Frontend Specification — Client–Accountant Collaboration Portal (Part 1/4)

Version: 1.0
Date: 2025-08-09
Prepared for: Wazeen
Prepared by: ChatGPT — Frontend Architect (React + Tailwind)

---

1. Project Overview

The **Client–Accountant Collaboration Portal** is a secure, scalable, and responsive web application that enables streamlined communication, document sharing, and task management between clients and accountants. It will be built with **React.js** and **Tailwind CSS**, integrated with a **Django backend** via REST APIs and WebSockets for real-time features.

Core Features: - Service request creation, assignment, and tracking. - Real-time chat with typing indicators, read receipts, and emoji reactions. - File management with preview, versioning, and secure download. - Role-based dashboards (Admin, Accountant, Client). - Multilingual UI (English + Arabic).

Platform Goals: - Provide an intuitive, Jira/Stripe-style modern interface. - Support both desktop and mobile usage. - Maintain high performance even with large datasets. - Ensure accessibility and compliance with WCAG 2.1 AA.

---

2. Objectives & Success Criteria

Objectives: 1. Deliver a **modern, clean, and compact** UI experience. 2. Achieve **real-time collaboration** between users. 3. Ensure **data consistency** across multiple devices and sessions. 4. Maintain **security** with JWT authentication, role-based access, and secure file handling. 5. Implement **responsive layouts** for different screen sizes.

Success Criteria: - Page load under **2 seconds** on a standard 4G connection. - 95%+ Lighthouse performance and accessibility score. - Zero critical accessibility violations at launch. - Seamless switch between English and Arabic without reload. - 99.9% uptime for chat and notifications.

---

3. Constraints & Considerations

- **Backend Alignment:** Must match provided PostgreSQL schema and API endpoints.
- **Deployment Region:** Optimized for Arab/MENA region latency.
- **Language Direction:** Arabic requires RTL layout support.
- **Performance:** Must support up to 50 concurrent chat rooms with 200+ active users without lag.
- **Security:** All file downloads must be tokenized and logged.
- **Browser Support:** Chrome, Firefox, Safari, Edge (last 2 versions).

---

4. Tech Stack & Tools

Frontend: - **React.js** (with Hooks) - **Tailwind CSS** (custom theme) - **React Query** (server state management) - **Zustand** (UI state management) - **React Router DOM** (routing) - **React Hook Form** (form handling) - **Framer Motion** (animations) - **i18next** (internationalization) - **Lucide Icons** (iconography)

Build & Tooling: - **Vite** (fast dev server + bundler) - **ESLint** + **Prettier** (code quality) - **Storybook** (component documentation) - **Jest** + **React Testing Library** (unit/integration tests) - **Playwright** (E2E tests)

Collaboration & CI/CD: - GitHub + GitHub Actions - Vercel/Netlify for frontend deployment - Sentry for error tracking

---

5. Design System Overview

Colors: - Primary: Blue (#1D4ED8) - Secondary: Slate Gray (#64748B) - Accent: Emerald (#10B981) - Neutral: Grayscale palette for backgrounds/borders

Typography: - Headings: Inter, 700 weight - Body: Inter, 400/500 weight - Sizes: Responsive scale (xs–4xl)

Spacing: - 4px baseline grid

Components: - Buttons (primary, secondary, icon-only) - Inputs (text, textarea, select, datepicker) - Modals, tooltips, dropdowns - Data tables with sorting, filtering, pagination - Tabs, accordions, and collapsibles

Accessibility: - All interactive elements keyboard navigable - Focus states clearly visible - Color contrast meets WCAG AA - Live regions for dynamic content updates

---

Next: Part 2 will cover **Architecture, State & Data Flow, Routing, and API Integration Principles**.

Frontend Specification — Client–Accountant Collaboration Portal (Part 2/4)

Version: 1.0
Date: 2025-08-09
Prepared for: Wazeen
Prepared by: ChatGPT — Frontend Architect (React + Tailwind)

---

6. Architecture Overview

The frontend will follow a **feature-first modular architecture** to ensure scalability, maintainability, and ease of onboarding for new developers.

Key Layers:

1. **Presentation Layer** — React components styled with Tailwind CSS and driven by props/state.
2. **State Management Layer** — React Query for server state, Zustand for UI state.
3. **Service Layer** — API abstraction layer for REST and WebSocket communication.
4. **Routing Layer** — Handled by React Router DOM with role-based route protection.
5. **Utility Layer** — Reusable helpers (formatting, validation, i18n wrappers, constants).

