

# CribLink Documentation

CribLink is a comprehensive real estate platform designed to connect clients with properties and agents efficiently. It features a robust backend built with Node.js and PostgreSQL, and a dynamic frontend developed using React. The system supports various user roles including clients, agents, and administrators, each with tailored functionalities.

## Table of Contents

* [Features](https://www.google.com/search?q=%23features)
  + [Client Features](https://www.google.com/search?q=%23client-features)
  + [Agent Features](https://www.google.com/search?q=%23agent-features)
  + [Agency Admin Features](https://www.google.com/search?q=%23agency-admin-features)
  + [Admin Features](https://www.google.com/search?q=%23admin-features)
  + [Core Platform Features](https://www.google.com/search?q=%23core-platform-features)
* [Technologies Used](https://www.google.com/search?q=%23technologies-used)
  + [Backend](https://www.google.com/search?q=%23backend)
  + [Frontend](https://www.google.com/search?q=%23frontend)
  + [Database](https://www.google.com/search?q=%23database)
* [Getting Started](https://www.google.com/search?q=%23getting-started)
  + [Prerequisites](https://www.google.com/search?q=%23prerequisites)
  + [Backend Setup](https://www.google.com/search?q=%23backend-setup)
  + [Frontend Setup](https://www.google.com/search?q=%23frontend-setup)
* [Project Structure](https://www.google.com/search?q=%23project-structure)
* [API Endpoints](https://www.google.com/search?q=%23api-endpoints)
* [Contributing](https://www.google.com/search?q=%23contributing)
* [License](https://www.google.com/search?q=%23license)
* [Contact](https://www.google.com/search?q=%23contact)

## Features

CribLink offers a rich set of features catering to its diverse user base:

### Client Features

* **Property Search & Filtering:** Advanced search capabilities to find properties by type, location, price range, bedrooms, and bathrooms.
* **Favourites Management:** Users can save properties, agents, and agencies to their favourites list.
* **Agent Connection:** Clients can send connection requests to agents, accept/reject agent requests, disconnect from agents, and view agent profiles.
* **Inquiry Management:** Create and manage inquiries for properties or general questions, with real-time messaging.
* **Profile Management:** Update personal information, profile pictures, and privacy settings (e.g., sharing favourites with agents, property preferences).
* **Recommended Listings:** View listings recommended by connected agents based on their preferences.
* **Client Activity Feed:** See recent activities related to their inquiries, listing matches, and agent connections.

### Agent Features

* **Client Management:** View, manage, archive, and restore client relationships. Add notes, toggle VIP status for clients, and send emails/messages.
* **Connection Management:** Send connection requests to clients, and manage incoming/outgoing requests (accept/reject).
* **Listing Management:** Create, retrieve, update, and manage property listings (including adding/removing images).
* **Inquiry Management:** View and respond to client inquiries, mark messages as read, and manage conversation status (archive/restore/delete).
* **Profile Management:** Update personal information, bio, location, social links, and profile pictures.
* **Dashboard & Stats:** Access performance metrics like total listings, total inquiries, and counts for "under offer" and "sold" listings.
* **Agent Activity Feed:** Monitor recent activities relevant to their interactions and listings.

### Agency Admin Features

* **Agency Management:** Create, update, and manage agency details including logo and address.
* **Member Management:** View all agents within their agency, approve/reject agent join requests, promote agents to agency admin, demote agency admins to agents, and remove members from the agency.
* **Listing Oversight:** View all listings associated with their agency, including counts for total, pending approvals, under offer, and sold listings.
* **Inquiry Management:** Access and manage all inquiries within their agency, with abilities to archive, restore, and permanently delete conversations.
* **Dashboard & Stats:** Comprehensive statistics for their agency, including agent count, client count, listing counts by status, and pending agent requests.
* **Agency Activity Feed:** Monitor recent activities across their agency, such as new agent joins, listings, and inquiries.
* **Legal Document Management:** Upload, view, and delete legal documents relevant to their agency.
* **Self-Management:** Agency admins can delete their own agency, which will revert their role to 'agent' and disassociate all agents from the deleted agency.

