BLOG SITE APPLICATION

IIHT

Time To Complete: 10 to 12 hr

CONTENTS

1 PROBLEM STATEMENT

Blog Site Application is a Restful Microservice application, where it allows users to search/add/delete the blogs from system for all the categories.

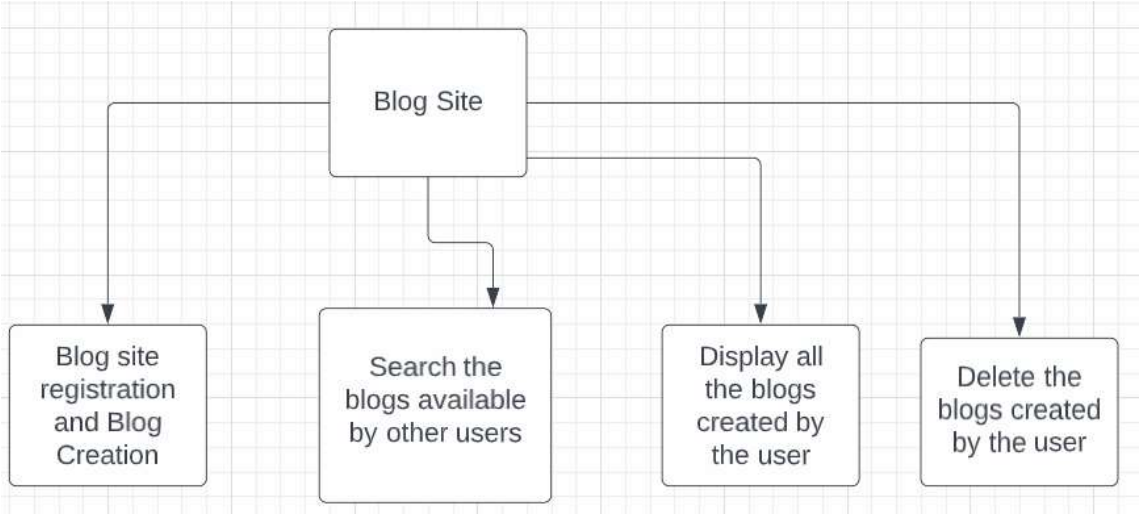
The core modules of BLOG SITE app are:

- Allows to add a new blog to the system
- Allows to delete an existing blog from the system
- Allows to search the blogs available for a particular category
- Allows to display all blogs created by the user

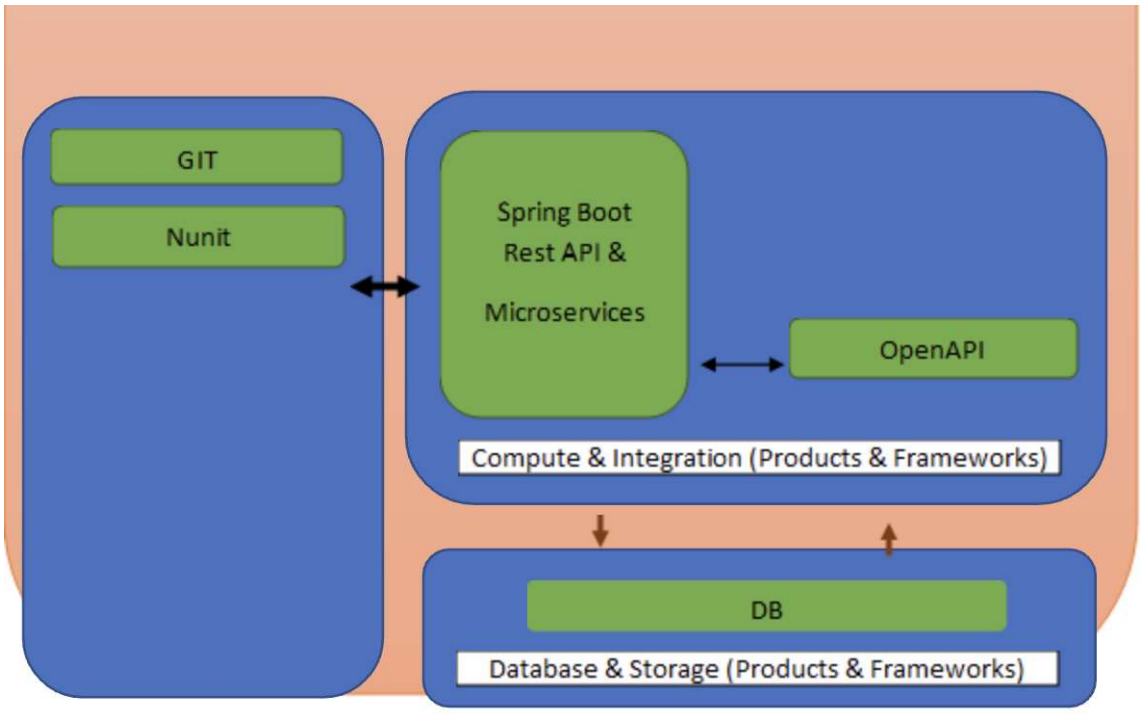
The scope includes developing the application using tool chain mentioned below.

