## **Main App Concept**

Build a photo-less dating app called "Kindred" that focuses on creating meaningful connections through conversations before revealing physical appearance. The app matches users based on compatibility and gradually reveals information as the relationship develops through timed audio calls.

## **Brand Messaging & Home Page**

* Design a pre-signin home page that communicates the Kindred philosophy with the following messaging:
  + **"In a world of shallow swipes and filtered photos, we're reimagining connection. Kindred isn't about judging a book by its cover—it's about exploring the chapters that truly matter. We believe love is deeper than a perfect selfie, more nuanced than an algorithm's first impression."**
* Include visuals that represent meaningful connection beyond physical appearance
* Implement clear sign-up/sign-in buttons prominently on the home page
* Add brief explanations of how the app works with simple icons or illustrations

## **Technical Requirements**

* Create a cross-platform mobile application using React Native
* Implement Firebase for backend services (authentication, database, storage)
* Design for iOS and Android compatibility
* Implement audio-only calling functionality with enforced time limits
* Create an AI matching algorithm that prioritizes location proximity, shared interests, values, communication style, and age range

## **User Flow & Core Functionality**

### **Signup & Authentication**

* Implement signup via Google, and email
* Include email verification via OTP for direct email signups
* Require ID verification during profile creation for safety
* Create secure authentication flow with password reset functionality

### **Profile Creation**

* Design an intuitive profile creation process collecting:
  + Personal information (name, age, gender, interested genders)
  + Location data (city)
  + Professional information (job)
  + Bio/personal description
  + Profile photo (hidden initially from matches)
  + Comprehensive questionnaire covering interests, free time activities, deal breakers, ideal partner traits, and pet ownership

### **Matching System**

* Develop an AI matching algorithm that:
  + Prioritizes users by proximity first
  + Considers compatibility based on questionnaire responses
  + Adapts based on user feedback from previous matches
  + Shows match likelihood percentage
* Generate non-human avatar placeholders using abstract shapes or objects (not faces)
* Limit standard users to 3 matches at a time
* Allow premium users up to 5 concurrent matches
* Implement strategic premium upsells within the matching interface

### **Communication System**

* Implement audio-only calling with enforced time limits:
  + Day 1: 5-minute call
  + Day 2: 10-minute call
  + Day 3: 20-minute call
  + After day 3: Unlimited call duration
* Allow premium users to have 2 calls on day one
* Create a private note-taking feature after each call (visible only to the note-taker)
* Unlock chat functionality after the first 2 calls
* Reveal photos only after 3 calls (premium users can view after day 2)
* Include premium upsell prompts at strategic points in the calling flow

### **Post-Call Interaction**

* Design options to:
  + Deepen connection (reveal additional profile information)
    - Take notes about connection, only the note taker can see the notes
  + Swap a match (provide reason, automatically receive new match)
* Implement additional questionnaire questions after each interaction to improve algorithm
* Create daily choice mechanism: deepen existing connection vs. try new match

### Personality Questionnaire

Communication Style Question: "How would you describe your communication style in relationships?"

* I prefer direct and straightforward communication
* I tend to be more diplomatic and gentle in my approach
* I communicate through actions more than words
* I'm expressive and share my feelings openly

Free Time Activities Question: "How do you prefer to spend your free time?"

* Socializing with friends/family
* Enjoying solo activities (reading, gaming, etc.)
* Outdoor adventures and physical activities
* Creative pursuits (art, music, writing, etc.)
* Learning new skills or topics

Values & Life Goals Question: "Which of these values is most important to you in a relationship?"

* Trust and honesty
* Growth and ambition
* Stability and security
* Independence and freedom
* Shared experiences and adventure

Conflict Resolution Question: "How do you typically handle disagreements in relationships?"

* Address issues immediately and directly
* Take time to process before discussing
* Prefer to compromise and find middle ground
* Focus on understanding the other person's perspective first

Love Language Question: "How do you primarily express affection in relationships?"

* Quality time together
* Acts of service
* Physical touch
* Verbal affirmation and compliments
* Thoughtful gifts

Relationship Pace Question: "What's your preferred pace when developing a new relationship?" Options:

* Taking things slowly and building friendship first
* Moderate pace with regular communication
* Diving deep quickly to establish emotional connection
* Following intuition rather than a set timeline

Dealbreakers Question: "Which of these would be most challenging for you in a relationship?" Options:

* Different lifestyle habits (smoking, drinking, diet)
* Misaligned future goals (children, career, location)
* Incompatible financial attitudes
* Different social needs (extroversion vs. introversion)
* Conflicting values or beliefs

