**1. Introduction**

The purpose of this document is to define the software requirements for GigConnect, a hyperlocal freelance marketplace designed to connect skilled freelancers with clients in the same community or geographic area. The platform addresses the growing need for localized freelance opportunities where clients can quickly find nearby professionals for small-scale or specialized tasks, while freelancers gain a trusted channel to showcase their expertise and secure job opportunities close to their location and globally.

This SRS serves as a foundation for the system’s design, development, and implementation. It ensures that developers, project stakeholders, and end-users have a common understanding of the project’s goals, features, and constraints.

**1.1 Purpose**

The purpose of GigConnect is to build a hyperlocal freelance marketplace that connects local clients with freelancers. The platform will allow clients to post jobs, search for nearby freelancers, and hire based on skills, location, and ratings. Freelancers will be able to showcase their expertise, apply for local jobs, and securely manage payments.

**1.2 Scope**

Gig-Connect will be a mobile-responsive web application that provides two primary roles: **Clients** and **Freelancers**.

* **Clients** :
  + Register and authenticate securely.
  + Post job listings (gigs) with clear requirements and budgets.
  + Search for freelancers based on skill, price, rating, and location.
  + Communicate with freelancers through an integrated messaging system.
  + Pay freelancers securely through a payment gateway with milestone support.
* **Freelancers** :
  + Register and authenticate securely.
  + Create detailed profiles with skills, service rates, portfolios, and availability.
  + Browse and apply for jobs posted by clients.
  + Chat with clients in real-time for clarifications.
  + Receive payments securely and provide feedback to clients.

**1.3 Definitions, Acronyms, Abbreviations**

* **Gig:** A job or service posted by a client.
* **JWT:** JSON Web Token, used for secure authentication.
* **UI:** User Interface.
* **API:** Application Programming Interface.

**1.4 References**

* Razorpay/Stripe API documentation.
* React.js & Node.js documentation.
* JWT authentication standards.

**2. Overall Description**

**2.1 Product** Perspective

Gig-Connect is designed as a **mobile-responsive web application** that acts as an intermediary platform between **clients** and **freelancers** at the community level. Unlike global freelance platforms, GigConnect focuses on **hyperlocal service delivery**, enabling trust and faster engagement.

* **System Architecture:**
  + **Front-End:** Built using React.js for responsive design.
  + **Back-End:** Powered by Node.js/Express.js with REST APIs.
  + **Database:** MongoDB/MySQL to store user profiles, gigs, transactions, and messages.
  + **Integration:** Payment gateway (Razorpay/Stripe), Google Maps API for location services, JWT for authentication.
  + **Communication:** Real-time messaging via WebSockets.
* **External Interfaces:**
  + **Clients:** Access through browsers on desktop or mobile devices.
  + **Freelancers:** Same interface, role-specific features enabled.
  + **Third-party services:** Payment gateway, email/SMS verification services, Google Maps API.

GigConnect will operate as a **centralized system**, where all operations (gig posting, search, communication, payment) are executed through the platform.

**2.2 Product Functions**

The main functional modules of GigConnect include:

1. **Dual-Role User Authentication**
   * Registration/login via email and mobile verification.
   * JWT-based secure session management.
   * Role-based dashboard for clients and freelancers.
2. **Freelancer Profile Management**
   * Add/edit personal information, skills, rates, and portfolio.
   * View and respond to client reviews and ratings.
   * Track earnings and job history.
3. **Gig Posting & Management**
   * Clients post gigs with title, description, budget, deadline, and category.
   * Edit, pause, or close gigs.
   * Manage applications from freelancers.
4. **Search & Filtering**
   * Hyperlocal search based on freelancer’s location.
   * Advanced filters: skills, price range, ratings, distance.
5. **Real-Time Messaging System**
   * Text-based chat with attachments.
   * Notification system for new messages.
6. **Review & Rating System**
   * Clients rate freelancers after job completion.
   * Freelancers provide feedback on clients.
   * Ratings displayed on profiles for credibility.
7. **Secure Payment Integration**
   * Razorpay/Stripe-based payments.
   * Escrow/milestone payment system.
   * Transaction history for both clients and freelancers.

**2.3 User Characteristics**

**Clients**

* **Demographics:** Individuals, households, or small businesses seeking localized freelance services.
* **Technical Skills:** Basic familiarity with web applications (registration, posting tasks, making payments).
* **Goals:** Quick access to trustworthy freelancers nearby, secure payments, and verified reviews.

**Freelancers**

* **Demographics:** Skilled professionals, students, part-time workers, or small business owners offering services.
* **Technical Skills:** Ability to create and manage an online profile, apply for gigs, and use chat/payment systems.
* **Goals:** Visibility in the local job market, fair compensation, building credibility through reviews.

**2.4 Constraints**

1. **Technical Constraints:**
   * The system must be **web-based** and **mobile-responsive**.
   * Payments must comply with **PCI DSS (**Payment Card Industry Data Security Standard**)** standards for financial security.
   * Real-time chat requires **stable internet connectivity** and server uptime.
2. **Operational Constraints:**
   * Users must verify their identity (email/phone verification).
   * Platform must comply with local tax and labor regulations.
   * Freelancers can only apply for gigs within a specified **geographical radius** (defined by Google Maps API).
3. **Business Constraints:**
   * Service fees may be deducted from freelancer payments for platform sustainability.
   * Dispute resolution mechanisms must be available for conflicts.

