Creator Boost
Software Requirements Specification

Version <1.0>

Group – 25
PID – 18
Mentor – Mr. Kavinda Rajapakse

220568T – Sandeep L.G.R.
220573E – Sandiw K.B.I.
220585R – Sathsara H.M.W.C.

-Contributions-

220568T – Sandeep L.G.R.

- Introduction
- Overall Description
- Specific Requirements

220573E – Sandiw K.B.I.

- Introduction
- Overall Description
- Usability Requirements

220585R – Sathsara H.M.W.C.

- Introduction
- Overall Description
- Reliability Requirements

Table of Contents

Software Requirements Specification

1. Introduction

1.1 Purpose

The purpose of this SRS is to define the requirements for the CreatorBoost platform, a digital marketplace that connects content creators, educators, and professionals with social media experts. This document describes both the functional requirements (like authentication, gig posting, order placement, payments, messaging) and non-functional requirements (such as reliability, security, and performance). It serves as a reference for developers, testers, and stakeholders.

1.2 Scope

CreatorBoost is an online marketplace focused on social media marketing services. Unlike generic platforms like Fiverr or Upwork, CreatorBoost is specialized for helping users grow their presence on platforms such as YouTube, TikTok, Facebook, and Instagram.

Key features include:

- User authentication (with OTP and JWT)
- Role-based dashboards (Buyer, Seller, Admin)
- Gig creation and management (with packages, FAQs, and images)
- Order booking and payment (Stripe integration)
- Reviews and ratings
- Messaging and notifications
- Admin monitoring and reporting

The system is built using a microservices architecture with Spring Boot, React.js, PostgreSQL, MongoDB, Kafka, Docker, and Stripe API.

1.3 Definitions, Acronyms, and Abbreviations

- **SRS** – Software Requirements Specification
- **JWT** – JSON Web Token (used for authentication)
- **API** – Application Programming Interface
- **UI** – User Interface
- **REST** – Representational State Transfer
- **OTP** – One-Time Password
- **CRUD** – Create, Read, Update, Delete
- **Stripe API** – Third-party payment gateway for secure transactions
- **Kafka** – Apache Kafka, a message broker for asynchronous communication

1.4 References

- IEEE 830-1998: IEEE Recommended Practice for Software Requirements Specifications
- Spring Boot Documentation – https://spring.io/projects/spring-boot
- React.js Documentation – https://react.dev
- Apache Kafka – https://kafka.apache.org/
- Stripe API Docs – https://stripe.com/docs/api

1.5 Overview

The rest of this SRS is structured as follows:

- Section 2 (Overall Description): General factors like product perspective, user roles, and system constraints.
- Section 3 (System Features & Requirements): Functional and non-functional requirements in detail.
- Section 4 (Supporting Information): Appendices, references, and supporting materials (e.g., diagrams, storyboards).

2. Overall Description

2.1 Product Perspective

CreatorBoost is a standalone platform developed as a web application using a microservices architecture. Each service (Auth, Gig, Order, Payment, Review, Messaging, Notification) runs independently but communicates through REST APIs (synchronous) and Kafka events (asynchronous).

2.2 Product Functions

- User signup/login (with OTP verification, JWT tokens)
- Create and manage gigs (with packages, FAQs, images)
- Browse and search gigs (filters, sorting, keyword search)
- Place and manage orders (status tracking, delivery timeline)
- Secure payments with Stripe API
- Messaging between buyers and sellers
- Review and rating system
- Notifications and alerts
- Admin control for moderation and reporting

2.3 User Characteristics

- Buyers (Creators/Teachers/Businesses): Non-technical, expect simple navigation to book gigs and pay.
- Sellers (Social Media Experts): Need dashboards to manage gigs, orders, and reviews.
- Admins: Require advanced controls for moderation, analytics, and user management.

2.4 Constraints

- Payments are handled only via Stripe .
- Deployed using Docker containers on AWS EC2.

- Must support 24/7 uptime with 99.5% availability.

2.5 Assumptions and Dependencies

- Users will have access to internet and modern browsers.
- Payment processing depends on Stripe API availability.
- File storage depends Cloudinary.
- Notifications depend on message broker (Kafka) availability.

2.6 Requirements Subsets

- MVP (Minimum Viable Product): Authentication, Gig Management, Order Placement, Stripe Payments.
- Future Extensions: AI-powered gig recommendations (Gemini API), mobile app support, advanced analytics dashboards.

3. Specific Requirements

3.1 Functionality

3.1.1 User Registration and Authentication (FR-01)

Description

The system shall provide secure user registration and authentication. New users must register with an email address and verify it via a One-Time Password (OTP). Returning users can log in using JWT (JSON Web Token) based authentication. This ensures secure identity management across all services.

Inputs

- Email address
- Password (for login)
- OTP code (during verifying)

Outputs

- Success message with account creation confirmation
- JWT token for authenticated sessions
- Error messages for invalid credentials or expired OTP

Preconditions

- The user must have a valid and unique email address
- The system must be connected to the authentication service and email service for OTP delivery

Postconditions

- A new user account is created and stored in the User Service database
- JWT token is generated upon successful authentication
- Authenticated user can now access role-based dashboards (Buyer, Seller, Admin)

Dependencies

- Email Service (for OTP delivery)
- Authentication Service (Spring Boot + Spring Security for user accounts and JWT issuance)
- Database: PostgreSQL (stores user records, OTPs, and hashed passwords)

3.1.2 Role-Based Access (FR-02)

Description

The system shall provide role-based access control (RBAC) to ensure that users only access features relevant to their roles. Users can register as Buyers (clients seeking services), Sellers (experts providing gigs), or be assigned as Admins (managing the platform). Each role has a dedicated dashboard and specific permissions.

