**EPLQ – Detailed Project Documentation**

**1. Introduction**

**1.1 Purpose**

EPLQ (Encrypted Privacy-preserving Location Query) is a secure, location-based query system that enables users to search for Points of Interest (POIs) without exposing sensitive spatial data to unauthorized parties.  
The core goals are:

* Enable location-based searches (e.g., “Find all POIs within 5 km of my location”).
* Encrypt all sensitive POI data at rest and decrypt only when necessary.
* Provide a full-stack solution with separate user and admin roles.
* Ensure privacy, compliance, and scalability for geospatial applications.

**1.2 Scope**

This documentation covers:

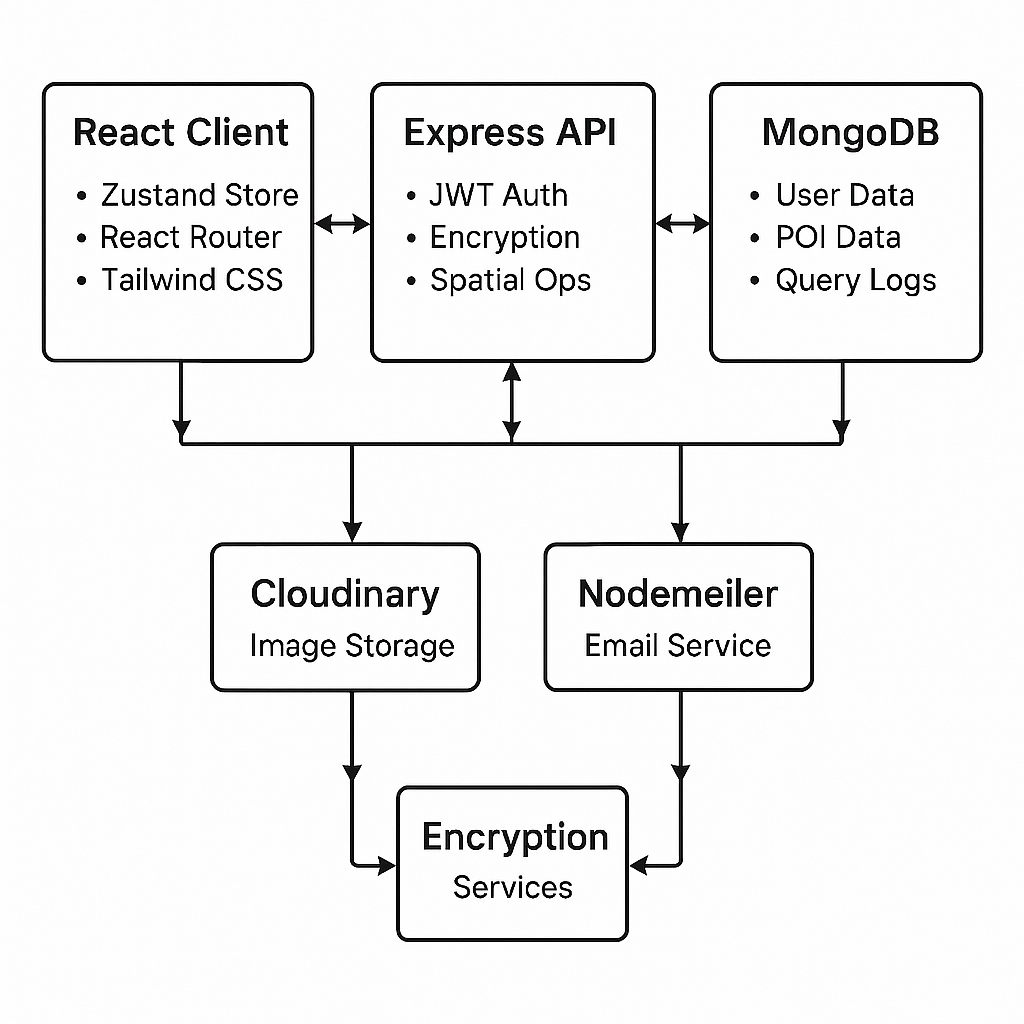
* **System architecture** and interaction flow.
* **Features** for both users and admins.
* **Detailed data models** and encryption flows.
* **API specifications** for frontend-backend communication.
* **Frontend components** for both admin and user interfaces.
* **Security model** including encryption, authentication, and authorization.
* **Deployment instructions** for local, Docker, and production environments.
* **Testing strategy** and future roadmap.

**1.3 Audience**

* **Developers** – Backend, frontend, and full-stack engineers implementing or extending the project.
* **System architects** – Reviewing the security, performance, and architecture decisions.
* **Administrators** – Deploying and managing the platform.
* **Stakeholders** – Understanding the platform capabilities and roadmap.

**2. System Overview**

**2.1 Architecture Diagram**



**Flow:**

1. **Frontend (React)** sends authenticated API requests.
2. **Backend (Express)** validates and processes requests.
3. **MongoDB** stores POIs, users, and query logs (with encryption for sensitive fields).
4. **Cloudinary** handles image uploads (profile pictures, POI images).
5. **Mailtrap** handles development email testing.
6. **Encryption Utils** ensure AES-256 encryption/decryption for POI details.

