And a booking is created with status "scheduled"  
  
Feature: In-App Messaging  
 As a user (seeker or caregiver)  
 I want to send messages related to a request  
 So that I can discuss details  
  
 Scenario: Send and read messages  
 Given a booking exists between user A and user B  
 When A sends a message to B with request\_id  
 Then a message record is created with is\_read=false  
 And B can fetch the message and mark it as read  
 And the is\_read flag updates to true  
  
Feature: Admin Approves Caregiver  
 As an admin  
 I want to approve or reject caregiver applications  
 So that only qualified caregivers are verified  
  
 Scenario: Approve an application  
 Given a pending caregiver\_application with id 300  
 When I POST /v1/admin/caregiver-applications/300/approve  
 Then the application status becomes "approved"  
 And the corresponding user's profile.is\_verified is set to true

(*Additional stories would cover login/logout, password reset, search/filter requests, user profile edits, notification preferences, dispute resolution, etc.*)

## Testing Plan

* **Unit Tests:** Each backend function and component (e.g., user service, bid service) will have unit tests covering success and failure cases (e.g., validation errors). Use frameworks like Jest or Mocha. Test password hashing logic and role-based access.
* **Integration Tests:** Test REST endpoints end-to-end (e.g., using Supertest). Verify sequences like registration→login→post request→bid→booking. Include authentication/authorization flows.
* **End-to-End (E2E) Tests:** Automate UI flows (e.g., Cypress) for key user journeys: new user signup, caregiver application, posting/searching requests, bidding, and messaging.
* **Performance Tests:** Load testing (e.g., JMeter or k6) to verify response time requirements (NFR-1). Scale to concurrent user targets.
* **Security Tests:**
* **Static Code Analysis:** Run SAST tool (ESLint security rules, NodeJS analyzers).
* **Dependency Scanning:** Check for known vulnerabilities (e.g., npm audit).
* **Dynamic Security Testing:** Use OWASP ZAP or Burp to scan the running app for OWASP Top 10 vulnerabilities (SQLi, XSS, CSRF, etc.). Ensure all critical issues are addressed.
* **Authentication Hardening:** Test brute-force lockout and session expiry.
* **Penetration Test:** Engage an external pentester or security firm before launch.
* **Compliance Tests:** Verify GDPR/CCPA flows: test data export (DSR), data deletion, cookie consent banner. For HIPAA, run a compliance checklist (encryption, audit logs, etc.).
* **Accessibility Tests:** Use automated tools (e.g., axe) and manual checks to confirm WCAG 2.2 AA criteria are met.

## Traceability Matrix

| **Requirement** | **User Story / Use Case** | **API Endpoint** | **DB Tables/Fields** |
| --- | --- | --- | --- |
| FR-1 | User Registration | POST /v1/users | users(email, password\_hash, ...) |
| FR-2 | User Login/Logout | POST /v1/auth/login | users(email,password\_hash) |
| FR-3 | Password Reset | POST /v1/auth/password-reset | users(email) + reset\_token store |
| FR-4 | Profile View/Edit | GET/PUT /v1/profile | profiles(address,bio,...) |
| FR-5 | Submit Caregiver Application | POST /v1/caregiver-applications | caregiver\_applications(...) |
| FR-6 | Admin Review Application | POST /v1/admin/caregiver-applications/{id}/approve | caregiver\_applications(status, reviewed\_at) |
| FR-7 | Post Service Request | POST /v1/requests | service\_requests(...) |
| FR-8 | Search/Filter Requests | GET /v1/requests?filters... | service\_requests(status, service\_type, location, budget) |
| FR-9 | Place Bid | POST /v1/requests/{id}/bids | bids(request\_id, caregiver\_id, bid\_amount) |
| FR-10 | Accept/Reject Bid | POST /v1/bids/{id}/accept | bids(status), service\_requests(status), bookings(...) |
| FR-11 | Booking Lifecycle | GET/POST /v1/bookings/... | bookings(status, booking\_date) |
| FR-12 | In-App Messaging | POST/GET /v1/messages | messages(sender\_id,receiver\_id,is\_read) |
| FR-13 | Notifications | GET/POST /v1/notifications | (backend triggers, not stored per-user) |
| FR-14 | Admin User Management | GET /v1/admin/users, POST /v1/admin/users/{id}/disable | users(user\_id, user\_type) |
| FR-15 | Admin Dispute Handling | *Use case management (no direct API)* | n/a |
| FR-16 | Content Moderation | DELETE /v1/requests/{id}, DELETE /v1/messages/{id} | service\_requests, messages |
| FR-17 | Data Export | GET /v1/admin/export?type=users | All tables as per type |
| FR-18 | Audit Logging | *System logging service* | audit\_logs (implicit in system) |

