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

CRC Card for Chat feature

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:

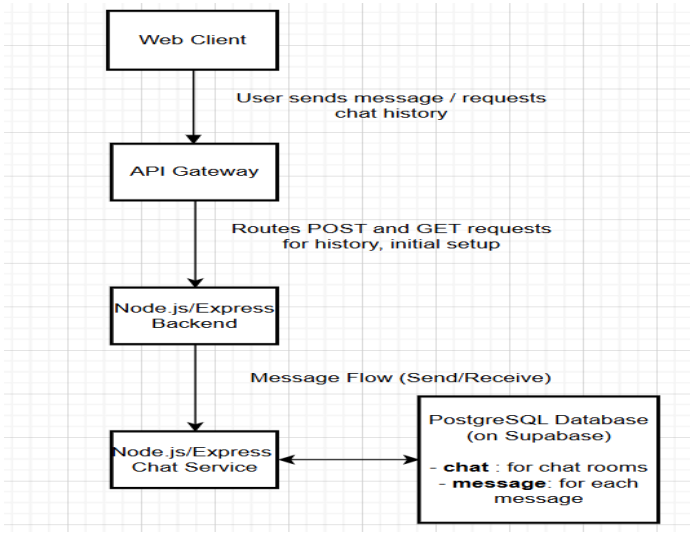
1. Frontend (Chat UI)
    a. React Chat Component: Handles chat message display, input, reactions, and file uploads
    b. Authentication: Uses Supabase Auth to ensure only logged-in users can access chat
    c. Api communication: Sends/receives chat messages and room data via HTTP requests to backend
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



CRC Card for Create Listing Feature

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 all listings	CreateListing, supabase

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

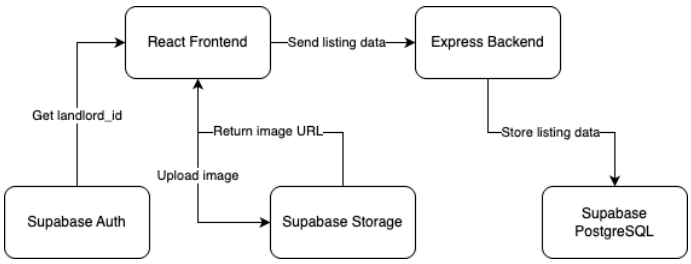
System architecture and reasoning

- 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
- Backend (Express)
    - Saves listing data to Supabase database
    - Retrieves listing data from Supabase database
- 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



CRC Cards for Landlord Profile Creation

Class Name	Parent	Responsibilities	Key Collaborators
LandlordProfile	React.FC	Render profile UI, handle form state, manage profile data	useAuth, landlordApi, useToast
landlordController	Express Controller	Handle CRUD operations, upsert logic, validation	Prisma ORM, Response Utils
landlordApi	API Utility	Frontend API calls, error handling, data transformation	apiFetch, useAuth
useAuth	React Hook	Access auth context, ensure proper usage	AuthContext
useToast	React Hook	Display success/error notifications	Toast Provider

System Interactions and assumptions

Operating System Dependencies

- Browser Environment: Modern web browsers supporting ES6+, File API, FormData
- Network: HTTPS required for secure file uploads and API calls
- Storage: Browser localStorage for role persistence, sessionStorage for temporary data
- File System: Support for file input, drag-and-drop, image preview

Database Dependencies

Supabase PostgreSQL:

- landlord_profiles table with upsert support
- Row Level Security (RLS) policies for data access control
- JSON fields for documents storage
- Timestamp fields for created_at and updated_at

External Service Dependencies

Supabase Services:

- Auth Service: User authentication and session management
- Storage Service: File upload for profile images and documents
- Database Service: Real-time data synchronization

File Storage:

- Supabase Storage buckets for profile images and documents

- Public URL generation for image display
- File type validation and size limits

Compiler/Build Dependencies

- Vite: Build tool and development server with hot reload
- TypeScript: Type checking and compilation with strict mode
- React 18: Component framework with hooks and context
- Prisma: Database ORM with type generation
- Tailwind CSS: Utility-first CSS framework for styling

