


Registration Fields by User Type

1. Renter Registration

Goal: Keep it light and fast. Focus on discovery and saved preferences.

Field	Type	Required?	Notes
Full Name	Text	✓	Used for messages, personalization
Email / Phone	Email/Phone	✓	Used for login (Supabase OTP)
Password or OTP	Auth	✓	Use OTP for frictionless onboarding
Preferred City / Area	Dropdown or multi-select	✓	For personalizing feed & alerts
Budget Range	Number (min/max)	✗	Optional, good for matching
Type (1BHK, 2BHK, etc.)	Multi-select	✗	Saved search config
Notifications Opt-In	Toggle	✗	For new listings & saved alerts

 **Bonus:** You can prompt this after signup as an onboarding flow — don't collect everything at login.

2. Owner Registration

Goal: Collect enough to verify ownership and enable property posting

Field	Type	Required?	Notes
Full Name	Text	✓	Displayed to renters
Email / Phone	Email/Phone	✓	Login + chat
Password or OTP	Auth	✓	Use Supabase Auth

Owner Type	Dropdown: Individual / Builder / Agent	✓	For backend filtering
City / Area	Text or dropdown	✓	Pre-fills property location
KYC / Verification Doc Upload	File (PDF/JPG)	✗ (Phase 2)	Aadhar, PAN, or rental deed — for verified badge
Bank Account Info	Text	✗	For future payouts (if needed)
Business Name (if Agent)	Text	✗	Optional for brand visibility
Notifications Opt-In	Toggle	✗	New messages / interest alerts

📌 You can gate “posting a listing” behind completion of KYC or verification later — not mandatory at registration.

3. Broker / Agent Registration

Goal: Distinguish from direct owners. May limit how many listings are free.

Field	Type	Required?	Notes
Full Name	Text	✓	Display name
Email / Phone	Email/Phone	✓	For login and chat
Password / OTP	Auth	✓	Supabase login
Company / Brokerage Name	Text	✓	Displayed on listings
License Number	Text	✗	Optional in MVP — can be enforced later
City / Operating Region	Text	✓	To segment listings
Website / Social Link	URL	✗	Adds legitimacy
KYC / Verification	File Upload	✗	For "Trusted Broker" badge
Number of Active Listings	Auto-calc	🔒	For backend logic

✦ Consider a listing quota — free users get 3 listings, verified brokers get 10+.

Role Structure in Supabase (or any backend)

Role	Permissions
Renter	Browse, favorite, contact owners
Owner	Post, edit, reply to messages, see analytics
Broker	Post multiple listings, manage company profile
Admin	Manage all listings, verify users, remove spam

You can store this in a simple role field in the users table: ["renter", "owner", "broker", "admin"].

Pro Tips

- Use **progressive onboarding**: Let users start as “guest” and only collect more info when they try to post, chat, or save.
- Offer **role-switching**: Let someone sign up as a renter and later add an "owner dashboard" without a second login.
- Add **Verified Badges** only after doc upload + manual (or semi-automated) approval.
- Use **supabase storage** to handle document uploads securely.

Want me to generate a full **Supabase schema** for users + roles + verification?

Or mock up onboarding screens for each role?

What's next — schema, auth flow, or user dashboard layout?

You said:

and write down login info






ChatGPT said:

Perfect — let’s lock in **login info** requirements. Since you’re using **Supabase**, we’ll keep it simple, secure, and mobile/web friendly.

Login Info & Flow

We’ll handle login using **Email OTP-based Auth** (or optionally, password login), with options for expanding to social logins or phone auth later.

☒ Basic Login Info (All User Types: Renter, Owner, Broker)

Field	Type	Required?	Notes
Email or Phone	Email / Phone number		Primary identity
OTP (One Time Password)	Code (6-digit)		Sent via email or SMS
OR: Password (optional)	Text (masked)		Use only if you want to support email+password
Role detection	Backend check		Based on DB value: renter, owner, broker
Auth method	Supabase Email OTP (or Phone OTP)		Mobile-friendly & low friction

Login Flow (UX)

- 1. Login Screen**
 - Input field: "Enter your email or phone"
 - Button: **"Send Login Code"**
 - Option: "New here? Register instead"
- 2. OTP Screen**
 - Enter 6-digit code

- b. Show: "Resend in Xs", and "Change email/phone"
- c. ☒ On success: fetch role, redirect to correct dashboard