2 PROPOSED BLOG SITE WIREFRAME

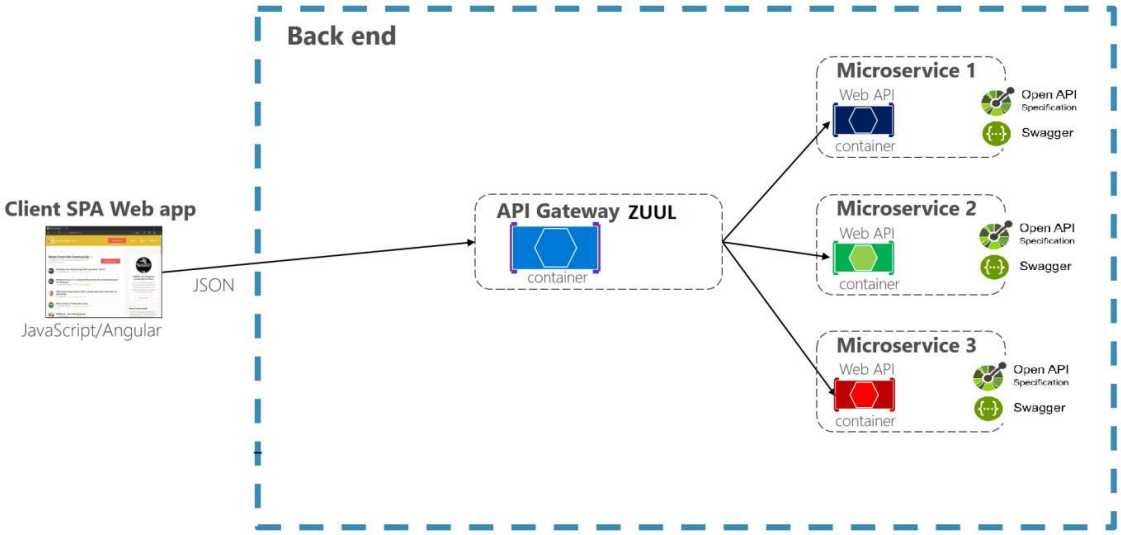
UI needs improvisation and modification as per given use case.



3 APPLICATION ARCHITECTURE



3.1 MICROSERVICE ARCHITECTURE (COMPUTE AND INTEGRATION/OPENAPI/NETWORKING AND CONTENT DELIVERY):


Back end
Microservice 1
Web API
container
Open API
Specification
Swagger
Client SPA Web app
JSON
JavaScript/Angular
API Gateway ZUUL
container
Microservice 2
Web API
container
Open API
Specification
Swagger
Microservice 3
Web API
container
Open API
Specification
Swagger

4 TOOL CHAIN

Competency	Skill	Skill Detail
Engineering Mindset	Networking and Content Delivery	
	Ways of Working	
	Consulting Mindset	
	DevOps	
Programming Languages	Application Language	Java
Products & Frameworks		
	Compute & Integration	Spring Boot
	Database & Storage	MySQL/SQL-Server
	Governance & Tooling	Git
		Junit
		Maven

5 DEVELOPMENT FLOW

No	MC	Competency	Section	Indicative Mechanism for Evaluation (Passing score of 60% in each MC)
Business Requirement				
1	**Backend**	Rest API, Database, Unit Testing, Debugging and Troubleshooting.	Click here	**Code Submission and Evaluation, Panel Presentation**

6 BUSINESS REQUIREMENTS:

As an application developer, develop microservices with below guidelines:

User Story #	User Story Name	User Story	Development
US_01	User Registration	As a user I should be able to register to the blog site application 1. As a user I am be able to furnish following details at the time of registration a. User Name b. User Email id c. Create Password Acceptance Criteria: • All Fields are mandatory • Email id should contain "@" and ".com" • Password should be Alphanumeric and at least 8 characters	Only API to be developed
US_02	Add new course	As an admin I should be able to add any new blog to the system Fields to be added: • Blog Name, Category, article, Author name, timestamp of creation Acceptance criteria: • Blog Name should be of minimum 20 characters • Category should be of minimum 20 characters • Article should be of minimum 1000 words • All the fields are mandatory • Timestamp should of current time	Only API to be developed

US_03	View and delete blogs	As a user I should be able to view and delete my blogs from the system. Acceptance criteria: • Should be able to view the blog created by the user • Should be able to delete only user from the system	Only API to be developed
US_04	View Blog Details	As a user I am able to view blogs details for any category created by other user in the same blog site Acceptance criteria: a. List the blogs based on provided category b. Along with listing the blogs, display the author details as well c. UI should provide a search button to enter the category d. UI should provide options to select the blogs to be displayed based on category and duration for blogs created within some dates	API and Frontend to be developed

