**Skill Swap App**

****

**Submitted by:  
 Tayyab Fareed (8831)**

**Supervised by:**

**Dr .Sadaf Tanvir**

**Department of Computing**

**Abasyn University Islamabad Campus**

**1. Introduction**

**1.1 Purpose**

This SRS describes the requirements for SkillSwap, a mobile app that lets university students trade skills peer-to-peer (no money involved). The goal is to let learners find short help (e.g., Data Structures tutoring) and offer something they can do in return (e.g., poster design).

**1.2 Scope**

SkillSwap is a React Native mobile application (MVP) that will allow students to:

* Create profiles and list skills they can teach.
* Post requests for skills they want to learn.
* Browse/filter skill offers and requests.
* Book session times and manage a simple schedule.
* Rate and review after a session.

**Out of scope for MVP:**

* In-app payments, video calls, or complex chat features.
* Enterprise integrations, advanced analytics.

**1.3 Definitions / Acronyms**

* Tutor: User offering a skill.
* Learner: User requesting a skill.
* Offer: A posted skill that someone wants to teach.
* Request: A posted skill someone wants to learn (can be implemented as inverted Offer).
* MVP: Minimum Viable Product.

**2. Overall Description**

**2.1 Product Perspective**

SkillSwap is a standalone mobile app that uses a backend (e.g., Node + Express + MongoDB) to store users, offers, sessions, and reviews. The frontend is a React Native app with simple navigation (Login → Home feed → Offer details → Booking → Profile).

**2.2 User Roles & Goals**

* Student (default role) — can be tutor or learner. Create/edit profile, post offers, book sessions, leave reviews.
* Admin — moderate content, remove bad actors, manage reports.

**2.3 Assumptions & Dependencies**

* Users are university students (age 16+).
* App requires internet connection for booking and profile sync.
* Backend services (auth, database) are available and secured.

**3. User Stories**

* As a learner, I want to filter skill offers by course (e.g., Data Structures) and rating.
* As a tutor, I want to set available time slots.
* As a user, I want a simple booking confirmation and calendar view of upcoming sessions.
* As an admin, I want to remove inappropriate offers.

**4. Functional Requirements**

List of priority requirements for the MVP, written as short FR statements:

1. FR1 — Account Management: Users can register, login, logout.
2. FR2 — Profile CRUD: Users can create/edit profile (name, bio, skills, profile picture).
3. FR3 — Post Offer: Users can create a skill offer (title, description, category, duration, location/online flag).
4. FR4 — Search & Filter: Users can search offers by keyword, category, and sort by rating or distance.
5. FR5 — Booking: Users can request a session at an available time slot of a tutor.
6. FR6 — Notifications: Tutor and learner receive notifications (push or in-app) on booking.
7. FR7 — Reviews: After a session, users can rate and write a review.
8. FR8 — Admin Actions: Admin can suspend accounts, delete offers, and view reports.
9. FR9 — Persistence: All data (users, offers, sessions, reviews) are saved in the database.
10. FR10 — Reporting: Users can report inappropriate content or behavior.

**5. Non-Functional Requirements**

* Usability: Simple onboarding. Posting an offer should be possible in 3 taps or less (goal).
* Performance: Screens load < 2s on standard Wi‑Fi.
* Security: Passwords hashed & salted; HTTPS for APIs.
* Reliability: Backups scheduled daily.
* Privacy: Only minimal public info is shown on profiles; email hidden unless user shares it.

**6. Data Model & Database Schema (MVP)**

**Users**

* \_id (ObjectId)
* email (string)
* passwordHash (string)
* name (string)
* bio (string)
* profilePicUrl (string)
* skills (array of {skillName, level, hourlyEquivalent?})
* avgRating (number)
* availableSlots (array of ISO datetime ranges)
* role (string) // e.g., "student" or "admin"

**Offers**

* \_id
* title (string)
* description (string)
* category (string)
* durationMinutes (number)
* createdBy (userId)
* createdAt (ISODate)
* location (string or geo)
* slots (array of available times)
* isRequest (boolean) // true if user requests to learn

**Sessions**

* \_id
* offerId
* tutorId
* learnerId
* scheduledTime (ISODate)
* status (string) // requested, confirmed, completed, cancelled

**Reviews**

* \_id
* fromUser (userId)
* toUser (userId)
* rating (1-5)
* comment (string)
* createdAt

**7. MVP Frontend Screens (short description)**

* Login / Signup: email, password, basic validation.
* Home Feed: list of offer cards.
* Community: offer details.
* Swaps: form with title, status, exchange.
* Profile: user info, skills, activities.
* Conversations: Conversations b/w tutor and learner.

**8. Test Cases**

1. TC1 — Registration: Valid email/password -> expect success and redirect to profile setup.
2. TC2 — Create Offer: Fill required fields -> expect offer appears in feed.
3. TC3 — Booking: Learner books available slot -> expect session status "requested" and notification to tutor.
4. TC4 — Review: After session complete, review updates.