

Swagger API Documentation – Layout, Endpoints, Schema, and Security

The Journal Application uses **Swagger UI (OpenAPI 3)** to provide interactive API documentation and testing. Swagger acts as a visual interface for the backend, allowing developers and users to view all available endpoints, understand request and response formats, and test APIs securely.

Swagger UI is accessible at:

<http://localhost:8080/swagger-ui/index.html>

1. Swagger Dashboard Overview

The Swagger dashboard is the main interface where all REST APIs are organized into logical groups based on functionality and access level.

It provides:

- * List of all available endpoints
- * HTTP methods (GET, POST, PUT, DELETE)
- * Request body format
- * Response format
- * Authentication options
- * Schema definitions

This makes it easy to understand and test backend APIs without external tools like Postman.

Swagger UI - localhost:8080/swagger-ui/index.html#/

Swagger
Supported by SMARTBEAR

/v3/api-docs

Explore

Journal App APIs

By parth

Servers: http://localhost:8080 - local Authorize

Public APIs

POST /public/sign-up

POST /public/login

GET /public/health-check

User APIs

GET /user

PUT /user

Journal APIs

PUT /journal/{id}

DELETE /journal/{id}

GET /journal

POST /journal

GET /journal/{myid}

Admin APIs

POST /Admin/create-newAdmin

GET /Admin/clear-app-cache

GET /Admin/all-users

Schemas

UserDTO ↴ {
 id
 userName*
 email
 password*
 sentimentAnalysis
}

ObjectId ↴ {
 timestamp
 date
}

JournalEntry ↴ {
 id
 date
 title
 content
}

2. Public Endpoints (No Authentication Required)

Public endpoints allow users to access basic system functionality without authentication.

Available endpoints include:

POST /public/sign-up

POST /public/login

GET /public/health-check

Purpose:

- * Register new users
- * Authenticate existing users
- * Verify application status

The login endpoint generates a JWT token, which is required to access protected endpoints.

Public APIs		
POST	/public/sign-up	^
POST	/public/login	▼ 🔒
GET	/public/health-check	▼ 🔒

3. JWT Authentication and Authorization

The application uses JWT (JSON Web Token) based authentication.

Authentication workflow:

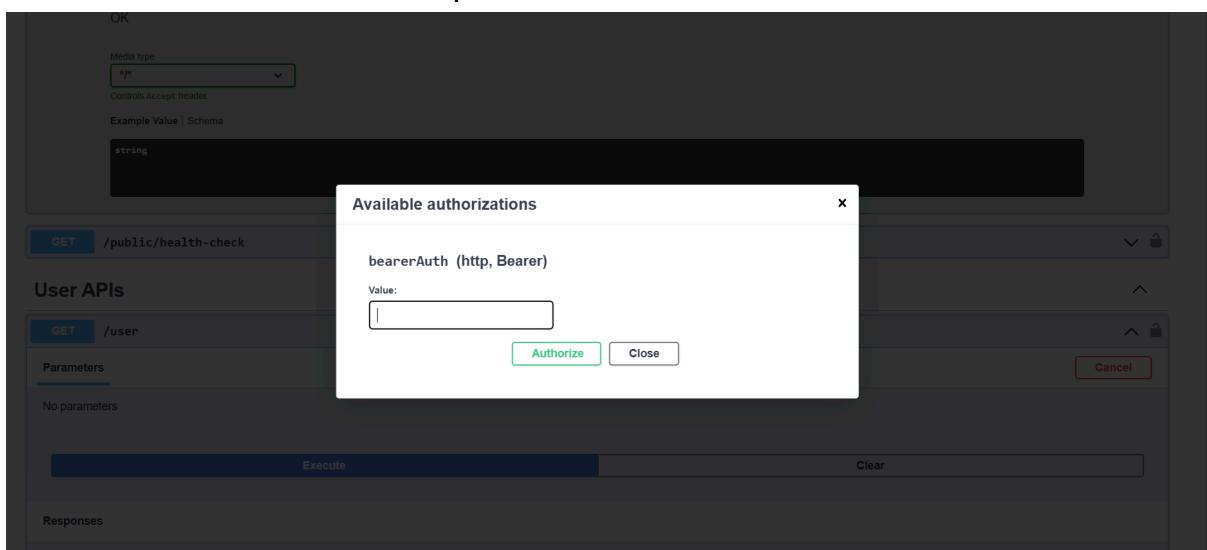
1. User logs in using `/public/login`
2. Server validates credentials

3. Server generates JWT token
4. User clicks "Authorize" button in Swagger
5. User enters JWT token
6. Swagger includes token in Authorization header automatically

Header format:

Authorization: Bearer <JWT_TOKEN>

This allows secure access to protected APIs.



