Issue Tracker API

REST API Implementation - Project Submission

Submitted By: Masud Zaman **Email:** masud.zmn@gmail.com

Date: May 29, 2025 **Branch:** solution/masud-zaman

GitHub: github.com/mzaman LinkedIn: linkedin.com/in/masudzaman

Overview

This submission demonstrates a production-ready REST API implementation that addresses all required tasks while incorporating additional engineering practices for maintainability, testing, and operational reliability. The solution emphasizes realistic data flows, comprehensive automation, and scalable architecture patterns.

Prerequisites

Before starting, ensure your system has:

- ✓ Docker & Docker Compose installed and running
- ✓ Bash available (Linux/macOS/WSL2 on shell Windows)

✓ Git for repository access

✓ Minimum 4GB RAM recommended for containers

Verify installation:

docker --version docker-compose --version

Quick Start Guide

1. Download Project and Checkout the Solution Branch

Download URL: https://drive.google.com/drive/folders/16mz6cIVCp61NRYRPBT2_XlgCbXHHKhqC?usp=sharing

git checkout solution/masud-zaman

2. One-Command Setup

chmod +x ./cmd/* & ./cmd/install

The install script will automatically:

- Build Docker images and start containers
- Initialize MySQL database with schema
- Run migrations and seed realistic sample data
- Configure all services and dependencies

3. Access Points

PAPPLICATION Access Points

Swagger UI

localhost:5555

API Base URL

localhost:8080

PhpMyAdmin

localhost:8081

Default Credentials

Admin User:

admin@example.com /

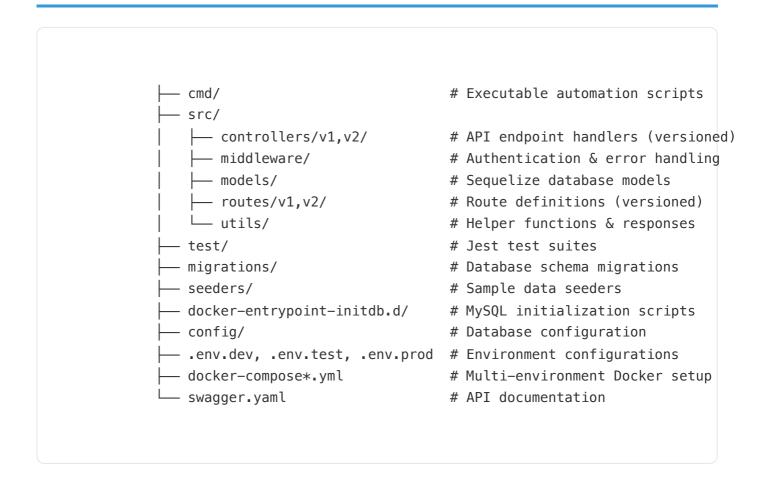
Password123

MySQL: root / abc123456 Valid Client ID:

my-client-id-123



Project Structure



X Technical Stack

The solution implements a modular Node.js/Koa.js REST API with the following components:

Backend

Node.js with Koa.js framework, highly structured routing and middleware

Database

MySQL with Sequelize ORM, migrations, and seeders

Authentication

JWT-based with middleware for secure user authentication

Documentation

Swagger/OpenAPI integration for API documentation

Testing

Jest test suite with Docker-based test database

Containerization

Docker & Multi-environment Docker Compose with automation scripts

DevOps

Bash automation scripts & Process Management



Core Features Implementation

Required Tasks (All Completed) **✓ COMPLETE**

REST API with proper HTTP methods, status Task 1-6: codes, and JSON responses

Sequelize ORM with migration and Database **Integration:** seeder support

Docker Compose configuration Multi-**Environment** for dev/test/prod Support:

Database Automated MySQL setup with **Initialization:** SQL scripts

Process Comprehensive bash scripts for **Automation:** container management

✓ API Interactive Swagger UI at **Documentation:** localhost:5555

Additional Engineering Enhancements

Realistic Data Modeling

- · Comprehensive seed data reflecting real-world usage patterns
- Complex relationships enabling realistic transaction
- · Meaningful sample data for effective testing

Operational Excellence

- · Unified automation via bash scripts and Makefile
- Environment-specific configuration management
- Docker-based development workflow with hot reload

Code Quality & Maintainability

- · Modular architecture with clear separation of concerns
- Structured middleware, services, and controllers
- Comprehensive error handling and logging

Testing Infrastructure

- Jest test suite covering API endpoints
- Docker-based test database isolation
- Automated test execution via ./cmd/app + npm run test , or, simply run: ./cmd/test