### **UI/UX Design**

* Maintain clean, minimalist aesthetic with pink "K" logo throughout
* Design intuitive navigation with emphasis on matches and conversations
* Create welcome/onboarding screens explaining the app concept (after profile creation)
* Implement simple call interface with timer display
* Design profile viewing and editing interfaces

## Matching Algorithm

/\*\*

\* Kindred - Photoless Dating Matching Algorithm

\* This algorithm matches users based on personality compatibility (60%),

\* location proximity (20%), and age compatibility (20%)

\*/

class MatchingAlgorithm {

constructor() {

// Define weights for each matching component

this.weights = {

personality: 0.6,

location: 0.2,

age: 0.2

};

// Define maximum distances (in km) for location scoring

this.distanceThresholds = {

sameCity: 5,

nearby: 20,

sameRegion: 100

};

}

/\*\*

\* Main matching function - returns array of potential matches with scores

\* @param {Object} user - Current user profile

\* @param {Array} candidates - Array of potential match candidates

\* @returns {Array} - Sorted array of matches with scores

\*/

findMatches(user, candidates) {

// Filter candidates by gender preference first (critical requirement)

const genderFilteredCandidates = this.filterByGenderPreference(user, candidates);

// Calculate match scores for remaining candidates

const scoredCandidates = genderFilteredCandidates.map(candidate => {

const personalityScore = this.calculatePersonalityScore(user, candidate);

const locationScore = this.calculateLocationScore(user, candidate);

const ageScore = this.calculateAgeScore(user, candidate);

// Apply weights to each component

const weightedScore = (

personalityScore \* this.weights.personality +

locationScore \* this.weights.location +

ageScore \* this.weights.age

);

return {

...candidate,

matchScore: Math.round(weightedScore \* 100), // Convert to percentage

compatibilityBreakdown: {

personality: Math.round(personalityScore \* 100),

location: Math.round(locationScore \* 100),

age: Math.round(ageScore \* 100)

}

};

});

// Sort by match score (highest first)

return scoredCandidates.sort((a, b) => b.matchScore - a.matchScore);

}

/\*\*

\* Filter candidates by gender preference - this is a critical requirement

\* @param {Object} user - Current user profile

\* @param {Array} candidates - All potential candidates

\* @returns {Array} - Filtered candidates matching gender preference

\*/

filterByGenderPreference(user, candidates) {

return candidates.filter(candidate => {

// Check if user's gender is in candidate's interested genders

const candidateInterestedInUser = candidate.interestedGenders.includes(user.gender);

// Check if candidate's gender is in user's interested genders

const userInterestedInCandidate = user.interestedGenders.includes(candidate.gender);

// Must be mutual interest

return candidateInterestedInUser && userInterestedInCandidate;

});

}

/\*\*

\* Calculate personality compatibility score (0-1)

\* @param {Object} user - Current user profile

\* @param {Object} candidate - Potential match candidate

\* @returns {number} - Compatibility score between 0-1

\*/

calculatePersonalityScore(user, candidate) {

const questionnaireResponses = [

this.compareResponses(user.communicationStyle, candidate.communicationStyle),

this.compareResponses(user.freeTimeActivities, candidate.freeTimeActivities),

this.compareResponses(user.values, candidate.values),

this.compareResponses(user.conflictResolution, candidate.conflictResolution),

this.compareResponses(user.loveLanguage, candidate.loveLanguage),

this.compareResponses(user.relationshipPace, candidate.relationshipPace),

this.compareDealbreakers(user.dealbreakers, candidate.dealbreakers)

];

// Average the scores from all questions

return questionnaireResponses.reduce((sum, score) => sum + score, 0) / questionnaireResponses.length;

}

/\*\*

\* Compare responses between two users for a single question

\* @param {string|Array} userResponse - Current user's response

\* @param {string|Array} candidateResponse - Candidate's response

\* @returns {number} - Similarity score between 0-1

\*/

compareResponses(userResponse, candidateResponse) {

// Handle array responses (multiple selections)

if (Array.isArray(userResponse) && Array.isArray(candidateResponse)) {

// Calculate Jaccard similarity (intersection over union)

const intersection = userResponse.filter(item => candidateResponse.includes(item));

const union = [...new Set([...userResponse, ...candidateResponse])];

return intersection.length / union.length;

}

// Handle single response (exact match)

return userResponse === candidateResponse ? 1 : 0;

}

/\*\*

\* Special handling for dealbreakers - these should heavily impact compatibility

\* @param {Array} userDealbreakers - User's dealbreakers

\* @param {Object} candidateProfile - Candidate's full profile

\* @returns {number} - Compatibility score (0 if dealbreaker present)

