Help me build a modular, scalable Airbnb-style web app called RooMe, tailored for the Zimbabwean market. This is an MVP build focused on core functionality — future prompts will follow for feature additions.

Emphasis:

- Prioritise a clean, well-structured backend-first architecture.

- Cleanly separate frontend from backend (frontend fetches only, no embedded logic).

- Use React for the frontend (clean, modular components).

- Use Supabase with PostgreSQL as the backend and database.

- Frontend components should be dynamically served/stored (as JSON configs if possible) so they can be repurposed in a future Flutter app.

- Use reusable UI components: listing cards, filters, navbar, search, etc.

- Include user roles (guest, host), each with appropriate access.

- Do NOT include admin features inside the user app:

- Instead, build secure backend endpoints for remote admin tools.

- Admin APIs should support moderation (flagging, approving), content control, user management, platform analytics, and listing review.

- Optimize for mobile-first design, but keep it responsive for web.

- Structure code for easy handoff and reuse in a \*future native build\*.

- Discard Airbnb features that don’t apply locally (e.g., international currency, automated payments, global listings).

- Adapted features for Zimbabwe:

- Listings for: boarding houses, rooms for rent, lodges, hotels

- Search by city/location (e.g., Harare, Bulawayo, Gweru)

- Listings include: title, image gallery, description, property type, price, location

- Manual bookings only (no payments yet)

- Messaging feature

- Contact via WhatsApp link, call, or SMS

- Simple auth with phone/email

- Output the project in a scalable folder structure (backend, frontend, utils, config).

Extended Backend-Ready Feature Checklist (For RooMe):

1. Versioning Support (Backend):

- Design the API and DB schema with versioning to avoid breaking older app clients during future updates.

2. Role Management (User Roles):

- Prepare for roles: guests and hosts.

- Backend should allow scalable permission-based access.

3. Notifications System:

- Include support for email, SMS, and \*in-app push notifications\* — with message queuing and user opt-in/out settings.

4. Document Verification via OCR & AI:

- Add backend endpoints for users to upload ID/docs.

- Prepare for OCR and AI checks for authenticity (even if AI model is added later).

- Store verification status and allow admin override.

5. Smart Recommendations Engine:

- Add hooks for a scoring algorithm: capture location, budget, preferences, behavior.

- Backend should allow attaching "tags" or "weights" to listings/users for ML models to use later.

6. User & Listing Badging System:

- Allow backend to assign badges like “Verified Agent,” “Top Landlord,” “Highly Rated.”

- Include logic for automatic or manual badge assignment.

7. Reporting & Moderation Tools:

- Flagging system for listings, messages, or users.

- Endpoints for admin moderation, warnings, and content visibility toggling.

8. Data Analytics & Logging:

- Track: search queries, clicks, conversions, time-on-page, device types.

- Prepare for dashboard analytics in future admin panel.

9. Multi-device & App Readiness:

- Ensure backend can support web, Flutter, and any native clients.

- Standardise API responses and UI-ready data structures.

10. Localization Readiness:

- Support for regional differences (e.g., area names, pricing units).

- Optional language key mapping for future multilingual support.

11. Services Section (Like Airbnb Experiences):

- Add backend support for a services/experiences module separate from accommodation.

- Each service should include:

- Title, description, host/provider, category (e.g., cleaning, movers, tours)

- Availability schedule & duration

- Pricing (per service or per hour/day)

- Image gallery & location

- Contact or booking link (manual or future in-app)

- Prepare endpoints for:

- Listing services

- Searching/filtering

- User reviews & ratings

- Booking requests (manual for now, payments optional)

Here’s the updated \*sub-prompt\* for the roommate feature, now expanded with notifications and scalability in mind:

---

\*Roommate Feature (Future-Ready – Backend Prep Now):\*

Include backend preparation for a future \*roommate matching system\*. This will allow users to either list a room \*with a preference for a roommate\*, or actively search for \*a compatible roommate\*.

- Collect relevant user data: lifestyle preferences, age, gender, occupation, budget range, habits (e.g. smoking, pets), preferred locations.

- Match users via smart filtering or later via recommendation algorithms.

- Support an \*optional profile visibility toggle\* for those looking for roommates.

- Notifications: Prepare a mechanism to send roommate recommendations via in-app notifications, showing basic info and a CTA (e.g. “You might match well with Tawanda in Borrowdale – View now”).

- Design all roommate-related logic and models in a \*modular, scalable way\* to allow seamless integration into future versions without disrupting existing listing/booking flows.

Backend Specs:

- Use a REST API design

- Modular structure with routes for users, listings, and admins

- Store listings and user info in a simple JSON-based mock database (or simulate one)

- Design the backend so it can later switch to a real database like PostgreSQL or Firebase

- Include sample endpoints: GET /listings, POST /listings, GET /user, POST /auth/login

Additionally, design the backend to store frontend UI component configurations (e.g., card layouts, filter options, labels, themes) in the database, so the frontend can dynamically retrieve and render components based on backend data. This will allow future versions or different platforms (like Flutter) to reuse the same UI structure and settings by fetching them from the backend.

Frontend:

- Use React (with clean JSX, CSS Modules or Tailwind) for the frontend to ensure modular, scalable components and a smooth developer experience.

- Avoid inline logic; keep layout files modular

- Add a home page with search, featured listings, and navigation

Additionally, ensure the frontend UI fully captures Airbnb’s clean, modern, and highly user-friendly aesthetic adapted for RooMe. Use minimalist design with ample white space, modern sans-serif fonts (like Inter or Poppins), and a cohesive color palette centered on trust and warmth (primary blue #2A72FF, gold accents #FFD166, dark charcoal #333333, and light gray #B0BEC5). Implement card-based layouts with high-quality images, subtle shadows, and rounded corners. Include smooth transitions and intuitive hover effects. The navigation should be simple, sticky, and include key elements like logo, search, user profile, and menu. The search bar must be dynamic with smart filters and autocomplete suggestions. Present listings with clear, concise info upfront (title, location, price, ratings, availability) using clean icons and minimalistic buttons to enhance trust and ease of use. Maintain consistent spacing, alignment, and padding for a polished, welcoming feel. Importantly, frontend components should be modular and stored in the backend database for dynamic rendering and easy updates. This aesthetic focus should blend seamlessly with the solid backend architecture emphasized earlier, ensuring smooth real-time data flow.