### Admin Features

* **User Management:** Comprehensive control over all user accounts (clients, agents, agency admins, staff).
* **Agency Management:** Full control over all agencies on the platform.
* **Listing Management:** Oversee all property listings, including approval/rejection.
* **Inquiry Management:** Full access and control over all inquiries across the platform.
* **Legal Document Management:** Full control over all legal documents.
* **Platform Analytics:** Access to platform-wide statistics like total users, revenue from sold listings, total deals closed, listing status distribution, user registration trends, and inquiry trends.
* **Platform Activity Feed:** Monitor recent activities across the entire platform.

### Core Platform Features

* **User Authentication & Authorization:** Secure login, registration, and role-based access control.
* **Real-time Communication:** Socket.IO integration for real-time messaging in inquiries.
* **Cloud Storage:** Integration with Cloudinary for secure storage of profile pictures, agency logos, and legal documents.
* **Responsive Design:** Optimized for seamless experience across various devices.
* **Search & Filters:** Robust search and filtering capabilities for properties, agents, and agencies.
* **Notifications:** System for displaying messages and confirmations to users.

## Technologies Used

### Backend

* **Node.js:** Runtime environment for server-side logic.
* **Express.js:** Web application framework for building RESTful APIs.
* **PostgreSQL:** Relational database for data storage.
* **Socket.IO:** For real-time, bidirectional event-based communication.
* **JWT (JSON Web Tokens):** For secure user authentication.
* **Cloudinary:** Cloud-based image and video management.
* **dotenv:** For managing environment variables.
* **cors:** Middleware for enabling Cross-Origin Resource Sharing.
* **slugify:** For generating URL-friendly slugs.

### Frontend

* **React:** JavaScript library for building user interfaces.
* **React Router DOM:** For declarative routing.
* **Tailwind CSS:** Utility-first CSS framework for styling.
* **Framer Motion:** For animations and transitions.
* **Lucide React / Heroicons:** For scalable vector icons.
* **jwt-decode:** For decoding JWT tokens on the client-side.
* **Axios:** Promise-based HTTP client for API requests.
* **React Fitty:** For responsive text.

### Database

* **PostgreSQL:** The primary database, managed with pg (node-postgres) client.
* **db.backup:** Database backup file.
* **schema.sql:** Database schema definition.
* **seed.sql:** Seed data for the database.

## Getting Started

### Prerequisites

* Node.js (v18 or higher recommended)
* npm or Yarn
* PostgreSQL database
* Cloudinary account (for image/document uploads)

### Backend Setup

1. **Clone the repository:**
2. git clone https://github.com/tobyt50/CribLink.git
3. cd CribLink/criblink-backend
4. **Install dependencies:**
5. npm install
6. **Configure environment variables:** Create a .env file in criblink-backend/ with the following:

PORT=5000

DATABASE\_URL="postgresql://user:password@host:port/database"

JWT\_KEY="your\_secret\_jwt\_key"

CLOUDINARY\_CLOUD\_NAME="your\_cloudinary\_cloud\_name"

CLOUDINARY\_API\_KEY="your\_cloudinary\_api\_key"

CLOUDINARY\_API\_SECRET="your\_cloudinary\_api\_secret"

Replace placeholders with your actual database and Cloudinary credentials.

1. **Database Setup:**
   * Create a PostgreSQL database.
   * Run the schema and seed scripts:
   * psql -d your\_database\_name -f db/schema.sql
   * psql -d your\_database\_name -f db/seed.sql
2. **Start the backend server:**
3. npm run dev

### Frontend Setup

1. **Navigate to the frontend directory:**
2. cd ../criblink-frontend
3. **Install dependencies:**
4. npm install
5. **Configure environment variables:** Create a .env.production (or .env.development) file in criblink-frontend/ with:

REACT\_APP\_API\_BASE\_URL="http://localhost:5000"

Adjust the URL if your backend is hosted elsewhere.

1. **Start the frontend development server:**
2. npm start

## Project Structure

The project is divided into criblink-backend and criblink-frontend directories.

.