\*/

compareDealbreakers(userDealbreakers, candidateProfile) {

// Check if any of user's dealbreakers match candidate's traits

const dealbreakersPresent = userDealbreakers.some(dealbreaker => {

return this.isDealbreaker(dealbreaker, candidateProfile);

});

return dealbreakersPresent ? 0 : 1;

}

/\*\*

\* Check if a specific dealbreaker applies to candidate

\* @param {string} dealbreaker - Dealbreaker category

\* @param {Object} candidateProfile - Candidate's profile

\* @returns {boolean} - Whether dealbreaker applies

\*/

isDealbreaker(dealbreaker, candidateProfile) {

// Implementation would need to check specific dealbreaker against candidate profile

// Simplified example:

switch (dealbreaker) {

case 'lifestyle\_habits':

return this.checkLifestyleConflicts(candidateProfile);

case 'future\_goals':

return this.checkFutureGoalConflicts(candidateProfile);

case 'financial\_attitudes':

return this.checkFinancialConflicts(candidateProfile);

case 'social\_needs':

return this.checkSocialNeedsConflicts(candidateProfile);

case 'values\_beliefs':

return this.checkValueConflicts(candidateProfile);

default:

return false;

}

}

/\*\*

\* Calculate location compatibility score (0-1)

\* @param {Object} user - Current user with location data

\* @param {Object} candidate - Potential match with location data

\* @returns {number} - Proximity score between 0-1

\*/

calculateLocationScore(user, candidate) {

const distance = this.calculateDistance(

user.location.latitude,

user.location.longitude,

candidate.location.latitude,

candidate.location.longitude

);

// Score based on distance thresholds

if (distance <= this.distanceThresholds.sameCity) {

return 1.0; // Same city (highest score)

} else if (distance <= this.distanceThresholds.nearby) {

return 0.8; // Nearby

} else if (distance <= this.distanceThresholds.sameRegion) {

return 0.5; // Same region

} else {

// Linear decrease in score as distance increases beyond region threshold

return Math.max(0, 0.5 - (distance - this.distanceThresholds.sameRegion) / 1000);

}

}

/\*\*

\* Calculate distance between two points using Haversine formula

\* @param {number} lat1 - User latitude

\* @param {number} lon1 - User longitude

\* @param {number} lat2 - Candidate latitude

\* @param {number} lon2 - Candidate longitude

\* @returns {number} - Distance in kilometers

\*/

calculateDistance(lat1, lon1, lat2, lon2) {

const R = 6371; // Earth's radius in km

const dLat = this.deg2rad(lat2 - lat1);

const dLon = this.deg2rad(lon2 - lon1);

const a =

Math.sin(dLat/2) \* Math.sin(dLat/2) +

Math.cos(this.deg2rad(lat1)) \* Math.cos(this.deg2rad(lat2)) \*

Math.sin(dLon/2) \* Math.sin(dLon/2);

const c = 2 \* Math.atan2(Math.sqrt(a), Math.sqrt(1-a));

const distance = R \* c;

return distance;

}

/\*\*

\* Convert degrees to radians

\* @param {number} deg - Degrees

\* @returns {number} - Radians

\*/

deg2rad(deg) {

return deg \* (Math.PI/180);

}

/\*\*

\* Calculate age compatibility score (0-1)

\* @param {Object} user - Current user with age/preferences

\* @param {Object} candidate - Potential match with age

\* @returns {number} - Age compatibility score between 0-1

\*/

calculateAgeScore(user, candidate) {

// Return 0 if candidate's age is outside user's preferred range or vice versa

if (

candidate.age < user.agePreference.min ||

candidate.age > user.agePreference.max ||

user.age < candidate.agePreference.min ||

user.age > candidate.agePreference.max

) {

return 0;

}

// If within range, calculate how centered they are in each other's preferences

const userCenteredness = this.calculateAgeCenteredness(

candidate.age,

user.agePreference.min,

user.agePreference.max

);

const candidateCenteredness = this.calculateAgeCenteredness(

user.age,

candidate.agePreference.min,

candidate.agePreference.max

);

// Average the two values

return (userCenteredness + candidateCenteredness) / 2;

}

/\*\*

\* Calculate how centered an age is within a preferred range (higher if closer to center)

\* @param {number} age - Age to check

\* @param {number} minPreferred - Minimum preferred age

\* @param {number} maxPreferred - Maximum preferred age

\* @returns {number} - Centeredness score between 0-1

\*/

calculateAgeCenteredness(age, minPreferred, maxPreferred) {

const rangeSize = maxPreferred - minPreferred;

const center = minPreferred + (rangeSize / 2);

const distanceFromCenter = Math.abs(age - center);

const maxDistance = rangeSize / 2;

