# **SkillSwap Mobile Application — Software Requirements Specification (SRS)**

## **1. Introduction**

### **1.1 Purpose**

This document defines the complete set of requirements for the SkillSwap mobile app. SkillSwap is a peer‑to‑peer skill exchange platform for university students. Users can offer skills and learn from others in return. The goal is to make it easy to discover, book, and complete skill sessions with no money involved.

### **1.2 Scope**

**In scope (MVP):**

* Account creation and secure authentication
* User profiles with skills to **offer** and skills to **learn**
* Search and discovery for skill offers and requests
* Booking and scheduling of sessions
* Notifications and activity history
* Ratings, reviews, and reporting
* Basic admin moderation

**Out of scope (MVP):** in‑app payments, voice/video calls, real‑time chat, advanced reputation badges, multi‑language support.

### **1.3 Definitions and Acronyms**

* **Tutor:** A user offering a skill.
* **Learner:** A user seeking a skill.
* **SkillSwap:** Trading skills without money.
* **CRUD:** Create, Read, Update, Delete.
* **MVP:** Minimum Viable Product.
* **UML:** Unified Modeling Language.

### **1.5 Overview**

Sections 2–7 describe users, features, constraints, data, quality attributes, and release criteria. Appendices include UML diagrams and API sketches to help development.

## **2. Overall Description**

### **2.1 Product Perspective**

SkillSwap is a new standalone mobile app. The backend provides REST APIs and notifications. The client is built with React Native. Data is stored in a cloud database. Authentication uses email and password (with OAuth optional later).

### **2.2 User Roles and Goals**

* **Student (default role):** Can act as Tutor and Learner. Create profile, list skills, request help, search, book, rate, review, and report.
* **Admin:** Moderate content, handle reports, suspend or delete accounts if needed.

### **2.3 User Stories**

* As a learner, I want to filter tutors by skill, rating, and time so I can find a reliable match.
* As a tutor, I want to set available time slots so learners can only book me when I am free.
* As a student, I want to see my upcoming and past sessions so I can plan my week.
* As an admin, I want to review reports and remove abusive content to keep the community safe.

### **2.4 Assumptions and Dependencies**

* Users have university email addresses.
* Users agree to a code of conduct and terms of use.
* Push notifications are available through platform services.
* Internet connection is required for core flows.

### **2.5 Constraints**

* Personal data must be protected under relevant privacy laws.
* Average screen load time under two seconds on standard Wi‑Fi.
* System uptime target 99.9%.

## **3. Functional Requirements**

### **3.1 Authentication and Accounts**

* **FR‑A1** Users can register with email and password.
* **FR‑A2** Users can log in, log out, and reset passwords.
* **FR‑A3** Email verification is required before full access.
* **FR‑A4** Sessions are secured with tokens. Expired tokens require re‑authentication.

### **3.2 Profiles and Skills**

* **FR‑P1** Users can create and edit a profile with name, photo, bio, university, program, year, location.
* **FR‑P2** Users can list skills they **offer** (title, description, category, duration, level) and skills they **want to learn**.
* **FR‑P3** A profile shows average rating, total reviews, completed sessions, and availability summary.

### **3.3 Discovery and Search**

* **FR‑D1** Users can browse a feed of offers with search by keyword, category, level, and location.
* **FR‑D2** Users can sort by relevance, rating, or most recent.
* **FR‑D3** Users can filter by time availability that matches their own.

### **3.4 Availability and Booking**

* **FR‑B1** Tutors can publish weekly availability blocks and session duration.
* **FR‑B2** Learners can request a slot from an offer page.
* **FR‑B3** The system validates conflicts and confirms booking.
* **FR‑B4** Both parties receive notifications for requests, confirmations, reminders, and changes.
* **FR‑B5** Users can cancel or reschedule under simple rules (e.g., no penalty if > 12 hours).

### **3.5 Sessions and History**

* **FR‑S1** A session has status: requested, confirmed, completed, canceled.
* **FR‑S2** Users see a calendar and a list of upcoming and past sessions.
* **FR‑S3** Optional notes field for pre‑session context.

### **3.6 Ratings, Reviews, and Reporting**

* **FR‑R1** After completion, each side can rate (1–5) and write a short review.
* **FR‑R2** Average rating updates on the profile.
* **FR‑R3** Users can report a profile, offer, or review for abuse. Reports go to Admin.

### **3.7 Admin and Moderation**

* **FR‑M1** Admin can view users, offers, sessions, reviews, and reports.
* **FR‑M2** Admin can remove content and suspend or delete accounts.
* **FR‑M3** Admin can export basic metrics.

### **3.8 Notifications**

* **FR‑N1** Push notifications for booking lifecycle and reminders.
* **FR‑N2** Email fallbacks when push fails.

### **3.9 Data Management**

* **FR‑G1** All data stored in a persistent database.
* **FR‑G2** Soft delete for users and offers to allow recovery.
* **FR‑G3** Audit fields: createdAt, updatedAt, createdBy, updatedBy.