├── .gitignore

**├── criblink-backend/**

│ ├── .gitignore

│ ├── .node-version

│ ├── app.js # Main Express app, Socket.IO setup, and route mounting

│ ├── controllers/ # Business logic for API endpoints

│ │ ├── activityController.js # Admin activity logs

│ │ ├── adminStatsController.js # Admin dashboard statistics

│ │ ├── agencyAdminController.js # Agency admin profile and management

│ │ ├── agencyController.js # Agency creation, updates, membership

│ │ ├── agencyStatsController.js # Agency-specific statistics

│ │ ├── agentController.js # Agent profile, client connection requests

│ │ ├── agentPerformanceController.js # Agent performance data (admin view)

│ │ ├── agentStatsController.js # Agent dashboard statistics

│ │ ├── analyticsController.js # Platform-wide analytics

│ │ ├── clientController.js # Client profile, agent connections, recommendations

│ │ ├── clientStatsController.js # Client-specific statistics

│ │ ├── documentController.js # Legal document management

│ │ ├── favouritesController.js # User favourites (properties, agents, clients, agencies)

│ │ ├── financeController.js # Financial data (placeholder)

│ │ ├── inquiriesController.js # Inquiry and messaging logic

│ │ ├── listingsController.js # Property listing CRUD

│ │ ├── settingsController.js # User settings (client, agent)

│ │ ├── ticketController.js # Support ticket management (placeholder)

│ │ └── userController.js # User authentication and profile management

│ ├── db/

│ │ └── index.js # PostgreSQL database connection pool

│ ├── middleware/

│ │ ├── authMiddleware.js # JWT authentication and role-based authorization

│ │ └── authenticateAdmin.js # Admin-specific authentication (can be merged into authMiddleware)

│ ├── package-lock.json

│ ├── package.json

│ ├── routes/ # API route definitions

│ │ ├── admin/ # Admin-specific routes

│ │ │ ├── activity.js

│ │ │ ├── adminStatsRoutes.js

│ │ │ ├── agentPerformanceRoutes.js

│ │ │ ├── analytics.js

│ │ │ ├── settings.js

│ │ │ ├── staff.js

│ │ │ └── users.js

│ │ ├── agencyAdminRoutes.js # Agency Admin specific routes

│ │ ├── agencyRoutes.js # Agency related routes

│ │ ├── agencyStatsRoutes.js # Agency statistics routes

│ │ ├── agentRoutes.js # Agent specific routes

│ │ ├── agentSettings.js # Agent settings routes

│ │ ├── agentStatsRoutes.js # Agent statistics routes

│ │ ├── clientRoutes.js # Client specific routes

│ │ ├── clientSettings.js # Client settings routes

│ │ ├── clientStatsRoutes.js # Client statistics routes

│ │ ├── documentRoutes.js # Legal document routes

│ │ ├── favouritesRoutes.js # Favourites routes

│ │ ├── financeRoutes.js # Finance routes

│ │ ├── inquiriesRoutes.js # Inquiry and messaging routes

│ │ ├── listingsRoutes.js # Listing routes

│ │ ├── ticketRoutes.js # Ticket routes

│ │ └── userRoutes.js # User authentication and profile routes

│ ├── uploads/ # (Deprecated for local storage, now uses Cloudinary)

│ │ └── [example\_image.png]

│ └── utils/

│ ├── cloudinary.js # Cloudinary integration for file uploads

│ └── logActivity.js # Utility for logging system activities

**├── criblink-frontend/**

│ ├── .env.production # Environment variables for production

│ ├── .gitignore

│ ├── README.md

│ ├── package-lock.json

│ ├── package.json

│ ├── postcss.config.js

│ ├── public/ # Public assets

│ │ ├── favicon.ico

│ │ ├── index.html

│ │ └── ...