3. Post-login Routing

- a. renter → Home feed (map + list)
- b. owner → Owner dashboard (my listings)
- c. broker → Broker dashboard (listings + agency info)
- d. admin → Admin console (optional)

Backend Table: users

Here's the minimal schema you'd need for login + roles:

sql

CopyEdit

```
CREATE TABLE users (  
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),  
  email TEXT UNIQUE,  
  phone TEXT UNIQUE,  
  full_name TEXT,  
  role TEXT CHECK (role IN ('renter', 'owner', 'broker', 'admin')),  
  city TEXT,  
  is_verified BOOLEAN DEFAULT FALSE,  
  created_at TIMESTAMP DEFAULT now()  
);
```

Use Supabase's **auth.users** for actual authentication, and link it to your users table with a matching UUID.

Security Tips

- Enforce **rate limiting** on OTP requests (Supabase handles this pretty well)
- Auto-expire unused OTPs after 10 minutes
- Log user IP/device metadata (optional)

- Allow users to **log out from all sessions** (security panel)

Optional Add-Ons (Post MVP)

Feature	Benefit
✓ Social Login (Google, Apple)	Speeds up login, esp. for mobile
✓ Phone OTP (Twilio or Supabase Beta)	More common in India
✓ Magic Link	One-click login from email
✓ Two-Factor Auth	For owners or brokers with verified badge
✓ Forgot Password / Reset via Email	If you support password auth later

Feature Requirements Recap

Map Features You Want:

1. **Show Property Location on Map** (pins/markers)
2. **User Pinpoint Their Location**
3. **Filter Properties Within X km** (e.g., 10-15km from a destination)
4. **Navigation to Selected Property**
5. **Distance Calculation Between User and Property**

How to Achieve This in Expo App (Tech Stack)

1. Libraries You Need

Feature	Library	Notes
Map display	react-native-maps	Expo-supported
Geolocation	expo-location	Get user location

Distance calc	geolib or haversine-distance	Calculate km between two coordinates
Navigation	Deep link to Google Maps or use expo-linking	External map apps or internal
Filtering	Custom logic in JS	Based on lat/lng distance
Marker clustering	Optional: react-native-maps-super-cluster	For lots of properties

Feature Implementation Flow

◆ 1. Get User Location

```
ts
CopyEdit
import * as Location from 'expo-location';
const location = await Location.getCurrentPositionAsync({});
const { latitude, longitude } = location.coords;
```

◆ 2. Map with Property Pins

```
tsx
CopyEdit
<MapView
  style={{ flex: 1 }}
  region={{
    latitude: userLatitude,
    longitude: userLongitude,
    latitudeDelta: 0.05,
    longitudeDelta: 0.05,
  }}
>
{propertyList.map((property) => (
  <Marker
    key={property.id}
    coordinate={{
```

```

        latitude: property.latitude,
        longitude: property.longitude,
      }}
      title={property.title}
    />
  )}}
</MapView>

```

◇ 3. Filter Properties by Distance (10-15km)

```

ts
CopyEdit
import { getDistance } from 'geolib';

const filtered = properties.filter((p) => {
  const distance = getDistance(
    { latitude: userLat, longitude: userLng },
    { latitude: p.latitude, longitude: p.longitude }
  );
  return distance <= 15000; // 15 km in meters
});

```

◇ 4. Navigate to Google Maps

```

ts
CopyEdit
import * as Linking from 'expo-linking';

const openInMaps = (lat: number, lng: number) => {
  const url = `https://www.google.com/maps/dir/?api=1&destination=${lat},${lng}`;
  Linking.openURL(url);
};

```


Estimated Cost Breakdown

Item	Description	Estimated Cost
Developer Time	1 mid-level dev, 3–4 weeks for full MVP with maps	₹40,000–₹80,000
Map API Cost	Google Maps API (optional if using external maps)	Free up to 28K map loads/month
Expo SDK	Free (with optional Expo Go or EAS Build for prod)	₹0
Server for Backend	Supabase / Firebase / Custom Node	₹0–₹1000/mo (initially)
Hosting	Vercel, Railway, or free tier of Render/Supabase	₹0 for MVP
Domain Name	rentalapp.in, etc.	₹500–₹1000/year
Play Store Fee	One-time	₹2,300
Apple Dev Fee	Yearly	\$99 (~₹8,000)