Directory Structure:

```
src/
  components/            # Reusable UI components
  features/      # Feature-specific modules (chat, files, requests)
  hooks/         # Custom hooks
  layouts/       # Layout components (AppShell, AuthLayout)
  pages/         # Page-level components
  services/      # API and WebSocket clients
  store/         # Zustand state stores
```

```
utils/          # Utility functions
i18n/           # Translation files and helpers
```

7. State & Data Flow

Server State:

- Managed by **React Query** with automatic cache updates, background refetching, and pagination.
- Query Keys Examples:
  - ['serviceRequests'] for listing all service requests.
  - ['chatMessages', roomId] for messages in a specific room.

UI State:

- Managed by **Zustand** for lightweight, predictable UI state (modals, sidebars, filters).

Example Flow:

1. User sends a chat message.
2. sendMessage() in chatService sends it to the backend.
3. WebSocket pushes the message to subscribed clients.
4. React Query invalidates and refetches ["chatMessages", roomId].
5. UI updates instantly.

8. Routing Structure

- **Public Routes:** /login, /register, /forgot-password.
- **Protected Routes:** /dashboard, /service-requests, /files, /chat, /settings.
- **Role-Based Access:** Admin, Accountant, Client.

Example Route Tree:

```
/
 /login
 /register
 /dashboard
 /service-requests
        /:id
 /files
        /:id
 /chat
        /:roomId
 /settings
```

Features:

- Lazy loading of feature modules.
- Bottom navigation for mobile.
- Sidebar navigation for desktop.

9. API Integration Principles

- Centralized apiClient module handles all API calls using Axios.
- Automatic JWT injection from auth store.
- Consistent response handling with success/error states.
- Toast notifications for API errors.
- Type-safe API calls with JSDoc/TypeScript.

Example:

```
import apiClient from '@/services/apiClient';

export const fetchServiceRequests = () => {
  return apiClient.get('/api/service-requests/');
};
```

Next: Part 3 will cover **UI Components, Feature Modules, and Detailed User Flows**.

Frontend Specification — Client–Accountant Collaboration Portal (Part 3/4)

Version: 1.0
Date: 2025-08-09
Prepared for: Wazeen
Prepared by: ChatGPT — Frontend Architect (React + Tailwind)

---

10. UI Component Library

The component system will be **atomic** (Atoms, Molecules, Organisms, Templates, Pages) to ensure reusability and consistent styling.

10.1 Atoms

- **Button** — Primary, Secondary, Icon, Text.

- **Input** — Text, Password, Email, Search, Multiline.
- **Select** — Single, Multi, Async.
- **Badge** — Status indicators (e.g., request priority).
- **Avatar** — User profile image with fallback initials.
- **Spinner** — Loading indicator.
- **Tooltip** — Info popups.

10.2 Molecules

- **FormGroup** — Label + Input + Error message.
- **DropdownMenu** — Contextual actions.
- **FileItem** — File name, size, category badge.
- **MessageBubble** — Chat message with timestamp and reactions.
- **TagList** — List of tags with remove option.

10.3 Organisms

- **ServiceRequestCard** — Summary of a request.
- **ChatWindow** — Message list, composer, typing indicators.
- **FileManager** — List/grid of files with folder navigation.
- **DashboardStats** — Request counts, charts, summaries.
- **SettingsForm** — User preferences and chat settings.

10.4 Templates & Layouts

- **AppShell** — Sidebar + Header + Content.
- **AuthLayout** — Centered form layout.
- **DetailView** — Split pane with details and related items.

---

11. Feature Modules

11.1 Authentication

- Login, Register, Forgot Password, Email Verification.
- JWT-based authentication flow.
- Inline field validation for forms.

11.2 Service Requests

- List view with filters (status, priority, tags).
- Detail view with notes, status updates, and reassignment.
- Wizard-style request creation.

- Real-time status updates via WebSockets.

11.3 File Management

- Folder-like structure with drag-and-drop uploads.
- File preview for supported types.
- Versioning and tagging.
- Logged downloads with IP & timestamp.

11.4 Chat

- Slack-like threaded conversations.
- Reactions and read receipts.
- Typing indicators.
- Searchable message history.
- Real-time updates.

11.5 Settings

- Profile & avatar update.
- Notification preferences.
- Language toggle (EN/AR).