│ ├── src/

│ │ ├── App.js # Main React application component

│ │ ├── api/

│ │ │ └── axiosInstance.js # Axios instance for API calls

│ │ ├── assets/ # Static assets (logos, images)

│ │ ├── components/ # Reusable UI components

│ │ │ ├── AgencyCard.js

│ │ │ ├── AgentInquiryModal.js

│ │ │ ├── AxiosErrorInterceptor.js

│ │ │ ├── ClientInquiryModal.js

│ │ │ ├── ConfirmDialog.js # Custom confirmation dialog

│ │ │ ├── DocumentCard.js # Card component for legal documents

│ │ │ ├── Footer.js

│ │ │ ├── GlobalMessageToasts.js

│ │ │ ├── Header.js

│ │ │ ├── HomeSearchFilters.js

│ │ │ ├── InquiryModal.js

│ │ │ ├── ListingCard.js

│ │ │ ├── LoadingSpinner.js # Global loading spinner

│ │ │ ├── ProfileMenu.js

│ │ │ ├── ProtectedBaseRoute.js # Route protection for authenticated users

│ │ │ ├── PurchaseCategoryFilter.js

│ │ │ ├── RoleProtectedRoute.js # Route protection for specific roles

│ │ │ ├── ScrollToTop.js

│ │ │ ├── SearchBar.js

│ │ │ ├── SearchFilters.js

│ │ │ ├── ShareModal.js # Listing sharing modal

│ │ │ ├── StatCard.js # Reusable statistic display card

│ │ │ ├── admin/ # Admin-specific components

│ │ │ ├── agencyadmin/ # Agency Admin-specific components

│ │ │ ├── agent/ # Agent-specific components

│ │ │ ├── client/ # Client-specific components

│ │ │ ├── profile/ # User profile components

│ │ │ └── ui/ # General UI components (e.g., Card)

│ │ ├── config.js # Frontend configuration (API base URL)

│ │ ├── context/ # React Context APIs for global state

│ │ │ ├── AuthContext.js # User authentication state

│ │ │ ├── ConfirmDialogContext.js # Confirmation dialog state

│ │ │ ├── LoadingContext.js # Global loading state

│ │ │ └── MessageContext.js # Global message/toast state

│ │ ├── hooks/

│ │ │ └── useSidebarState.js # Custom hook for sidebar management

│ │ ├── index.css # Main CSS (Tailwind imports and custom styles)

│ │ ├── index.js # React app entry point

│ │ ├── layouts/

│ │ │ ├── AppShell.js # Main application layout

│ │ │ └── MainLayout.js

│ │ ├── pages/ # Main application pages

│ │ │ ├── AboutUs.js

│ │ │ ├── AddLegalDocument.js # Page for uploading legal documents

│ │ │ ├── AddListing.js

│ │ │ ├── Agencies.js # Page to browse all agencies

│ │ │ ├── AgencyProfile.js # Public agency profile page

│ │ │ ├── ContactUs.js

│ │ │ ├── EditListing.js

│ │ │ ├── Favourites.js # User favourites page

│ │ │ ├── Home.js

│ │ │ ├── LegalDocuments.js # Page to manage legal documents

│ │ │ ├── ListingDetails.js

│ │ │ ├── Listings.js

│ │ │ ├── ManageProfile.js # User profile management hub

│ │ │ ├── NotFoundPage.js

│ │ │ ├── ResetPassword.js

│ │ │ ├── SearchPage.js

│ │ │ ├── SelectRole.js

│ │ │ ├── SignIn.js

│ │ │ ├── SignUp.js

│ │ │ ├── admin/ # Admin dashboard pages

│ │ │ ├── agencyadmin/ # Agency Admin dashboard pages

│ │ │ ├── agent/ # Agent dashboard pages

│ │ │ ├── client/ # Client dashboard pages

│ │ │ └── profile/ # User profile sub-pages (General, Privacy, Security)

│ │ ├── socket.js # Socket.IO client setup

│ │ └── utils/

│ │ ├── authUtils.js # Authentication utility functions

│ │ └── handleApiError.js # Centralized API error handling

│ └── tailwind.config.js

├── db connect.txt # Database connection strings

├── db.backup # Database backup file

├── docs/ # Additional documentation files

│ ├── Agency Admin Roles and Priviledges.docx

│ ├── CribLink Business Proposal.docx

│ ├── CribLink\_Agency\_Integration\_Proposal.pdf

│ ├── CribLink\_Documentation\_Complete.docx # This document

│ ├── CribLink\_Information\_System\_Blueprint.pdf

│ ├── admin account creation.txt

│ ├── criblink\_information\_system\_UPDATED.xlsx

│ ├── id\_sequence fix.sql

│ └── pgAdmin4\_Backup\_Guide.pdf

├── git pull.txt

└── schema.sql # Database schema

## API Endpoints

The backend exposes a comprehensive set of RESTful API endpoints. Key routes include:

### User Management

* /users: User registration, login, profile management.
  + POST /users/register: Register a new user.
  + POST /users/login: Authenticate user and issue JWT.
  + GET /users/profile: Get authenticated user's profile.
  + PUT /users/update: Update authenticated user's profile.
  + PUT /users/profile/picture/upload: Upload/update profile picture.
  + DELETE /users/profile/picture/delete: Delete profile picture.
  + POST /users/logout: Invalidate user session.
  + GET /users/agency-status: Get an agent's agency affiliation status.
  + GET /users/:userId/agency-memberships: Get all agency memberships for a specific agent.
  + PUT /users/update-password: Update user's password.
  + POST /users/forgot-password: Initiate password reset.
  + POST /users/reset-password: Complete password reset.
  + GET /admin/users: (Admin) Get all users.
  + GET /admin/users/:id: (Admin) Get user by ID.
  + PUT /admin/users/:id: (Admin) Update user by ID.
  + DELETE /admin/users/:id: (Admin) Delete user by ID.
  + GET /admin/staff: (Admin) Get all staff users.
  + POST /admin/staff: (Admin) Add new staff user.
  + PUT /admin/staff/:id: (Admin) Update staff user.
  + DELETE /admin/staff/:id: (Admin) Delete staff user.

### Property Listings

* /listings: Property listing creation, retrieval, updates, and search.
  + POST /listings: Create a new listing.
  + GET /listings: Get all listings (with search, filter, pagination).
  + GET /listings/:id: Get a single listing by ID.
  + PUT /listings/:id: Update a listing.
  + DELETE /listings/:id: Delete a listing.
  + POST /listings/:id/images: Add images to a listing.
  + DELETE /listings/:id/images/:imageId: Delete a listing image.
  + PUT /listings/:id/status: Update listing status (e.g., 'available', 'sold', 'pending').
  + GET /listings/:id/similar: Get similar listings.
  + GET /listings/agent/:agentId: Get listings by a specific agent.

### Inquiries and Communication

* /inquiries: Managing property inquiries and general communication.
  + POST /inquiries: Create a new property inquiry (guest or client).
  + POST /inquiries/general: Create a new general inquiry (initiated by agent from client profile).
  + POST /inquiries/message: Send a message within an existing conversation.
  + GET /inquiries/client: (Client) Fetch all inquiries for the client.
  + PUT /inquiries/client/mark-read/:conversationId: (Client) Mark messages as read.
  + DELETE /inquiries/client/delete-conversation/:conversationId: (Client) Delete a conversation.
  + GET /inquiries/agent: (Agent/Admin/Agency Admin) Fetch all active inquiries.
  + GET /inquiries/agent/archived: (Agent/Admin/Agency Admin) Fetch all archived inquiries.
  + PUT /inquiries/:conversationId/archive-for-agent: (Agent/Admin/Agency Admin) Archive an inquiry.
  + PUT /inquiries/:conversationId/restore-for-agent: (Agent/Admin/Agency Admin) Restore an inquiry.
  + DELETE /inquiries/:conversationId/permanently-delete: (Admin/Agency Admin) Permanently delete an inquiry.
  + GET /inquiries/agent/:agentId/client/:clientId/conversation: Fetch a specific conversation between an agent and a client.
  + PUT /inquiries/agent/assign/:inquiryId: (Agent/Admin/Agency Admin) Assign an inquiry to an agent.
  + PUT /inquiries/agent/mark-read/:conversationId: (Agent/Admin/Agency Admin) Mark messages as read.
  + PUT /inquiries/agent/mark-opened/:conversationId: (Agent/Admin/Agency Admin) Mark conversation as opened.
  + PUT /inquiries/agent/mark-responded/:conversationId: (Agent/Admin/Agency Admin) Mark conversation as responded.
  + DELETE /inquiries/agent/delete-conversation/:conversationId: (Agent/Admin/Agency Admin) Soft delete/hide a conversation.
  + GET /inquiries/agent/count/all-inquiries: (Agent/Admin/Agency Admin) Get total inquiries count.
  + GET /inquiries/agent/count/unread-inquiries: (Agent/Admin/Agency Admin) Get unread inquiries count.

