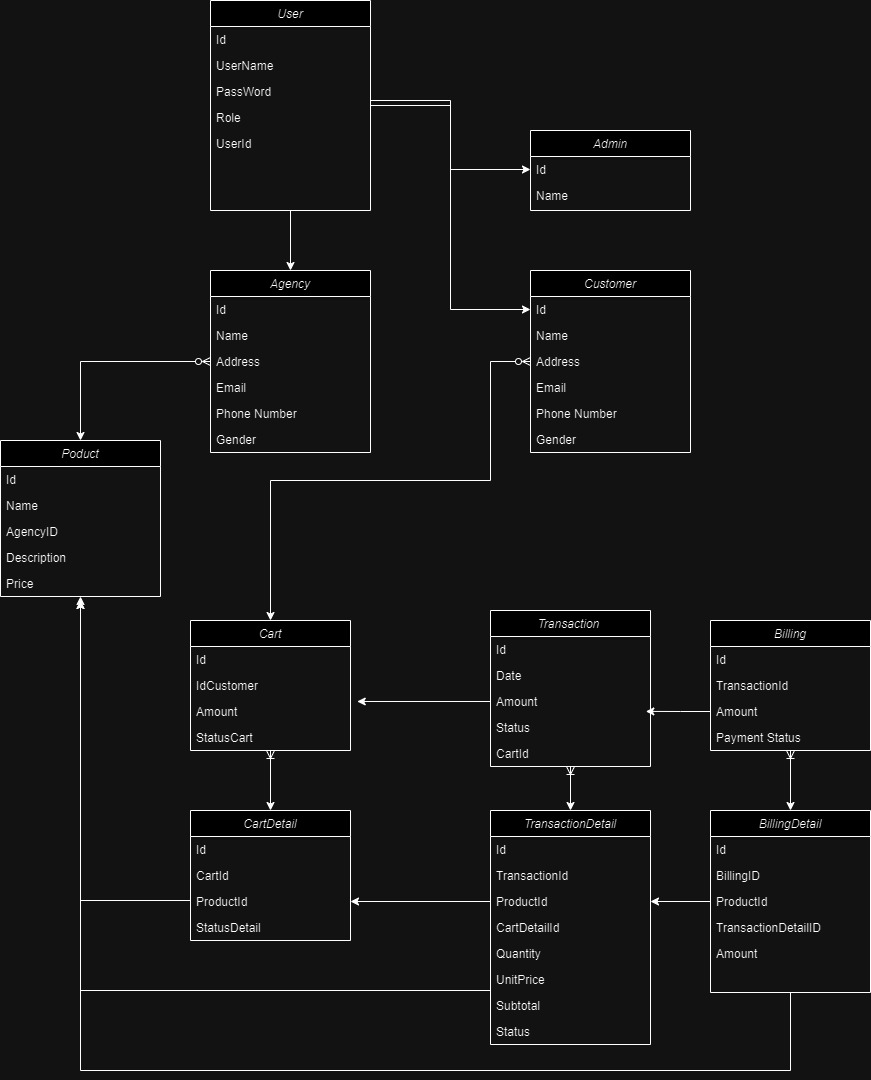
Q2

  
  
  
Q3  
1. I use MongoDb for my system

2.

I use it because my database during development can constantly change and if I continue with the project I can quickly expand in the direction I want.

**Benefits of Using MongoDB (NoSQL):**

* Schema Flexibility: MongoDB is schema-less, offering flexibility in handling evolving data models.
* Horizontal Scalability: MongoDB supports horizontal scaling for improved performance and handling large data volumes.
* JSON-Like Documents: MongoDB stores data in a flexible, JSON-like BSON format, making it developer-friendly.
* Rich Query Language: MongoDB provides a powerful query language with support for various types of queries and indexing.
* Geospatial Capabilities: MongoDB has built-in support for geospatial data and queries, suitable for location-based applications.

**Drawbacks of Using MongoDB (NoSQL):**

* Eventual Consistency: MongoDB follows an eventual consistency model, where changes may take time to propagate across nodes.
* Learning Curve: Adapting to MongoDB's document-oriented approach may pose a learning curve for users familiar with relational databases.
* Transaction Limitations: MongoDB supports transactions within a single document, limiting complex transactions.
* Data Size and Storage Overhead: MongoDB's BSON format may result in larger storage requirements compared to more compact binary formats.
* No Built-in Joins: MongoDB lacks support for traditional SQL-style joins, requiring careful data model design.

3 . The file is attached to the API project

A black screen with orange text

Description automatically generated

Q4

1. There will be an FE, but I will temporarily ignore it because I don't deal with it too much in the authentication role.

A screenshot of a computer

Description automatically generated

3.

Some database relationships are complicated

I will merge a table into 1 and delegate permissions to them

4.

1. Define Permission Structure:

Define roles (e.g., Admin, User, Guest) and specific permissions associated with each role.

Assign permissions to APIs based on their functionality.

2. Automated Testing with Postman:

Use Postman for API testing.

Create collections representing different APIs and endpoints.

Write test scripts within Postman to cover various scenarios:

Test user authentication and token generation.

Test API endpoints for different roles and permissions.

Check for proper error responses for unauthorized requests.

3. Data Generation:

Develop scripts to generate test data that represents a variety of scenarios, including edge cases.

Populate databases with test data before running tests.

4. Role-Based Access Control (RBAC):

Implement RBAC in your application to manage roles and permissions.

Ensure that roles are assigned correctly during user creation or updates.