Inputs

- User ID (from Authentication Service)
- Role selection during registration (Buyer/Seller)
- Admin role assigned by system administrators

Outputs

- Buyer Dashboard with gig browsing, order placement, messaging, and payments
- Seller Dashboard with gig management, order management, earnings, and messaging
- Admin Dashboard with user/gig management, dispute resolution, and reports
- Access denied message if unauthorized user attempts to access restricted features

Preconditions

- The user must be authenticated and have a valid JWT token
- The role must be assigned in the User Service

Postconditions

- The system enforces access restrictions based on role
- Buyers cannot create gigs, Sellers cannot manage users, Admins can access all controls

Dependencies

- Authentication Service (JWT token validation)
- User Service (stores role information)
- API Gateway (checks roles and routes requests accordingly)

3.1.3 Profile Management (FR-03)

Description

The system shall allow users (Buyers, Sellers, and Admins) to manage their personal profiles. Users can update their profile information such as name, bio, profile picture, skills (for Sellers), and contact details. This ensures that each user maintains an up-to-date presence on the platform.

Inputs

- User ID (from Authentication Service)
- Updated profile data (name, email, bio, profile picture URL, skills, location, etc.)

Outputs

- Success message confirming profile update
- Error messages for invalid inputs (e.g., invalid email format, missing required fields)
- Updated profile information displayed in the user dashboard and visible to others (for Sellers/Buyers)

Preconditions

- The user must be authenticated with a valid JWT token
- The user account must exist in the User Service

Postconditions

- Updated user information is stored in the User Service database
- For Sellers, updated profiles are reflected in their gig listings
- For Buyers, updated profiles are displayed when interacting with Sellers

Dependencies

- User Service (stores and manages user profiles)
- Cloud Storage (Cloudinary) for storing profile images
- Database: PostgreSQL for structured profile data

3.1.4 Gig Creation and Management (FR-04)

Description

The system shall allow Sellers to create and manage gigs that represent the services they offer. Each gig includes a title, description, category, platform (e.g., YouTube, TikTok, Facebook), images, packages (Basic, Standard, Premium), and FAQs. Sellers shall be able to update, activate/deactivate, or delete their gigs. Buyers can view active gigs while browsing or searching.

Inputs

- Seller ID (from User Service)
- Gig details:
    - Title
    - Description
    - Category & Subcategory
    - Platform (YouTube, TikTok, etc.)
    - Status (ACTIVE / INACTIVE)
    - Images (uploaded via Cloudinary/S3, stored as URLs)
    - Packages (name, price, delivery days, description)
    - FAQs (question, answer)

Outputs

- Confirmation message upon successful gig creation/update
- Error message for missing/invalid fields (e.g., missing title, negative price)
- Updated gig details available in Buyer's browsing/search results

Preconditions

- The seller must be authenticated with a valid JWT token
- The seller must have a valid Seller profile
- The gig details must comply with platform rules (e.g., categories available)

Postconditions

- A new gig record is stored in the Gig Service database
- Gig status defaults to ACTIVE upon creation
- Updated gig information is visible to Buyers in search/browse results
- If deactivated, the gig will not appear in Buyer's browsing/searching

Dependencies

- Gig Service (handles gig creation, updates, and retrieval)
- User Service (validates Seller role and profile)
- Database: PostgreSQL (for structured gig data)
- Cloudinary (stores gig images, URLs saved in DB)

3.1.5 Gig Browsing and Search (FR-05)

Description

The system shall allow Buyers to browse, filter, and search available gigs. Buyers can view all active gigs and refine results using filters such as platform, category, price range, and rating. Additionally, Buyers can search gigs by keywords (e.g., "YouTube SEO," "Content Strategy") for quick discovery.

Inputs

- Search query (keywords entered by Buyer)
- Filter parameters (platform, category, price range, rating)
- Sorting preferences (price low-to-high, popularity, rating, newest)

Outputs

- A list of matching gigs with summary details (title, price, seller rating, thumbnail image)
- No results message if no gigs match the filters/search criteria

Preconditions

- Buyer must be authenticated with a valid JWT token
- Gigs must exist in the Gig Service with status = ACTIVE

Postconditions

- A list of gigs is retrieved and displayed to the Buyer based on filters/search criteria

- Selected gig can be clicked to view full gig details

Dependencies

- Gig Service (stores gig information, provides search and filtering functionality)
- Database: PostgreSQL (gig data, searchable by indexed fields like title, category, platform)
- API Gateway (routes Buyer's search and filter requests to the Gig Service)

3.1.6 Gig Details (FR-06)

Description

The system shall allow Buyers to view the complete details of a selected gig. This includes gig title, description, images, packages, FAQs, seller profile information, ratings, and reviews. This feature ensures Buyers have enough information to make informed purchasing decisions before placing an order.