7 PROPOSED REST ENDPOINTS TO BE EXPOSED

7.1 REST APIS:

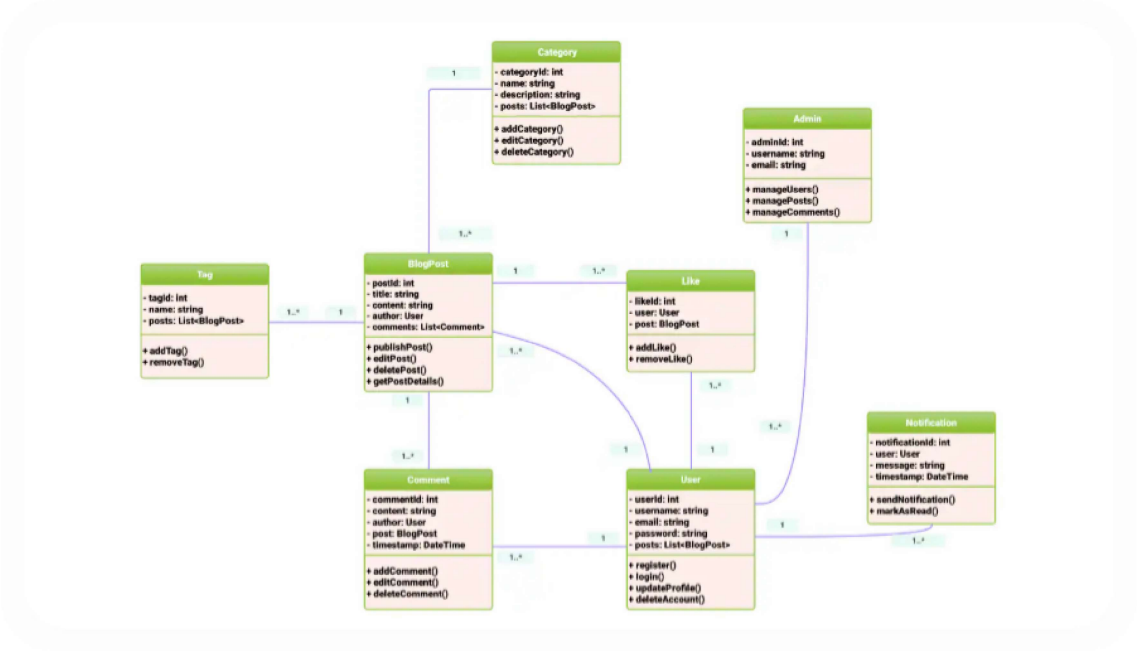
POST	**/api/v1.0/blogsite/user/register**	**Register as a new user**
GET	**/api/v1.0/blogsite/blogs/info/<category>**	**Fetches the blog based on the given category**
GET	**/api/v1.0/blogsite/user/getall**	**Fetches all the blogs of the user**
DELETE	**/api/v1.0/blogsite/user/delete/<blogname>**	**Deletes a blog**
POST	**/api/v1.0/blogsite/user/blogs/add/<blogname>**	**Add new log to the system**
GET	**/api/v/1.0/blogsite/blogs/get/<category>/ <durationToRange>/<durationToRange>**	**Fetches blogs based on the given duration**

Use Creational design pattern for composing the model object to be sent back as response on following end-point:

a.

/api/v/1.0/blogsite/blogs/get/<category>/<durationFromRange>/<durationToRange>

8 RUBRICS/EXPECTED DELIVERABLES



8.1 ENGINEERING CONCEPTS (COMPUTE & INTEGRATION):

1. As an application developer:
    a. Develop the application as a microservice architecture.
    b. Implementation as follows:
        i. Use Domain Driven Design to implement distributed architecture
        ii. Follow the Single Data Store per microservice practice
        iii. Document REST endpoints with OpenAPI.
        iv. Add CQRS pattern for Event Sourcing for saving and retrieving course details
        v. Expose all rest Endpoints using a common API Gateway.

8.2 ENGINEERING CONCEPTS (SECURITY & IDENTITY):

1. As an Application Developer:
    a. Restrict the access over all write operation (secured operations) by adding authentication
    b. Secure all Rest Endpoints by configuring SSL Certificate for Cloud

8.3 PRODUCTS & FRAMEWORK (DATABASE & STORAGE):

1. As an application developer:
    a. Implement ORM with Spring Data MongoRepository and MongoDB / JPARepository for other databases like SQL Server/MySQL. For complex and custom queries, create custom methods and use @Query, Aggregations (Aggregation Operation, Match Operation, Aggregation Results), implementation of MongoTemplateetc as necessary.

b. Use MySQL for maintaining data for at least one of the microservice
c. Introduce a backup mechanism, such that when record count crosses 10,000 rows, a backup should trigger

8.4 DEBUGGING & TROUBLESHOOTING

1. Generate bug report & error logs - Report must be linked with final deliverables which should also suggest the resolution for the encountered bugs and errors.

8.5 UNIT TESTING:

1. As an application developer:

a. Implement Unit testing using JUnit 5 for all the layers Exp: Controller layer, Service layer, and Repository layer.
b. Generate the test results with positive and negative test scripts.

