**ASSIGNMENT NO 1**

**MOBILE APPLICATION**

**Software Requirements Specification (SRS)**

**Author:** Hamna Hoor Khan  
**Date:** 2025-09-25  
**Reg no:** 8408

**1. Purpose**

This SRS describes what the **SkillSwap** mobile app must do and how it should behave. SkillSwap is a peer-to-peer mobile application where university students post skills they can teach and find other students to teach them — all without money. This document explains the features, data, constraints and acceptance criteria so developers and testers can build the MVP and later expansions.

**2. Scope — what this SRS covers**

Included for the MVP:

* User authentication (simple login for prototype).
* User profiles (name, bio, skills).
* Posting and listing skill offers (create / read / delete).
* Basic booking placeholder (UI and session records).
* Ratings & reviews (post-session).
* Admin moderation (delete/report content).
* A React Native frontend prototype containing Login, HomeFeed, CreatePost, Profile screens (the code you provided maps to these screens).

Not included in MVP:

* In-app payments or marketplace.
* Real-time chat or video calling.
* Advanced search with geo-location routing (can be added later).

**3. Definitions & terms**

* **Tutor**: user offering a skill.
* **Learner**: user requesting a skill.
* **Offer**: a posted skill (title, description, duration, tutor).
* **Session**: a booked exchange between users.
* **Admin**: platform moderator.

**4. Stakeholders**

* Students (primary users — both tutors and learners).
* University admin / course staff (may use admin interface).
* Developers & testers building the MVP.

**5. Key user stories**

* *As a Learner*, "I want to find a Python tutor with 4+ rating so I can finish my assignment."
* *As a Tutor*, "I want to post my guitar lessons and set available time slots."
* *As an Admin*, "I want to remove inappropriate offers quickly."

**6. Functional requirements (explicit, testable)**

**FR1 — Registration & Login**

* The system shall allow users to register (email + password) and login.
* *MVP:* support login with a test account (email test@student.com, password 12345), as in your prototype.

**FR2 — Profile management**

* Users can create and edit name, bio, list of skills, and a profile picture.
* Average rating is displayed on profile.

**FR3 — Skill Offers (CRUD)**

* Users can create, view, edit, and delete skill offers. Offer fields: id, title, description, category, duration, createdBy, createdAt.

**FR4 — Browse & Search**

* Users can browse offers (sorted by newest or rating).
* *MVP:* keyword search and simple filtering by category.

**FR5 — Booking / Session**

* Users can request/book a session for an offer and select a time slot. System records a Session with status (requested, confirmed, completed, cancelled).

**FR6 — Notifications**

* The system shall send notification (push/email) for booking confirmations and cancellations.
* *MVP:* simple in-app toast or alert; push can be added later.

**FR7 — Ratings & Reviews**

* After a completed session, learner can rate and write a review for the tutor. Average rating updated.

**FR8 — Admin controls**

* Admins can view all users and offers and delete inappropriate content.

**FR9 — Reporting**

* Users can report a profile or offer for review.

**FR10 — Persistence**

* All data persists in a database (MVP may use local storage or a simple backend).

**7. Non-functional requirement**

* **Usability:** Non-technical students should post an offer within ~3 taps from the Home screen. Clear labels and error messages.
* **Performance:** Screen load ≤ 2s on standard Wi-Fi.
* **Security:** Passwords stored hashed & salted. Communication over HTTPS for any remote API.
* **Reliability:** Persistent store for offers & sessions; data backed up.
* **Scalability:** The design should allow moving from local prototype to a server + DB.
* **Accessibility:** Text sizes readable; buttons large enough for mobile tapping.

**8. Data model (suggested collections / tables)**

**Users**

* \_id (string)
* email (string)
* passwordHash (string)
* name (string)
* bio (string)
* skills (array of strings)
* profilePicUrl (string)
* avgRating (number)

**Offers**

* \_id
* title
* description
* category
* duration (minutes)
* createdBy (Users.\_id)
* createdAt
* availableSlots (optional array)

**Sessions**

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

**Reviews**

* \_id
* fromUser
* toUser
* rating (1–5)
* comment
* createdAt

**9. System architecture**

* **Mobile client** (React Native) — screens: Login, HomeFeed, CreatePost, Profile. (Your code implements these screens.)
* **Optional backend API** — REST endpoints to manage users, offers, sessions, reviews.
* **Database** — NoSQL (MongoDB) or relational DB. For the prototype, local state (React useState) or AsyncStorage is acceptable.

