EPLQ – Detailed Project Documentation

**1. Introduction**

1.1. Purpose  
Describe the goals of EPLQ: a privacy-preserving, encrypted location-based query system for secure spatial searches.

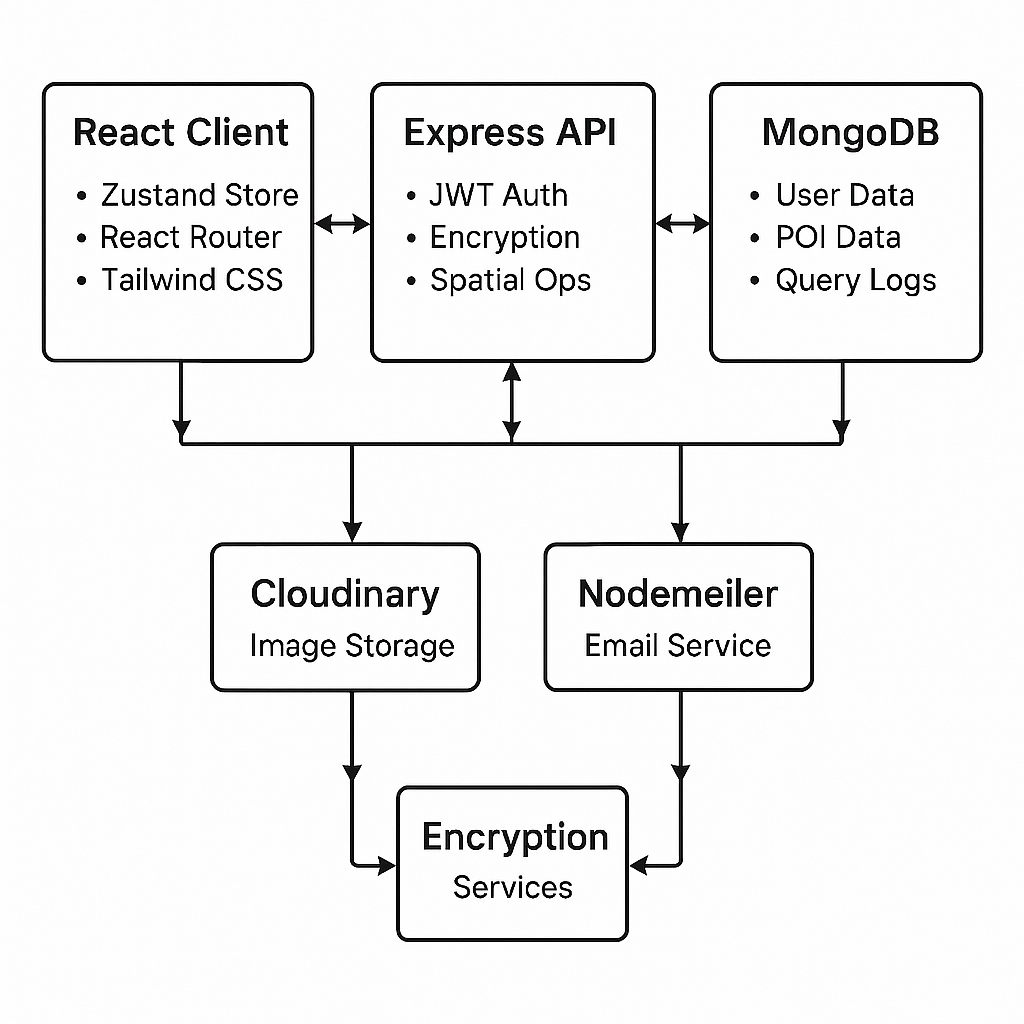
1.2. Scope  
Covers system architecture, features, user/admin flows, API specs, security models, deployment, and future roadmap.

1.3. Audience  
Developers, architects, system administrators, stakeholders.

**2. System Overview**

2.1. Architecture Diagram

* React client ↔ Express API ↔ MongoDB
* Auxiliary services: Cloudinary, Mailtrap, Encryption Utils



2.2. Key Components

* Frontend (React, Zustand, Tailwind CSS)
* Backend (Node.js, Express, Mongoose, JWT)
* Database (MongoDB, 2dsphere indexes)
* Utils (Encryption, Token, Email)

**3. Features**

3.1. User Features

* Signup, email verification, login/logout
* Profile management with Cloudinary image upload
* Search POIs by location & radius
* Decrypt POI data on demand
* View personal query history

3.2. Admin Features

* Dashboard with stats (users, POIs, logs)
* CRUD for POIs (create, list, manage)
* View & delete all query logs
* View/manage all users

**4. Data Models & Encryption**

4.1. User Model  
Fields: fullname, email, password (bcrypt), contact, address, city, country, profilePicture, admin, lastLogin, isVerified, tokens

4.2. POI Model  
Fields: title, description, location {Point, coordinates}, encryptedData (AES-256), createdBy, isActive

4.3. QueryLog Model  
Fields: user (ref), queryType, queryParams, resultCount, timestamp

4.4. Encryption Flow

* encryptPOIData(plainData) → Base64/AES-256
* Store encryptedData
* Decrypt with decryptPOIUtility on demand

**5. API Documentation**

5.1. Authentication

* POST /user/signup
* POST /user/login
* POST /user/verify-email
* GET /user/check-auth
* POST /user/forgot-password
* POST /user/forgot-password/:token
* PUT /user/profile/update

5.2. POI Endpoints

* POST /poi/admin/poi
* GET /poi/admin/pois
* POST /poi/user/pois/search
* GET /poi/user/poi/:id/decrypt

5.3. Query Logs

* GET /querylog/all
* GET /querylog/me
* DELETE /querylog/:id

Example Request & Response  
See section 5.4 in README for cURL/Postman templates.

**6. Frontend Components**

6.1. Admin

* AdminDashboard, AdminData, AdminLogs, ManagePOI, UploadData

6.2. User

* UserDashboard, HeroSection, SearchPOI, DecryptPOI, History, Profile, Navbar

6.3. State Management

* useUserStore, usePOIStore, useQueryLogStore

**7. Security & Privacy**

7.1. Authentication & Authorization

* JWT tokens, protect and admin middleware

7.2. Encryption

* AES-256 or Base64 stub for POC
* Data encrypted in transit (HTTPS) and at rest

7.3. Best Practices

* Bcrypt for passwords
* Input validation & sanitization
* CORS, rate limiting, security headers

**8. Deployment & DevOps**

8.1. Local Development

* npm run dev for backend & frontend
* .env configurations for both

8.2. Docker

* docker-compose up --build

8.3. Production

* Build React (npm run build)
* Start Node with pm2 or Docker

**9. Testing**

9.1. Backend Tests (npm test)  
9.2. Frontend Tests (npm test)  
9.3. Postman Collection (docs/postman/EPLQ.postman\_collection.json)

**10. Roadmap**

* Real-time notifications
* Advanced analytics & ML integration
* Mobile app (React Native)
* Enhanced encryption algorithms
* Multi-tenant support

**11. Contribution Guidelines**

* Fork → Branch → Commit → PR
* ESLint & Prettier rules applied
* Conventional commits

**12. License**

MIT License