### Client-Agent Connections

* /clients: Agent-specific client management.
  + GET /clients/all-agents: (Client) Get all agents for browsing.
  + GET /clients/agent/:agentId/clients: (Agent/Admin/Agency Admin) Get all clients for a specific agent.
  + GET /clients/:clientId: Get a specific client's profile.
  + GET /clients/:clientId/preferences: Get client's property preferences.
  + PUT /clients/:clientId/preferences: Update client's property preferences.
  + GET /clients/:clientId/recommendations: (Agent) Get recommended listings for a client.
  + POST /clients/:clientId/recommendations/:propertyId: (Agent) Add a recommended listing.
  + DELETE /clients/:clientId/recommendations/:propertyId: (Agent) Remove a recommended listing.
  + GET /clients/:clientId/recommendations/agent/:agentId: (Client/Admin) Get recommended listings for a client from a specific agent.
  + POST /clients/:clientId/connection-requests/send-to-agent/:agentId: (Client) Send connection request to an agent.
  + GET /clients/:clientId/connection-requests/incoming: (Client) Get incoming connection requests.
  + GET /clients/:clientId/connection-requests/outgoing: (Client) Get outgoing connection requests.
  + GET /clients/:clientId/pending-agent-requests: (Client) Get all pending (incoming/outgoing) connection requests.
  + POST /clients/:clientId/connection-requests/:requestId/accept-from-agent: (Client) Accept connection request from an agent.
  + POST /clients/:clientId/connection-requests/:requestId/reject-from-agent: (Client) Reject connection request from an agent.
  + GET /clients/:clientId/connection-requests/status/:agentId: Get connection status between a client and an agent.
  + POST /clients/:clientId/disconnect-agent/:agentId: (Client) Disconnect from an agent.
  + GET /clients/:clientId/connected-agent: (Client) Get details of connected agent(s).
  + POST /clients/agent/:agentId/clients/:clientId/email: (Agent/Admin) Send email to client.
  + POST /clients/agent/:agentId/clients/:clientId/respond: (Agent/Admin) Respond to client inquiry (deprecated by inquiries/message).
  + PUT /clients/agent/:agentId/clients/:clientId/note: (Agent) Add/update note for a client.
  + POST /clients/agent/:agentId/clients/:clientId/message: (Agent/Admin) Send message to client (deprecated by inquiries/message).
  + PUT /clients/agent/:agentId/clients/:clientId/vip: (Agent) Toggle VIP status for client.
  + DELETE /clients/agent/:agentId/clients/:clientId: (Agent) Archive client.
  + DELETE /clients/agent/:agentId/archived-clients/:clientId: (Agent/Admin) Delete archived client permanently.
  + GET /clients/agent/:agentId/archived-clients: (Agent/Admin) Get archived clients.
  + POST /clients/agent/:agentId/archived-clients/:clientId/restore: (Agent) Restore archived client.
  + GET /clients/agency/:agencyId/clients: (Agency Admin/Admin) Get clients by agency ID.

### Agent Management

* /agents: Agent profile and connection management.
  + GET /agents/profile/:agentId: Get agent profile details.
  + POST /agents/:agentId/connection-requests/send-to-client/:clientId: (Agent/Agency Admin) Send connection request to a client.
  + GET /agents/:agentId/connection-requests/incoming: (Agent/Agency Admin) Get incoming connection requests from clients.
  + GET /agents/:agentId/connection-requests/outgoing: (Agent/Agency Admin) Get outgoing connection requests to clients.
  + POST /agents/:agentId/connection-requests/:requestId/accept-from-client: (Agent/Agency Admin) Accept client connection request.
  + POST /agents/:agentId/connection-requests/:requestId/reject-from-client: (Agent/Agency Admin) Reject client connection request.