Inputs

- Gig ID (selected by Buyer during browsing/search)

Outputs

- Detailed gig information:
    - Title and description
    - Platform and category
    - Seller details (name, profile picture, rating, number of completed orders)
    - Packages (Basic, Standard, Premium with pricing and delivery timelines)
    - FAQs
    - Reviews and ratings
    - Gig images (with one primary image highlighted)

Preconditions

- Buyer must be authenticated with a valid JWT token
- The selected gig must exist in the Gig Service and be in ACTIVE status

Postconditions

- The Buyer is able to view complete gig details
- Buyer may proceed to place an order by selecting one of the packages

Dependencies

- Gig Service (retrieves gig details, packages, FAQs, and images)
- User Service (retrieves seller profile details for display)
- Review Service (retrieves reviews and ratings associated with the gig)
- Database: PostgreSQL for gig metadata, MongoDB (optional) for reviews/ratings storage
- Cloud Storage (Cloudinary) for gig images

3.1.7 Order Placement (FR-07)

Description

The system shall allow Buyers to place an order for a selected gig by choosing a package (Basic, Standard, or Premium). An order represents a formal request from a Buyer to a Seller and includes details such as gig, package, buyer, seller, delivery timeline, and amount. Once placed, the order is stored in the Order Service and tracked until completion.

Inputs

- Buyer ID (from User Service)
- Gig ID (from Gig Service)
- Package ID (selected by Buyer from Gig details)
- Payment amount (from Gig Service package)

Outputs

- Confirmation message for order creation
- Unique Order ID
- Initial order status = PENDING
- Error message if gig/package is invalid or inactive

Preconditions

- The Buyer must be authenticated with a valid JWT token
- The selected gig must exist in ACTIVE status
- The selected package must belong to the chosen gig

Postconditions

- A new Order record is created in the Order Service database
- The Buyer receives confirmation and can proceed to payment
- The Seller is notified of a new order request

Dependencies

- Order Service (handles order creation and tracking)
- Gig Service (validates gig and package details, fetches price)
- User Service (validates Buyer and Seller IDs)
- Database: PostgreSQL for storing order data
- Notification Service (to alert Seller of a new order)

3.1.8 Order Management (FR-08)

Description

The system shall allow Buyers and Sellers to manage orders after placement. Buyers can track the status of their orders, while Sellers can accept, reject, or update progress. Orders transition through predefined statuses such as PENDING, ACCEPTED, IN PROGRESS, DELIVERED, PAID, COMPLETED, and CANCELLED. This ensures transparency and accountability throughout the transaction lifecycle.

Inputs

- Order ID (generated during order placement)
- Buyer or Seller actions (track, cancel, update, mark as delivered, confirm completion)
- System updates (e.g., payment completed)

Outputs

- Updated order status
- Notifications sent to Buyer and Seller on status change
- Error messages if unauthorized actions are attempted (e.g., Buyer trying to mark an order as "Delivered")

Preconditions

- The Buyer and Seller must be authenticated with valid JWT tokens
- The Order must exist in the Order Service database

Postconditions

- Order status is updated and stored in the database
- Buyers can view the updated status in their dashboard
- Sellers can track their order pipeline (active, delivered, completed)

Dependencies

- Order Service (manages order lifecycle and statuses)
- User Service (validates Buyer and Seller identities)
- Notification Service (informs parties of updates)
- Database: PostgreSQL for storing order records and statuses

3.1.9 Payment Processing (FR-09)

Description

The system shall provide secure payment processing for all orders. Buyers can pay for services using the Stripe API. Payments are linked to specific orders and recorded in the system. Once payment is confirmed, the corresponding order status is updated to PAID. Failed payments result in a FAILED status, and the Buyer is prompted to retry.

Inputs

- Order ID (from Order Service)
- Buyer payment details (card information, tokenized via Stripe)
- Payment amount (retrieved from Gig package details)

Outputs

- Payment confirmation (success/failure)
- Updated order status (PAID if successful, FAILED if unsuccessful)
- Payment record stored in Payment Service database
- Notification sent to both Buyer and Seller

Preconditions

- The Buyer must be authenticated with a valid JWT token
- The Order must exist with status = PENDING
- The payment amount must match the gig package price

Postconditions

- A new payment transaction is created and stored in the Payment Service database
- The order status is updated to PAID upon successful payment
- The Seller is notified that payment has been made and can proceed with service delivery

Dependencies

- Payment Service (handles payment transactions and status updates)
- Stripe API (or payment gateway) (processes card payments securely)
- Order Service (updates order status upon payment confirmation)
- Database: PostgreSQL for payments
- Notification Service (alerts Buyer and Seller on payment events)

3.1.10 Review and Rating (FR-10)

Description

The system shall allow Buyers to leave reviews and ratings for Sellers after an order is marked as COMPLETED. Reviews include written feedback, while ratings are numeric (e.g., 1–5 stars). This feature ensures service quality and helps future Buyers evaluate Sellers before placing orders.

Inputs

- Order ID (from Order Service)
- Buyer ID (from User Service)
- Rating (numeric value, e.g., 1–5)
- Review text (optional feedback from Buyer)

Outputs

- Confirmation message upon successful review submission
- Updated Seller profile rating (average rating recalculated)
- Review displayed under the corresponding gig details page
- Error messages if invalid rating (e.g., outside 1–5 range) or unauthorized user submits review

Preconditions

- The Buyer must be authenticated with a valid JWT token
- The order must exist and have status = COMPLETED
- The Buyer must be the one who placed the order

Postconditions

- Review is stored in the Review Service database and linked to the Seller and Gig

- Seller's rating is updated (average rating recalculated)
- Review is visible to all future Buyers under that gig

Dependencies

- Review Service (stores reviews and calculates average ratings)
- Order Service (validates order completion before allowing review submission)
- User Service (links review to Buyer and Seller identities)
- Database: MongoDB (for scalable storage of reviews and ratings)

3.1.11 Messaging and Communication (FR-11)

Description

The system shall provide a messaging feature that allows Buyers and Sellers to communicate directly regarding gigs, orders, and project requirements. Messages are stored securely and delivered in real-time using WebSockets. This ensures transparency, reduces disputes, and helps clarify order details.