## Key Performance Indicators (KPIs) & Metrics

* **Fill Rate:** Percentage of posted service requests that result in a booking. (High fill rate indicates good matching.)
* **Time-to-Approval:** Average time (hours) from caregiver application submission to admin decision. (Measure admin responsiveness.)
* **Bid-to-Booking Conversion:** Ratio of accepted bids to total bids. (Higher means seekers find suitable bids quickly.)
* **Dispute Rate:** Percentage of bookings that result in a formal dispute. (Measure quality/fit of matches.)
* **User Growth/Activation:** Number of new registered users per week and percent completing profile/application.
* **Retention:** Rate of repeat bookings or repeat bids per user.

## Analytics Events

The application will emit analytics events (to a tool like Segment/GA):  
- UserRegistered: { user\_id, timestamp }  
- LoginSuccess / LoginFailure: { user\_id/email, timestamp }  
- CaregiverApplicationSubmitted: { application\_id, user\_id, timestamp }  
- ServiceRequestCreated: { request\_id, service\_type, timestamp }  
- BidPlaced: { bid\_id, request\_id, caregiver\_id, bid\_amount, timestamp }  
- BidAccepted: { bid\_id, request\_id, timestamp }  
- BookingCreated: { booking\_id, request\_id, caregiver\_id, seeker\_id, timestamp }  
- MessageSent: { message\_id, sender\_id, receiver\_id, request\_id?, timestamp }  
- NotificationSent: { type, user\_id, medium, timestamp }  
Each event includes relevant IDs, roles, and properties (e.g., location or device info for geo analysis).

## Release Criteria

Before production release, the following must be satisfied:  
- All functional requirements (FRs) implemented and passing acceptance tests.  
- Non-functional requirements (performance, security, accessibility) verified by automated tests and manual review.  
- 100% of unit tests and >90% of integration tests pass.  
- No critical or high-severity defects open (defined by bug triage).  
- Automated security scan (e.g., OWASP ZAP) shows no exploitable vulnerabilities.  
- GDPR/CCPA compliance checklist completed (DSR flows tested).  
- Documentation (API docs, SRS, user guide) is complete and reviewed.

## Out-of-Scope

* **Payment Processing:** Integrating billing or payments is not implemented; only referral to an external payment gateway is planned.
* **Mobile Apps:** No native mobile application (only responsive web app).
* **Home Automation:** Integration with devices (alarms, sensors) is not supported.
* **Video Conferencing:** Live video or telehealth features are out of scope.
* **Multi-language UI:** Initially, only the English UI is supported (other languages are deferred to the future).
* **Offline Access:** The app requires network connectivity; there is no offline mode.

## Open Questions

* Should service requests allow multiple caregivers (group care) or only one caregiver per request?
* How to handle time zones for service\_date if users are in different regions?
* What refund/cancellation policy applies to budget and payouts (if payment is in scope)?
* How to escalate disputes automatically (e.g., mediation workflow)?
* Are there any age or background check requirements beyond document KYC for caregivers?
* Will the platform support recurring requests (e.g., weekly services)?