### Favourites

* /favourites: User favourite listings, agents, clients, and agencies.
  + POST /favourites/properties: Add a property to favourites.
  + DELETE /favourites/properties/:property\_id: Remove a property from favourites.
  + GET /favourites/properties: Get all favourite properties for the authenticated user.
  + GET /favourites/properties/status/:property\_id: Check if a property is favourited.
  + POST /favourites/agents: Add an agent to favourites.
  + DELETE /favourites/agents/:agent\_id: Remove an agent from favourites.
  + GET /favourites/agents: Get all favourite agents for the authenticated user.
  + GET /favourites/agents/status/:agent\_id: Check if an agent is favourited.
  + POST /favourites/clients: Add a client to favourites (for Agents/Agency Admins).
  + DELETE /favourites/clients/:client\_id: Remove a client from favourites.
  + GET /favourites/clients: Get all favourite clients for the authenticated user.
  + GET /favourites/clients/status/:client\_id: Check if a client is favourited.
  + POST /favourites/agencies: Add an agency to favourites.
  + DELETE /favourites/agencies/:agency\_id: Remove an agency from favourites.
  + GET /favourites/agencies: Get all favourite agencies for the authenticated user.
  + GET /favourites/agencies/status/:agency\_id: Check if an agency is favourited.

### Agency Management

* /agencies: Agency creation, management, and agent affiliation.
  + POST /agencies: (Admin) Create a new agency.
  + POST /agencies/register-agent-agency: (Agent) Register a new agency and become its agency admin.
  + GET /agencies: Get all agencies.
  + GET /agencies/:id: Get a single agency by ID.
  + PUT /agencies/:id: (Agency Admin/Admin) Update an agency's details.
  + DELETE /agencies/:id: (Admin) Delete an agency.
  + POST /agencies/request-to-join: (Agent) Request to join an agency.
  + PUT /agencies/approve-join-request/:requestId: (Agency Admin/Admin) Approve an agent's join request.
  + PUT /agencies/reject-join-request/:requestId: (Agency Admin/Admin) Reject an agent's join request.
  + GET /agencies/:agencyId/agents: (Agency Admin/Admin) Get all agents (members) for a specific agency.
  + GET /agencies/:agencyId/pending-requests: (Agency Admin/Admin) Get pending join requests for an agency.
  + DELETE /agencies/:agencyId/members/:agentId: (Agency Admin/Admin/Agent) Remove an agent from an agency.
  + DELETE /agencies/:agencyId/admin-delete: (Agency Admin) Agency admin deletes their agency.
  + PUT /agencies/:agencyId/members/:memberId/promote-to-admin: (Agency Admin/Admin) Promote a member to agency admin.
  + PUT /agencies/:agencyId/members/:memberId/demote-to-agent: (Agency Admin/Admin) Demote a member to agent.
  + PUT /agencies/:agencyId/members/:memberId/status: (Agency Admin/Admin) Update member status (VIP/Regular).
  + GET /agencies/:agencyId/admin-count: (Agency Admin/Admin) Get count of agency administrators.
  + GET /agencies/:agencyId/listings: Get all listings associated with a specific agency.

### Agency Admin Specific Routes

* /agency-admins: Agency admin profile and management.
  + GET /agency-admins/profile/:adminId: Get agency admin profile details.