---

12. Detailed User Flows

12.1 Client Creates Service Request

1. Client clicks **New Request**.
2. Wizard opens → Step 1: Title & Category → Step 2: Description & Priority → Step 3: Due Date & Tags.
3. Client submits form.
4. API call: POST /api/service-requests/.
5. Request appears in dashboard instantly.

12.2 Accountant Uploads File

1. Accountant opens request detail.
2. Navigates to **Files** tab.
3. Clicks **Upload File**, selects category & folder.
4. API call: POST /api/file-management/requests/{id}/files/upload/.
5. File list updates in real-time.

12.3 Real-Time Chat

1. User opens chat room.
2. WebSocket connects to /ws/chat/{roomId}/.
3. Messages load via GET /api/chat/rooms/{id}/messages/.
4. New message sent → pushed to all participants.
5. Typing indicators update live.

Next: Part 4 will cover **Performance Optimization, Security, Testing, and Deployment Plan**.

Frontend Specification — Client–Accountant Collaboration Portal (Part 4/4)

Version: 1.0
Date: 2025-08-09
Prepared for: Wazeen
Prepared by: ChatGPT — Frontend Architect (React + Tailwind)

---

13. Performance Optimization

To ensure a fast, responsive experience globally, the frontend will use:

13.1 Code Splitting & Lazy Loading

- Use **React.lazy** and **dynamic imports** for routes and heavy components.
- Split vendor bundles to improve initial load time.

13.2 Asset Optimization

- Compress and optimize images via build pipeline.
- Use WebP/AVIF for modern browsers.
- Serve static assets from a CDN.

13.3 Caching & Prefetching

- Implement **React Query** caching for API responses.
- Prefetch frequently accessed data (e.g., dashboard stats) when idle.

13.4 Virtualization

- Use **react-window** or similar for large lists (e.g., chat messages, files).

---

14. Security Considerations

Frontend will strictly follow secure coding standards:

14.1 Authentication & Authorization

- Secure JWT storage in HttpOnly cookies or secure memory.
- Guard routes based on **role** from auth_user.role.

14.2 Input Validation & Sanitization

- Validate form inputs client-side.
- Sanitize rich-text or HTML before rendering to prevent XSS.

14.3 API Communication

- All requests over HTTPS.
- CSRF protection for non-GET requests when required.

14.4 File Handling

- Validate file size and type before upload.
- Show warning for potentially harmful files flagged by backend.

---

15. Testing Strategy

15.1 Unit Tests

- Jest + React Testing Library for components and hooks.
- Mock API calls for isolated testing.

15.2 Integration Tests

- Test complete flows (e.g., request creation, file upload).
- Use MSW (Mock Service Worker) for API mocking.

15.3 End-to-End Tests

- Cypress for simulating real user interactions.
- Test across roles: admin, accountant, client.

15.4 Performance & Accessibility Audits

- Lighthouse audits for performance, accessibility, SEO.
- axe-core for accessibility compliance.

---

16. Deployment Plan

16.1 Build & CI/CD

- GitHub Actions or GitLab CI for automated builds.
- Run tests on every pull request.
- Deploy via containerized environment or static hosting (Vercel/Netlify) depending on backend integration.

16.2 Environment Management

- .env for environment variables.
- Separate configs for development, staging, and production.

16.3 Monitoring

- Integrate Sentry for error tracking.
- Google Analytics / Plausible for usage tracking.

16.4 Rollback Plan

- Maintain previous build artifacts.
- Enable quick rollback from CI/CD.

