Secure Authentication

Enterprise-grade JWT-based authentication service for modern applications

Authentication Service

Complete API Reference and Implementation Guide

A comprehensive microservice built with Node.js and Express.js, providing secure user authentication, token management, and profile access with industry-standard security practices.

- Secure JWT-based authentication
- Password hashing with bcryptjs
- Rate limiting and security headers
- MySQL database with connection pooling

Service Overview

User Management

Secure user registration and profile management with email validation and account status tracking.

- → User registration
- → Email validation
- → Profile retrieval
- → Account status

Authentication

Industry-standard JWT-based authentication with secure password hashing and token management.

- → JWT tokens
- → Bcryptjs hashing
- → Token refresh
- → Session logout

Security

Comprehensive security measures including rate limiting, input validation, and security headers.

- → Rate limiting
- → Input validation
- → Helmet headers
- → CORS support

6 API Endpoints for complete authentication lifecycle

100 Requests per 15 minutes rate limit

Architecture: Layered Design

Routes

Express Route Handlers

HTTP endpoint definitions for registration, login, refresh, logout, and user profile retrieval

↓

Services

Business Logic

Core authentication logic including user registration, login, token generation, and profile retrieval

↓

Repository

Data Access

Database operations for user CRUD operations with parameterized queries and connection pooling

↓

Middleware

Authentication & Security

JWT token validation, user verification, rate limiting, CORS, and security headers

Next: Database Layer, Utilities, and Connection Pooling

Architecture: Data and Utilities

Database Layer

MySQL Data Store

Persistent storage for user accounts, credentials, and account status with connection pooling

- → User account storage
- → Credential management
- → Account status tracking
- → Timestamp management

Connection Pooling

Efficient database connection management with configurable pool size and queue limits

- → Min connections: 2
- → Max connections: 10
- → Queue limit: 0
- → Connection timeout: 10s

Utility Functions

JWT Token Generation

Create secure JWT tokens with user claims and expiration

- → HS256 algorithm
- → Access token: 15 min
- → Refresh token: 7 days
- → User ID & email payload

Password Security

Hash and verify passwords using bcryptjs with strong salt rounds

- → Bcryptjs hashing
- → Salt rounds: 12
- → Secure comparison
- → No plaintext storage

Email & Input Validation

Validate and sanitize user inputs for data integrity

- → Email format validation
- → Password complexity check
- → Input sanitization
- → XSS prevention

Secure by Design

All sensitive operations use industry-standard cryptographic functions and secure comparison algorithms

Performance Optimized

Connection pooling and efficient query patterns ensure fast response times and resource utilization

API Endpoints Summary

POST

/v1/api/auth/register

Authentication: Optional

Create new user account with email, password, and optional name. Returns user data and JWT tokens.

POST

/v1/api/auth/login

Authentication: Optional

Authenticate user with email and password. Returns user profile and access/refresh tokens.

POST

/v1/api/auth/refresh

Authentication: Optional

Generate new access and refresh tokens using valid refresh token for seamless session renewal.

POST

/v1/api/auth/logout

Authentication: Required

Invalidate current user session. Requires valid JWT token in Authorization header.

GET

/v1/api/users/me

Authentication: Required

Retrieve current authenticated user's profile information including email, name, and account status.

GET

/health

Authentication: Optional

Check service health status. Returns status and timestamp for monitoring and uptime verification.