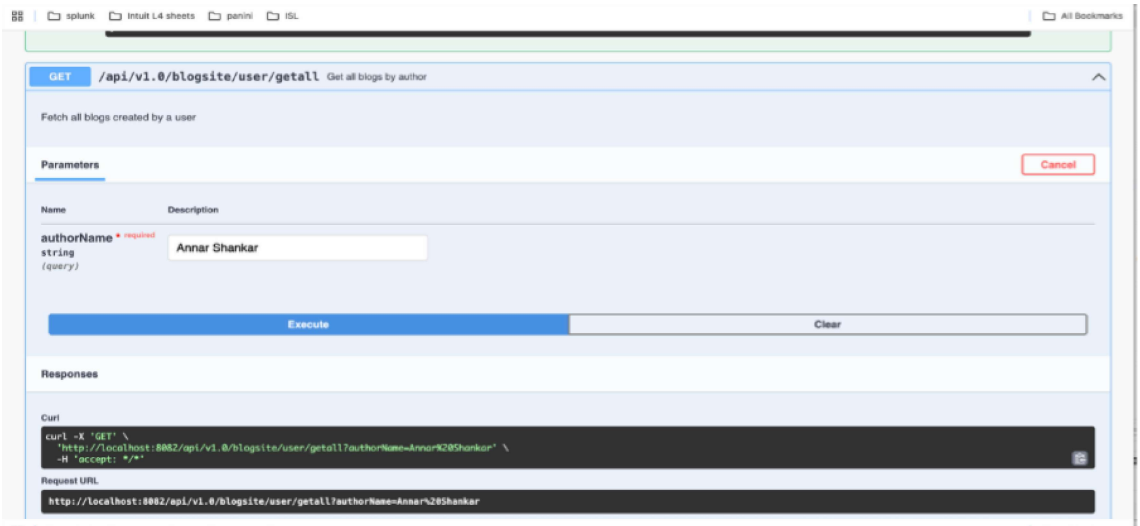
8.6 CODE QUALITY & CODE COVERAGE:

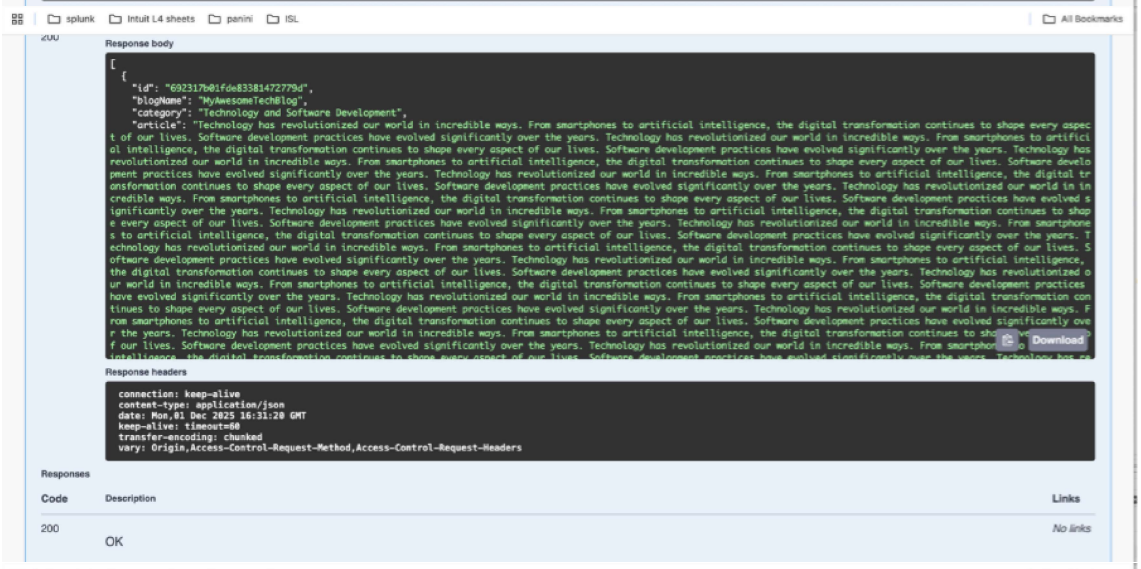
1. Ensure code quality with static analysis tools.
2. Code coverage for API should be > 80%

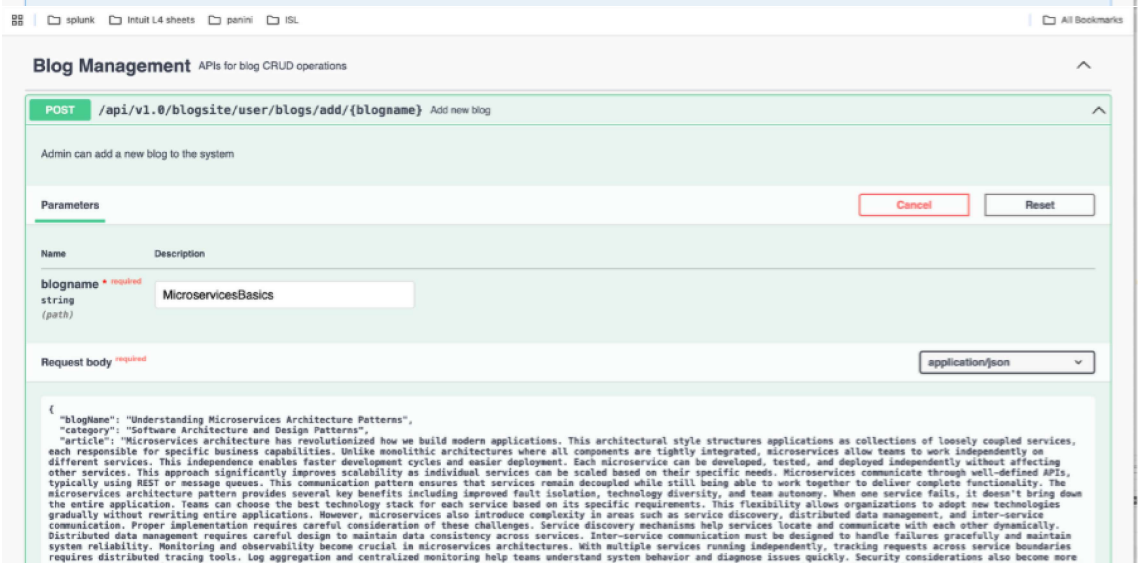
8.7 GOOD TO HAVE:

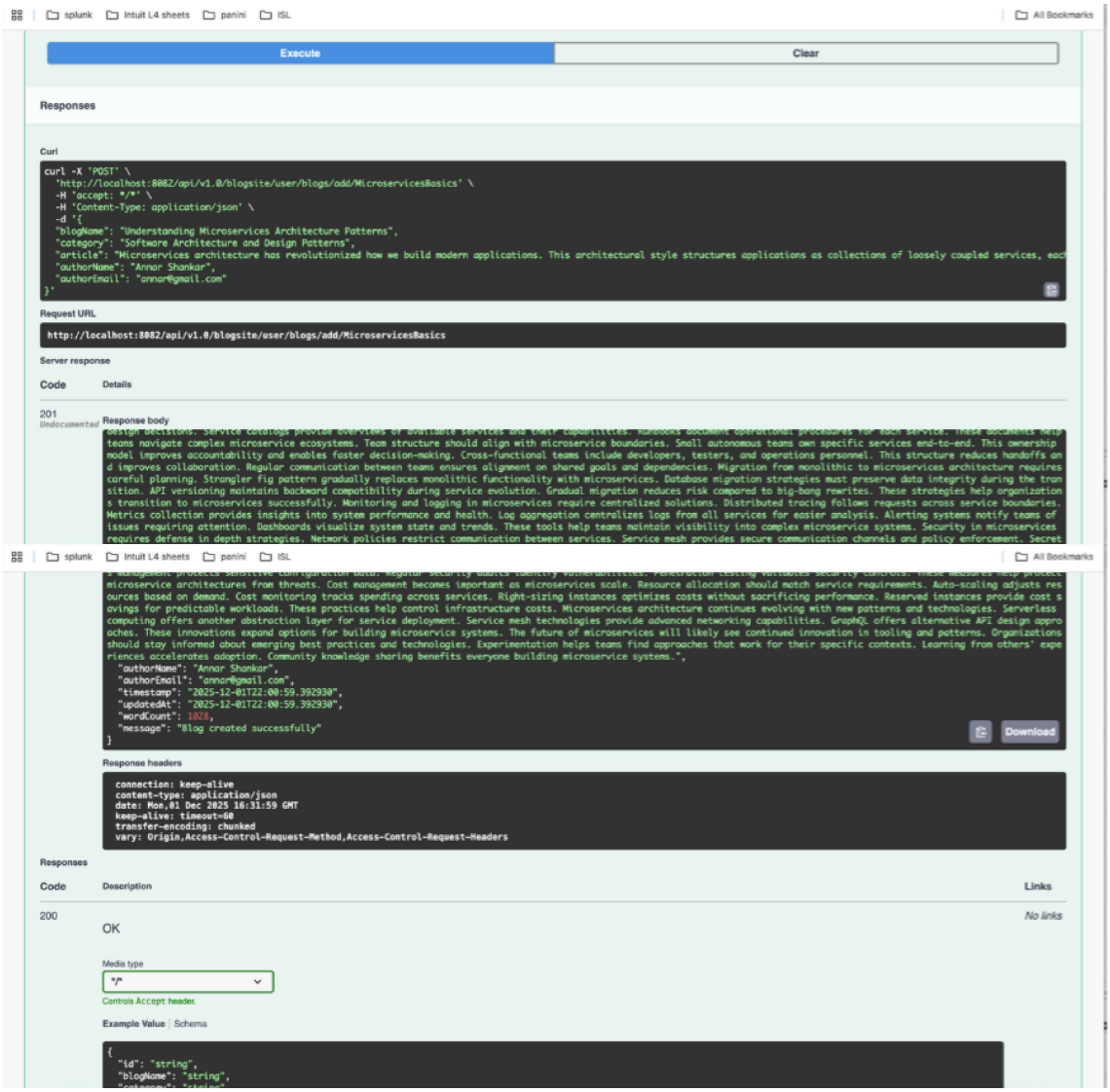
1. Implement error/exception handling mechanisms.
2. Implement logging and monitoring to detect and troubleshoot issues.
3. Implement rate limiting to prevent abuse of API endpoints.
4. Blogs should be fetched and displayed within 30 seconds.
5. Use containerization to package and deploy the application.
6. Improve maintainability with modular codebase, automated testing, CI/CD pipelines