5. Test Environments:

Set up test environments mirroring production, including databases, authentication systems, and configurations.

Use separate environments for development, testing, and staging.

6. Scalability Testing:

Gradually increase the number of APIs and users in your tests.

Monitor system performance and response times.

Ensure that the solution scales horizontally to handle increased load.

7. Use Mocking for External Services:

For APIs depending on external services, use tools like WireMock or Postman Mock Servers.

Create mocks that simulate responses from external services.

8. Continuous Integration (CI):

Integrate Postman tests into your CI/CD pipeline using tools like Jenkins, Travis CI, or GitLab CI.

Trigger automated tests on every code commit or pull request.

9. Performance Testing:

Implement performance tests using tools like Apache JMeter or k6.

Simulate high loads and verify that the system maintains correct permissions under stress.

10. Security Testing:

Integrate security testing tools such as OWASP ZAP or SonarQube into your pipeline.

Identify and address security vulnerabilities related to permissions.

11. Monitoring and Logging:

Implement logging to record user activities and permissions checks.

Set up monitoring tools to detect anomalies in user behavior or permission-related issues.

12. Documentation:

Document the entire testing process, including test cases, environments, and scripts.

Provide clear instructions for running tests and interpreting results.

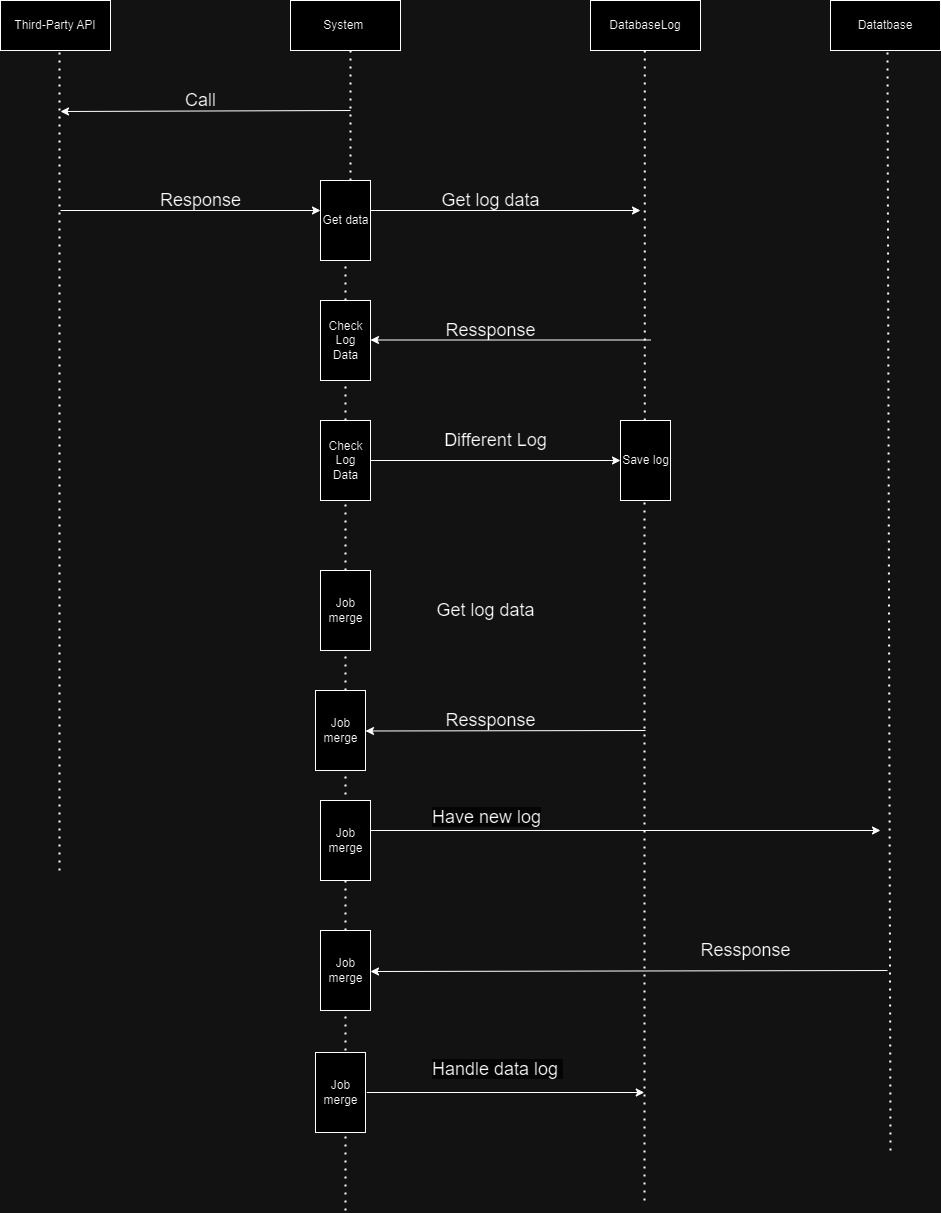
13. Regular Review and Update:

Periodically review and update test cases to accommodate changes in APIs, permissions, or business requirements.

Keep the testing suite aligned with the evolving nature of the application.

Q5

1



2.

The strength is that it can reduce the pressure on the server at a time, save past data if needed, and ensure data integrity and optimization in real time.  
In some cases where data is needed, it causes an overload on the system because it has to access both ends.

To solve the problem, we can upgrade the system hardware or build our system to perform data consolidation, providing rules on performing data consolidation in cases  
  
Q6  