End of Frontend Specification Document for Wazeen

```
-- Database Schema for Client-Accountant Collaboration Portal

-- Users Table
CREATE TABLE auth_user (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   email VARCHAR(255) UNIQUE NOT NULL,
   password VARCHAR(255) NOT NULL,
   first_name VARCHAR(100) NOT NULL,
   last_name VARCHAR(100) NOT NULL,
   role VARCHAR(20) CHECK (role IN ('admin', 'accountant', 'client'))
DEFAULT 'client',
   phone_number VARCHAR(20),
   preferred_language VARCHAR(5) CHECK (preferred_language IN ('en',
'ar')) DEFAULT 'en',
   email_verified BOOLEAN DEFAULT false,
```

```sql
   avatar VARCHAR(500),
   is_online BOOLEAN DEFAULT false,
   last_activity TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
   is_active BOOLEAN DEFAULT true,
   is_staff BOOLEAN DEFAULT false,
   is_superuser BOOLEAN DEFAULT false,
   created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
   updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
   CONSTRAINT valid_email CHECK (email ~*
'^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$')
);

-- User Profiles Table
CREATE TABLE user_profiles (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   user_id UUID REFERENCES auth_user(id) ON DELETE CASCADE,
   bio TEXT,
   company VARCHAR(100),
   job_title VARCHAR(100),
   address TEXT,
   city VARCHAR(50),
   country VARCHAR(50),
   timezone VARCHAR(50) DEFAULT 'Asia/Dubai',
   notification_preferences JSONB DEFAULT '{}',
   created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
   updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Email Verification Tokens Table
CREATE TABLE email_verification_tokens (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   user_id UUID REFERENCES auth_user(id) ON DELETE CASCADE,
   token UUID DEFAULT gen_random_uuid(),
   created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
   expires_at TIMESTAMP WITH TIME ZONE NOT NULL,
   used BOOLEAN DEFAULT false
```

```sql
);

-- Password Reset Tokens Table
CREATE TABLE password_reset_tokens (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   user_id UUID REFERENCES auth_user(id) ON DELETE CASCADE,
   token UUID DEFAULT gen_random_uuid(),
   created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
   expires_at TIMESTAMP WITH TIME ZONE NOT NULL,
   used BOOLEAN DEFAULT false
);

-- Service Request Categories Table
CREATE TABLE service_request_categories (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   name VARCHAR(100) NOT NULL,
   name_ar VARCHAR(100),
   description TEXT,
   description_ar TEXT,
   is_active BOOLEAN DEFAULT true,
   created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Service Requests Table
CREATE TABLE service_requests (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   client_id UUID REFERENCES auth_user(id) ON DELETE CASCADE,
   accountant_id UUID REFERENCES auth_user(id) ON DELETE SET NULL,
   title VARCHAR(200) NOT NULL,
   description TEXT NOT NULL,
   category_id UUID REFERENCES service_request_categories(id) ON DELETE
SET NULL,
   status VARCHAR(20) DEFAULT 'new' CHECK (status IN ('new',
'in_progress', 'review', 'completed', 'closed')),
   priority VARCHAR(10) DEFAULT 'medium' CHECK (priority IN ('low',
'medium', 'high', 'urgent')),
   due_date DATE,
   created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
   updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
   started_at TIMESTAMP WITH TIME ZONE,
   completed_at TIMESTAMP WITH TIME ZONE,
   closed_at TIMESTAMP WITH TIME ZONE,
   estimated_hours DECIMAL(5,2),
   actual_hours DECIMAL(5,2),
   tags JSONB DEFAULT '[]',
   custom_fields JSONB DEFAULT '{}'
);

-- Request Assignments Table
CREATE TABLE request_assignments (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   request_id UUID REFERENCES service_requests(id) ON DELETE CASCADE,
   from_accountant_id UUID REFERENCES auth_user(id) ON DELETE SET NULL,
   to_accountant_id UUID REFERENCES auth_user(id) ON DELETE CASCADE,
   assigned_by_id UUID REFERENCES auth_user(id) ON DELETE CASCADE,
   reason TEXT,
   assigned_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Request Notes Table
CREATE TABLE request_notes (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   request_id UUID REFERENCES service_requests(id) ON DELETE CASCADE,
   author_id UUID REFERENCES auth_user(id) ON DELETE CASCADE,
   content TEXT NOT NULL,
   is_internal BOOLEAN DEFAULT false,
   created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
   updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Request Status History Table
CREATE TABLE request_status_history (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   request_id UUID REFERENCES service_requests(id) ON DELETE CASCADE,
   from_status VARCHAR(20),
   to_status VARCHAR(20) NOT NULL,
   changed_by_id UUID REFERENCES auth_user(id) ON DELETE CASCADE,
   reason TEXT,
   changed_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- File Categories Table
CREATE TABLE file_categories (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   name VARCHAR(100) NOT NULL,
   name_ar VARCHAR(100),
   description TEXT,
   description_ar TEXT,
   color VARCHAR(7) DEFAULT '#007bff',
   is_active BOOLEAN DEFAULT true,
   created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Files Table
CREATE TABLE files (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   request_id UUID REFERENCES service_requests(id) ON DELETE CASCADE,
   uploaded_by_id UUID REFERENCES auth_user(id) ON DELETE CASCADE,
   original_filename VARCHAR(255) NOT NULL,
   stored_filename VARCHAR(255) NOT NULL UNIQUE,
   file_path VARCHAR(1000) NOT NULL,
   file_size BIGINT NOT NULL,
   mime_type VARCHAR(100) NOT NULL,
   file_hash VARCHAR(64) NOT NULL,
   category_id UUID REFERENCES file_categories(id) ON DELETE SET NULL,
   folder_path VARCHAR(500) DEFAULT '/',
   tags JSONB DEFAULT '[]',
   version_number INTEGER DEFAULT 1,
   parent_file_id UUID REFERENCES files(id) ON DELETE CASCADE,
   preview_status VARCHAR(20) DEFAULT 'pending' CHECK (preview_status IN
('pending', 'processing', 'ready', 'failed', 'not_supported')),
   preview_path VARCHAR(1000),
   thumbnail_path VARCHAR(1000),
   is_deleted BOOLEAN DEFAULT false,
   is_virus_scanned BOOLEAN DEFAULT false,
   virus_scan_result VARCHAR(20),
   metadata JSONB DEFAULT '{}',
   created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
   updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- File Downloads Table
CREATE TABLE file_downloads (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   file_id UUID REFERENCES files(id) ON DELETE CASCADE,
   downloaded_by_id UUID REFERENCES auth_user(id) ON DELETE CASCADE,
   ip_address INET,
   user_agent TEXT,
   download_token VARCHAR(255),
   downloaded_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- File Shares Table
CREATE TABLE file_shares (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   file_id UUID REFERENCES files(id) ON DELETE CASCADE,
   shared_by_id UUID REFERENCES auth_user(id) ON DELETE CASCADE,
   shared_with_id UUID REFERENCES auth_user(id) ON DELETE CASCADE,
   share_token UUID DEFAULT gen_random_uuid(),
   can_download BOOLEAN DEFAULT true,
   can_view_preview BOOLEAN DEFAULT true,
   expires_at TIMESTAMP WITH TIME ZONE,
   max_downloads INTEGER,
   download_count INTEGER DEFAULT 0,
   is_active BOOLEAN DEFAULT true,
   created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Chat Rooms Table
CREATE TABLE chat_rooms (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   request_id UUID REFERENCES service_requests(id) ON DELETE CASCADE,
   is_active BOOLEAN DEFAULT true,
   allow_file_sharing BOOLEAN DEFAULT true,
   max_file_size INTEGER DEFAULT 52428800, -- 50MB
   created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
   updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Chat Messages Table
CREATE TABLE chat_messages (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   room_id UUID REFERENCES chat_rooms(id) ON DELETE CASCADE,
   sender_id UUID REFERENCES auth_user(id) ON DELETE CASCADE,
   message_type VARCHAR(20) DEFAULT 'text' CHECK (message_type IN ('text',
'file', 'system', 'image', 'document')),
   content TEXT NOT NULL,
   file_id UUID REFERENCES files(id) ON DELETE SET NULL,
   is_read BOOLEAN DEFAULT false,
   is_deleted BOOLEAN DEFAULT false,
   is_edited BOOLEAN DEFAULT false,
   metadata JSONB DEFAULT '{}',
   created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
   updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
   edited_at TIMESTAMP WITH TIME ZONE
);

-- Message Reactions Table
CREATE TABLE message_reactions (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   message_id UUID REFERENCES chat_messages(id) ON DELETE CASCADE,
   user_id UUID REFERENCES auth_user(id) ON DELETE CASCADE,
   emoji VARCHAR(10) CHECK (emoji IN ('👍', '👎', '❤️', '😂', '😮', '😢',
'😡', '✅', '❌')) NOT NULL,
   created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
