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 shell available (Linux/macOS/WSL2 on Windows)

✓ Git for repository access

✓ Minimum 4GB RAM recommended for containers

Verify installation:

docker --version
docker-compose --version

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

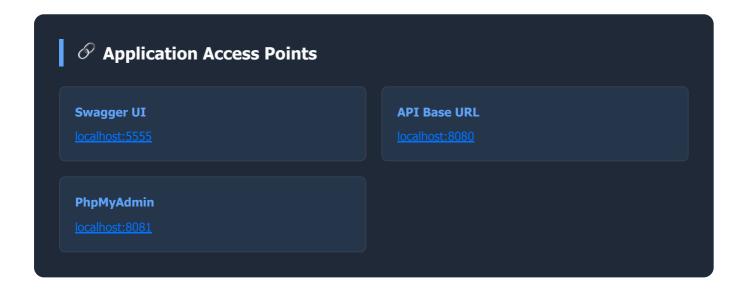
2. One-Command Setup

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



Default Credentials

Admin User:

admin@example.com /
Password123

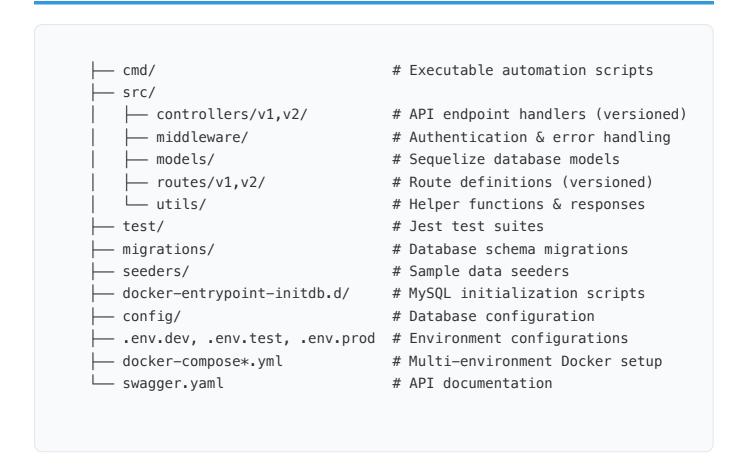
MySQL:

root / abc123456

Valid Client ID:

my-client-id-123

Project Structure



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

Documentation

Swagger/OpenAPI integration for API documentation

Database

MySOL with Sequelize ORM, migrations, and seeders

Testing

Jest test suite with Docker-based test database

Authentication

JWT-based with middleware for secure user authentication

Containerization

Docker & Multi-environment Docker Compose with automation scripts

DevOps

Bash automation scripts & Process Management

Required Tasks (All Completed)

✓ COMPLETE

- √ Task 1-6: REST API with proper HTTP methods, status codes, and JSON responses
- ✓ Database Integration: Sequelize ORM with migration and seeder support
- ✓ Multi-Environment Support: Docker Compose configuration for dev/test/prod
- ✓ **Database Initialization:** Automated MySQL setup with SQL scripts
- ✓ Process Automation: Comprehensive bash scripts for container management
- ✓ API Documentation: Interactive Swagger UI at localhost:5555

Additional Engineering Enhancements

Realistic Data Modeling

- Comprehensive seed data reflecting real-world usage patterns
- Complex relationships enabling realistic transaction flows
- · 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

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
install	Complete setup with seeding	./cmd/install
up	Start containers	./cmd/up
stop	Stop containers	./cmd/stop
down	Stop and remove containers	./cmd/down
restart	Restart all services	./cmd/restart

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

Environment-Specific Operations

```
Install for specific environment

./cmd/install dev  # Default if no environment specified

./cmd/install test  # Testing environment with isolated database

./cmd/install prod  # Production—ready configuration

Environment switching

./cmd/restart dev  # Switch to development environment

./cmd/restart test  # Switch to testing environment

Complete reinstallation with fresh database

./cmd/reinstall dev  # Clean volumes, rebuild, migrate, and seed
```

Database Management

Automated Setup

- SQL initialization scripts in docker-entrypoint-initab.d 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 audit trail with created by / updated by tracking

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 <code>created_by</code> / <code>updated_by</code> 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.

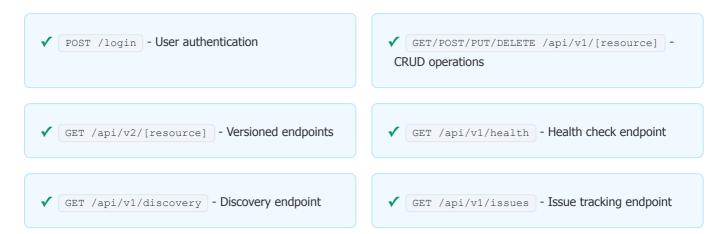
This API allows clients to perform discovery, health checks, and issue tracking.

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.)

▲ 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

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

✓ **Direct MySQL:** Connect to localhost:3307 with provided credentials

Log Monitoring

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

Future Enhancements

✓ TypeScript migration for better type safety

✓ Redis caching layer implementation

✓ API rate limiting

✓ Enhanced monitoring and logging



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:

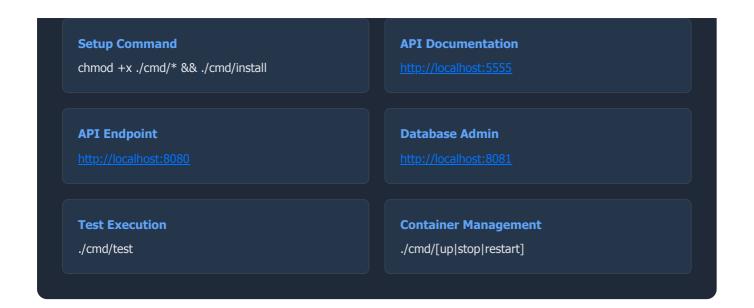
- ✓ Complete Task Implementation: All 6 required tasks fully implemented with proper REST APT conventions
- ✓ Comprehensive Testing: Jest test suite with Docker-based isolation and automated execution
- ✓ Developer Experience: Interactive documentation, realistic sample data, and streamlined workflow

- ✓ Production-Ready Architecture: Modular design with clear separation of concerns and scalable patterns
- ✓ Operational Excellence: Full automation scripts, multi-environment support, and deployment tooling
- ✓ Security Implementation: 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 enterpriselevel engineering practices including automation, testing, documentation, and operational considerations that would be expected in a real-world production environment.



Project Submission Complete

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

Issue Tracker API - Production-Ready REST API Implementation