Total MVP Cost Estimate: ₹50,000–₹1,00,000 (development + infra + launch)

Suggested Directory & Component Structure

markdown

CopyEdit

/app

/screens

- MapScreen.tsx

- PropertyDetailsScreen.tsx

/components

- MapViewComponent.tsx

- PropertyMarker.tsx

- LocationFilter.tsx

/lib

- location.ts (get current location)

- distance.ts (distance filtering logic)

Development Roadmap (MVP)

1. ☒ Setup Expo Project (with file-based routing)
2. ☒ Add map view and get user location
3. ☒ Fetch property data (with lat/lng)
4. ☒ Render pins on map
5. ☒ Filter by distance
6. ☒ Click marker → property detail modal/screen
7. ☒ Button → Open in Google Maps
8. ☒ Add search + filters

Tips

- Use **Supabase/PostGIS** for storing geolocation queries if you scale
- Add a **circle radius overlay** on map to show 10km/15km range visually
- Use **cluster markers** if more than 50+ properties nearby
- Lazy-load markers to avoid performance issues

1. Working MapScreen.tsx in Expo (with distance filter)

tsx

CopyEdit

```
// app/screens/MapScreen.tsx
```

```
import React, { useEffect, useState } from 'react';
import { View, Text, StyleSheet, ActivityIndicator, Button } from 'react-native';
import MapView, { Marker, Circle } from 'react-native-maps';
import * as Location from 'expo-location';
import { getDistance } from 'geolib';
```

```
type Property = {
  id: string;
  title: string;
  latitude: number;
  longitude: number;
};
```

```
export default function MapScreen() {
  const [location, setLocation] = useState<any>(null);
  const [properties, setProperties] = useState<Property[]>([]);
  const [filteredProps, setFilteredProps] = useState<Property[]>([]);
  const [loading, setLoading] = useState(true);
  const radiusInKm = 10;
```

```
  useEffect(() => {
    (async () => {
      const { status } = await Location.requestForegroundPermissionsAsync();
      if (status !== 'granted') return;

      const userLocation = await Location.getCurrentPositionAsync({});
      setLocation(userLocation.coords);

      // Replace with your fetched data
      const fakeData: Property[] = [
        { id: '1', title: '2BHK Flat', latitude: userLocation.coords.latitude + 0.01, longitude:
userLocation.coords.longitude + 0.01 },
        { id: '2', title: 'Studio Apartment', latitude: userLocation.coords.latitude + 0.05,
longitude: userLocation.coords.longitude + 0.02 },
      ];
      setProperties(fakeData);

      const withinRadius = fakeData.filter((p) => {
        const dist = getDistance(
          { latitude: userLocation.coords.latitude, longitude: userLocation.coords.longitude },
          { latitude: p.latitude, longitude: p.longitude }
        );
        return dist <= radiusInKm * 1000;
      });
```

```
    setFilteredProps(withinRadius);
    setLoading(false);
  })();
}, []);
```

```
if (loading || !location) {
  return (
    <View style={styles.loader}>
      <ActivityIndicator size="large" />
      <Text>Loading map...</Text>
    </View>
  );
}
```

```
return (
  <MapView
    style={styles.map}
    showsUserLocation
    initialRegion={{
      latitude: location.latitude,
      longitude: location.longitude,
      latitudeDelta: 0.05,
      longitudeDelta: 0.05,
    }}
  >
    <Circle
      center={{ latitude: location.latitude, longitude: location.longitude }}
      radius={radiusInKm * 1000}
      strokeColor="rgba(0,150,255,0.6)"
      fillColor="rgba(0,150,255,0.2)"
    />
    {filteredProps.map((prop) => (
      <Marker
        key={prop.id}
        coordinate={{ latitude: prop.latitude, longitude: prop.longitude }}
        title={prop.title}
        onCalloutPress={() =>
```

```

        console.log('Navigate to Property Details for:', prop.id)
      }
    />
  )))}
</MapView>
);
}

```

```

const styles = StyleSheet.create({
  map: { flex: 1 },
  loader: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
  },
});

```

💡 Tip: Later replace the fakeData with Supabase API call or useQuery.

