# Lex Assist Backend Architecture Documentation

## Overview

This document provides a comprehensive overview of the Lex Assist backend architecture, detailing the implementation of role-based access control, Supabase integration, and AI components including InLegalBERT integration. The architecture is designed to support a scalable, secure, and feature-rich legal assistance platform for Indian lawyers.

## System Architecture

### High-Level Architecture

The Lex Assist backend follows a modular architecture with the following key components:

1. **Authentication & Authorization Layer**
   * User authentication via email/password and OTP
   * Role-based access control (Super Admin, Admin, User)
   * JWT token management with role and subscription claims
2. **Database Layer**
   * Supabase PostgreSQL database
   * Row-level security policies
   * Subscription and user management tables
3. **API Layer**
   * RESTful API endpoints
   * Request validation and error handling
   * Rate limiting and security measures
4. **AI Processing Layer**
   * Model Context Protocol for standardized AI interactions
   * InLegalBERT integration for Indian legal text processing
   * Response parsing and formatting
5. **Document Generation Layer**
   * Case file drafting
   * Export functionality (PDF, DOCX, RTF)
   * Sharing capabilities

### Component Diagram

┌─────────────────────────────────────────────────────────────────┐

│ Frontend Application │

└───────────────────────────────┬─────────────────────────────────┘

│

▼

┌─────────────────────────────────────────────────────────────────┐

│ API Gateway │

└───────────────┬─────────────────┬────────────────┬──────────────┘

│ │ │

▼ ▼ ▼

┌───────────────────┐ ┌────────────────┐ ┌──────────────────┐

│ Authentication │ │ Legal Brief │ │ Subscription │

│ Service │ │ Analysis │ │ Service │

└─────────┬─────────┘ └────────┬───────┘ └────────┬─────────┘

│ │ │

▼ ▼ ▼

┌─────────────────────────────────────────────────────────────────┐

│ Supabase Database │

└─────────────────────────────────────────────────────────────────┘

│

▼

┌─────────────────────────────────────────────────────────────────┐

│ Model Context Protocol │

└───────────────────────────────┬─────────────────────────────────┘

│

▼

┌─────────────────────┐ ┌─────────────────┐ ┌───────────────────┐

│ InLegalBERT │ │ Indian Legal │ │ Document │

│ Processor │ │ Databases │ │ Generator │

└─────────────────────┘ └─────────────────┘ └───────────────────┘

## Core Components

### 1. Authentication & Authorization

#### Implementation Files

* /backend/api/auth\_endpoints.py: API endpoints for authentication
* /backend/api/role\_based\_access\_control.py: Role-based access control middleware

#### Key Features

* JWT-based authentication with custom claims for roles and subscription tiers
* Email/password and OTP authentication methods
* Role-based access control with three distinct roles
* Subscription tier-based feature access

#### Authentication Flow

1. User registers or logs in via email/password or OTP
2. Backend validates credentials and generates JWT with role and subscription claims
3. Frontend stores JWT and includes it in subsequent API requests
4. Backend validates JWT and enforces access control based on claims

### 2. Database Schema

#### Implementation Files

* /legal\_app/supabase\_schema\_design.md: Detailed database schema design

#### Key Tables

* users: User accounts with role information
* profiles: Extended user profile information
* subscription\_tiers: Available subscription plans
* subscriptions: User subscription records
* case\_briefs: User-submitted case briefs
* analysis\_results: AI-generated analysis results
* case\_files: Generated case file documents

#### Security Features

* Row-level security policies for all tables
* Role-based access restrictions at the database level
* Encrypted sensitive data

### 3. Model Context Protocol

#### Implementation Files

* /backend/api/model\_context\_protocol.py: Protocol definition and implementation
* /backend/api/inlegalbert\_processor.py: InLegalBERT model integration

#### Key Components

* LegalModelInterface: Abstract interface for all legal AI models
* ModelRequest and ModelResponse: Standardized data structures
* PromptTemplate: Task-specific prompt engineering
* ResponseParser: Structured parsing of model outputs
* ModelContextManager: Context management for model interactions

#### Supported Tasks

* Statute identification
* Case history retrieval
* Legal analysis
* Judgment prediction
* Case file drafting