8.8 EXPECTED OUTCOMES:

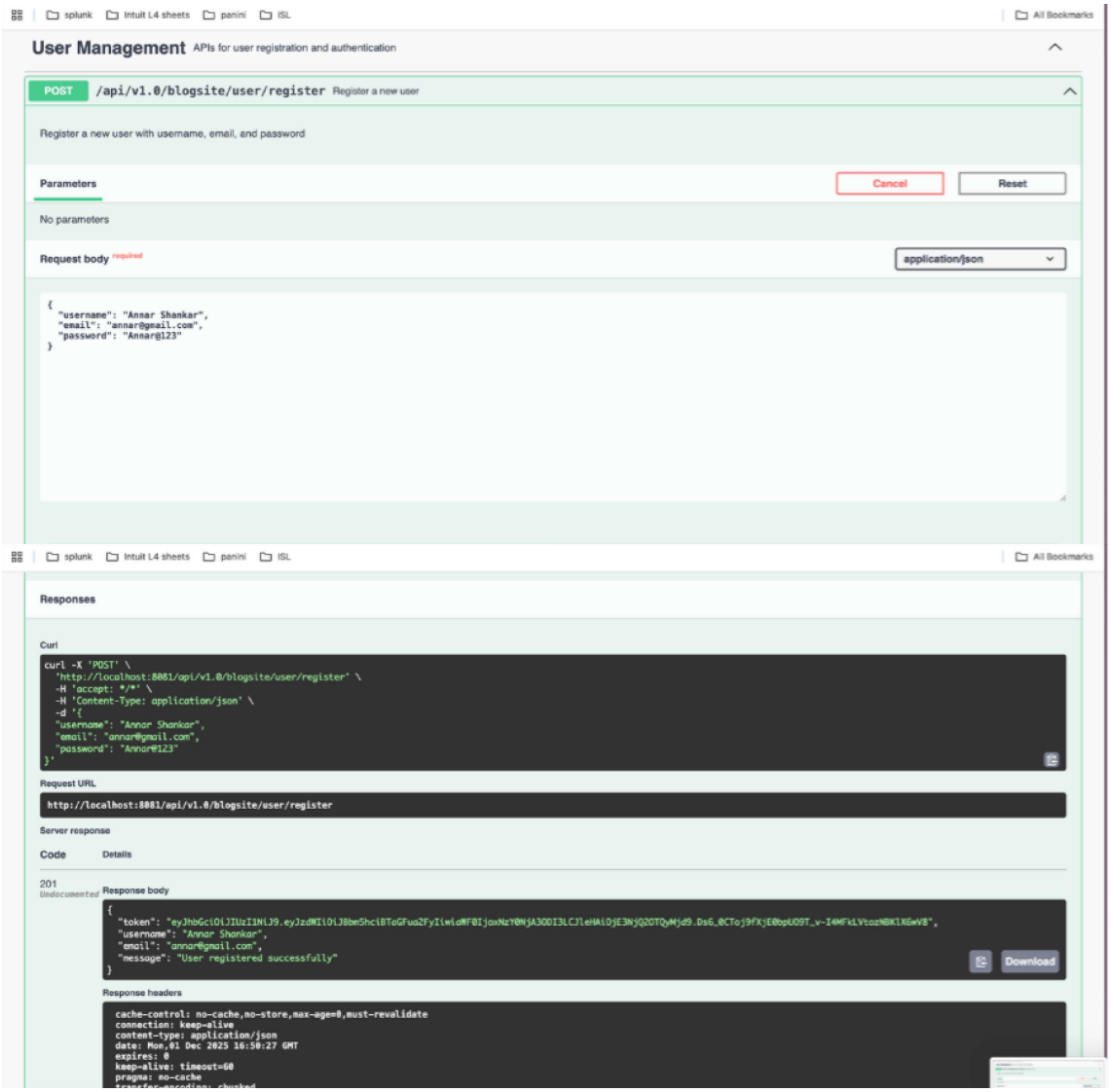

splunk
Intuit L4 sheets
panini
ISL
All Bookmarks
GET
/api/v1.0/blogsite/user/getall
Get all blogs by author
Fetch all blogs created by a user
Parameters
Cancel
Name
Description
authorName * required
string
(query)
Annar Shankar
Execute
Clear
Responses
Curl
curl -X 'GET' \
'http://localhost:8082/api/v1.0/blogsite/user/getall?authorName=Annar%20Shankar' \
-H 'accept: */*'
Request URL
http://localhost:8082/api/v1.0/blogsite/user/getall?authorName=Annar%20Shankar