   CONSTRAINT unique_reaction UNIQUE (message_id, user_id, emoji)
);

-- Chat Participants Table
CREATE TABLE chat_participants (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   room_id UUID REFERENCES chat_rooms(id) ON DELETE CASCADE,
   user_id UUID REFERENCES auth_user(id) ON DELETE CASCADE,
   is_active BOOLEAN DEFAULT true,
   is_muted BOOLEAN DEFAULT false,
   last_read_message_id UUID REFERENCES chat_messages(id) ON DELETE SET
NULL,
   last_seen TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
   joined_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
   CONSTRAINT unique_participant UNIQUE (room_id, user_id)
);

-- Typing Indicators Table
CREATE TABLE typing_indicators (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   room_id UUID REFERENCES chat_rooms(id) ON DELETE CASCADE,
   user_id UUID REFERENCES auth_user(id) ON DELETE CASCADE,
   is_typing BOOLEAN DEFAULT true,
   created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
   updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
   CONSTRAINT unique_typing UNIQUE (room_id, user_id)
);

-- Message Threads Table
CREATE TABLE message_threads (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   parent_message_id UUID REFERENCES chat_messages(id) ON DELETE CASCADE,
   reply_message_id UUID REFERENCES chat_messages(id) ON DELETE CASCADE,
   created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
   CONSTRAINT unique_thread UNIQUE (parent_message_id, reply_message_id)
);

-- Chat Settings Table
CREATE TABLE chat_settings (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   user_id UUID REFERENCES auth_user(id) ON DELETE CASCADE,
   email_notifications BOOLEAN DEFAULT true,
   push_notifications BOOLEAN DEFAULT true,
   desktop_notifications BOOLEAN DEFAULT true,
   sound_notifications BOOLEAN DEFAULT true,
   show_typing_indicators BOOLEAN DEFAULT true,
   show_read_receipts BOOLEAN DEFAULT true,
   auto_download_files BOOLEAN DEFAULT false,
   theme VARCHAR(20) CHECK (theme IN ('light', 'dark')) DEFAULT 'light',
   allow_direct_messages BOOLEAN DEFAULT true,
   created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
   updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
   CONSTRAINT unique_user_settings UNIQUE (user_id)
);

-- Indexes for Performance
CREATE INDEX idx_auth_user_email ON auth_user(email);
CREATE INDEX idx_auth_user_role ON auth_user(role);
CREATE INDEX idx_auth_user_is_active ON auth_user(is_active);
CREATE INDEX idx_user_profiles_user_id ON user_profiles(user_id);
CREATE INDEX idx_service_requests_client_id ON
service_requests(client_id);
CREATE INDEX idx_service_requests_accountant_id ON
service_requests(accountant_id);
CREATE INDEX idx_service_requests_status ON service_requests(status);
CREATE INDEX idx_service_requests_priority ON service_requests(priority);
```

```sql
CREATE INDEX idx_service_requests_created_at ON
service_requests(created_at DESC);
CREATE INDEX idx_service_requests_due_date ON service_requests(due_date);
CREATE INDEX idx_request_assignments_request_id ON
request_assignments(request_id);
CREATE INDEX idx_request_notes_request_id ON request_notes(request_id);
CREATE INDEX idx_request_status_history_request_id ON
request_status_history(request_id);
CREATE INDEX idx_files_request_id ON files(request_id);
CREATE INDEX idx_files_uploaded_by_id ON files(uploaded_by_id);
CREATE INDEX idx_files_file_hash ON files(file_hash);
CREATE INDEX idx_files_is_deleted ON files(is_deleted);
CREATE INDEX idx_file_downloads_file_id ON file_downloads(file_id);
CREATE INDEX idx_file_shares_share_token ON file_shares(share_token);
CREATE INDEX idx_chat_rooms_request_id ON chat_rooms(request_id);
CREATE INDEX idx_chat_messages_room_id_created_at ON
chat_messages(room_id, created_at);
CREATE INDEX idx_chat_messages_sender_id ON chat_messages(sender_id);
CREATE INDEX idx_chat_messages_is_read ON chat_messages(room_id, is_read);
CREATE INDEX idx_message_reactions_message_id ON
message_reactions(message_id);
CREATE INDEX idx_chat_participants_room_id_is_active ON
chat_participants(room_id, is_active);
CREATE INDEX idx_typing_indicators_room_id_is_typing ON
typing_indicators(room_id, is_typing);
CREATE INDEX idx_message_threads_parent_message_id ON
message_threads(parent_message_id);

-- Full-Text Search Indexes
CREATE INDEX idx_service_requests_search ON service_requests
USING gin(to_tsvector('english', title || ' ' || description));
CREATE INDEX idx_chat_messages_search ON chat_messages
USING gin(to_tsvector('english', content));
```

