COMPONENT-2 RUBRICS/EXPECTED DELIVERABLES

1.1 REST API (PRODUCTS & FRAMEWORKS -> COMPUTE & INTEGRATION):

a. Use Spring Boot to version and implement the REST endpoints.

b. Implement HTTP methods like GET, POST, PUT, DELETE, PATCH to implement RESTful resources:

POST	**/api/v1.0/tweets/register**	**Register as new user**
GET	**/api/v1.0/tweets/login**	**Login**
GET	**/api/v1.0/tweets/<username>/forgot**	**Forgot password**
GET	**/api/v1.0/tweets/all**	**Get all tweets**
GET	**/api/v1.0/tweets/users/all**	**Get all users**
GET	**/api/v/1.0/tweets/user/search/username***	**Search by username**
GET	**/api/v1.0/tweets/username**	**Get all tweets of user**
POST	**/api/v1.0/tweets/<username>/add**	**Post new tweet**
PUT	**/api/v1.0/tweets/<username>/update/<id>**	**Update tweet**
DELETE	**/api/v1.0/tweets/<username>/delete/<id>**	**Delete tweet**
PUT	**/api/v1.0/tweets/<username>/like/<id>**	**Like tweet**
POST	**/api/v1.0/tweets/<username>/reply/<id>**	**Reply to tweet**

c. ***username may be partial or complete username**

d. Use necessary configuration in place for REST API in application.properties or bootstrap.properties or application.yml; whichever is applicable.

e. Package Structure for Spring Boot Project will be like com.tweetapp.* with proper naming conventions for package and beans.

f. Use configuration class annotated with @Configuration and @Service for business layer.

g. Use constructor-based dependency injection in few classes and setter-based dependency injection in few classes.

h. Follow Spring Bean Naming Conventions

1.2 DATABASE (PRODUCTS & FRAMEWORKS -> DATABASE & STORAGE):

1. As an application developer:
   a. Implement ORM with Spring Data MongoRepository and MongoDB. For complex and custom queries, create custom methods and use @Query, Aggregations (AggregationOperation, MatchOperation, AggregationResults), implementation of MongoTemplate etc as necessary.
   b. Have necessary configuration in place for REST API in application.properties or bootstrap.properties or application.yml OR Java based configuration; whichever is applicable.

1.3 API Documentation (Products & Frameworks -> Compute & Integration):

1. As an application developer:
    a. Document REST endpoints with OpenAPI or Swagger

1.4 Messaging (Products & Frameworks -> Compute & Integration):

1. As an application developer:
    a. Have a centralized logging system
    b. Be able to communicate using a messaging infrastructure.
    c. Use KafkaTemplate for communication with Springboot and topics in Kafka/RabbitMQ/ActiveMQ/KubeMQ.
    d. Use Kafka/RabbitMQ/ActiveMQ/KubeMQ for messaging infrastructure and implement producers to write messages/tweets to topic and consumers to read messages/tweets from topic.
    e. Configure Springboot app to log all logging messages to Kafka/RabbitMQ/ActiveMQ/KubeMQ.
    f. Configure all Kafka/RabbitMQ/ActiveMQ/KubeMQ related configuration needed for Spring Boot in *.properties or *.yml file.

1.5 Log/ Monitoring (Products & Frameworks -> Governance & Tooling):

1. As an application developer:
    a. Containerize the complete application, which includes front-end, middleware and Kafka/RabbitMQ/ActiveMQ/KubeMQ (consumers and producers) using docker and Dockerfile.
    b. Use .dockerignore as necessary to avoid containerizing un-necessary packages.
    c. Integrate Spring Boot Actuator with Prometheus and Grafana to monitor middleware.
    d. Implement logs with logstash.
    e. Open the preconfigured Logstash in Kibana and check if it successfully connect to Elasticsearch Server.

1.6 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.