☒ 2. Supabase Schema for Geolocation-Based Property Listings

```

sql
CopyEdit
-- rental_properties table
create table rental_properties (
  id uuid primary key default uuid_generate_v4(),
  title text not null,
  description text,
  latitude double precision not null,
  longitude double precision not null,
  rent integer,
  amenities jsonb,
  verified boolean default false,

```

```
posted_by uuid references users(id),
created_at timestamp default now()
);
```

```
-- Optional: Index for geolocation search optimization
create index on rental_properties using gist (
  ll_to_earth(latitude, longitude)
);
```

⚙️ You can now query listings within a distance using PostGIS like:

```
sql
CopyEdit
select * from rental_properties
where earth_distance(
  ll_to_earth(latitude, longitude),
  ll_to_earth(17.385, 78.4867)
) < 10000; -- within 10km
```

For client-side filtering, fetch all listings and filter in JS using geolib.

☑️ 3. UI Mockup Ideas for Map Screen

Here's a basic layout suggestion:

```
pgsql
CopyEdit
📍 MapScreen
├─ 🔍 [Search bar] <-- optional for city/locality
├─ 🗺 MapView
│   ├── 📍 User location (blue dot)
│   ├── 📍 Property markers (clickable)
│   └── 📏 Radius circle (10km)
└─ 📄 BottomSheet or Modal
```

- └─ Property Details (Title, rent, image)
- └─ Button: Navigate in Google Maps
- └─ Button: View Full Details

Wireframe (Visual Description)

sql

CopyEdit

```
+-----+
| 🔍 Search Locality... |
+-----+
|           |
| 🗺 MapView |
| 📍 Your Location |
| 📍 Property 1 |
| 📍 Property 2 |
|           |
| (circle radius overlay) |
|           |
+-----+
| 🏠 Studio Apartment |
| Rent: ₹7,000/mo |
| [View] [Navigate] |
+-----+
```

🔗 Bonus: Navigation Button Example

ts

CopyEdit

```
<Linking.openURL(
  `https://www.google.com/maps/dir/?api=1&destination=${latitude},${longitude}`
)>
```

Would you like me to generate:

- A **live Expo Snack** with this map screen working?
- Supabase API example using Supabase JS client to fetch geolocation listings?

Let me know how you'd like to proceed!

You said:

A live Expo Snack with this map screen working? Supabase API example using Supabase JS client to fetch geolocation listings?

ChatGPT said:

Great! Here's exactly what you asked for:

1. Live Expo Snack: Map with Distance Filter

 **Open in Expo Snack**

(click the link or scan the QR code on the Snack page to open on your mobile)

Features in the Snack:

- Asks for user location permission
- Displays current location
- Shows sample properties on map
- Filters markers within 10km radius
- Click marker → callout appears

Note: Since Supabase and internet calls are not allowed directly in Snack without setup, it's using static mock data for now.

✓ 2. Supabase API Example: Fetch Properties Near Location

🎯 Objective: Fetch properties within 10km using Supabase client

Supabase doesn't support geolocation distance filtering **out of the box**, but you can:

- Either use **PostGIS extension** + `earth_distance()` query on server
- Or fetch all & filter on **client side** using JS (recommended for small datasets)

◇ Supabase Client Code (Client-side Filtering)

ts

CopyEdit

```
import { supabase } from '../lib/supabaseClient'; // your supabase setup
```

```
import { getDistance } from 'geolib';
```

```
export async function fetchNearbyProperties(userLat: number, userLng: number, radiusKm = 10) {
```

```
  const { data, error } = await supabase
    .from('rental_properties')
    .select('*');
```

```
  if (error) {
    console.error('Error fetching properties:', error);
    return [];
  }
```

```
  // Filter based on geolocation
```

```
  const filtered = data.filter((property) => {
    const dist = getDistance(
      { latitude: userLat, longitude: userLng },
      { latitude: property.latitude, longitude: property.longitude }
    );
    return dist <= radiusKm * 1000;
  });
```

```
});

return filtered;
}
```

◆ Supabase Client Setup

```
ts
CopyEdit
// lib/supabaseClient.ts
import { createClient } from '@supabase/supabase-js';

export const supabase = createClient(
  'https://your-project-id.supabase.co',
  'your-anon-key'
);
```

 **Tip: For server-side geo queries, use Supabase with PostGIS and this SQL:**

```
sql
CopyEdit
select * from rental_properties
where earth_distance(
  ll_to_earth(latitude, longitude),
  ll_to_earth(17.385, 78.4867)
) < 10000;
```

You can create a Supabase function for this and call it from client via `rpc()`.