splunk
Intuit L4 sheets
panini
ISL
All Bookmarks
200
Response body
[
{
"id": "692317b01fde83381472779d",
"blogName": "MyAwesomeTechBlog",
"category": "Technology and Software Development",
"article": "Technology has revolutionized our world in incredible ways. From smartphones to artificial intelligence, the digital transformation continues to shape every aspec
t of our lives. Software development practices have evolved significantly over the years. Technology has revolutionized our world in incredible ways. From smartphones to artifici
al intelligence, the digital transformation continues to shape every aspect of our lives. Software development practices have evolved significantly over the years. Technology has
revolutionized our world in incredible ways. From smartphones to artificial intelligence, the digital transformation continues to shape every aspect of our lives. Software develo
pment practices have evolved significantly over the years. Technology has revolutionized our world in incredible ways. From smartphones to artificial intelligence, the digital tr
ansformation continues to shape every aspect of our lives. Software development practices have evolved significantly over the years. Technology has revolutionized our world in in
credible ways. From smartphones to artificial intelligence, the digital transformation continues to shape every aspect of our lives. Software development practices have evolved s
ignificantly over the years. Technology has revolutionized our world in incredible ways. From smartphones to artificial intelligence, the digital transformation continues to shap
e every aspect of our lives. Software development practices have evolved significantly over the years. Technology has revolutionized our world in incredible ways. From smartphone
s to artificial intelligence, the digital transformation continues to shape every aspect of our lives. Software development practices have evolved significantly over the years. T
echnology has revolutionized our world in incredible ways. From smartphones to artificial intelligence, the digital transformation continues to shape every aspect of our lives. S
oftware development practices have evolved significantly over the years. Technology has revolutionized our world in incredible ways. From smartphones to artificial intelligence,
the digital transformation continues to shape every aspect of our lives. Software development practices have evolved significantly over the years. Technology has revolutionized o
ur world in incredible ways. From smartphones to artificial intelligence, the digital transformation continues to shape every aspect of our lives. Software development practices
have evolved significantly over the years. Technology has revolutionized our world in incredible ways. From smartphones to artificial intelligence, the digital transformation con
tinues to shape every aspect of our lives. Software development practices have evolved significantly over the years. Technology has revolutionized our world in incredible ways. F
rom smartphones to artificial intelligence, the digital transformation continues to shape every aspect of our lives. Software development practices have evolved significantly ove
r the years. Technology has revolutionized our world in incredible ways. From smartphones to artificial intelligence, the digital transformation continues to sh
f our lives. Software development practices have evolved significantly over the years. Technology has revolutionized our world in incredible ways. From smartpho
Download
Response headers
connection: keep-alive
content-type: application/json
date: Mon,01 Dec 2025 16:31:20 GMT
keep-alive: timeout=60
transfer-encoding: chunked
vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
Responses
Code
Description
Links
200
OK
No links


splunk
Intuit L4 sheets
panini
ISL
All Bookmarks
Blog Management
APIs for blog CRUD operations
POST
/api/v1.0/blogsite/user/blogs/add/{blogname}
Add new blog
Admin can add a new blog to the system
Parameters
Cancel
Reset
Name
Description
blogname * required
string
(path)
MicroservicesBasics
Request body required
application/json
{
"blogName": "Understanding Microservices Architecture Patterns",
"category": "Software Architecture and Design Patterns",
"article": "Microservices architecture has revolutionized how we build modern applications. This architectural style structures applications as collections of loosely coupled services,
each responsible for specific business capabilities. Unlike monolithic architectures where all components are tightly integrated, microservices allow teams to work independently on
different services. This independence enables faster development cycles and easier deployment. Each microservice can be developed, tested, and deployed independently without affecting
other services. This approach significantly improves scalability as individual services can be scaled based on their specific needs. Microservices communicate through well-defined APIs,
typically using REST or message queues. This communication pattern ensures that services remain decoupled while still being able to work together to deliver complete functionality. The
microservices architecture pattern provides several key benefits including improved fault isolation, technology diversity, and team autonomy. When one service fails, it doesn't bring down
the entire application. Teams can choose the best technology stack for each service based on its specific requirements. This flexibility allows organizations to adopt new technologies
gradually without rewriting entire applications. However, microservices also introduce complexity in areas such as service discovery, distributed data management, and inter-service
communication. Proper implementation requires careful consideration of these challenges. Service discovery mechanisms help services locate and communicate with each other dynamically.
Distributed data management requires careful design to maintain data consistency across services. Inter-service communication must be designed to handle failures gracefully and maintain
system reliability. Monitoring and observability become crucial in microservices architectures. With multiple services running independently, tracking requests across service boundaries
requires distributed tracing tools. Log aggregation and centralized monitoring help teams understand system behavior and diagnose issues quickly. Security considerations also become more