Authentication APIs (/api/authentication/):

- POST /api/authentication/login/: Authenticate user with email and password, returns JWT tokens.
- POST /api/authentication/refresh/: Refresh JWT access token using refresh token.
- POST /api/authentication/register/: Register a new user (email, first_name, last_name, role, etc.).
- GET/PUT /api/authentication/profile/: Get or update user profile (User and UserProfile fields).
- GET /api/authentication/profile/detail/: Get detailed user profile information.
- GET /api/authentication/status/: Get user status (is_online, last_activity).
- POST /api/authentication/password/change/: Change user password.
- POST /api/authentication/password/reset/: Request password reset (sends email with token).
- POST /api/authentication/password/reset/confirm/: Confirm password reset using token.
- POST /api/authentication/email/verify/: Verify email using token.
- POST /api/authentication/email/verify/resend/: Resend email verification link.

Service Request APIs (/api/service-requests/):

- GET /api/service-requests/categories/: List service request categories (ServiceRequestCategory).
- GET/POST /api/service-requests/: List or create service requests (ServiceRequest).
- GET/PUT/DELETE /api/service-requests/<uuid:pk>/: Get, update, or delete a specific service request.
- POST /api/service-requests/<uuid:request_id>/assign/: Assign an accountant to a request (RequestAssignment).
- POST /api/service-requests/<uuid:request_id>/status/: Update request status (RequestStatusHistory).
- GET/POST /api/service-requests/<uuid:request_id>/notes/: List or create notes for a request (RequestNote).
- GET /api/service-requests/dashboard/stats/: Get dashboard statistics (e.g., request counts by status).

File Management APIs (/api/file-management/):

- GET /api/file-management/categories/: List file categories (FileCategory).
- GET /api/file-management/requests/<uuid:request_id>/files/: List files for a specific request (File).
- POST /api/file-management/requests/<uuid:request_id>/files/upload/: Upload a file to a request.
- GET/PUT/DELETE /api/file-management/files/<uuid:pk>/: Get, update, or delete a specific file.
- GET /api/file-management/files/<uuid:file_id>/download/: Download a file (returns signed URL).
- GET /api/file-management/files/<uuid:file_id>/preview/: Get file preview if available.
- GET /api/file-management/files/<uuid:file_id>/thumbnail/: Get file thumbnail.

- POST /api/file-management/files/<uuid:file_id>/share/: Create a file share (FileShare).
- GET /api/file-management/shares/: List file shares.
- GET /api/file-management/share/<uuid:share_token>/: Access a shared file using share token.

Chat APIs (/api/chat/):

- GET /api/chat/rooms/: List chat rooms (ChatRoom) accessible to the user.
- GET /api/chat/rooms/<uuid:room_id>/: Get details of a specific chat room.
- GET /api/chat/rooms/<uuid:room_id>/messages/: List messages in a room (paginated).
- POST /api/chat/rooms/<uuid:room_id>/messages/: Create a new message (ChatMessage).
- GET /api/chat/rooms/<uuid:room_id>/messages/<uuid:message_id>/: Get message details.
- PATCH /api/chat/rooms/<uuid:room_id>/messages/<uuid:message_id>/: Update a message (if editable).
- DELETE /api/chat/rooms/<uuid:room_id>/messages/<uuid:message_id>/: Delete a message (if allowed).
- POST /api/chat/rooms/<uuid:room_id>/messages/<uuid:message_id>/react/: Add a reaction (MessageReaction).
- DELETE /api/chat/rooms/<uuid:room_id>/messages/<uuid:message_id>/react/<emoji>/: Remove a reaction.
- POST /api/chat/rooms/<uuid:room_id>/messages/mark_as_read/: Mark messages as read (ChatParticipant).
- POST /api/chat/rooms/<uuid:room_id>/typing/: Set typing indicator (TypingIndicator).
- GET /api/chat/settings/: Get user chat settings (ChatSettings).
- PATCH /api/chat/settings/: Update chat settings.
- GET /api/chat/rooms/<uuid:room_id>/stats/: Get chat statistics (e.g., message count, participants).
- GET /api/chat/rooms/<uuid:room_id>/search/?q=query: Search messages in a room.
- POST /api/chat/rooms/<uuid:room_id>/export/: Export chat history.
- DELETE /api/chat/rooms/<uuid:room_id>/clear/: Clear chat (admin only).
- GET /api/chat/rooms/<uuid:room_id>/messages/<uuid:message_id>/thread/: Get threaded replies (MessageThread).

WebSocket Endpoints:

- ws://domain/ws/chat/<uuid:room_id>/: Connect to a chat room for real-time messaging, typing indicators, and reactions.
- ws://domain/ws/notifications/: Receive system notifications (e.g., new messages, request updates).