1. Project Overview

Project Name: SkillSwap (MVP)
Concept: A peer-to-peer educational marketplace where individuals exchange skills directly (e.g., "I teach you Python, you teach me Guitar"). No currency is exchanged.
Core Value: Democratizing education through mutual exchange. "Learn by Teaching."

---

2. Functional Requirements & "Skill Economy" Logic

2.1 The Swap Lifecycle

1. **Proposal:** User A posts a card: "Offering [Skill X], Seeking [Skill Y]."
2. **Discovery:** User B searches/filters and finds the card.
3. **Request (Pitch):** User B clicks "Request" and writes a pitch message (e.g., "I'm an expert in Y, let's chat").
4. **Negotiation:** User A and User B chat to agree on time/modality.
5. **Agreement:** User A accepts the application. The Proposal becomes "In Progress," and a **Swap** record is created.
6. **Completion:** Both mark the swap as done.
7. **Review & Endorsement:** Both leave a rating. If positive, the skill taught is added to the teacher's profile as a "Verified/Endorsed" skill.

2.2 Profile Integrity

- **Manual Skills:** Users can add any skill manually.
- **Endorsed Skills:** Skills gained/verified via completed swaps.
- **Visibility:** Users cannot delete Endorsed Skills (to prevent gaming the system), but they can **Hide** them from their public profile.
- **Identity:** Phone number is **not** required for signup (to reduce friction) but reserved for a future verification step.

---

3. User Experience (UX) Architecture

3.1 Views & Pages

- **Landing Page (Public):**
    - Hero Section with Value Prop.
    - **Instant Search Bar** (allows exploration before signup).
    - Login/Signup (Email/Pass -> Profile Setup).
- **Dashboard (Main View):**
    - **Search & Filter Bar:** Filter by "Skill Needed," "Skill Offered," "Remote/In-Person."
    - **Grid View:** Cards showing the split visual (Offering vs. Seeking).
    - **Tabs:** [Browse] | [My Proposals] | [Active Swaps].
- **Proposal Detail (Modal):**
    - Expanded details.
    - **Action:** "Request/Message" (Opens pitch input).
- **Profile Page:**
    - Header: Avatar, Name, Industry, Stats (Swaps, Endorsements, Skill Count).
    - Skill List: Distinguishes between Verified and Manual skills.
    - Edit Mode: Ability to hide skills or edit bio.

3.2 Owner View (Managing Requests)

When a user views their *own* proposal:

- **Button:** "Rescind Offer" (Deletes proposal if no active swap).
- **Button:** "View Applicants" (instead of "Request").
    - Shows list: Applicant Name, Avatar, Skills, and **Pitch Message**.
    - Action: [Chat] or [Accept].

---

4. Technical Architecture

4.1 Stack

- **Framework:** Next.js (App Router)
- **Language:** TypeScript
- **Pattern:** MVC (Model-View-Controller)
- **Styling:** Tailwind CSS (Shadcn UI components recommended).

4.2 Infrastructure

- **Database:** PostgreSQL (Hosted on **Supabase**).
- **ORM:** Prisma.
- **Auth:** Supabase Auth (or NextAuth).
- **Storage:** Supabase Storage (for Profile Pics/Proposal Images).

---

5. Data Model (Schema)

This SQL schema (via Prisma) manages the relationships required for the marketplace.

