# **CBL Documentation — Usof Backend (Q&A API)**

## **Short project summary**

Usof is a backend API for a Q&A web/mobile application. It implements user registration and authentication, CRUD for questions and answers, likes, basic admin panel actions and file uploads. Built with Node.js, Express and MySQL using MVC and OOP principles.

## **CBL stages — progress report**

### **1. Engage**

**Goal:** build a minimal, secure, and extendable API for a Q&A service.

**Activities:**

* Defined essential features: users, questions, answers, likes, admin role, file upload.
* Collected references: REST principles, Node.js/Express best practices, SQL design guides.
* Planned stack: Node.js, Express, MySQL, bcrypt, jsonwebtoken, multer.

**Problems found:**

* Scope could grow quickly (many optional features).
* Unsure about exact database relationships (likes, tags, votes).

**Decisions / Solutions:**

* Set minimum scope for the challenge (required features only).
* Designed simplified DB schema (users, questions, answers, likes) to be extended later.

**Evidence:** short project plan

### **2. Investigate**

**Goal:** choose specific designs, libraries, and API contracts before coding.

**Activities:**

* Designed database tables and keys.
* Chose authentication model: JWT in httpOnly cookie + refresh tokens.
* Prepared API routes and request/response shape (Postman collection).
* Reviewed Node.js and SQL security best practices.

**Problems found:**

* Handling relational integrity (deleting users or cascading deletes).
* Race conditions on likes and rating counters.

**Decisions / Solutions:**

* Use foreign keys with ON DELETE CASCADE where appropriate; soft-delete for posts if needed.
* Use transactions for counter updates or read-modify-write with optimistic checks.

**Evidence:** docs/Postman\_collection.json, src/backend/sql/init.sql and src/backend/models/README.md.

### **3. Act (implementation)**

**Goal:** implement functioning API and basic admin panel endpoints.

**Completed features:**

* User registration and login with password hashing (bcrypt).
* JWT-based authentication and role check middleware (admin role).
* CRUD for questions and answers with fields validation.
* Like/unlike endpoints with safe counter update.
* File upload for avatars using multer (stored in src/backend/upload).
* Database initialization scripts that create the schema and seed test data.

**Problems during implementation:**

* Asynchronous errors and unhandled promise rejections.
* Proper error messages and status codes for client errors.

**Solutions applied:**

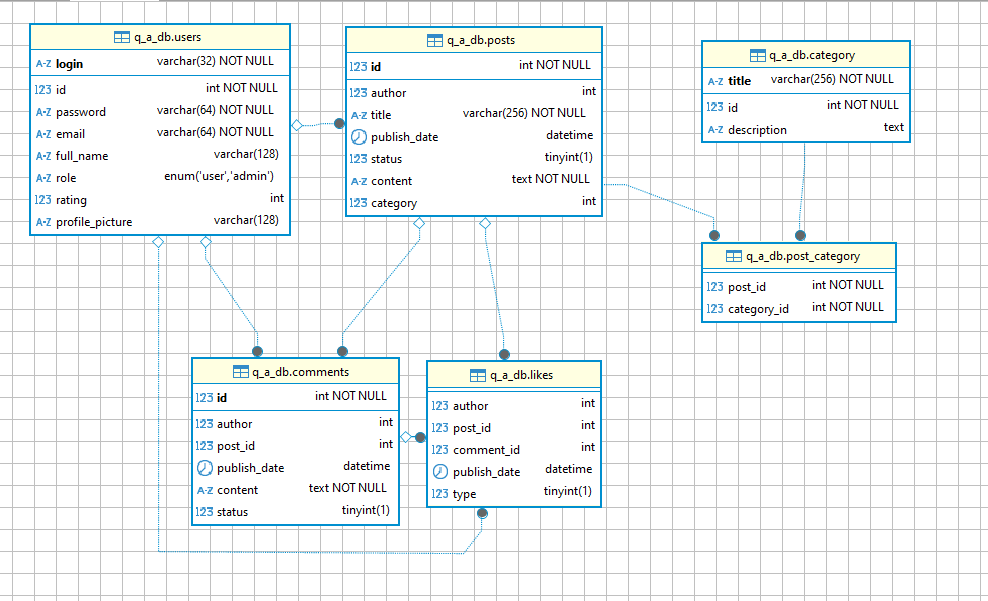
* Centralized error middleware and consistent response format { success: boolean, data, error }.
* Input validation (server-side checks) and explicit HTTP status codes.

**Evidence:** working Postman examples, testData.sql seeds, and unit/manual test notes in docs/tests.md.

## **Application algorithm (high-level)**

1. Client sends HTTP request.
2. Router matches path and method.
3. Authentication middleware checks JWT and sets req.user.
4. Controller validates input and calls model methods.
5. Model performs SQL queries using mysq and returns results.
6. Controller formats the response and returns JSON.
7. Error middleware logs and returns a consistent error response.

## **Database schema (summary)**



Seed: at least 5 rows per table exist in src/backend/sql/testData.sql.

## **API endpoints**

* *router*.use("/auth", *authRoutes*); registration and authorization
* *router*.use("/users", *usersRoutes*); users CRUD
* *router*.use("/upload", express.static(path.join(\_\_dirname,
* "src/backend/upload"))); download avatars
* *router*.use("/categories", *categoriesRoutes*); categories CRUD
* *router*.use("/posts", *postsRoutes*); posts CRUD
* *router*.use("/comments", *commentsRoutes*); comments CRUD

## **Testing and validation**

* Manual testing with Postman collection /Postman\_collection.json.
* SQL seeds used to verify business flows.

## **Reflection and lessons learned**

* Writing clear API contracts first speeds up development.
* Deciding on DB relations early avoids refactoring.
* Centralized error handling and validation improve client experience.

## **Next steps (Act: Creative ideas)**

* Add tags and full-text search for questions.
* Implement soft-delete + admin restore.
* Add automated tests (API, unit).
* Add pagination, sorting, and rate limiting.

## **Where to find artifacts in this repo**

* README.md — short project summary and quick start
* CBL\_Documentation — this file
* docs/Q&A.postman\_collection.json — detailed API list and examples for Postman
* src/backend/sql — init.sql, source.sql, testData.sql
* src/backend/models, src/backend/controllers, src/backend/routers