### Statistics and Analytics

* /admin-stats: Admin dashboard statistics.
  + GET /admin-stats/users/count: Total users count.
  + GET /admin-stats/agencies/count: Total agencies count.
  + GET /admin-stats/agents/count: Total agents count.
  + GET /admin-stats/clients/count: Total clients count.
  + GET /admin-stats/listings/count: Total listings count.
  + GET /admin-stats/listings/available/count: Total available listings count.
  + GET /admin-stats/listings/sold/count: Total sold listings count.
  + GET /admin-stats/inquiries/count: Total inquiries (conversations) count.
  + GET /admin-stats/inquiries/responses/count: Total agent responses count.
  + GET /admin-stats/documents/count: Total legal documents count.
  + GET /admin-stats/activity/recent-activity: Platform-wide recent activity.
* /agent-stats: Agent dashboard statistics.
  + GET /agent-stats/dashboard/stats: Agent's total listings and inquiries.
  + GET /agent-stats/dashboard/activity: Agent's recent activity.
  + GET /agent-stats/listings/under-offer/count: Agent's under offer listings count.
  + GET /agent-stats/listings/sold/count: Agent's sold listings count.
* /agency-stats: Agency-specific statistics.
  + GET /agency-stats/:agencyId/agents/count: Count of agents affiliated with an agency.
  + GET /agency-stats/:agencyId/admins/count: Count of agency administrators for an agency.
  + GET /agency-stats/:agencyId/clients/count: Count of clients connected to agents in an agency.
  + GET /agency-stats/:agencyId/listings/count: Count of listings associated with an agency.
  + GET /agency-stats/:agencyId/listings/pending-approvals: Count of pending listings for an agency.
  + GET /agency-stats/:agencyId/pending-agent-requests/count: Count of pending agent join requests for an agency.
  + GET /agency-stats/:agencyId/listings/under-offer/count: Count of 'under offer' listings for an agency.
  + GET /agency-stats/:agencyId/listings/sold/count: Count of 'sold' listings for an agency.
  + GET /agency-stats/:agencyId/recent-activity: Recent activity for a specific agency.
* /client-stats: Client-specific statistics.
  + GET /client-stats/saved-listings/count: Client's saved listings count.
  + GET /client-stats/:clientId/recommendations/agent/:agentId/count: Recommended listings count for a client from a specific agent.
  + GET /client-stats/activity/recent-activity: Client's recent activity.
  + GET /client-stats/inquiries/client/count/all-inquiries: Client's total inquiries count.
* /admin/analytics: Platform-wide analytics for administrators.
  + GET /admin/analytics/stats/listings-count: Total listings count.
  + GET /admin/analytics/stats/inquiries-count: Total inquiries count.
  + GET /admin/analytics/total-users-count: Total users count.
  + GET /admin/analytics/revenue-sold-listings: Revenue from sold listings.
  + GET /admin/analytics/total-deals-closed: Total deals closed.
  + GET /admin/analytics/listing-status: Listing status distribution.
  + GET /admin/analytics/listings-over-time: Listings added over time.
  + GET /admin/analytics/user-registrations: User registration trends.
  + GET /admin/analytics/inquiry-trends: Inquiry trends.
  + GET /admin/analytics/property-type-distribution: Property type distribution.
  + GET /admin/analytics/user-role-distribution: User role distribution.

### Legal Documents

* /docs: Legal documents.
  + POST /docs/upload: (Admin/Agency Admin) Upload a new legal document.
  + GET /docs: (Admin/Agency Admin) Get all legal documents (with filters/pagination).
  + DELETE /docs/:id: (Admin/Agency Admin) Delete a legal document.

### Financials

* /finances: Financial overview (currently a placeholder).
  + GET /finances: Get all financial data.

### Support Tickets

* /tickets: Support ticket management (currently a placeholder).
  + GET /tickets: Get all tickets.

For detailed API documentation, refer to the Postman collection or explore the criblink-backend/routes directory.

## Contributing

Contributions are welcome! If you'd like to contribute, please follow these steps:

1. Fork the repository.
2. Create a new branch (git checkout -b feature/your-feature-name).
3. Make your changes.
4. Commit your changes (git commit -m 'Add new feature').
5. Push to the branch (git push origin feature/your-feature-name).
6. Open a Pull Request.

Please ensure your code adheres to the existing style and includes relevant tests.

## License

This project is licensed under the MIT License.

## Contact

For any inquiries or feedback, please contact:

Matthew Tedunjaiye  
[tedunjaiyem@gmail.com](mailto:tedunjaiyem@gmail.com)

[https://github.com/tobyt50/CribLink/tree/main](https://www.google.com/search?q=https://github.com/tobyt50/CribLink/tree/main)