Reasoning Behind Architecture Selection

- Upsert Pattern over Create/Update
    - Simplicity: Single operation handles both new and existing profiles
    - Reliability: Eliminates race conditions and duplicate key errors
    - User Experience: Seamless profile updates without error messages
- Prisma ORM over Raw SQL
    - Type Safety: Auto-generated types from schema
    - Developer Experience: Intuitive API, better IDE support
    - Performance: Optimized queries and connection pooling
- Supabase Integration over Multiple Services
    - Unified Platform: Auth, database, and storage in one service
    - Security: Built-in authentication and authorization
    - Cost Efficiency: Single billing and reduced operational overhead
- Component-Based Architecture
    - Reusability: Shared components across different profile types
    - Maintainability: Clear separation of concerns
    - Testing: Isolated components easier to unit test
- Error Handling Strategy
    - User-Friendly: Toast notifications with clear error messages
    - Graceful Degradation: Fallback mechanisms for failed operations
    - Logging: Comprehensive error logging for debugging
- File Upload Architecture
    - Security: File type validation and size limits
    - Performance: Client-side image compression and optimization
    - User Experience: Drag-and-drop interface with preview
- State Management Approach
    - React Context: Simple state management for auth and user data

    - Local Storage: Persistent role and preference storage
    - Optimistic Updates: Immediate UI feedback with background sync
- API Design Principles
    - RESTful: Standard HTTP methods and status codes
    - Consistent: Uniform response format across all endpoints
    - Documented: Clear API documentation and examples
- Security Considerations
    - Authentication: JWT-based authentication with proper expiration
    - Authorization: Role-based access control at component and API levels
    - Data Validation: Input sanitization on both client and server

CRC cards for tenant profile creation

Class	Parent	Responsibilities	Collaborators
TenantProfile		Create profile,	AuthContext apiFetch supabase useToast useNavigate CameraButton
TenantProfileService		Handle profile creation business rules, Convert between frontend and database formats, Implement complex validation rules, Manage service-level errors	TenantProfileController ValidationService FileService AuthService
CameraButton		Capture Photos Preview Images Handle File Input Validate Files Notify Parent	Input Components: Text inputs, selects, textareas Validation Rules: Field validation logic Form State: Track form data and validation state Submit Handler: Process form submission
ProfileValidation		Validate Email Validate Phone	Form Fields

		Validate Required Fields Validate File Types Show Error Messages	Error Display Validation Rules User Feedback

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

CRC Cards for Financial Account

Class	Parent	Responsibilities	Collaborators
FinancialAccount	React Component	Display financial overview, Show payment progress, Present payment history	UpcomingPaymentBanner,AuthContext, UI Components
UpcomingPaymentBanner	React Component	Display payment alerts, Show amount and due date	AlertCircle icon, Calendar icon

TenantMyHouse	React Component	Provide navigation to Financial Account, Display button in Quick Actions	Button component, React Router
AuthContext	React Context	Provide user authentication data	FinancialAccount component
Card UI Components	React Component	Provide layout containers, Display summary cards	FinancialAccount component
Button Components	React Component	Handle navigation, Provide support functionality	FinancialAccount component, React Router

System Interactions and assumptions

Operating System Dependencies

- Browser Environment: Modern browsers supporting ES6+, File API, FormData (for uploading payment proof and rendering UI)
- Network: HTTPS required for secure API calls and file uploads
- Storage: Browser localStorage/sessionStorage for user session and role persistence

Database Dependencies

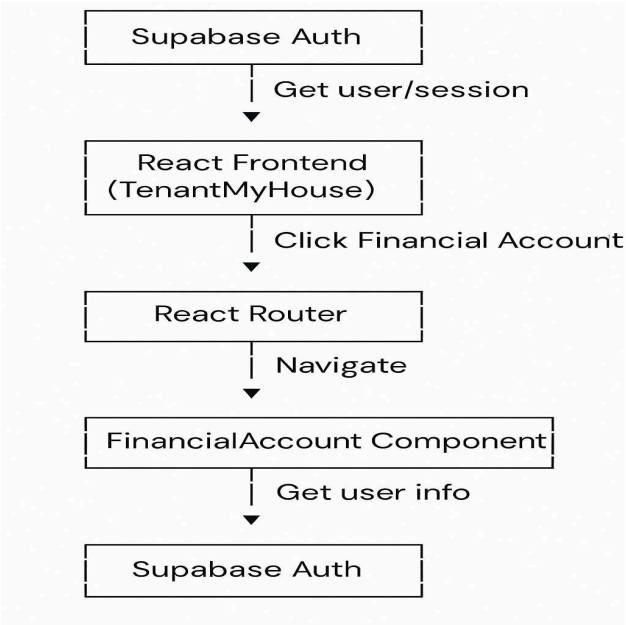
- Supabase PostgreSQL (via Prisma):
    - Used indirectly for authentication (to determine if user is logged in and has access to the button/page