### 4. InLegalBERT Integration

#### Implementation Files

* /backend/api/inlegalbert\_processor.py: InLegalBERT model implementation

#### Key Features

* Integration with law-ai/InLegalBERT model
* Specialized for Indian legal text processing
* Embedding generation for semantic search
* Task-specific processing pipelines

#### Model Capabilities

* Trained on 5.4 million Indian legal documents
* 110M parameters (bert-base-uncased configuration)
* Optimized for Indian legal terminology and concepts
* Supports multiple legal NLP tasks

### 5. API Endpoints

#### Implementation Files

* /backend/api/auth\_endpoints.py: Authentication endpoints
* /backend/app.py: Main API application
* /backend/api/legal\_brief\_analyzer.py: Brief analysis endpoints
* /backend/api/case\_file\_drafter.py: Case file drafting endpoints
* /backend/api/document\_generator.py: Document generation endpoints

#### Key Endpoints

* /auth/\*: Authentication and user management
* /users/\*: User profile and management
* /subscriptions/\*: Subscription management
* /briefs/\*: Case brief submission and analysis
* /export/\*: Document export and sharing

#### Security Measures

* Input validation for all endpoints
* Rate limiting to prevent abuse
* CORS configuration for frontend access
* Error handling and logging

### 6. Frontend Integration

#### Implementation Files

* /legal\_app\_frontend/src/api/LexAssistApiClient.ts: API client for frontend

#### Key Features

* Secure API communication
* Token management and refresh
* Role and subscription-based UI adaptation
* Error handling and user feedback

## Data Flow

### Case Brief Analysis Flow

1. User submits case brief through frontend
2. Frontend sends brief to /briefs/analyze endpoint
3. Backend validates user authentication and subscription tier
4. Brief is processed through Model Context Protocol:
   * Statute identification task extracts relevant law sections
   * Case history task retrieves relevant precedents
   * Legal analysis task generates structured analysis
5. Results are stored in database and returned to frontend
6. Frontend displays results in structured tabs/cards

### User Authentication Flow

1. User enters credentials on frontend login page
2. Frontend sends credentials to /auth/login endpoint
3. Backend validates credentials against Supabase Auth
4. JWT token with role and subscription claims is generated
5. Token is returned to frontend and stored
6. Frontend uses token for subsequent API requests

### Subscription Management Flow

1. Super Admin configures subscription tiers
2. Users select and purchase subscription plans
3. Payment is processed and subscription record is created
4. User's JWT claims are updated with new subscription tier
5. User gains access to tier-specific features

## Security Considerations

### Authentication Security

* Passwords stored using bcrypt hashing
* JWT tokens with appropriate expiry
* Refresh token rotation
* Two-factor authentication via OTP

### Data Security

* HTTPS for all communications
* Database encryption for sensitive data
* Row-level security in Supabase
* Input validation and sanitization

### Access Control

* Role-based permissions at API level
* Subscription-based feature access
* Super Admin actions require additional verification

## Deployment Architecture

### Production Environment

* Frontend: Netlify static hosting
* Backend API: Supabase Edge Functions
* Database: Supabase PostgreSQL
* AI Models: Dedicated compute instances or serverless functions

### Scaling Considerations

* Horizontal scaling for API servers
* Database read replicas for high traffic
* Caching for frequent queries
* Batch processing for heavy AI workloads

## Development and Testing

### Development Environment

* Local development with Docker containers
* Supabase local development setup
* Mock AI responses for faster iteration

### Testing Strategy

* Unit tests for individual components
* Integration tests for API endpoints
* End-to-end tests for complete flows
* Role-based access testing
* Performance testing for AI components

## Future Enhancements

### Planned Improvements

* Additional language support beyond English
* Integration with more Indian legal databases
* Advanced analytics for legal professionals
* Collaborative features for law firms
* Mobile application development

### Scalability Enhancements

* Distributed AI processing for faster analysis
* Enhanced caching strategies
* Microservices architecture for independent scaling

## Conclusion

The Lex Assist backend architecture provides a robust foundation for a legal assistance platform tailored to Indian lawyers. The modular design allows for easy maintenance and future enhancements, while the integration of InLegalBERT ensures high-quality legal analysis specific to Indian law. The role-based access control and subscription tier system enable a sustainable business model with feature differentiation based on user needs.