Project Specification: SkillForge

Status: Draft / In-Development

1. Project Overview

SkillForge is a web platform that facilitates the exchange of skills between users via a virtual credit system. The platform is built on a **Unified User Model**, meaning every user can act as both a learner and an instructor. Users create profiles to list skills they can teach (e.g., Java, Graphic Design) while simultaneously using their credits to book sessions to learn new skills.

1.1 Terminology Note

- **User:** A registered account holder on the platform.
- **Student (Role):** Refers to a User who is currently searching for or booking a learning session.
- **Teacher (Role):** Refers to a User who is currently listing a skill or conducting a session.

2. Actors and Roles

Actor	Description
User	The core participant. Automatically possesses capabilities to both teach and learn.
Admin	System administrator responsible for dispute resolution and platform health.
Support Staff	Responsible for handling tickets and user inquiries.

3. Functional Requirements

Requirements are grouped by architectural module.

3.1 User Management & Authentication

Requirement Description	Priority
The system must allow users to register using Email and Password.	**High**
Unified Access: Registration must create a single identity that allows the user to access both "Teach" and "Learn" dashboards without re-logging.	**High**
Upon registration, the system must grant the user a "Welcome Bonus" (Set Amount of Credits).	**Medium**
Users must be able to generate a unique Referral Link. If a new user signs up via this link, both parties receive bonus credits.	**Low**

3.2 Skill Discovery & Search (The "Teach" Context)

Requirement Description	Priority
Any User must be able to create "Skill Listings" (e.g., "I teach Java") with a description and tags.	**High**
Users (acting as Students) must be able to search for listings by keyword and filter by the listing owner's Reputation Score.	**High**
Users (acting as Students) must be able to look at reviews (if there are any) of a the teacher's listing.	**High**
Pre-Booking Chat: Students must be able to initiate a chat with a Teacher directly from their profile to discuss requirements before requesting a slot.	**Medium**
The system must display a Public Profile for every user, showing their offered skills, bio, and reviews received as a teacher or as a student.	**Medium**

3.3 Scheduling & Sessions

Requirement Description	Priority
A User (acting as Student) must be able to request a specific time slot from another User's (Teacher's) availability calendar.	**High**
Negotiation: If a requested slot is inconvenient, the Teacher must be able to **propose an alternative time**, sending the request back to the Student for approval.	**Medium**
The system must prevent "Double Booking" (a user cannot be scheduled to teach and learn at the same time).	**High**
Session Lifecycle: The system must manage state transitions: `Requested` -> `Counter-Offered` -> `Accepted` -> `Completed` -> `Disputed`.	**High**
Upon `Accepted` status, the system must enable the exchange of video links (Zoom/Meet) within the chat.	**Medium**
Request Expiration: If a session request remains in `Requested` or `Counter-Offered` state for more than **48 hours**, the system must automatically expire the request, refund any locked credits, update the status to `Expired`, and notify the Student.	**High**
Teacher Cancellation: A Teacher may cancel a scheduled session at any time. Upon cancellation, the Student must receive a **full refund** of locked credits. The Teacher receives a cancellation warning.	**High**
Cancellation Warnings: After **5 cancellations**, the system must display a notice on the Teacher's public profile indicating their cancellation frequency (e.g., "This teacher has cancelled X sessions in the last 30 days").	**Medium**
No-Show Reporting: Either party may report a no-show if the other party fails to attend within a **30-minute grace period** from the scheduled start time.	**High**
No-Show Resolution: When a no-show is reported, the session is marked as `Cancelled` and locked credits are **refunded to the Student**. No credits are transferred to the Teacher.	**High**

3.4 Credits & Wallet (The Ledger)

Requirement Description	Priority
Escrow Logic: When a session is scheduled, credits must be deducted from the requesting User and held in a "Locked" state.	**High**
Upon Session Completion, locked credits must be transferred to the teaching User's available balance.	**High**
Users must be able to purchase credit packs via a Payment Gateway mock (e.g., Stripe API).	**Low**
The system must record an immutable Audit Log of every credit transaction (Earned, Spent, Purchased).	**High**
Insufficient Credits Prevention: If a User has insufficient credits to book a session, the booking button must be **disabled (greyed out)** in the UI. The system must display the credit shortfall (e.g., "You need 5 more credits") and prompt the user to purchase credits.	**High**