Inputs

- Sender ID (Buyer or Seller)
- Receiver ID (Buyer or Seller)
- Message content (text, attachments, links)
- Timestamp (auto-generated by system)

Outputs

- Confirmation of message sent
- Real-time delivery of message to receiver's inbox/chat window
- Error messages for invalid inputs (e.g., empty message, blocked user)

Preconditions

- Both sender and receiver must be authenticated with valid JWT tokens
- Both must have an active Buyer/Seller account
- Sender and receiver must have either a gig interaction (inquiry) or an order relationship

Postconditions

- Messages are stored in the Messaging Service database
- Receiver is notified in real-time of the new message
- Message history is retrievable for future reference and dispute resolution

Dependencies

- Messaging Service (manages message persistence and retrieval)
- WebSockets (for real-time delivery of chat messages)
- User Service (validates sender and receiver identities)
- Database: MongoDB (for storing chat logs and message history)
- Notification Service (sends alerts for unread messages)

3.1.12 Notifications (FR-12)

Description

The system shall provide real-time notifications to Buyers and Sellers about important events such as new orders, order status updates, payments, reviews, and unread messages. Notifications may be delivered via in-app alerts, email, or push notifications. This ensures that users remain informed and engaged with the platform.

Inputs

- Trigger events:
  - New order placed
  - Order status change (e.g., Accepted, Delivered, Completed)
  - Payment confirmation or failure
  - New message received
  - New review posted
- User ID (recipient of notification)

Outputs

- In-app notification (bell icon with count, alert popup)
- Optional email notification (order confirmation, payment receipt, etc.)
- Optional push notification (if mobile app or web push enabled)

Preconditions

- The user must be authenticated with a valid JWT token
- A valid event (order, payment, message, review) must be generated by the respective service

Postconditions

- Notifications are recorded in the Notification Service database
- User receives the notification in-app (and optionally by email/push)
- Unread notifications are highlighted until the user views them

Dependencies

- Notification Service (manages notification creation and delivery)
- Order Service (triggers order-related notifications)
- Payment Service (triggers payment-related notifications)
- Messaging Service (triggers message-related notifications)
- Review Service (triggers review-related notifications)
- Database: MongoDB (for scalable notification storage and retrieval)
- Kafka (for asynchronous event-based notifications)

3.1.13 Admin Management (FR-13)

Description

The system shall provide an Admin Dashboard to manage users, gigs, transactions, and overall platform activities. Admins ensure compliance, resolve disputes, and maintain service quality. They also monitor system health,

generate reports, and handle user bans or gig removals when necessary.

Inputs

- Admin ID (validated through authentication & role-based access control)
- Admin actions:
    - View all users (Buyers & Sellers)
    - Suspend or ban users
    - Approve/Remove gigs
    - Monitor transactions (orders, payments)
    - Resolve disputes between Buyers and Sellers
    - Generate system reports (revenue, active users, popular categories)

Outputs

- Confirmation messages for admin actions (e.g., "User banned successfully")
- Updated system state (user removed, gig deactivated, report generated)
- Error messages for invalid actions or insufficient permissions

Preconditions

- Admin must be authenticated and authorized with Admin role
- Relevant data (users, gigs, payments, orders) must exist in the system

Postconditions

- Users, gigs, and transactions are updated in their respective services based on admin actions
- Reports are generated and stored for analytics
- Banned or suspended accounts cannot access the system

Dependencies

- User Service (for managing users and roles)
- Gig Service (for activating/deactivating gigs)
- Order Service (for monitoring transactions and disputes)
- Payment Service (for transaction monitoring and fraud detection)
- Review Service (for moderating abusive reviews)
- Notification Service (to inform users about admin actions)
- Database: PostgreSQL for structured admin actions

3.2 Usability Requirements

This section includes all requirements that affect usability, including training time for users, measurable task times, and conformance to usability standards.

3.2.1 Intuitive User Interface (UR-01)

Description:
The system shall provide an intuitive and user-friendly interface for Buyers, Sellers, and Admins with clear navigation, simple workflows, and consistent layouts.

Rationale:

Ensures that users with minimal technical knowledge can easily access and use the platform.

3.2.2 Multi-Device Accessibility (UR-02)

Description:
The platform shall be usable across desktop, tablet, and mobile devices with responsive design.

Rationale:
Supports users accessing the marketplace from different devices and screen sizes.

3.2.3 Search and Filter Usability (UR-04)

Description:
The system shall allow users to search and filter gigs by category, platform, rating, price, and relevance.

Rationale:
Helps Buyers quickly find suitable services without unnecessary effort.

3.3 Reliability Requirements

Requirements for reliability should specify availability, Mean Time Between Failures (MTBF), Mean Time To Repair (MTTR), accuracy, and defect rates.

3.3.1 Availability (RR-01)

Description:
The system shall maintain an availability of at least 99.5% uptime excluding scheduled maintenance. Scheduled maintenance shall occur during non-peak hours with prior user notification.

Rationale:
Ensures that Buyers and Sellers can reliably access the marketplace whenever needed.

3.3.2 Mean Time Between Failures (MTBF) (RR-02)

Description:
The system shall have an MTBF of at least 1,000 hours under normal operating conditions.