POST Create or modify resources

GET Retrieve resources

Required Bearer token required in Authorization header

User Registration Flow: Validation

1

Client Request

User submits email, password, and name through mobile app

2

Email Validation

Verify email format matches valid pattern

3

Password Check

Validate password meets complexity requirements

Password Requirements

- ✓ Minimum 8 characters long
- ✓ At least one uppercase letter (A-Z)
- ✓ At least one lowercase letter (a-z)
- ✓ At least one number (0-9)
- ✓ At least one special character (!@#$%^&*)

Next: Duplicate Check, Password Hashing, and Token Generation

User Registration Flow: Storage

4

Duplicate Check

Query database to ensure email is unique

5

Password Hashing

Hash password using bcryptjs (12 salt rounds)

6

Store & Generate

Save user to database and generate JWT tokens

Bcryptjs Password Hashing Details

- Algorithm: Bcryptjs with 12 salt rounds
- Output: 60-character hash stored in database
- Security: Resistant to rainbow table and brute-force attacks
- Verification: Secure comparison prevents timing attacks

↓

Registration Success Response

→ User object with id, email, name, account status

→ Access token (15 minute expiration)

→ Refresh token (7 day expiration)

→ HTTP 201 Created status code

User Login Flow

1
User Input
Email & Password

→

2
Email Lookup
Query Database

→

3
User Found?
Validation Check

4
Password Verify
Bcryptjs Compare

→

5
Account Status
Check Active

→

6
Generate Tokens
JWT Creation

✓
Return Response
User Data + Tokens

Password Verification

Bcryptjs compares provided password with stored hash using secure comparison algorithm

Token Generation

Access token (15 min) and refresh token (7 days) generated with user ID and email

Token Management and Refresh

JWT Token Structure

Access Token

Purpose:	**API Requests**
Expiration:	**15 minutes**
Algorithm:	**HS256**
Payload:	**userId, email**

Refresh Token

Purpose:	**Token Renewal**
Expiration:	**7 days**
Algorithm:	**HS256**
Payload:	**userId, email**

Token Refresh Flow

1. **Client Detects Expiration**
   Access token nears expiration or returns 401 error
2. **Send Refresh Request**
   POST /v1/api/auth/refresh with refresh token
3. **Verify Refresh Token**
   Server validates token signature and expiration
4. **Generate New Tokens**
   New access and refresh tokens created and returned
5. **Continue Requests**
   Client uses new tokens for subsequent API calls

Seamless Experience
Users stay logged in without re-entering credentials during long sessions

Enhanced Security
Short-lived access tokens minimize risk if credentials are compromised

Database Schema and Data Model

Column Name	Data Type	Constraints	Description
`id`	`CHAR(36)`	PRIMARY KEY NOT NULL	Unique identifier using UUID v4 format for distributed system compatibility
`email`	`VARCHAR(255)`	UNIQUE NOT NULL	User email address with unique constraint to prevent duplicate accounts
`password`	`VARCHAR(255)`	NOT NULL	Bcryptjs hashed password (60 characters) - never stored in plaintext
`name`	`VARCHAR(255)`	NULLABLE	User's full name or display name (optional field)
`accountStatus`	`ENUM`	NOT NULL	Account status: 'active', 'inactive', 'suspended' (default: 'active')
`createdAt`	`TIMESTAMP`	NOT NULL	Automatic timestamp when user account is created (DEFAULT CURRENT_TIMESTAMP)
`updatedAt`	`TIMESTAMP`	NOT NULL	Automatic timestamp updated on any record modification (ON UPDATE CURRENT_TIMESTAMP)

Indexes and Constraints

PRIMARY KEY: id (UUID)

UNIQUE INDEX: email

AUTO TIMESTAMP: createdAt, updatedAt

COLLATION: utf8mb4_unicode_ci

Table Configuration

Table Name: users

Engine: InnoDB

Charset: utf8mb4

Connection Pool: 10 connections

Technology Stack: Runtime & Data

Runtime & Framework

Node.js

JavaScript runtime for building scalable server-side applications

✓ Non-blocking I/O, event-driven architecture perfect for concurrent requests

Express.js

Lightweight web framework for routing and middleware

✓ Minimal overhead, flexible middleware chain, industry standard

Data & Storage

MySQL

Relational database for persistent user data storage

✓ ACID compliance, reliable transactions, proven stability

Connection Pooling

Efficient database connection management

✓ Reduces overhead, improves response times, resource optimization

Next: Security & Utilities - bcryptjs, JWT, Helmet.js, CORS, Rate Limiting, and scalability benefits

