# Project Submission Document

## Overview

This submission demonstrates a production-ready REST API implementation that addresses all required tasks while incorpo

## 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:
```bash
docker --version
docker-compose --version
```

## Quick Start Guide

### 1. Download Project and Checkout the Solution Branch
Download URL: https://drive.google.com/drive/folders/16mz6cIVCp6lNRYRPBT2_XlgCbXHHKhqC?usp=sharing

```bash
git checkout solution/masud-zaman
```

### 2. One-Command Setup
```bash
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
- **Swagger UI:** [http://localhost:5555](http://localhost:5555)
- **API Base URL:** [http://localhost:8080](http://localhost:8080)
- **PhpMyAdmin:** [http://localhost:8081](http://localhost:8081)

**Default Credentials:**
- Admin User: `admin@example.com / Password123`
- MySQL: `root / abc123456`
- Valid Client ID: `my-client-id-123`

---

## Project Structure

```
■■■ cmd/                    # Executable automation scripts
■■■ src/
■   ■■■ controllers/v1,v2/      # API endpoint handlers (versioned)
■   ■■■ middleware/             # Authentication & error handling
■   ■■■ models/                 # Sequelize database models
■   ■■■ routes/v1,v2/           # Route definitions (versioned)
■   ■■■ utils/                  # Helper functions & responses
```

■■■ test/                    # Jest test suites
■■■ migrations/              # Database schema migrations
■■■ seeders/                 # Sample data seeders
■■■ docker-entrypoint-initdb.d/   # MySQL initialization scripts
■■■ config/                  # Database configuration
■■■ .env.dev, .env.test, .env.prod # Environment configurations
■■■ docker-compose*.yml      # Multi-environment Docker setup
■■■ swagger.yaml             # API documentation
```

---

## Architecture Overview

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

- **API Layer:** Koa.js with structured routing and middleware
- **Database:** MySQL with Sequelize ORM, migrations, and seeders
- **Authentication:** JWT-based with role management
- **Documentation:** Swagger UI integration
- **Testing:** Jest test suite with Docker-based test database
- **DevOps:** Multi-environment Docker Compose with automation scripts

---

## Core Features Implementation

### ■ Required Tasks (All Completed)

- **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
- Automated test execution via `./cmd/app` + `npm run test` or `./cmd/test`

---

## Engineering Approach
```

Rather than implementing a minimal solution, I focused on building a system that demonstrates real-world engineering pract

**Trade-off Decision:** TypeScript migration was planned but deferred in favor of ensuring robust core functionality, compre

---

## Environment Configuration

The project supports multiple environments with dedicated configuration:

- `.env.dev` — Local development environment
- `.env.test` — Testing and CI environment
- `.env.prod` — Production deployment

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` |
| `rebuild` | Rebuild with no cache | `./cmd/rebuild` |
| `reinstall` | Fresh installation with clean database | `./cmd/reinstall` |

### Environment-Specific Operations

```bash
# 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/env dev        # Switch to development environment
./cmd/env 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-initdb.d`
- Sequelize migrations for schema versioning
- Comprehensive seeders with realistic sample data

### Manual Operations

```bash
./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

```bash
# Login
POST /login
{
  "email": "admin@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

```bash
./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](http://localhost:5555) via Swagger UI with complete endpo

**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

---

## Development Workflow

### Container Access

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

### Database Access

- **PhpMyAdmin:** [http://localhost:8081](http://localhost:8081)
- **Direct MySQL:** Connect to `localhost:3306` with provided credentials

### Log Monitoring

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

---

## Summary

This submission delivers a complete, production-ready REST API that exceeds the basic requirements through thoughtful e