Rationale:
Indicates system robustness and reduces user frustration caused by frequent failures.

3.3.3 Mean Time To Repair (MTTR) (RR-03)

Description:
In the event of system failure, the system shall be restored within 4 hours or less to minimize downtime.

Rationale:
Quick recovery ensures service continuity for critical business transactions.

3.3.4 Accuracy of Transactions (RR-04)

Description:
All monetary transactions (orders, payments, refunds) shall be 100% accurate with respect to amounts, user accounts, and order references. Gig details retrieved during booking shall match the original listing without discrepancies.

Rationale:
Ensures trust in the system's payment and order processing.

3.3.5 Critical Bug Handling (RR-06)

Description:
Critical bugs (defined as complete service outages, loss of payment data, or security breaches) must be fixed within 24 hours of detection.

Rationale:
Prevents severe disruption of business operations and protects user trust.

3.4 Performance and Security Requirements

3.4.1 Response Time (PR-01)

Description:
The system shall provide an average response time of ≤ 2 seconds for all standard operations (gig search, booking, profile load). Payment processing and authentication shall complete in ≤ 5 seconds under normal load.

Rationale:
Fast response ensures a smooth user experience and prevents drop-offs.

3.4.2 Throughput (PR-02)

Description:
The system shall support at least 200 transactions per second (TPS) during peak usage, including gig searches, orders, and payments.

Rationale:
Ensures the platform can scale with high user activity.

3.4.3 Capacity (PR-03)

Description:
The system shall support:

- 100,000 registered users (Buyers + Sellers)
- 10,000 concurrent active users
- 50,000 total gigs listed

Rationale:

Defines the growth scalability of the platform.

3.4.4 Resource Utilization (PR-05)

Description:
The system shall maintain optimal utilization:

- CPU usage ≤ 70% average load
- Memory usage ≤ 75% average load
- Database queries optimized to ≤ 100ms average execution

Rationale:
Prevents resource bottlenecks and ensures stability.

3.4.5 Authentication & Authorization (SR-01)

Description:
The system shall enforce JWT-based authentication and role-based access control (RBAC) for Buyers, Sellers, and Admins.

3.4.6 Payment Security (SR-02)

Description:
The system shall comply with PCI DSS standards for handling payment data via Stripe API, ensuring no raw card data is stored in the system.

3.4.7 Security Monitoring (SR-03)

Description:
The system shall implement audit logs, anomaly detection, and intrusion detection systems (IDS) to monitor suspicious activity in real time.

3.5 Supportability Requirements

3.5.1 Standardized Naming Conventions (SUP-01)

Description:
All code, APIs, database tables, and microservice components shall follow consistent naming conventions (camelCase for variables, PascalCase for classes, snake_case for database tables).

Rationale:
Improves readability and makes ongoing support easier across teams.

3.5.2 Class Libraries and Frameworks (SUP-02)

Description:
The system shall use standard and widely supported libraries/frameworks such as Spring Boot for backend services, React.js for frontend, and PostgreSQL/MongoDB for data persistence.

Rationale:
Reduces maintenance risks by avoiding unsupported or niche technologies.

3.5.3 Maintenance Access (SUP-03)

Description:
The system shall provide admin dashboards, database management access (pgAdmin, MongoDB Compass), and API monitoring tools (Postman) for maintenance and troubleshooting.

Rationale:
Ensures that administrators and support engineers can quickly address issues.

3.5.4 Monitoring and Alerts (SUP-04)

Description:
The system shall support health checks, logging, and automated alerts using tools like Grafana

Rationale:
Helps support teams detect failures before they impact end-users.

3.5.5 Backup and Recovery Utilities (SUP-05)

Description:
The system shall implement automated database backup scripts (daily) and provide recovery utilities for restoring services in case of failures.

Rationale:
Ensures business continuity and reduces downtime during system failures.

3.5.6 Environment Configuration (SUP-06)

Description:
The system shall maintain separate environments for development, testing, staging, and production, with configuration managed via environment variables and Docker.

Rationale:
Prevents production issues caused by untested changes and simplifies deployments.

3.6 Design Constraints

3.6.1 Standards Compliance (DC-01)

Description:
The system shall comply with RESTful API design principles, OAuth2/JWT authentication standards, and PCI DSS for payment handling.

Rationale:

Ensures interoperability, security, and industry-standard compliance.

3.6.2 Programming Languages (DC-02)

Description:
The backend shall be implemented in Java (Spring Boot), and the frontend shall be implemented using React.js with Tailwind CSS.

Rationale:
Mandates the chosen technology stack for maintainability and consistency.

3.6.3 Database Technologies (DC-03)

Description:
The system shall use PostgreSQL for structured relational data (users, orders, payments) and MongoDB for flexible/non-relational data (messages, reviews).

Rationale:
Ensures scalability and efficient handling of both structured and unstructured data.

3.6.4 Architectural Constraints (DC-04)

Description:
The system shall follow a microservices architecture with services being independently deployable and communicating via REST APIs (synchronous) and Apache Kafka (asynchronous).

Rationale:
Supports scalability, modularity, and fault isolation.

3.6.5 Development Tools (DC-06)

Description:
The development shall use IntelliJ IDEA / VS Code for coding, Postman/Swagger for API testing, GitHub/GitLab for version control, and Jenkins/GitHub Actions for CI/CD.

Rationale:
Specifies approved tools for consistency in development workflow.