**2.5 Assumptions and Dependencies**

* **Assumptions:**
  1. Users (clients and freelancers) have access to stable internet connections.
  2. Users possess devices (mobile/PC) with modern browsers.
  3. Clients and freelancers act in good faith, respecting platform rules.
  4. Payment gateways (Razorpay/Stripe) are accessible in the target region.
  5. Google Maps API will provide accurate location-based services.
* **Dependencies:**
  1. System reliability depends on third-party APIs (Google Maps, payment gateway).
  2. Secure authentication depends on JWT and encrypted database storage.
  3. Platform performance depends on hosting infrastructure (e.g., AWS, Azure).
  4. Community adoption depends on effective marketing and trust-building mechanisms.

**3. Specific Requirements**

**3.1 Functional Requirements**

The system must provide the following **functional capabilities**:

**1. User Authentication & Authorization**

* The system shall allow users to register as either **Client** or **Freelancer**.
* The system shall use **JWT (JSON Web Token)** for secure session management.
* The system shall require email/phone verification during registration.
* Users shall have role-based dashboards (Client dashboard, Freelancer dashboard).

**2. Freelancer Profile Management**

* Freelancers shall be able to create/edit profiles with name, bio, skills, hourly/daily rate, portfolio, and availability.
* The system shall allow uploading of portfolio files (images, documents, videos).
* Freelancers shall be able to view client reviews and ratings.

**3. Gig Posting & Management**

* Clients shall be able to post new gigs with job title, description, budget, deadline, and required skills.
* Clients shall be able to edit or delete posted gigs.
* Freelancers shall be able to apply for gigs.
* Clients shall be able to shortlist, hire, or reject freelancers.

**4. Search & Filtering**

* Users shall be able to search freelancers/gigs based on skill, location, price range, and ratings.
* The system shall provide hyperlocal search using Google Maps API.
* Advanced filters shall include sorting by highest rated, nearest location, and lowest price.

**5. Real-Time Messaging System**

* Clients and freelancers shall be able to exchange text messages within the platform.
* The system shall allow file sharing (documents, images, small videos).
* The system shall provide notifications for new messages.

**6. Review & Rating System**

* After gig completion, clients shall rate freelancers (1–5 stars) and provide written feedback.
* Freelancers shall also rate and review clients.
* Reviews shall be displayed on profiles and gig history.

**7. Secure Payment Integration**

* The system shall integrate Razorpay/Stripe for secure payments.
* The system shall support escrow/milestone payments.
* Clients shall view payment history and invoices.
* Freelancers shall view earnings and withdrawal history.

**3.2 Non-Functional Requirements (NFRs)**

**1. Performance Requirements**

* The system shall support at least **500 concurrent users** with <2 second average response time.
* The system shall ensure database queries execute within **1 second** under normal load.
* The system shall scale horizontally using cloud infrastructure (e.g., AWS).

**2. Security Requirements**

* All sensitive data (passwords, payment info) shall be stored in **encrypted format**.
* Communication shall use **HTTPS with SSL/TLS** encryption.
* Authentication shall be handled using **JWT tokens with expiration policies**.
* The platform shall comply with **PCI DSS standards** for payment security.

**3. Usability Requirements**

* The system shall be **mobile-responsive**, accessible via desktop, tablet, and mobile browsers.
* UI shall follow **intuitive navigation** with minimal clicks for major tasks.
* Accessibility features shall include **keyboard navigation** and support for screen readers.

**4. Reliability & Availability Requirements**

* The system shall provide **99.9% uptime**.
* Automatic backup shall occur daily to prevent data loss.
* Recovery time after a system crash shall be **less than 2 hours**.

**5. Maintainability & Scalability**

* Code shall follow **modular structure** to support future enhancements.
* The system shall support scaling to **5000+ users** in the future.
* The system shall provide **error logging and monitoring** for maintenance.

**3.3 System Interfaces**

The system will interact with several external components:

* **Payment Gateway Interface (Razorpay/Stripe):**
  + Provides secure transactions and escrow functionality.
  + Requires API keys and compliance with financial regulations.
* **Google Maps API:**
  + Provides location-based services for hyperlocal search.
  + Requires integration of API for distance and map rendering.
* **Email/SMS Service:**
  + Sends verification codes, gig updates, and notifications.
* **Hosting Platform (AWS/Azure/Google Cloud):**
  + Ensures scalability, high availability, and secure infrastructure.

**3.4 User Interfaces (UI/UX Requirements)**

The application will provide intuitive, role-specific dashboards:

**Client Dashboard**

* Post, edit, and manage gigs.
* View freelancer profiles and applications.
* Access payment and transaction history.
* Messaging interface for communication.

**Freelancer Dashboard**

* Create/edit profile (skills, portfolio, availability).
* Browse and apply for gigs.
* Manage accepted gigs and payment history.
* Messaging interface for communication.

**Admin Dashboard (optional future module)**

* Monitor users, gigs, transactions, and disputes.
* Generate reports and analytics.

**General UI Features**

* Mobile-first design using **responsive layout**.
* **Navigation bar** with role-specific options.
* **Notifications panel** for messages and job updates.
* **Profile section** for account settings.

**4. System Models & Diagrams**

**4.1 Use Case Diagram**

**Actors:** Client, Freelancer, Payment Gateway.

* Client: Register/Login → Post Gig → Search Freelancer → Chat → Hire → Payment → Review.
* Freelancer: Register/Login → Create Profile → Browse Gigs → Apply → Chat → Get Hired → Payment → Review.

**4.2 Data Flow Diagram (Level 0)**

* User → [Authentication System] → Dashboard → [Gig Module / Profile Module / Messaging / Payments].

**4.3 Entity-Relationship Diagram (ERD)**

**Entities:**

* User (Client/Freelancer)
* Profile
* Gig
* Message
* Review
* Transaction

**5. External Interface Requirements**

* **User Interfaces:** Responsive web interface (React.js).
* **Hardware Interfaces:** Runs on desktops, tablets, mobiles.
* **Software Interfaces:**
  + Razorpay/Stripe API for payments.
  + Google Maps API for location.