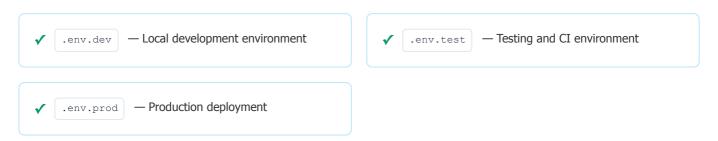
Engineering Approach

Rather than implementing a minimal solution, I focused on building a system that demonstrates real-world engineering practices. This included investing time in automation, realistic data modeling, comprehensive testing, and operational tooling that would be expected in a production environment.

Trade-off Decision: TypeScript migration was planned but deferred in favor of ensuring robust core functionality, comprehensive testing, and operational reliability. The current JavaScript implementation prioritizes stability and thorough feature coverage.

Environment Configuration

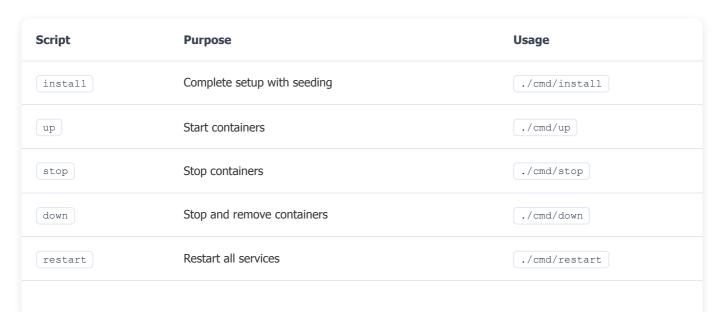
The project supports multiple environments with dedicated configuration:



Each environment maintains isolated database credentials, ports, and service configurations.

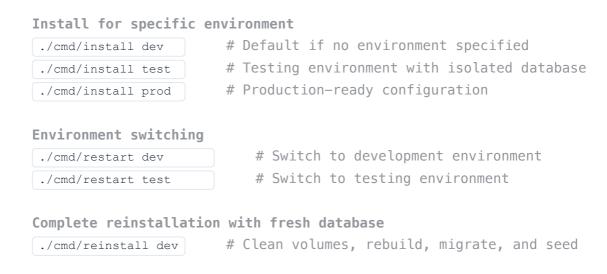
Container Management

Quick Commands



Script	Purpose	Usage
rebuild	Rebuild with no cache	./cmd/rebuild
reinstall	Fresh installation with clean database	./cmd/reinstall

Environment-Specific Operations



Database Management

Automated Setup

- SQL initialization scripts in <code>docker-entrypoint-initdb.d</code> is available, but not used due to automatic database initialization setup using docker-compose, customized scripts, and environment variables for migrations and seeds.
- Sequelize migrations for schema versioning
- Comprehensive seeders with realistic sample data

Manual Operations

```
./cmd/app
npm run migrate
npm run seed
npm run migrate:undo
```

npm run migrate:undo:all



Authentication & Authorization

Implementation

- ✓ User management with secure password hashing
- ✓ JWT token-based authentication via POST /login
- ✓ Required X-Client-ID header validation
- ✓ Automatic created by / updated by tracking audit trail with

Usage Example

```
# Login curl -X 'POST' \ 'http://localhost:8080/api/v1/auth/login' \ -H
'accept: application/json' \ -H 'x-client-id: my-client-id-123' \ -H 'Content-
Type: application/json' \ -d '{ "email": "user-1@example.com", "password":
"Password123" }'
```

Use token for authenticated requests

Headers:

Authorization: Bearer <jwt token> X-Client-ID: my-client-id-123

Token Features

- Configurable expiry via JWT EXPIRES IN environment variable
- Email claims automatically recorded as created by updated by in audit trails
- Secure JWT secret configuration via JWT SECRET

Testing

Test Execution

./cmd/test # or ./cmd/app npm run test

Test Coverage

✓ API endpoint validation

Authentication flow testing

✓ Database integration tests

✓ Error handling verification



API Documentation

Interactive Documentation: Available at http://localhost:5555 via Swagger UI with complete endpoint testing capabilities.

YAML Documentation: Full endpoint documentation available at \(\)./swagger.yaml \(\) for offline reference and integration testing.

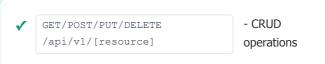
This API allows clients to perform discovery, health checks, and issue tracking.