3.7 On-line User Documentation and Help System Requirements

3.7.1 User Guides (UDR-01)

Description:
The system shall provide an online user guide accessible via the platform. It will include step-by-step instructions for account creation, browsing gigs, placing orders, managing services, and making payments.

Rationale:

Ensures that both new and experienced users can easily learn how to use the system

3.7.2 System Notifications & Alerts (UDR-05)

Description:
The system shall provide helpful notifications and error messages that clearly describe issues (e.g., "Payment failed: please verify card details" instead of generic error codes).

Rationale:
Improves usability and reduces frustration during errors.

3.8 Licensing, Legal, Copyright, and Other Notices

3.8.1 Licensing Enforcement

The system shall enforce licensing restrictions for any third-party components integrated into the platform (e.g., Stripe API, Google Maps API, Gemini API, Cloudinary, Kafka).

The development team shall comply with each API's Terms of Service (ToS), ensuring usage within free tier quotas or under valid commercial licenses.

Any open-source libraries (Spring Boot, React.js, Tailwind CSS, PostgreSQL, MongoDB, Docker) must be used in accordance with their licenses (Apache 2.0, MIT, GPL, etc.).

The system shall not remove or obscure required license notices included with any open-source distribution.

3.8.2 Legal Disclaimers

The platform will include a Terms of Service (ToS) and Privacy Policy to inform users of rights and responsibilities regarding account usage, data collection, and payment activities.

Disclaimer of Liability: CreatorBoost is not liable for any direct, indirect, or incidental damages resulting from service listings, payments, or external links.

Payment Disclaimer: CreatorBoost does not store credit card details. All payments are securely processed through Stripe under PCI-DSS compliance.

Content Ownership: Sellers (creators/marketing experts) are solely responsible for the legality of their service descriptions, uploaded media, and reviews.

3.8.3 Copyright Notices

Unauthorized duplication, distribution, or modification of platform code, branding, or content is strictly prohibited without prior written consent.

User-generated content (e.g., gig descriptions, images, reviews) remains the intellectual property of the respective users, but CreatorBoost reserves the right to display such content for platform functionality.

3.8.4 Trademarks and Branding

The CreatorBoost logo, name, and associated brand elements shall be protected under applicable trademark laws.

The system shall not allow third parties to use the CreatorBoost logo or name without explicit permission.

3.9 Interfaces

3.9.1 User Interfaces

The CreatorBoost platform will provide intuitive and responsive user interfaces accessible via web browsers. User interfaces will follow responsive web design principles to ensure usability on desktops, tablets, and mobile devices. Key interfaces include:

Authentication Pages

- Login Page: Email/Password login, JWT authentication, OTP verification field
- Registration Page: Input fields for name, email, password, role (User/Creator/Expert)
- Password Reset Page: Email input, OTP/verification, new password

User Dashboards

- Creator Dashboard: Post gigs, manage services, view bookings, track earnings
- Expert Dashboard: Browse available gigs, submit offers, manage clients, track performance
- Admin Dashboard: Manage users, moderate content, approve/reject gigs, view analytics

Gig Management Interfaces

- Add/Edit Gig: Input forms for title, description, platform, category, sub-category, pricing packages, FAQs, and media uploads
- Gig Details Page: Display full gig description, packages, reviews, seller profile

Order & Booking Interfaces

- Order Placement: Select gig, package, confirm booking, payment form
- Order Tracking: View active/completed/cancelled orders with timeline
- Review & Rating Form: Star rating, text input for review

Messaging & Notifications

- Internal chat between buyers and sellers (websocket-based)
- Notification panel for booking updates, payment confirmations, and messages

Navigation

- Global navbar: Home, Browse Gigs, My Dashboard, Messages, Notifications, Settings
- Search bar with filters: Platform, category, price, rating, popularity

3.9.2 Hardware Interfaces

The system is primarily web-based, with no dedicated hardware requirements apart from client devices and server

infrastructure.

Client-Side Requirements:

- RAM: Minimum 4GB
- Processor: Intel i3 or equivalent ARM processor
- Storage: 500MB free disk space for browser cache
- Browser: Chrome, Firefox, or Safari (latest versions)
- Internet: Minimum 2 Mbps connection for smooth media uploads

Server-Side Requirements:

- CPU: Minimum 4 vCPUs
- Memory: Minimum 8GB RAM
- Disk: Minimum 50GB SSD storage (scalable)
- Network: Configurable VPC with load balancer

3.9.3 Software Interfaces

The CreatorBoost platform consists of microservices that interact with each other and with third-party APIs:

Internal Microservice Interfaces:

- Auth Service ↔ All Services: Provides JWT tokens and validates user identities
- Gig Service ↔ Order Service: Provides gig details, package data, pricing
- Order Service ↔ Payment Service: Sends order/payment requests, confirms transaction status
- Order Service ↔ Review Service: Adds and retrieves reviews linked to gigs and orders
- Notification Service: Subscribes to Kafka topics for real-time updates

External APIs:

- Stripe API: Payment processing (REST API, Webhooks)
- Cloudinary API: Media storage and retrieval
- SMTP/Email API : OTP and transactional emails

Database Interfaces:

- PostgreSQL: Auth, Orders, Payments, Reviews
- MongoDB: Gigs, FAQs, Images, Packages

3.9.4 Communications Interfaces

The system will support multiple communication protocols for internal and external integrations:

- RESTful APIs: JSON over HTTPS for synchronous communication between services
- Apache Kafka: Asynchronous event streaming for order events, notifications, and analytics
- WebSockets: For real-time chat and notification delivery

Authentication Protocols:

- JWT tokens for service-to-service and frontend-backend communication

- OAuth2 for external API integration (Stripe)

Ports & Protocols:

- HTTPS (443) for all public-facing endpoints
- Kafka broker communication over TCP (9092)
- Docker container networks configured for internal communication

3.10 Database Requirements

3.10.1 Overview

The platform shall use a polyglot persistence approach:

- PostgreSQL for strongly relational, transactional data (Auth/Users, Orders, Payments, Reviews)
- MongoDB for flexible, document-centric data (Messaging; optional for Gig rich documents if needed)
- Cloudinary for media objects; only URLs & metadata are stored in DBs

Databases shall run in separate instances per microservice (no shared schema) to ensure loose coupling and independent scaling.