Technology Stack: Security & Utilities

Security & Utilities

bcryptjs

Password hashing with adaptive salt rounds

✓ Resistant to brute force, industry standard, secure comparison

jsonwebtoken

JWT token generation and verification

✓ Stateless authentication, secure claims, standard format

Helmet.js

HTTP security headers middleware

✓ Protects against common vulnerabilities, XSS prevention

CORS & Rate Limit

Cross-origin support and request throttling

✓ Prevents abuse, controls traffic, enables mobile integration

Key Benefits

- Production-ready security
- Industry best practices
- Strong community support
- Proven performance

Local Development Setup: Windows & macOS

Windows

Step 1: Node.js

Download & Install

Visit nodejs.org, download LTS version, run installer

Step 2: MySQL

Install MySQL Server

Download MySQL Community Server, run MSI installer

Step 3: Extract

Unzip Service

```
unzip auth-service.zip
```

Step 4: Install

Dependencies

```
npm install
```

Step 5: Configure

Environment

```
copy example.env .env
```

Step 6: Start

Run Service

```
npm start
```

macOS

Step 1: Homebrew

Install Package Manager

/bin/bash -c "$(curl -fsSL ...)"

Step 2: Node.js

Install via Homebrew

```
brew install node
```

Step 3: MySQL

Install via Homebrew

```
brew install mysql
```

Step 4: Extract

Unzip Service

```
unzip auth-service.zip
```

Step 5: Install

Dependencies

```
npm install
```

Step 6: Start

Run Service

```
npm start
```

Next: Linux installation steps, database configuration, and service verification

Local Development Setup: Linux & Verification

Linux Installation

Step 1: Update

System Packages

```
sudo apt update
```

Step 2: Node.js

Install Node

```
sudo apt install nodejs npm
```

Step 3: MySQL

Install MySQL

```
sudo apt install mysql-server
```

Step 4: Extract

Unzip Service

```
unzip auth-service.zip
```

Step 5: Install

Dependencies

```
npm install
```

Step 6: Start

Run Service

```
npm start
```

Database Configuration

- → Create MySQL user and database
- → Update .env with DB credentials
- → Set DB_HOST, DB_USER, DB_PASSWORD
- → Set DB_NAME and DB_PORT

Environment Variables (.env)

- → PORT=3000
- → NODE_ENV=development
- → JWT_SECRET=your_secret_key
- → JWT_EXPIRE=15m
- → REFRESH_TOKEN_EXPIRE=7d

Verification Steps

- ✓ Service starts without errors
- ✓ Database connection successful
- ✓ Health endpoint responds
- ✓ No port conflicts on 3000

Health Check Endpoint

```
GET http://localhost:3000/health
```

Returns service status and timestamp to confirm successful startup

Postman API Testing

Collection Structure

📦 **Auth Service API**

- **Authentication**
  - POST ├─ Register User
  - POST ├─ Login User
  - POST ├─ Refresh Token
  - POST └─ Logout
- **Users**
  - GET └─ Get Current User
- **Health**
  - GET └─ Health Check

Testing Features

Automated Test Scripts

Each request includes pre-request and test scripts for validation and data extraction

Environment Variables

Pre-configured variables: baseUrl, accessToken, refreshToken, userId, testEmail

Token Auto-Extraction

Tokens automatically saved to environment after login and refresh requests

Collection Runner

Execute all requests sequentially with detailed test results and reporting

Test Assertions

Validate response status, structure, content, and data types automatically

Easy Import

Import JSON collection file directly into Postman with one click

Testing Workflow

Import collection → Select environment → Run individual tests or collection → Review results and logs

Quick Start

Complete API testing setup in 2 minutes. All endpoints tested and documented with examples.