// Convert to a 0-1 score, higher when closer to center

return 1 - (distanceFromCenter / maxDistance);

}

/\*\*

\* Additional helper methods for specific dealbreaker checks

\* These would be filled out based on how your data is structured

\*/

checkLifestyleConflicts(candidateProfile) {

// Implementation depends on how lifestyle data is structured

return false; // Placeholder

}

checkFutureGoalConflicts(candidateProfile) {

// Implementation depends on how goals data is structured

return false; // Placeholder

}

checkFinancialConflicts(candidateProfile) {

// Implementation depends on how financial data is structured

return false; // Placeholder

}

checkSocialNeedsConflicts(candidateProfile) {

// Implementation depends on how social needs data is structured

return false; // Placeholder

}

checkValueConflicts(candidateProfile) {

// Implementation depends on how values data is structured

return false; // Placeholder

}

}

/\*\*

\* Example usage:

\*

\* // Initialize the matching algorithm

\* const matcher = new MatchingAlgorithm();

\*

\* // Get potential matches for a user

\* const matches = matcher.findMatches(currentUser, candidateUsers);

\*

\* // Display top 5 matches

\* console.log(matches.slice(0, 5));

\*/

## **Monetization Strategy**

* Implement two-tier user system:  
    
   **Free Tier Features**
  + 3 matches at a time
  + 1 call per day with time limits
  + Photo reveal after 3 calls
  + Ad-supported experience with:
    - Banner ads on appropriate screens
    - Interstitial ads at strategic moments
    - Option to close ads which triggers premium subscription prompt
* **Premium Subscription Features**
  + Ad-free experience throughout
  + Increased matches (5 instead of 3)
  + Additional calls per day (2 on day one)
  + Earlier photo reveal (after day 2 instead of day 3)
* **Conversion Strategy**
  + Design strategic premium upsell touchpoints:
    - When users reach their match limit
    - Before/after calls with time limits
    - When attempting to view photos early
    - After closing ads
    - Within app notification center
  + Implement subscription management interface
  + Create tiered subscription options (monthly, quarterly, annual)
  + Include free trial option for new users

## **Safety & Moderation**

* Implement ID verification system
* Create user reporting system for inappropriate behavior
* Design moderation dashboard for admin review
* Implement content filtering for inappropriate messages

## **Data Privacy & Management**

* Implement standard privacy practices and GDPR compliance
* Retain user data for 2 years after account deletion
* Create data export functionality for user data portability
* Implement privacy controls for user information

## **Scalability Considerations**

* Design for initial target of 10,000 users
* Plan infrastructure for international expansion
* Implement localization framework targeting US, UK, and Australia markets

## **Testing Requirements**

* Implement comprehensive testing strategy including:  
    
   **User Authentication Testing**
  + Test all authentication methods (Google, Apple, email)
  + Verify OTP functionality and error handling
  + Test account recovery and password reset flows
  + Validate session management and token refresh
  + Test ID verification process and edge cases
* **Profile Creation Testing**
  + Verify all form validations and error states
  + Test image upload, cropping, and storage
  + Validate questionnaire flow and data saving
  + Test profile editing and information updating
  + Verify profile completion indicators
* **Matching Algorithm Testing**
  + Create test scenarios with controlled user profiles
  + Validate proximity-based matching accuracy
  + Test interest and values matching weights
  + Verify algorithm adaptation based on user feedback
  + Benchmark matching performance with increasing user base
  + Test match percentage calculation accuracy
* **Call Functionality Testing**
  + Verify call initiation, connection, and termination
  + Test enforced time limits for different relationship stages
  + Validate audio quality across different network conditions
  + Test call notifications and availability indicators
  + Verify note-taking functionality after calls
  + Test edge cases (interrupted calls, network switching)
* **Chat Functionality Testing**
  + Verify message sending, delivery, and read receipts
  + Test media sharing capabilities if applicable
  + Validate chat history persistence
  + Test notification system for new messages
  + Verify chat unlocking after appropriate call progression
* **Monetization Testing**
  + Test ad implementation and display
  + Verify ad dismissal flow triggers premium prompts
  + Test subscription purchase flow
  + Verify payment gateway integration
  + Test subscription management (upgrades, cancellations)
  + Validate receipt validation and purchase restoration
  + Test subscription benefits activation
  + Verify subscription renewal and expiration handling
  + Test all premium upsell touchpoints
* **Performance Testing**
  + Load testing for concurrent users
  + API response time benchmarking
  + Battery usage optimization
  + Memory usage monitoring
  + Cold and warm startup time measurement
* **Security Testing**
  + Penetration testing of authentication system
  + Data encryption validation
  + API endpoint security testing
  + Input validation and injection prevention
  + Session management security