3.5 Reputation & Reviews

Requirement Description	Priority
After a session, the Student must be able to rate the Teacher (1-5 Stars) and leave a text review.	**High**
Bidirectional Ratings: After a session, the Teacher may rate the Student (1-5 Stars) and leave a text review.	**Medium**
Reputation Score: The system must calculate and display a Reputation Score based on ratings received. This score must be visible on the public profile.	**Medium**

3.6 Disputes & Resolution

Requirement Description	Priority
Raising a Dispute: Either the Student or the Teacher may raise a dispute within **3 days** after a session has been marked as `Completed`.	**High**
Dispute Grounds: Disputes must be based on predefined platform rules that both parties agree to upon registration (e.g., session quality, content mismatch, unprofessional conduct).	**High**
Dispute Review: When a dispute is raised, the session status transitions to `Disputed` and an Admin is notified for review.	**High**
Admin Resolution Actions: An Admin reviewing a dispute may take one or more of the following actions: issue a full refund to the Student, issue a partial refund, transfer credits to the Teacher, issue bonus credits to the affected party as compensation, issue a warning to either party, or suspend a User account.	**High**
Audit Logging: All dispute resolutions and Admin actions must be recorded in the immutable Audit Log.	**High**

Requirement Description	Priority
Appeal Process: A User may appeal an Admin's dispute resolution by sending an email to the support team.	**Low**

4. Non-Functional Requirements (Quality Attributes)

4.1 Data Strategy & Persistence

- **Hybrid Storage:**
  - **Relational (SQL):** User Profiles, Wallet Balances, Session metadata (Strict consistency).
  - **NoSQL (Document):** Chat history, Activity/Audit Logs (High volume, unstructured).
- **Integrity:** Financial calculations must use precise decimal types to avoid floating-point errors.

4.2 Security

- All user passwords must be hashed before storage.
- Administrative endpoints must be protected via Role-Based Access Control.
- All data in transit must be encrypted via HTTPS.

4.3 Performance & Scalability

- Search queries must return results in under 2 seconds for up to 50 concurrent users.
- The backend architecture must follow a **Modular Monolith** pattern to allow for clean separation of concerns (Billing vs. Scheduling).

4.4 Availability

- The system should be available 99% of the time during business hours.
- Failure of external APIs (e.g., Payment Gateway) must not crash the core application (Graceful Degradation).

4.5 Real-Time Communication

- The chat system must use WebSocket connections for real-time message delivery between users.
- Session status changes must trigger **real-time notifications** to users.
- The system must gracefully fall back to polling if WebSocket connections fail.

4.6 Responsiveness & Cross-Platform

- The web application must be **responsive** and usable on desktop, tablet, and mobile browsers.
- The interface must support the following browsers: Chrome, Firefox, Safari, and Edge (at least last 2 major versions).

4.7 Observability & Monitoring

- The system must log all critical operations (session requests, credit transactions, disputes) with timestamps and user IDs.
- Logs must be structured (JSON format) to enable easy querying and analysis.
- The system must expose **health check endpoints** for monitoring service availability.

- Errors must be logged with sufficient context for debugging (stack traces, request IDs).

4.8 Reliability & Recovery

- The system must implement **automatic retry logic** for transient failures (e.g., database timeouts, payment API errors).
- Database transactions involving credits should respect ACID rules.
- The system should support **daily database backups** with a Recovery Point Objective (RPO) of 24 hours.

4.9 Testability

- The codebase must maintain a minimum of **70% unit test coverage** for business logic (session lifecycle, credit operations).
- All state transitions must have corresponding test cases to prevent regression.
- External dependencies (Payment Gateway, Email Service) must be abstracted behind interfaces to enable mocking in tests.

4.10 Maintainability

- Code must follow consistent naming conventions and be documented with inline comments for complex logic.
- API Swagger documentation must be available at a discoverable endpoint and kept up-to-date with all REST endpoints.

---

5. Technology Constraints

- **Frontend:** Web Application (Browser based).
- **Backend:** Modular Monolith application exposing REST APIs.
- **External Dependencies:** Zoom/Google Meet (for video), Stripe (Mock for payments).