3.10.2 Logical Data Model by Service

A) Auth & User Service (PostgreSQL)

Tables:

- users(id UUID PK, email UNIQUE, password_hash, full_name, is_verified BOOL, created_at, updated_at)
- roles(id SERIAL PK, name UNIQUE)
- user_roles(user_id FK->users.id, role_id FK->roles.id, PK(user_id, role_id))
- otp_codes(id UUID PK, user_id FK->users.id, code, expires_at, used BOOL)

Constraints/Requirements:

- Email unique
- Password_hash using strong algorithm (e.g., bcrypt)
- otp_codes.expires_at enforced; delete/expire job required

Indexes:

- users(email)
- otp_codes(user_id)
- user_roles(user_id, role_id)

B) Gig Service (PostgreSQL)

Tables:

- gigs(id UUID PK, seller_id UUID, title, description, platform, category, subcategory, status, created_at, updated_at)
- gig_images(id UUID PK, gig_id FK, url, is_primary BOOL)
- gig_packages(id UUID PK, gig_id FK, name, price NUMERIC(10,2), delivery_days INT, description)
- gig_faqs(id UUID PK, gig_id FK, question, answer)

Constraints/Requirements:

- gig_packages.price >= 0, delivery_days > 0
- status ∈ {ACTIVE, INACTIVE, DRAFT}
- gig_images.is_primary max 1 per gig_id (partial unique index)

Indexes:

- gigs(platform, category)
- gigs(status)
- index on gigs(title, description) for search
- gig_packages(gig_id, price)

C) Order Service (PostgreSQL)

Tables:

- orders(id UUID PK, buyer_id UUID, seller_id UUID, gig_id UUID, package_id UUID, amount NUMERIC(10,2), currency, status, placed_at, updated_at, due_date)

Constraints/Requirements:

- status ∈ {PENDING, ACCEPTED, IN_PROGRESS, DELIVERED, PAID, COMPLETED, CANCELLED}
- amount >= 0

Indexes:

- orders(buyer_id)
- orders(seller_id)
- orders(status)
- orders(placed_at DESC)
- order_events(order_id, created_at)

D) Payment Service (PostgreSQL)

Tables:

- payments(id UUID PK, order_id FK->orders.id, stripe_payment_intent_id UNIQUE, amount NUMERIC(10,2), currency, status, method, created_at, updated_at)
- refunds(id UUID PK, payment_id FK->payments.id, stripe_refund_id UNIQUE, amount NUMERIC(10,2), status, created_at)

Constraints/Requirements:

- status ∈ {PENDING, AUTHORIZED, COMPLETED, FAILED, REFUNDED}
- Idempotency: stripe_payment_intent_id unique, used for retries

Indexes:

- payments(order_id)
- payments(status)
- refunds(payment_id)

E) Review Service (PostgreSQL)

Tables:

- reviews(id UUID PK, order_id FK->orders.id, gig_id UUID, reviewer_id UUID, rating INT, review_text TEXT, created_at)
- gig_ratings(gig_id PK, avg_rating NUMERIC(2,1), rating_count INT, updated_at) (Optional aggregated, maintained via events or triggers)

Constraints/Requirements:

- rating ∈ [1..5]
- One review per order_id (unique)

Indexes:

- reviews(gig_id)
- reviews(reviewer_id)
- reviews(created_at DESC)

F) Messaging Service (MongoDB)

Collections:

- conversations: { _id, memberIds: [UUID], createdAt, lastMessageAt }
- messages: { _id, conversationId, senderId, body, attachments: [{ url, type }], createdAt, readBy: [UUID] }

Indexes:

- conversations.memberIds (multikey)
- messages.conversationId + createdAt
- messages.senderId

3.10.3 Transactions and Consistency

- ACID transactions required for orders, payments, reviews in PostgreSQL
- Read-your-writes consistency for user actions
- Eventual consistency acceptable for analytics and rating aggregates
- Cross-service operations shall use sagas/outbox pattern with Kafka to avoid 2PC
- Example: Order→Payment→Order status update; publish events and reconcile via consumers

3.10.4 Referential Integrity & Constraints

- Foreign keys enforced in PostgreSQL between domain tables (e.g., payments.order_id->orders.id)
- Cascade rules carefully applied: typically restrict delete for auditability
- Soft-delete pattern (is_deleted) where needed
- Constraints: non-negative prices, valid enums, unique (gig_id, is_primary TRUE) for primary image

3.10.5 Performance & Indexing

- Create composite indexes aligned to query patterns (e.g., orders(seller_id, status, placed_at DESC))