## Risks and Mitigations

* **Fraudulent Users:** (Risk: fake caregivers or scammers.) Mitigation: Enforce KYC checks and duplicate detection [22], and monitor unusual activity (e.g., multiple accounts from one device).
* **Data Breach:** (Risk: PII/PHI leak.) Mitigation: Strong encryption (in transit and at rest) [[2]](https://owasp.org/Top10/A02_2021-Cryptographic_Failures/#:~:text=,all%20sensitive%20data%20at%20rest), regular security audits, and SOC 2 processes [[6]](https://trustnetinc.com/resources/soc-2-best-practices-by-trustnet-an-expert-compliance-guide/#:~:text=SOC%202%20compliance%20ensures%20that,negotiable).
* **Performance Bottlenecks:** (Risk: slow response under load.) Mitigation: Implement caching (e.g., Redis), scale-out deployment, and use indexes (e.g., on search fields). Load-test early.
* **User Drop-off:** (Risk: complexity deters users.) Mitigation: Ensure smooth UX, minimal mandatory steps, and clear messaging. Iteratively refine based on user feedback.
* **Scope Creep:** (Risk: unplanned features delay the project.) Mitigation: Adhere strictly to SRS; new requests logged as backlog items and reviewed against core vision.
* **Legal Compliance Changes:** (Risk: new regulations.) Mitigation: Regularly review the legal landscape; use privacy by design.

## Change Log

| Version | Date | Description | Author |
| --- | --- | --- | --- |
| 0.1 | 2025-08-20 | Initial draft of SRS | Md. Zaheer Hassan Khan |
| 0.2 | YYYY-MM-DD | Revised after stakeholder review | [Name] |
| … | … | … | … |

*(This template will be updated as the document evolves.)*

**Sources:** This SRS follows the IEEE 29148 structure [[1]](https://www.cin.ufpe.br/~in1020/docs/publicacoes/IEEE29148-srs_example.pdf#:~:text=The%20main%20purpose%20of%20this,2%20Scope) and incorporates best practices from security and compliance standards (OWASP ASVS, WCAG, GDPR/CCPA guidelines, HIPAA, PCI-DSS) [[2]](https://owasp.org/Top10/A02_2021-Cryptographic_Failures/#:~:text=,all%20sensitive%20data%20at%20rest)[[3]](https://www.w3.org/TR/WCAG22/#:~:text=Web%20Content%20Accessibility%20Guidelines%20,laptops%2C%20kiosks%2C%20and%20mobile%20devices)[[5]](https://cppa.ca.gov/faq.html#:~:text=,subject%20to%20some%20exceptions). It is self-contained and ready for implementation.

[[1]](https://www.cin.ufpe.br/~in1020/docs/publicacoes/IEEE29148-srs_example.pdf#:~:text=The%20main%20purpose%20of%20this,2%20Scope) Example Software Requirements Specification Document for ReqView

<https://www.cin.ufpe.br/~in1020/docs/publicacoes/IEEE29148-srs_example.pdf>

[[2]](https://owasp.org/Top10/A02_2021-Cryptographic_Failures/#:~:text=,all%20sensitive%20data%20at%20rest) [[8]](https://owasp.org/Top10/A02_2021-Cryptographic_Failures/#:~:text=,not%20retained%20cannot%20be%20stolen) A02 Cryptographic Failures—OWASP Top 10:2021

<https://owasp.org/Top10/A02_2021-Cryptographic_Failures/>

[[3]](https://www.w3.org/TR/WCAG22/#:~:text=Web%20Content%20Accessibility%20Guidelines%20,laptops%2C%20kiosks%2C%20and%20mobile%20devices) Web Content Accessibility Guidelines (WCAG) 2.2

<https://www.w3.org/TR/WCAG22/>

[[4]](https://regulaforensics.com/blog/marketplace-identity-verification/#:~:text=KYC%20is%20not%20mandatory%20but,is%20often%20applied%20in%20ecommerce) [[21]](https://regulaforensics.com/blog/marketplace-identity-verification/#:~:text=An%20identity%20verification%20process%20with,to%20prevent%20online%20marketplace%20fraud) Marketplace Identity Verification to Prevent Fraud

<https://regulaforensics.com/blog/marketplace-identity-verification/>

[[5]](https://cppa.ca.gov/faq.html#:~:text=,subject%20to%20some%20exceptions) Frequently Asked Questions (FAQs)—California Privacy Protection Agency (CPPA)