Security Features: Core Protection

Authentication

Password Hashing

Bcryptjs with 12 salt rounds for secure password storage

- 60-character hash output
- Resistant to brute force attacks
- No plaintext storage
- Secure comparison prevents timing attacks

JWT Authentication

HS256 signed tokens with configurable expiration

- Access token: 15 minutes
- Refresh token: 7 days
- Stateless authentication
- Secure token verification

Data Protection

Rate Limiting

Prevent brute force and DDoS attacks

- 100 requests per 15 minutes
- Per IP address tracking
- 429 Too Many Requests response
- Configurable thresholds

Input Validation

Comprehensive validation and sanitization

- Email format validation
- Password complexity requirements
- SQL injection prevention
- XSS attack prevention

Next: Infrastructure Security, CORS, Helmet Headers, Database Security, and Production Best Practices

Security Features: Infrastructure & Compliance

Infrastructure Security

CORS Configuration

Cross-Origin Resource Sharing for mobile apps

- Whitelist allowed origins
- Restrict HTTP methods
- Control credential sharing
- Preflight request handling

Helmet Security Headers

HTTP security headers for browser protection

- Content Security Policy (CSP)
- X-Frame-Options (clickjacking)
- X-Content-Type-Options
- Strict-Transport-Security (HSTS)

Data & Deployment

Database Security

Secure database operations and access control

- Parameterized queries
- Connection pooling
- Least privilege access
- Encrypted connections

Production Deployment

Best practices for production environments

- Use HTTPS/TLS encryption
- Environment variable secrets
- Monitoring and logging
- Regular security updates

Error Handling and Response Codes

Status Code	Meaning	Common Scenarios	Error Type
200	**OK**	Successful GET request, login successful, token refresh successful	Success
201	**Created**	User registration successful, new resource created	Success
400	**Bad Request**	Invalid email format, password too weak, missing required fields	Validation Error
401	**Unauthorized**	Invalid credentials, expired token, missing Authorization header	Authentication Error
404	**Not Found**	User not found, endpoint does not exist	Resource Error
429	**Too Many Requests**	Rate limit exceeded (100 requests per 15 minutes)	Rate Limit Error
500	**Server Error**	Database connection failure, unexpected server error	Server Error

Example: Validation Error (400)

```
{
  "status": "error",
  "message": "Validation failed",
  "errors": {
    "email": "Invalid",
    "password": "Too weak"
  }
}
```

Invalid input format or missing required fields

Example: Authentication Error (401)

```
{
  "status": "error",
  "message": "Invalid credentials",
  "code": "AUTH_FAILED"
}
```

Wrong password, expired token, or missing credentials

Example: Not Found Error (404)

```
{
  "status": "error",
  "message": "User not found",
  "code": "USER_NOT_FOUND"
}
```

Requested resource does not exist in database

Example: Rate Limit Error (429)

```
{
  "status": "error",
  "message": "Too many requests",
  "retryAfter": 900
}
```

Exceeded rate limit, retry after 15 minutes

Getting Started and Next Steps

Quick Start Timeline

1 — Extract & Install
Unzip service and run npm install
2 minutes

2 — Configure Environment
Copy example.env and update database credentials
1 minute

3 — Start Service
Run npm start and verify health endpoint
1 minute

4 — Test with Postman
Import collection and run automated tests
2 minutes

Total Setup Time: 5-6 minutes

Key Resources

API Documentation
Complete endpoint reference with request/response examples

Postman Collection
Pre-configured requests with automated test scripts

Source Code
Well-documented codebase with clear architecture

Security Guide
Best practices and production deployment recommendations

Database Schema
SQL schema with migration scripts and indexes

Team Collaboration
Share documentation with backend, frontend, and QA teams. Use Postman collection for collaborative testing and API validation across environments.

Ready to Deploy?
Follow the setup guide, run tests, and deploy to development environment. Monitor logs and performance metrics in production.