System Design Document

1. CRC Cards for Login

Class Name	Parent	Responsibilities	Key Collaborators
AuthProvider	React Context	Global auth state, session management, auto-redirects	Supabase, React Router
useAuth	React Hook	Access auth context, ensure proper usage	AuthContext
SupabaseAuth	React.FC	Render auth UI, handle magic links/social auth	Supabase Auth UI, useAuth
AuthCallback	React.FC	Process auth callbacks, handle redirects	Supabase client, useAuth
ProtectedRoute	React.FC	Guard routes requiring authentication	useAuth, React Router
RoleProtectedRoute	React.FC	Guard routes by user role (tenant/landlord)	useAuth, auth utilities
useUserRole	React Hook	Manage user roles, sync metadata/localStorage	useAuth, Supabase

System Interactions and assumptions

Operating System Dependencies

- Browser Environment: Modern web browsers supporting ES6+, WebCrypto API
- Network: HTTPS required for secure authentication
- Storage: Browser localStorage and sessionStorage support

Database Dependencies

- Supabase PostgreSQL:
  - User authentication tables
  - User metadata storage
  - Session management
  - Row Level Security (RLS) policies

External Service Dependencies

- Supabase Auth Service:
  - Magic link email delivery
  - Social OAuth providers (Google, Facebook)
  - JWT token management
  - Session persistence

Compiler/Build Dependencies

- Vite: Build tool and development server
- TypeScript: Type checking and compilation
- React 18: Component framework
- React Router v6: Client-side routing

Reasoning Behind Architecture Selection

1. Context API over Redux

Simplicity: Authentication state is relatively simple

Performance: Minimal re-renders with proper context splitting

Bundle Size: No additional dependencies

2. Supabase Auth over Custom Auth

Security: Battle-tested authentication service

Features: Built-in magic links, social auth, session management

Maintenance: Reduced security maintenance burden

Compliance: GDPR, SOC2 compliant

3. Magic Links over Passwords

Security: No password storage or management

UX: Simplified user experience

Accessibility: Better for users with password management issues

4. Role-based Architecture

Scalability: Easy to add new roles

Security: Clear separation of concerns

Maintainability: Centralized role logic

5. Multi-layer Route Protection

Defense in Depth: Multiple protection layers

Flexibility: Different protection levels for different routes

User Experience: Smooth redirects based on auth state

2. <u>Chat feature</u>

Class Name	Parent	Responsibilities	Collaborators
ChatRouter	None	Create new room, Post a new message, Get messages in a specific room	RoomController, MessageController, Supabase
Message	None	Represent a single message within the chat, store messages, Manage reactions associated with message	Reactions
ChatWindowProps	None	Provide access to message related data, hold properties for the chat window	Message, Chatwindow
ChatService	None	Get all chats for user, Create a chat, send a message	User, Chat, Message, Supabase
Chat	None	Represent a generic chat, Manage common chat functionalities	Message, ChatWindow, User
LandlordChats	None	Manage chats related to landlords, provide interface for landlords to view conversations	Chat, Landlord, Tenant, Property

System Interactions and assumptions

Operating System Dependencies

- Browser Environment: Modern web browsers that support ES6+, file input, and emoji rendering

- Network: HTTP/HTTPS required for API communication
- Storage: Browser localStorage and sessionStorage support for authentication and user data
- File Handling: Browser file input support for image and attachment uploads

Database Dependencies

- Supabase PostgreSQL:
    - Chat room table
    - Message table
    - Foreign key constraints for use and property references
    - Timestamp and status fields for messages
    - Row Level Security (RLS) policies for chat data

External Service Dependencies

- Supabase Auth Service
    - Session persistence

Compiler/Build Dependencies

- Vite: Build tool and development server
- Typescript: Type checking and compilation
- React 18: Component Framework
- React Router v6: Client-side routing

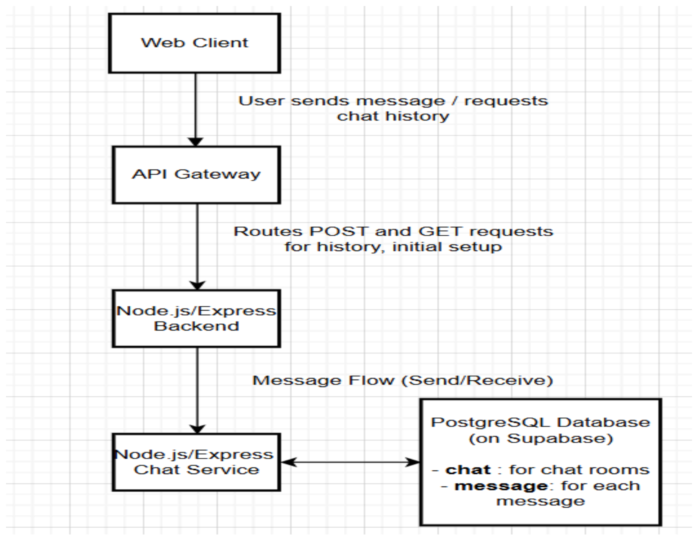
System architecture:

