**Architecture:**

The socialmedia microservices follows a modular architecture to achieve scalability and advanced features. Key components include:

* **Microservices:**
  + Entities such as User, Post, Comment, Friendship, and others form the core microservices.
  + Efficient communication is achieved through WebClient, facilitating seamless inter-service communication.
  + Microservices are registered with the Eureka server for efficient service discovery.
* **API Gateway:**
  + Implemented using Spring Cloud Gateway to streamline communication between microservices.
  + Routes requests intelligently to the appropriate microservices based on their functionality.
* **Security:**
  + Basic security mechanisms are in place for safeguarding data modification endpoints.
  + Keycloak is seamlessly integrated for robust Identity Access Management (IAM), enhancing overall security.
* **Resilience & Monitoring:**
  + Resilience4J's Circuit Breaker pattern is employed to ensure graceful degradation during service failures, enhancing overall system resilience.
  + Spring Actuator is incorporated for monitoring the health and performance of microservices.
  + Zipkin is utilized for distributed tracing, offering insights into the flow of requests across microservices.

**Challenges Faced:**

1. **Inter-Service Communication:**
   * Managing cascading failures during microservices communication presented challenges.
   * Ensuring data consistency across services demanded careful coordination.
2. **Security Integration:**
   * Configuring Keycloak to seamlessly integrate with SpringSecurity posed challenges.
   * Balancing security measures without compromising overall system performance required careful consideration.
3. **Fault Tolerance:**
   * Fine-tuning Resilience4J parameters for optimal fault tolerance presented challenges.
   * Addressing issues related to service degradation during peak loads required focused attention.

**Lessons Learned:**

1. **Clear API Contracts:**
   * Importance of clear API contracts became evident for effective communication between microservices.
   * Balancing modularity and service independence played a crucial role in maintaining a flexible architecture.
2. **Security Best Practices:**
   * Integrating IAM is crucial for establishing a secure microservices architecture.
   * Adhering to OAuth 2.0 standards ensures a robust and standardized security framework.
3. **Resilience in Distributed Systems:**
   * Implementing the Circuit Breaker pattern emerged as a key strategy for enhancing system reliability.
   * Monitoring and tracing became vital components for identifying bottlenecks and optimizing system performance.
4. **Microservices Independence:**
   * Emphasis on designing microservices with a high level of independence to avoid unnecessary dependencies.
   * Ensuring that changes in one microservice do not adversely affect others.
5. **Load Testing and Scalability:**
   * Conducting effective load testing to identify system limitations and areas for scalability improvement.
   * Strategies for scaling microservices horizontally to handle increased load and ensure optimal performance.