4. User Endpoints (Authentication Required)

User endpoints allow authenticated users to access personal account information.

Example endpoint:

GET /user

Function:

* Returns currently authenticated user details

These endpoints require a valid JWT token.

If the token is missing or invalid, access is denied.

User APIs

GET /user

PUT /user

5. Journal Endpoints (Core Application Functionality)

Journal endpoints allow users to perform CRUD operations on journal entries.

Available operations:

Create entry:

...

POST /journal

...

Get all entries:

...

GET /journal

...

Get entry by ID:

...

GET /journal/id/{id}

...

Update entry:

...

PUT /journal/id/{id}

Delete entry:

DELETE /journal/id/{id}

These endpoints are protected and require authentication.

Each user can manage their own journal entries securely.

The screenshot shows a list of five API endpoints under the heading "Journal APIs". The endpoints are:

- PUT** /journal/id/{id} (highlighted with an orange border)
- DELETE** /journal/id/{id} (highlighted with a red border)
- GET** /journal
- POST** /journal
- GET** /journal1/id/{myid}

Each endpoint has a small lock icon and a dropdown arrow to its right.

6. Schema Representation

Swagger automatically displays schema definitions for request and response objects.

Example: JournalEntry schema contains:

- * id → Unique identifier
- * title → Journal title
- * content → Journal content
- * date → Entry creation date

Schema visualization helps developers understand data structure clearly.

Schemas	^
<pre>UserDTO ↵ { id > [...] userName* > [...] email > [...] password* > [...] sentimentAnalysis > [...] }</pre>	
<pre>ObjectId ↵ { timestamp > [...] date > [...] }</pre>	
<pre>JournalEntry ↵ { id ObjectId > {...} date > [...] title > [...] content > [...] }</pre>	

7. Security Enforcement

The application uses Spring Security to enforce authentication and authorization.

If a request is made without a valid JWT token, the server returns:

``

403 Forbidden

``

This ensures that protected resources cannot be accessed without proper authentication.

Security implementation includes:

- * JWT Authentication
- * Stateless sessions
- * Role-based access control
- * Secure API access

Responses

Curl

```
curl -X 'GET' \
'http://localhost:8080/user' \
-H 'accept: */*'
```

Request URL

```
http://localhost:8080/user
```

Server response

Code	Details
403 <i>Undocumented</i>	Error: response status is 403

Response body

```
{  
  "timestamp": "2026-02-17T13:38:17.605+00:00",  
  "status": 403,  
  "error": "Forbidden",  
  "path": "/user"
```

Download

Response headers

```
cache-control: no-cache,no-store,max-age=0,must-revalidate  
connection: keep-alive  
content-type: application/json  
date: Tue,17 Feb 2026 13:38:17 GMT  
expires: 0  
keep-alive: timeout=60  
pragma: no-cache  
transfer-encoding: chunked  
x-content-type-options: nosniff  
x-frame-options: DENY  
x-xss-protection: 1; mode=block
```

Responses

8. Role-Based Access Control

The system implements role-based authorization.

Access levels:

Public → No authentication required

User → JWT authentication required

Admin → Admin role required

Admin endpoints include:

...

GET /Admin/all-users

POST /Admin/create-newAdmin

...

These endpoints are restricted to admin users only.

Admin APIs		^
POST	/Admin/create-newAdmin	✓ 🔒
GET	/Admin/clear-app-cache	✓ 🔒
GET	/Admin/all-users	✓ 🔒

9. Benefits of Swagger in this Project

Swagger provides several advantages:

- * Interactive API testing
- * Clear API documentation
- * Integrated authentication support
- * Schema visualization
- * Easy debugging and testing
- * Improved backend transparency

Swagger also helps recruiters and developers understand the backend system architecture easily.

10. Backend Architecture Representation

Swagger represents the Controller layer of the backend architecture:

...

Client

↓

Controller

↓

Service

↓

Repository

↓

Database (MongoDB)

...

This layered architecture ensures:

- * Scalability
- * Maintainability
- * Security
- * Clean code structure

Summary

Swagger UI provides a complete interactive interface to explore, test, and understand the Journal Application backend. It clearly documents all endpoints, enforces JWT-based authentication, displays schema definitions, and ensures secure access to protected resources. This improves developer productivity, application security, and system maintainability.