External Service Dependencies

- Supabase Auth Service
    - Provides user authentication and session management (to determine if the button should be shown/enabled)
- React Router
    - Handles navigation from TenantMyHouse to the Financial Account page when the button is clicked

Compiler/Build Dependencies

- Vite: Build tool and dev server
- TypeScript: Type checking and compilation
- React: Component framework for UI (TenantMyHouse, Button)
- ESLint/Prettier: Code linting and formatting



CRC Card for Landlord Payment Tracking

Class Name	**Parent/Subclasses**	**Responsibilities**	**Key Collaborators**
Payments	None	Renders Payment page including a summary of payment data from all listings	landlordController
LandlordPayments	None	Renders a list of all active payment requests and gets approval status changes from landlord	landlordController
ManageListing	None	Renders Manage Listing page for each listing, gets approval updates from the landlord, and sends the updated status to the backend	landlordController, LandlordPayments
landlordController	None	Fetches payment request data from the database	Supabase

System Interactions and assumptions

Operating System Dependencies

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

Database Dependencies

- Supabase PostgreSQL:
    - Stores payment data in payment_requests table (tenant id, listing id, payment amount, approval status)

External Service Dependencies

- Supabase Storage
    - Stores uploaded proof of payment documents
- Supabase Auth
    - Authenticates landlord before getting their listings and each listing's payments

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

Frontend (React)

- Renders Payment page and Manage Listing page with Chart.js bar charts
- Sends approval status updates to the backend
- Uploads listing images to Supabase storage
- Uses Supabase Auth to get the current landlord's id who creates the listing

Backend (Express)

- Retrieves payment request data from Supabase database
- Gets approval status updates from Frontend

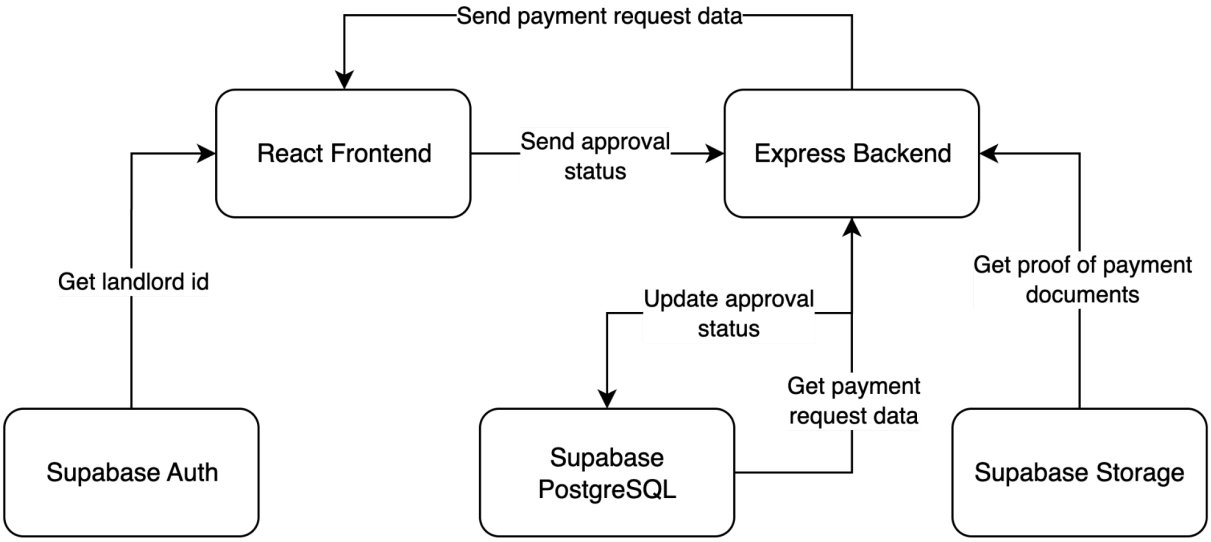
Database (Supabase PostgreSQL, Storage, & Auth)

- Stores payment request data in PostgreSQL table
- Stores proof of payment documents in Supabase storage bucket
- Authenticates landlord, ensuring the landlord only views payment data associated with their own listings

Reasoning Behind Architecture

- React: Flexible with state, effect, and auth hooks for fetching payment 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 payment data
- Supabase Storage: Stores proof of payment documents for easy retrieval
- Supabase Auth: Authenticates current landlord to only render the current landlord's listings and payment data

Diagram


Send payment request data
React Frontend
Send approval status
Express Backend
Get landlord id
Update approval status
Get proof of payment documents
Get payment request data
Supabase Auth
Supabase PostgreSQL
Supabase Storage