Application Architecture & Flow — User Feedback System

Overview

The User Feedback System is a full-stack web application designed to collect, store, and display user feedback.
It provides a simple interface for users to submit feedback, and a dashboard to view all feedback with search and sorting capabilities.

System Architecture

Tech Stack:

- Frontend: React
- Backend: Node.js + Express
- Database: MongoDB using Mongoose ORM
- Data Transfer: RESTful API (JSON format)

Architecture Diagram (Text Representation):

Application Flow

1. Frontend (src/app.js)

- Built using React
- Contains:
  - Feedback form: username, email, feedbackText
  - Dashboard to list all submitted feedback
  - Features:
    - Search by name or email
    - Sort by timestamp

2. Backend (src/backend/)

- Built with Express
- Manages:
  - API routing (/feedback)
  - Database operations
- Key files:
  - routes/feedback.js – defines POST and GET endpoints
  - models/Feedback.js – Mongoose schema for feedback data
  - server.js – main entry point for backend

3. Database (MongoDB)

- Hosted on MongoDB Atlas or local
- Stores feedback with fields:
  - username: String
  - email: String
  - feedbackText: String
  - timestamp: Date (auto-generated)

API Design

Method	Endpoint	Purpose
POST	/feedback	Submit new feedback
GET	/feedback	Retrieve all feedbacks

Project Structure (Simplified)

```
user-feedback-system/
├── src/
│   ├── app.js          # React frontend entry
```

```
│   └── backend/
│       ├── server.js     # Backend entry point
│       ├── routes.js/
│       │
│       └── Databaseschema.js
│                         # Mongoose schema
├── assets/             # (Optional) Screenshots for docs
├── README.md           # Main documentation
├── ARCHITECTURE.md     # System architecture & flow
```

Technologies Used

Layer	Technology
Frontend	React
Backend	Node.js, Express
Database	MongoDB, Mongoose
HTTP	Fetch
Env Mgmt	dotenv
Dev Tools	nodemon (optional)

Summary

This architecture allows clean separation of concerns between client-side UI, API logic, and persistent data storage.
It's designed to be easily deployed, extended, or containerized for production environments.