System Design Document

Journaling Application

25th MARCH 2025

Table of Contents

Technology Stack

- Frontend – EJS (Embedded JavaScript)
- Backend – MyExpress (Custom Express Framework)
- Database – MySQL
- Storage – Local Directory

System Architecture

Architecture Overview

The system follows a MVC architecture, integrating the frontend, backend, and database in a structured manner.

Components used are as follows

- Frontend (EJS Templates) – Server-side rendering for fast performance.
- Backend (MyExpress Framework) – Build and powered by Nodejs & Express Js. https://github.com/Gicehajunior/myexpress-framework. Handles routing, authentication, and API logic.
- Service Layer – Business logic for journals, users, and categories.
- MySQL Database – Stores users details, journal entries, and categories.
- Local Storage – Stores journal attachments.

Data Model Design & Relationships

Database Schema

The Application uses a relational database model using MySQL.

1. Users – Stores user details and authentication information.
2. Categories – Organizes journals into predefined groups.
3. Journals – Stores user-created journal entries with category associations.

Users Table

```
CREATE TABLE users (
    id INT PRIMARY KEY AUTO_INCREMENT,
    fullname VARCHAR(255) NOT NULL,
    username VARCHAR(255) UNIQUE NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    contact VARCHAR(20) UNIQUE,
    password VARCHAR(255) NOT NULL,
    role ENUM('admin', 'user') DEFAULT 'user',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

Categories Table

```
CREATE TABLE categories (

    id INT PRIMARY KEY AUTO_INCREMENT,
    category_name VARCHAR(255) UNIQUE NOT NULL,
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

Journals Table

```
CREATE TABLE journals (
    id INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT NOT NULL,
    title VARCHAR(255) NOT NULL,
    description TEXT NOT NULL,
    category_id INT,
    attachments TEXT,
    status ENUM('draft', 'published', 'archived') DEFAULT 'draft',
    date DATE NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
    FOREIGN KEY (category_id) REFERENCES categories(id) ON DELETE SET NULL
);
```

Security Measures

Beyond Basic Authentication

1. Password Hashing – Uses bcrypt for secure password storage.
2. JWT-based Authentication – Stateless session management.
3. File Upload Security – Limits file types to prevent malicious uploads.
4. Role-Based Access Control (RBAC) – Restricts certain actions to users role'd as 'users'.
5. Cross-Site Scripting (XSS) Protection – Sanitizes user input.

Scalability Considerations

Challenges & Solutions

Challenge	**Solution**
Local storage limits	Migrate to AWS S3 or Google Cloud Storage
High database load	Implement indexing & caching (Redis)
Slow query performance	Optimize queries & add replication
User concurrency	Load balancing across multiple servers
Increased file storage needs	Use a separate file server or CDN

Scaling to 1M+ Users

- Database Sharding – Split data into multiple MySQL instances.
- Read/Write Splitting – Use Read Replicas to balance queries.
- Caching – Implement Redis/Memcached for fast data retrieval.
- Load Balancer – Distribute traffic across multiple MyExpress server instances.
- Cloud Storage – Move from local to AWS S3 or Google Cloud Storage.

Bottlenecks & Redesign Considerations

Potential Bottlenecks

Component	Problem	Solution
Database Reads	High reads on journals	Use Redis to cache responses
Large Attachments	Storage fills up	Move to a dedicated file storage system
High API Traffic	Server overload	Introduce API rate limiting & caching
Slow Queries	Complex queries	Indexing & query optimization

Components to Redesign at Scale

1. File Storage – Migrate to cloud storage/CDN.
2. Authentication – Offload to an OAuth2 provider (e.g., Auth0, Firebase).
3. Database – Implement Read/Write Replication & Sharding.

Technical Decision Log

Key Decisions

Using MySQL for the Database

1) Problem – Needed a reliable relational database.
2) Options Considered;
    - MySQL
    - PostgreSQL
    - MongoDB
3) Reason – MyExpress only supports MySQL, and luckily, relational structure suits journals.
4) Trade-offs – Might need replication at scale.

Using Local Storage for Attachments

1. Problem – Where to store journal file attachments.
2. Options Considered;
    - Local Directory (root)
    - Cloud Storage (AWS S3, Google Cloud)
3. Reason – Simpler setup, no extra costs.
4. Trade-offs – Not scalable beyond a single server.

Using EJS for Frontend Rendering

1. Problem: Needed a lightweight frontend solution.
2. Options Considered;
    - EJS
    - React.js
    - Vue.js
3. Reason – SSR (Server-Side Rendering) speeds up load times & integrates well with MyExpress.
4. Trade-offs – Less interactive compared to React/Vue.

Implementing JWT for Authentication

1. Problem – Needed a secure way to manage sessions.
2. Options Considered;
    - JWT
    - Sessions
3. Reason – JWT is stateless and works well for API-based apps.
4. Trade-offs – Requires token management.

Setup Guide

Cloning the Repository

```
> git clone https://github.com/Gicehajunior/journal-application
```

```
> cd journal-application
```

Database Setup

Import the MySQL dump file found in the db directory.

Use a MySQL client:

```
> mysql -u root -p myexpress < db/myexpress.sql
```

Install Dependencies

```
> npm install
```

Build Static Assets

```
> npm run build
```

Create and Save JWT Secret

After npm installation, you now have the ability to create JWT web token by using the below command on terminal. After creating, save the token in the respective config, JWT_SECRET.

```
> node -e "console.log(require('jsonwebtoken').sign({ user: 'JohnDoe' }, 'your-secret-key', { algorithm: 'HS256', expiresIn: '1h' }));"
```

Configurations

Configurations are stored in config/config.js.

```
const path = require('path');
require('dotenv').config();
module.exports = {
   APP: {
      APP_NAME: process.env.APP_NAME || 'MyExpress',
      APP_ENV: process.env.APP_ENV || 'development',
      APP_PORT: process.env.APP_PORT || 5000,
   },
   DATABASE: {
      DB_CONNECTION: process.env.DB_CONNECTION || 'mysql',
      DB_HOST: process.env.DB_HOST || '127.0.0.1',
      DB_NAME: process.env.DB_NAME || 'myexpress',
   },
};
```

Example .env file

```
APP_NAME='MyExpress'
APP_ENV='development'
APP_PORT=8200
APP_URL=http://127.0.0.1
DB_CONNECTION='mysql'
DB_HOST='127.0.0.1'
DB_PORT='3306'
DB_NAME='myexpress'
DB_USER='root'
DB_PASSWORD=''
JWT_SECRET='your_jwt_secret'
```

Run the Application

```
> npm run dev
```

Running Tests

To run Jest tests just type;

```
> npm test
```

Conclusion

This Journaling Application is structured with MyExpress, MySQL, and EJS. Future improvements should focus on handling high concurrency and offloading storage needs.