**Example REST endpoints (for later)**

* POST /api/auth/login
* GET /api/offers
* POST /api/offers
* POST /api/sessions
* POST /api/reviews
* DELETE /api/offers/:id (admin)

**10. UI / Screen details — what each screen must do (and acceptance criteria)**

**Login Screen**

* Inputs: email, password.
* Success: if creds match test account, navigation.replace('Home').
* Error message for invalid creds.
* *Acceptance:* Logging in with test@student.com / 12345 navigates to Home.

**HomeFeed Screen**

* Displays list of offers as cards (title + tutor).
* Buttons: Create Post (navigates to CreatePost), Profile.
* *Acceptance:* New offers posted are visible on Home immediately.

**CreatePost Screen**

* Inputs: title, user name (and optionally description/category).
* Submit creates new offer with unique id (example uses Date.now().toString()), adds to offers list, shows success alert and navigates back.
* *Acceptance:* After submission, new offer appears in HomeFeed.

**Note about implementation detail:** passing offers and setOffers through navigation params works for a quick prototype (as you have), but for stability / scale use React Context or a global store so screens don't rely on route params. This prevents errors when route.params is undefined.

**Profile Screen**

* Shows name, skills and bio.
* *Acceptance:* Editable in later iteration.

**11. Acceptance tests (examples)**

* **AT-01** Login with correct test credentials → lands on HomeFeed.
* **AT-02** Create a new offer with title & name → success alert shown → HomeFeed list length increases by 1 with the new title visible.
* **AT-03** Delete an offer (admin flow) → offer removed from list.
* **AT-04** After marking a session completed, learner can leave a rating; avgRating updates.

**12. Security & privacy**

* Do not store plaintext passwords. Hash + salt on server.
* If using third-party analytics, inform users in README (privacy).
* Provide a “report user” flow for harassment/content moderation.

**13. Constraints, assumptions & risks**

* **Constraint:** MVP uses local state (no server). This limits multi-device sync and real bookings.
* **Assumption:** Users are students comfortable installing an app; no heavy security is required for prototype.
* **Risk:** Passing state through navigation can cause undefined errors if navigation params are missing (use safe defaults or Context). Mitigation: add checks and fallbacks (your route?.params?.offers || [] is good).

**14. MVP features & priorities**

**Must have (MVP):**

1. Login (test account)
2. HomeFeed with list of offers
3. CreatePost (add offers)
4. Profile view
5. Local persistence option (AsyncStorage) or in-memory for demo

**Should have (next):**

* Ratings & simple booking placeholder
* Search & filter

**Nice to have (post-MVP):**

* Real backend + auth, push notifications, messaging, geo/availability calendar.

**15. UML diagrams to include with submission (what to draw)**

* **Use Case Diagram:** Actors — Student (Tutor/Learner), Admin; use cases: Login, Post Offer, Book Session, Rate User, Report Content.
* **Class Diagram:** Classes: User, Offer, Session, Review with attributes and relationships.
* **Sequence Diagram:** Booking flow (Learner requests → Tutor confirms → Session created).
* **Activity Diagram:** Create Offer flow.

(Place these diagrams in the A1 folder when you push to GitHub.)

A1 (1)

**16. Developer notes**

* **IDs:** use Date.now().toString() for quick unique IDs in prototype or install uuid for robust IDs.
* **State management:** for reliability replace route.params passing of setOffers with React.createContext + provider or use navigation setParams sparingly.
* **Keyboard:** wrap CreatePost form in KeyboardAvoidingView (you already do — good!).
* **Alerts:** for smoother UX replace Alert.alert with a non-blocking toast later.
* **Persistence:** to show offers persistently across app restarts, use AsyncStorage in the MVP or connect to a simple backend.

**17. Delivery checklist**

* SRS document (this file) — place in A1/ on GitHub.
* UML diagrams images/PDFs in A1/.
* React Native MVP code in skillswap-mvp repo main branch. Include README.md with run instructions (how to run with Expo or React Native CLI).
* Make sure the Create Post screen in your code updates the Home feed (you already pass offers and setOffers when navigating — ensure the route name matches registration in your navigator).

**18. Quick example: acceptance for CreatePost**

1. Open app → Login with test@student.com / 12345.
2. Home shows 4 initial offers.
3. Tap Create Post. Fill Title French Lessons, Name Rania. Tap Submit.
4. Alert Success shown; app returns to Home.
5. Home now lists French Lessons at top — test passed.