3.11 Licensing, Legal, Copyright, and Other Notices

3.11.1 Licensing Enforcement

- The system shall enforce licensing restrictions for any third-party components integrated into the platform (e.g., Stripe API, Google Maps API, Gemini API, Cloudinary, Kafka)
- The development team shall comply with each API's Terms of Service (ToS), ensuring usage within free tier quotas or under valid commercial licenses
- Any open-source libraries (Spring Boot, React.js, Tailwind CSS, PostgreSQL, MongoDB, Docker) must be used in accordance with their licenses (Apache 2.0, MIT, GPL, etc.)
- The system shall not remove or obscure required license notices included with any open-source distribution

3.11.2 Legal Disclaimers

- The platform will include a Terms of Service (ToS) and Privacy Policy to inform users of rights and responsibilities regarding account usage, data collection, and payment activities
- Disclaimer of Liability: CreatorBoost is not liable for any direct, indirect, or incidental damages resulting from service listings, payments, or external links
- Payment Disclaimer: CreatorBoost does not store credit card details. All payments are securely processed through Stripe under PCI-DSS compliance
- Content Ownership: Sellers (creators/marketing experts) are solely responsible for the legality of their service descriptions, uploaded media, and reviews

3.11.3 Copyright Notices

- Copyright 2025 CreatorBoost. All rights reserved
- Unauthorized duplication, distribution, or modification of platform code, branding, or content is strictly prohibited without prior written consent
- User-generated content (e.g., gig descriptions, images, reviews) remains the intellectual property of the respective users, but CreatorBoost reserves the right to display such content for platform functionality

3.12 Legal and Regulatory Standards

- GDPR (General Data Protection Regulation – EU): All user data (personal details, payment information, communications) must be stored and processed in compliance with GDPR requirements for privacy, data retention, and the right to erasure
- CCPA (California Consumer Privacy Act – US): Ensures that US users' data collection and usage comply

with CCPA, including user rights to request access and deletion of personal data
- PCI-DSS (Payment Card Industry Data Security Standard): Required for integration with Stripe API to ensure secure payment processing and protection of financial transactions

3.12.1 Operating System and Cloud Standards

- Docker & Kubernetes Best Practices: For containerization and orchestration of microservices
- Apache Kafka Protocol Standards: Event-driven communication must follow Kafka's message serialization and partitioning standards for scalability and fault tolerance

4. Supporting Information

The supporting information makes this SRS easier to use, reference, and verify. It includes appendices, references, and supplementary details related to the technologies, algorithms, and methodologies applied in developing the CreatorBoost platform.

4.1 Index

Key terms such as Microservices, JWT, Stripe API, Gig Service, Order Service, Review Service, Kafka, AWS, Docker, Authentication, PostgreSQL, MongoDB are indexed for quick lookup.

4.2 Appendices

Appendix A – Use Case Storyboards

User Registration & Authentication A new user signs up using email → system sends OTP verification → user logs in with JWT token.

Gig Creation by Seller Seller creates a new gig by entering title, description, category, packages, FAQs, and uploading images.

Placing an Order Buyer selects gig → chooses package → frontend fetches gig info (gigId, packageId, price) → order is placed via Order Service.

Processing Payment Order Service interacts with Payment Service (Stripe API) → payment is confirmed → order status updated to PAID.

Review Submission After order completion, buyer adds a review (rating, feedback) → Review Service stores and links it to order & gig.

Admin Dashboard Admin manages users, bans malicious accounts, moderates gigs, and generates system reports.

Appendix B – Sample User Interface

- Login Page: Email, password, OTP option, login button
- Gig Details Page: Gig title, description, packages, images, seller info, FAQs
- Order Page: Selected gig, package details, total price, payment button
- Admin Dashboard: User list, gig approval panel, reports section

Appendix C – References

[1] Fiverr - Online Marketplace for Freelance Services. https://www.fiverr.com (Accessed on 14 July 2025)

[2] Upwork - Hire Freelancers. https://www.upwork.com (Accessed on 14 July 2025)

[3] React.js Documentation. https://react.dev (Accessed on 14 July 2025)

[4] PostgreSQL Docs. https://www.postgresql.org (Accessed on 14 July 2025)

[5] "Spring Boot Reference Documentation," Spring.io, [Online]. Available: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/ (Accessed on: July 19, 2025)

[6] "MongoDB Documentation," MongoDB Inc., [Online]. Available: https://www.mongodb.com/docs/ (Accessed on: July 19, 2025)

[7] "Firebase Authentication," Google Firebase, [Online]. Available: https://firebase.google.com/docs/auth (Accessed on: July 19, 2025)

[8] "Stripe API Reference," Stripe Inc., [Online]. Available: https://stripe.com/docs/api (Accessed on: July 19, 2025)

[9] "Google Maps Platform Documentation," Google Cloud, [Online]. Available: https://developers.google.com/maps/documentation (Accessed on: July 19, 2025) 10

[10] "Gemini API by Google DeepMind," Gemini – Google AI, [Online]. Available: https://deepmind.google/technologies/gemini/ (Accessed on: July 19, 2025)

[11] "Docker Documentation," Docker Inc., [Online]. Available: https://docs.docker.com/ (Accessed on: July 19, 2025)

[12] Tailwind Labs, "Tailwind CSS Documentation." [Online]. Available: https://tailwindcss.com/docs(Accessed on: July 19, 2025)

[13] OpenJS Foundation, "Node.js Documentation." [Online]. Available: https://nodejs.org/en/docs(Accessed on: July 19, 2025)

[14] Supabase, "PostgreSQL and Supabase Documentation." [Online]. Available: https://supabase.com/docs(Accessed on: July 19, 2025)