1. Frontend (Chat UI)
    a. React Chat Component: Handles chat message display, input, reactions, and file uploads
    b. Authentication: Uses Supabase Auth to ensure only logged-in users can access chat
    c. Api communication: Sends/receives chat messages and room data via HTTP requests to backend
2. Backend (Chat API)
    a. Express.js Chat Endpoints: Receives chat-related requests (send, fetch, delete messages, create chat rooms)
    b. Prisma ORM: Handles database operations for chat rooms and messages
    c. Authorization: Verifies user identify (via JWT/session from Supabase Auth) before processing chat actions.
3. Database (Supabase PostgreSQL)
    a. Chat Tables: Stores chat rooms and messages, with foreign keys to user and property tables.
    b. RLS Policies: Ensures only authorized users can access their chat data.
4. External Service (Supabase Auth)
    a. Session/JWT: Provides user authentication and session management for chat access

Reasoning Behind Chat Feature Architecture:

- React: Enables a responsive, real-time chat UI with state management for messages and reactions.
- Express.js: Provides a clear separation between chat logic and other backend features, making chat endpoints easy to maintain and scale
- Prisma ORM: Simplifies database access and enforces data integrity for chat messages and rooms
- Supabase PostgreSQL: Offers a reliable, scalable, and managed database for storing chat data
- Supabase Auth: Ensures only authenticated users can participate in chats, and simplifies session management

Diagram


Web Client
User sends message / requests chat history
API Gateway
Routes POST and GET requests for history, initial setup
Node.js/Express Backend
Message Flow (Send/Receive)
Node.js/Express Chat Service
PostgreSQL Database (on Supabase)
- chat : for chat rooms
- message: for each message

Create Listing Feature

CRC Card

Class Name	Parent/Subclasses	Responsibilities	Key Collaborators
CreateListing	None	Gets data from the Create Listing form, and sends requests to the backend for posting and getting the current landlord's listing data	landlordController, useAuth, supabase
landlordController	None	Manage landlord actions, including storing a listing in the database and retrieving	CreateListing, supabase

		all listings	

System Interactions and assumptions

Operating System Dependencies

- Modern web browsers that support ES6+ and file uploads (for images)

Database Dependencies

- Supabase PostgreSQL:
    - Stores listing data in listings table (landlord id, image URL, other property details)

External Service Dependencies

- Supabase Storage
    - Stores uploaded listing images and creates image URLs
- Supabase Auth
    - Authenticates landlord before creating the listing

Compiler/Build Dependencies

- Vite: Build tool and development server
- TypeScript: Type checking and compilation
- React 18: Component framework
- React Router v6: Client-side routing

System architecture and reasoning

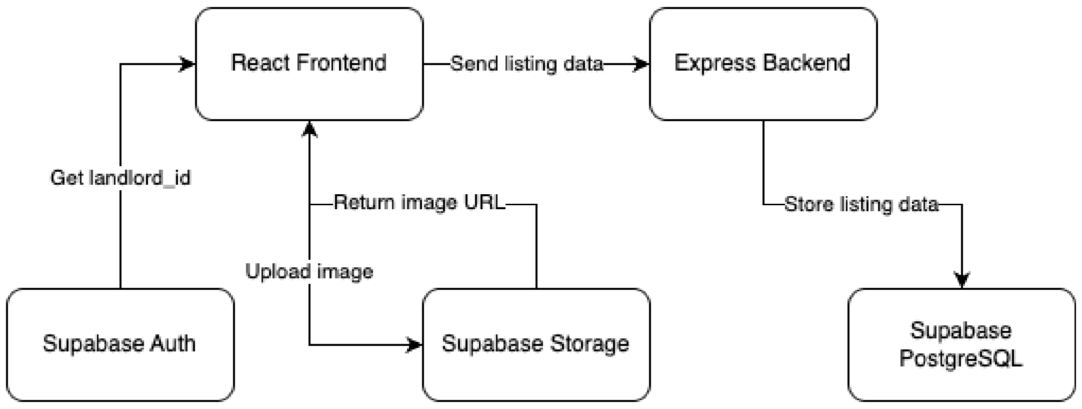
1. Frontend (React)
    - Renders Create Listing form and sends inputs to the backend
    - Uploads listing images to Supabase storage
    - Uses Supabase Auth to get the current landlord's id who creates the listing
2. Backend (Express)
    - Saves listing data to Supabase database
    - Retrieves listing data from Supabase database
3. Database (Supabase PostgreSQL, Storage, & Auth)
    - Stores listings in PostgreSQL table
    - Stores listing images in Supabase storage bucket
    - Authenticates landlord, ensuring only landlords can create listings and that listings belong to a landlord

Reasoning Behind Architecture

- React: Flexible with state, effect, and auth hooks for fetching listing data
- Express.js: Simple backend setup with middleware and routing
- Supabase PostgreSQL: Reliable database listings table, combines PostgreSQL, Storage, and Auth for easy management of all listings data
- Supabase Storage: Stores images and creates image URLs to be stored in the listings table

- Supabase Auth: Authenticates current landlord to ensure listings are created with the correct landlord information

Diagram


React Frontend
Send listing data
Express Backend
Get landlord_id
Return image URL
Store listing data
Upload image
Supabase Auth
Supabase Storage
Supabase PostgreSQL