## **4. Non‑Functional Requirements**

### **4.1 Usability**

* Clear navigation and consistent UI. A user should be able to post an offer within three taps after login.
* Accessible touch targets and readable typography.

### **4.2 Performance**

* P95 screen load under 2 seconds on Wi‑Fi. API round‑trip under 600 ms P95.
* Support 5k DAU without noticeable slowdowns in MVP.

### **4.3 Security**

* Passwords hashed and salted (e.g., bcrypt/Argon2).
* All traffic over HTTPS. JWT with short TTL and refresh.
* Server‑side validation and rate limiting on auth and search.

### **4.4 Reliability and Availability**

* Target 99.9% uptime monthly. Automatic daily backups with 7‑day retention.
* Graceful degradation if notifications fail.

### **4.5 Privacy and Compliance**

* Minimize stored PII. Clear terms and code of conduct. Opt‑in analytics.

### **4.6 Maintainability and Scalability**

* Layered architecture with well‑defined APIs.
* CI checks, linting, tests, and staged deployments.

## **5. Data Model (Logical)**

### **5.1 Core Entities**

* **User**: id, email, passwordHash, name, photoUrl, bio, university, program, year, location, roles[], avgRating, reviewCount, stats{completedSessions}, availability[]
* **SkillOffer**: id, tutorId, title, description, category, level, durationMins, tags[], isActive, createdAt, updatedAt
* **SkillRequest** (optional future): id, learnerId, title, description, desiredCategory, desiredLevel
* **Session**: id, offerId, tutorId, learnerId, startTime, endTime, status, notes
* **Review**: id, sessionId, fromUserId, toUserId, rating, comment, createdAt
* **Report**: id, reporterId, targetType, targetId, reason, details, status, createdAt
* **AdminAction**: id, adminId, actionType, targetType, targetId, reason, createdAt

### **5.2 Relationships**

* User 1‑N SkillOffer
* User 1‑N Review (as author)
* User 1‑N Session (as tutor or learner)
* SkillOffer 1‑N Session
* Session 0‑2 Review

## **6. System Architecture (High Level)**

* **Client:** React Native app (Expo). Screens: Auth, Home/Feed, Search/Filters, Offer Details, Create/Edit Offer, Availability, Booking, Calendar/History, Profile, Settings.
* **API:** REST JSON over HTTPS. Auth, Users, Offers, Sessions, Reviews, Reports.
* **Data:** Cloud database (MongoDB or Firestore). Storage for images.
* **Notifications:** FCM/APNs via backend.

## **7. API Sketch (MVP)**

* POST /auth/register, POST /auth/login, POST /auth/forgot
* GET /users/:id, PATCH /users/:id
* GET /offers?search=&category=&level=&near=, POST /offers, PATCH /offers/:id, DELETE /offers/:id
* POST /sessions, GET /sessions?role=&status=, PATCH /sessions/:id, DELETE /sessions/:id
* POST /reviews, GET /users/:id/reviews
* POST /reports, GET /admin/reports, PATCH /admin/reports/:id

## **8. Acceptance Criteria (Samples)**

* A new user can register, verify email, create a profile, post an offer, and receive a booking within one session.
* Booking prevents double‑booking of the same time slot.
* After a session is marked completed, both users can submit a single review.
* Admin can remove an offer and it no longer appears in search.

## **9. Risks and Mitigations**

* **Low liquidity:** Seed with campus ambassadors and sample offers.
* **Trust and safety:** Clear reporting, moderation, and code of conduct.
* **Scheduling conflicts:** Strong validation and calendar UX.

## **10. Release Plan (MVP)**

* **Sprint 1:** Auth, Profile, Create Offer
* **Sprint 2:** Search, Offer Details, Availability
* **Sprint 3:** Booking, Notifications
* **Sprint 4:** Reviews, Reports, Admin console (basic)

## **Sample Data (for prototyping)**

export const dummyOffers = [

{ id: '1', title: 'Data Structures Help', user: 'Aisha', rating: 4.9 },

{ id: '2', title: 'Public Speaking Coaching', user: 'Bilal', rating: 4.8 },

{ id: '3', title: 'Poster Design in Figma', user: 'Hira', rating: 4.7 },

{ id: '4', title: 'Guitar Basics', user: 'Usman', rating: 4.6 },

];

# **Appendix**

## **A. Requirements Traceability Matrix (sample)**

| **ID** | **Requirement** | **Source** | **Design Artifact** |
| --- | --- | --- | --- |
| FR‑B3 | Validate slot conflicts | Booking story | Sequence diagram C |
| FR‑R1 | Post‑session review | Reviews story | Class model Review |
| NFR‑SEC | Password hashing and HTTPS | Security | Architecture §6 |

## **B. Test Cases (samples)**

* Register → Verify email → Login → Create offer → Book from a second account → Complete → Review → Rating updates.
* Try to double‑book the same slot. Expect validation error.
* Report abusive review. Admin removes it. It no longer appears.