<https://cppa.ca.gov/faq.html>

[[6]](https://trustnetinc.com/resources/soc-2-best-practices-by-trustnet-an-expert-compliance-guide/#:~:text=SOC%202%20compliance%20ensures%20that,negotiable) SOC 2 Best Practices by TrustNet: An Expert Compliance Guide - TrustNet

<https://trustnetinc.com/resources/soc-2-best-practices-by-trustnet-an-expert-compliance-guide/>

[[7]](https://uptime.is/#:~:text=,Yearly%3A%208h%2045m%2057s) SLA & Uptime calculator: How much downtime corresponds to 99.9 % uptime

<https://uptime.is/>

[[9]](https://www.geeksforgeeks.org/software-engineering/non-functional-requirements-in-software-engineering/#:~:text=,growing%20workload%20or%20user%20base) [[10]](https://www.geeksforgeeks.org/software-engineering/non-functional-requirements-in-software-engineering/#:~:text=2) [[11]](https://www.geeksforgeeks.org/software-engineering/non-functional-requirements-in-software-engineering/#:~:text=Maintainability%20requirements%20describe%20the%20ease,easy%20to%20test%20and%20debug) Non-Functional Requirements in Software Engineering - GeeksforGeeks

<https://www.geeksforgeeks.org/software-engineering/non-functional-requirements-in-software-engineering/>

[[12]](https://www.ibm.com/think/insights/observability-pillars#:~:text=Three%20pillars%20of%20observability%3A%20Logs%2C,easier%20to%20visualize%20and%20understand) Three pillars of observability: Logs, metrics, and traces—IBM

<https://www.ibm.com/think/insights/observability-pillars>

[[13]](https://www.hipaavault.com/resources/what-makes-a-platform-hipaa-compliant/#:~:text=HIPAA%20compliance%20applies%20when%20a,for%20radiology%20all%20handle%20PHI) What Makes a Platform HIPAA Compliant? | Essential Elements

<https://www.hipaavault.com/resources/what-makes-a-platform-hipaa-compliant/>

[[14]](https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#:~:text=To%20sum%20up%20our%20recommendations%3A) Password Storage - OWASP Cheat Sheet Series

<https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html>

[[15]](https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html#:~:text=The%20,ID%20from%20web%20browser%20traffic) [[16]](https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html#:~:text=See%20also%3A%20HttpOnly) Session Management—OWASP Cheat Sheet Series

<https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html>

[[17]](https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html#:~:text=,consider%20%2038%20using%20custom) Cross-Site Request Forgery Prevention - OWASP Cheat Sheet Series

<https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html>

[[18]](https://owasp-aasvs4.readthedocs.io/en/latest/14.4.3.html#:~:text=Verify%20that%20a%20content%20security,JSON%2C%20and%20JavaScript%20injection%20vulnerabilities) 14.4.3 A content security policy is in place—OWASP Annotated Application Security Verification Standard documentation

<https://owasp-aasvs4.readthedocs.io/en/latest/14.4.3.html>

[[19]](https://cheatsheetseries.owasp.org/cheatsheets/Query_Parameterization_Cheat_Sheet.html#:~:text=SQL%20Injection%20is%20best%20prevented,queries%20within%20a%20web%20application) Query Parameterization - OWASP Cheat Sheet Series

<https://cheatsheetseries.owasp.org/cheatsheets/Query_Parameterization_Cheat_Sheet.html>

[[20]](http://owasp-aasvs.readthedocs.io/en/latest/requirement-2.20.html#:~:text=2,attacks%C2%B6) 2.20 Protects against brute force attacks—OWASP Annotated Application Security Verification Standard 3.0.0 documentation

<http://owasp-aasvs.readthedocs.io/en/latest/requirement-2.20.html>

[[22]](https://kycaid.com/duplicates/#:~:text=Multiple%20Account%20Detection) Duplicate Detection for KYC Compliance - KYCAID

<https://kycaid.com/duplicates/>