**2.2 Key Components**

* **Frontend:**  
  React + Zustand + Tailwind CSS for a responsive, state-managed UI.
* **Backend:**  
  Node.js + Express + Mongoose for API services.
* **Database:**  
  MongoDB with 2dsphere indexes for efficient geospatial queries.
* **Utils:**
  + Encryption/Decryption service (AES-256).
  + JWT token generation/verification.
  + Email utilities (nodemailer + Mailtrap).

**3. Features**

**3.1 User Features**

* Signup, login, logout with JWT.
* Email verification after registration.
* Profile management (edit details, change profile picture via Cloudinary).
* Search POIs by location & radius.
* On-demand decryption of encrypted POI details.
* Personal query history view.

**3.2 Admin Features**

* Dashboard with overall statistics (users, POIs, query logs).
* Full POI CRUD operations (create, list, edit, activate/deactivate).
* View and delete query logs (global).
* Manage all users (activation, deactivation, deletion).

**4. Data Models & Encryption**

**4.1 User Model**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| fullname | String | User’s full name |
| email | String | Unique, validated |
| password | String (bcrypt hash) | Hashed for security |
| contact, address, city, country | String | Profile details |
| profilePicture | String | Cloudinary URL |
| admin | Boolean | Role flag |
| lastLogin | Date | Last login timestamp |
| isVerified | Boolean | Email verification flag |
| tokens | Array | Active JWT refresh tokens |

**4.2 POI Model**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| title | String | POI name |
| description | String | Brief description |
| location | GeoJSON Point | { type: 'Point', coordinates: [lng, lat] } |
| encryptedData | String | AES-256 encrypted content |
| createdBy | ObjectId (ref: User) | Creator |
| isActive | Boolean | Active status flag |

**4.3 QueryLog Model**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| user | ObjectId (ref: User) | Query initiator |
| queryType | String | e.g., "radius-search" |
| queryParams | Object | Original search parameters |
| resultCount | Number | Matches found |
| timestamp | Date | Time of query |

**4.4 Encryption Flow**

1. encryptPOIData(plainData) → AES-256 → Base64 encode.
2. Store in encryptedData field in MongoDB.
3. When user/admin requests details:
   * Verify authorization.
   * Call decryptPOIUtility with stored Base64.
   * Return plain text to frontend.

**5. API Documentation**

**5.1 Authentication**

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| POST | /user/signup | Create new account |
| POST | /user/login | Authenticate user |
| POST | /user/verify-email | Verify new account |
| GET | /user/check-auth | Validate current token |
| POST | /user/forgot-password | Request reset |
| POST | /user/forgot-password/:token | Reset password |
| PUT | /user/profile/update | Update user profile |

**5.2 POI Endpoints**

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| POST | /poi/admin/poi | Create POI |
| GET | /poi/admin/pois | List POIs |
| POST | /poi/user/pois/search | Search POIs by location |
| GET | /poi/user/poi/:id/decrypt | Decrypt POI details |

**5.3 Query Logs**

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| GET | /querylog/all | View all logs (admin) |
| GET | /querylog/me | View own logs (user) |
| DELETE | /querylog/:id | Delete log |

**6. Frontend Components**

**6.1 Admin**

* AdminDashboard – Overview metrics.
* AdminData – Detailed tables of users & POIs.
* AdminLogs – View and delete query logs.
* ManagePOI – Create/edit/delete POIs.
* UploadData – Import bulk POIs.

**6.2 User**

* UserDashboard – Main landing for logged-in users.
* HeroSection – Homepage banner/search entry.
* SearchPOI – Map and results list.
* DecryptPOI – View decrypted details.
* History – Personal query log.
* Profile – Update profile info.
* Navbar – Navigation header.

**7. Security & Privacy**

**7.1 Authentication & Authorization**

* JWT for authentication.
* protect middleware for private routes.
* admin middleware for admin-only actions.

**7.2 Encryption**

* AES-256 encryption for sensitive POI data.
* HTTPS for all data in transit.

**7.3 Best Practices**

* Bcrypt for passwords (10+ rounds).
* Input validation & sanitization.
* CORS configuration.
* Rate limiting & security headers.

**8. Deployment & DevOps**

**8.1 Local Development**

bash

CopyEdit

npm run dev # start frontend & backend

* Use .env for secrets (Mongo URI, JWT secret, Cloudinary keys).

**8.2 Docker**

bash

CopyEdit

docker-compose up --build

* Includes backend, frontend, and MongoDB containers.

**8.3 Production**

* Build React: npm run build
* Serve backend with pm2 or in Docker container.
* Ensure SSL termination (NGINX or reverse proxy).

**9. Testing**

* Backend unit & integration tests via Jest.
* Frontend tests via React Testing Library.
* Postman collection in docs/postman/EPLQ.postman\_collection.json.

**10. Roadmap**

* Real-time notifications.
* ML-powered location recommendations.
* Mobile app via React Native.
* Stronger encryption algorithms.
* Multi-tenant support.

**11. Contribution Guidelines**

* Fork → Branch → Commit → Pull Request.
* ESLint + Prettier enforced.
* Follow Conventional Commits.

**12. License**

MIT License – free to use, modify, and distribute.