splunk
Intuit L4 sheets
panini
ISL
All Bookmarks
Execute
Clear
Responses
Curl
curl -X 'POST' \
'http://localhost:8082/api/v1.0/blogsite/user/blogs/add/MicroservicesBasics' \
-H 'accept: */*' \
-H 'Content-Type: application/json' \
-d '{
"blogName": "Understanding Microservices Architecture Patterns",
"category": "Software Architecture and Design Patterns",
"article": "Microservices architecture has revolutionized how we build modern applications. This architectural style structures applications as collections of loosely coupled services, eac
"authorName": "Annar Shankar",
"authorEmail": "annar@gmail.com"
}'
Request URL
http://localhost:8082/api/v1.0/blogsite/user/blogs/add/MicroservicesBasics
Server response
Code
Details
201
Undocumented
Response body
design decisions. Service catalogs provide overviews of available services and their capabilities. Runbooks document operational procedures for each service. These documents help teams navigate complex microservice ecosystems. Team structure should align with microservice boundaries. Small autonomous teams own specific services end-to-end. This ownership model improves accountability and enables faster decision-making. Cross-functional teams include developers, testers, and operations personnel. This structure reduces handoffs an d improves collaboration. Regular communication between teams ensures alignment on shared goals and dependencies. Migration from monolithic to microservices architecture requires careful planning. Strangler fig pattern gradually replaces monolithic functionality with microservices. Database migration strategies must preserve data integrity during the tran sition. API versioning maintains backward compatibility during service evolution. Gradual migration reduces risk compared to big-bang rewrites. These strategies help organization s transition to microservices successfully. Monitoring and logging in microservices require centralized solutions. Distributed tracing follows requests across service boundaries. Metrics collection provides insights into system performance and health. Log aggregation centralizes logs from all services for easier analysis. Alerting systems notify teams of issues requiring attention. Dashboards visualize system state and trends. These tools help teams maintain visibility into complex microservice systems. Security in microservices requires defense in depth strategies. Network policies restrict communication between services. Service mesh provides secure communication channels and policy enforcement. Secret
splunk
Intuit L4 sheets
panini
ISL
All Bookmarks
microservice architectures from threats. Cost management becomes important as microservices scale. Resource allocation should match service requirements. Auto-scaling adjusts res ources based on demand. Cost monitoring tracks spending across services. Right-sizing instances optimizes costs without sacrificing performance. Reserved instances provide cost s avings for predictable workloads. These practices help control infrastructure costs. Microservices architecture continues evolving with new patterns and technologies. Serverless computing offers another abstraction layer for service deployment. Service mesh technologies provide advanced networking capabilities. GraphQL offers alternative API design appro aches. These innovations expand options for building microservice systems. The future of microservices will likely see continued innovation in tooling and patterns. Organizations should stay informed about emerging best practices and technologies. Experimentation helps teams find approaches that work for their specific contexts. Learning from others' expe riences accelerates adoption. Community knowledge sharing benefits everyone building microservice systems.",
"authorName": "Annar Shankar",
"authorEmail": "annar@gmail.com",
"timestamp": "2025-12-01T22:00:59.392930",
"updatedAt": "2025-12-01T22:00:59.392930",
"wordCount": 1028,
"message": "Blog created successfully"
}
Download
Response headers
connection: keep-alive
content-type: application/json
date: Mon,01 Dec 2025 16:31:59 GMT
keep-alive: timeout=60
transfer-encoding: chunked
vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
Responses
Code
Description
Links
200
OK
No links
Media type
/
Controls Accept header.
Example Value | Schema
{
"id": "string",
"blogName": "string",


splunk
Intuit L4 sheets
panini
ISL
All Bookmarks
DELETE
/api/v1.0/blogsite/user/delete/{blogname}
Delete blog
User can delete their own blog
Parameters
Cancel
Name
Description
blogname * required
string
(path)
MyAwesomeTechBlog
authorName * required
string
(query)
Annar Shankar
Execute
Clear
Responses
Curl
curl -X 'DELETE' \
'http://localhost:8082/api/v1.0/blogsite/user/delete/MyAwesomeTechBlog?authorName=Annar%20Shankar' \
-H 'accept: */*'
Request URL
http://localhost:8082/api/v1.0/blogsite/user/delete/MyAwesomeTechBlog?authorName=Annar%20Shankar
Server response
Code
Details
200
Response body
"blogName": "MyAwesomeTechBlog",
"category": "Technology and Software Development",
"authorName": "Annar Shankar",
"authorEmail": "annar@example.com",
"timestamp": "2025-11-23T19:48:24.351",
"updatedAt": "2025-11-23T19:48:24.351",
"wordCount": 3300,
"message": "Blog deleted successfully"
}
Download
Response headers
connection: keep-alive
content-type: application/json
date: Mon,01 Dec 2025 16:34:35 GMT
keep-alive: timeout=60
transfer-encoding: chunked
vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
Responses
Code
Description
Links
200
OK


splunk
Intuit L4 sheets
panini
ISL
All Bookmarks
User Management APIs for user registration and authentication
POST /api/v1.0/blogsite/user/register Register a new user
Register a new user with username, email, and password
Parameters
Cancel
Reset
No parameters
Request body required
application/json
{
"username": "Annar Shankar",
"email": "annar@gmail.com",
"password": "Annar@123"
}
Responses
Curl
curl -X 'POST' \
'http://localhost:8081/api/v1.0/blogsite/user/register' \
-H 'accept: */*' \
-H 'Content-Type: application/json' \
-d '{
"username": "Annar Shankar",
"email": "annar@gmail.com",
"password": "Annar@123"
}'
Request URL
http://localhost:8081/api/v1.0/blogsite/user/register
Server response
Code
Details
201
Undocumented
Response body
{
"token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJBbmShciBTaGFua2FyIiwidWF0IjoxNzY0NjA3ODI3LCJleHAiOjE3NjQ2OTQyMjd9.Ds6_0CToj9fXjE0bpUO9T_v-I4MFkLVtozNBKlXGwV8",
"username": "Annar Shankar",
"email": "annar@gmail.com",
"message": "User registered successfully"
}
Download
Response headers
cache-control: no-cache,no-store,max-age=0,must-revalidate
connection: keep-alive
content-type: application/json
date: Mon,01 Dec 2025 16:50:27 GMT
expires: 0
keep-alive: timeout=60
pragma: no-cache