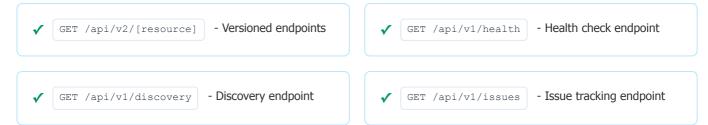
This API requires:

- Setting the X-Client-ID header for all requests. Set this value: my-client-id-123 to X-Client-ID field in Authorize area.
- Authorizing using Bearer Token after login. Set token via Authorization tab or Authorization header.

API Endpoints Summary







Versioning and Prefix

Flexible API Versioning

- You can switch between versions using the path parameters like /v1 , /v2 , or omit when not needed.
- Prefix can also be changed dynamically depending on environment setup.
- For example, /api/v1 or /api/v2 or /v1 or /v2 depending on the setup.
- All APIs are accessible in all combinations of:
 - No prefix (/)
 - Version only (/v1, /v2, etc.)
 - API prefix only (/api)
 - API prefix + version (/api/v1, /api/v2, etc.)

When version grows (v1, v2, and so on), as a future-proof solution, this setup allows: $\label{eq:condition}$ /v1/health $\label{eq:condition}$ /v2/health $\label{eq:condition}$ /api/v1/health $\label{eq:condition}$ etc...

▲ Important Requirements:

- All endpoints require X-Client-ID (Value: my-client-id-123)
- Some endpoints require Bearer Token from login response

Postman Collection

Postman Collection: Import Trail Day REST API.postman collection.json for automated testing workflows:

- 1. Import the collection into Postman
- 2. Authenticate via /login endpoint
- 3. Token automatically applied to subsequent requests
- 4. Complete test coverage for all endpoints

Production Deployment Notes

- Use [./cmd/install prod] for production setup
- Ensure environment variables are properly configured
- · Database credentials should be updated for production
- · Consider load balancing and scaling requirements

Troubleshooting

Port Conflicts

Ensure ports 8080, 5555, 8081, 3307 are available

Docker Issues

 $Run \ \, | \ \, \text{docker system prune} \ \, \text{if containers fail to start}$

Database Connection Errors

Verify MySQL container is fully initialized

Development Workflow

Container Access

```
./cmd/app # Enter application container
./cmd/exec dev # Execute shell in dev environment
```

Database Access

✓ PhpMyAdmin: http://localhost:8081



Log Monitoring

```
docker-compose logs -f app
docker-compose logs -f mysql
```

9 Future Enhancements

- ▼ TypeScript migration for better type safety
- ✓ Redis caching layer implementation

✓ API rate limiting

✓ Enhanced monitoring and logging

Summary

This submission delivers a complete, production-ready REST API that exceeds the basic requirements through thoughtful engineering practices. The solution emphasizes maintainability, operational reliability, and realistic data flows while providing comprehensive tooling for development and deployment workflows.

Key Achievements:

✓ Complete Task All 6 required tasks fully Implementation: implemented with proper

REST API conventions

✓ Production Ready
 Architecture:
 Modular design with clear
 separation of concerns and
 scalable patterns

✓ **Comprehensive** Jest test suite with Docker-**Testing:** based isolation and automated execution ✓ **Operational** Full automation scripts, multi-**Excellence:** environment support, and deployment tooling

✓ **Developer** Interactive documentation, realistic **Experience:** sample data, and streamlined workflow

✓ Security JWT authentication, secure password handling, and audit trail tracking

Technical Highlights:

- Modern Tech Stack: Node.js/Koa.js with MySQL and Sequelize ORM
- API Versioning: Flexible versioning system supporting multiple URL patterns
- Container Orchestration: Docker Compose with environment-specific configurations
- Database Management: Automated migrations, seeders, and initialization scripts
- **Documentation:** Interactive Swagger UI and comprehensive API documentation
- Authentication: JWT-based security with configurable expiration and audit tracking

Engineering Philosophy: Rather than implementing a minimal viable product, this solution demonstrates enterprise-level engineering practices including automation, testing, documentation, and operational considerations that would be expected in a real-world production environment.

© Quick Access Summary

Setup Command

API Documentation http://localhost:5555 API Endpoint
http://localhost:8080

chmod +x ./cmd/* && ./cmd/install

Database Admin

http://localhost:8081

Test Execution

./cmd/test

Container Management

./cmd/[up|stop|restart]

Project Submission Complete

Masud Zaman | May 29, 2025 | solution/masud-zaman

Issue Tracker API - Production-Ready REST API Implementation