codePrisma

```
// prisma/schema.prisma

model User {
  id            String    @id @default(cuid())
  email         String    @unique
  name          String
  industry      String?
  bio           String?
  avatarUrl     String?
  createdAt     DateTime  @default(now())

  // Relations
  skills        UserSkill[]
  proposals     Proposal[]    @relation("ProposalOwner")
  applications  Application[] @relation("Applicant")

  // Chat & Swaps
  sentMessages    Message[]   @relation("Sender")
  receivedMessages Message[]  @relation("Receiver")
  swapsAsTeacher  Swap[]      @relation("Teacher")
```

```
  swapsAsStudent  Swap[]      @relation("Student")
}

model Skill {
  id            String    @id @default(cuid())
  name          String    @unique // e.g., "Python", "Guitar"

  users         UserSkill[]
  neededIn      Proposal[]
  offeredIn     Proposal[] @relation("OfferedSkills")
}

// Junction Table: User <-> Skill
model UserSkill {
  id        String   @id @default(cuid())
  userId    String
  skillId   String
  user      User     @relation(fields: [userId], references: [id])
  skill     Skill    @relation(fields: [skillId], references: [id])

  source    SkillSource @default(MANUAL) // MANUAL or ENDORSED
  isVisible Boolean     @default(true)   // User can hide verification
  endorsementCount Int  @default(0)

  @@unique([userId, skillId])
}

enum SkillSource {
  MANUAL
  ENDORSED
}

model Proposal {
  id          String   @id @default(cuid())
  ownerId     String
  owner       User     @relation("ProposalOwner", fields: [ownerId],
references: [id])

  title       String
```

```
  description String
  modality    String   // "Remote", "In-Person"
  status      ProposalStatus @default(OPEN)

  offeredSkills Skill[] @relation("OfferedSkills")
  neededSkills  Skill[]
  applications  Application[]
  swaps         Swap[]
}

enum ProposalStatus {
  OPEN
  IN_PROGRESS
  CLOSED
}

model Application {
  id          String   @id @default(cuid())
  proposalId  String
  applicantId String
  proposal    Proposal @relation(fields: [proposalId], references: [id])
  applicant   User     @relation("Applicant", fields: [applicantId],
references: [id])

  pitchMessage String  // The negotiation starter
  status       ApplicationStatus @default(PENDING)
}

enum ApplicationStatus {
  PENDING
  ACCEPTED
  REJECTED
}

model Swap {
  id          String   @id @default(cuid())
  proposalId  String

  teacherId   String
```

```
  studentId   String
  status      SwapStatus @default(ACTIVE)

  reviews     Review[]
}

enum SwapStatus {
  ACTIVE
  COMPLETED
  CANCELLED
}

// (Review, Message, Notification models implied as per previous discussion)
```

---

6. File Structure (MVC Implementation)

The project is organized to separate Data (Model), UI (View), and Logic (Controller/Actions).

codeText

```
src/
├── actions/                  # [CONTROLLERS]
│   ├── auth.ts               # Login/Signup logic
│   ├── proposals.ts          # createProposal, getProposals, deleteProposal
│   ├── applications.ts       # applyToProposal, acceptApplication
│   └── swaps.ts              # completeSwap (logic for endorsements)
│
├── app/                      # [VIEW - Pages]
│   ├── (public)/             # Landing page layout
│   │   └── page.tsx
│   ├── (auth)/               # Login/Register layout
│   │   ├── login/page.tsx
│   │   └── register/page.tsx
│   └── (dashboard)/          # App layout (Sidebar/Nav)
│       ├── page.tsx          # The Browse Grid
│       ├── my-proposals/     # Manage own offers
│       ├── active-swaps/     # Manage ongoing learning
│       ├── messages/         # Chat
│       └── profile/[id]/     # Public profile view
│
├── components/               # [VIEW - Components]
│   ├── features/
│   │   ├── proposals/        # ProposalCard, PitchModal
│   │   ├── profile/          # SkillTag, StatBox
│   │   └── chat/             # ChatWindow
│   └── ui/                   # Generic Atoms (Button, Input, Modal)
│
├── lib/                      # [INFRASTRUCTURE]
│   ├── prisma.ts             # DB Connection
│   └── supabase.ts           # Storage Connection
│
└── types/                    # [MODEL - Types]
    └── index.ts              # TS Interfaces
```

7. Key Workflows (Data Flow)

Workflow A: Posting a Proposal

1. **View:** User fills form (Title, Skills Offered, Skills Needed, Modality) -> Clicks "Post".
2. **Controller (`actions/proposals.ts`):** Validates input -> Calls `prisma.proposal.create` -> Connects Skills.
3. **View:** Redirects to Dashboard -> Shows new card.

Workflow B: The Exchange

1. **View:** Applicant clicks card -> Clicks "Request" -> Enters Pitch.
2. **Controller (`actions/applications.ts`):** Creates `Application` record -> Creates `Notification` for Owner.
3. **Owner View:** Sees notification -> Opens Proposal -> Sees Applicant List -> Clicks "Accept".
4. **Controller:** Updates Application to `ACCEPTED` -> Updates Proposal to `IN_PROGRESS` -> Creates `Swap` record.

Workflow C: Completion & Reputation

1. **View:** Teacher clicks "Mark Complete" on Swap Card.
2. **View:** Teacher leaves 5-star review.
3. **Controller (`actions/swaps.ts`):**
    - Updates `Swap` to `COMPLETED`.
    - Finds Student's `UserSkill` entry for the learned skill.
    - **Logic:** If exists, `endorsementCount++`. If not, create new `UserSkill` with `source: ENDORSED`.

8. Future Considerations (Post-MVP)

1. **Verification:** Implement Phone Number verification via SMS before a Swap is finalized (Trust & Safety).
2. **Calendar Integration:** Allow users to schedule the Zoom/In-person meet directly in the app.
3. **Video:** Integrate WebRTC for in-browser video calls.