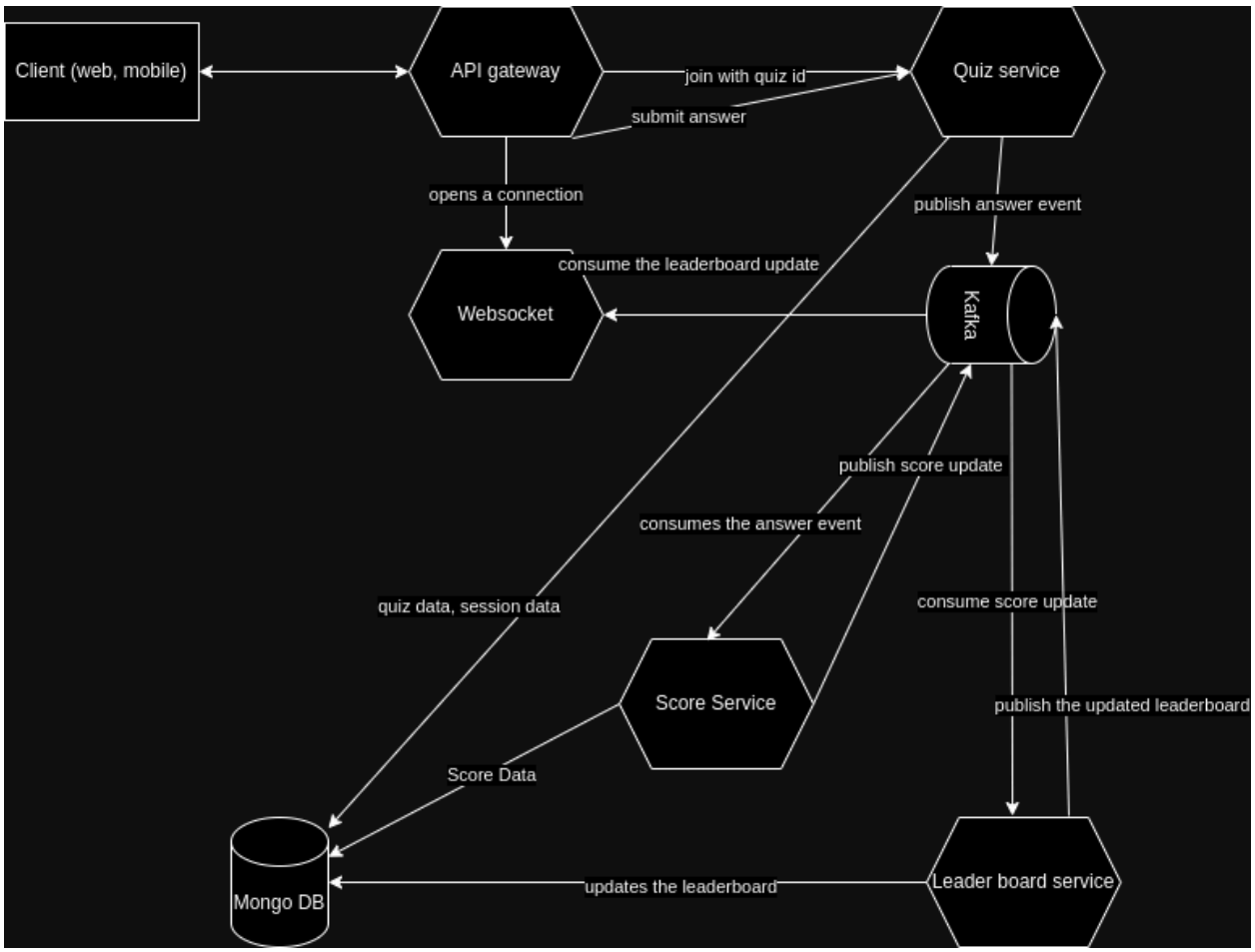
System Design

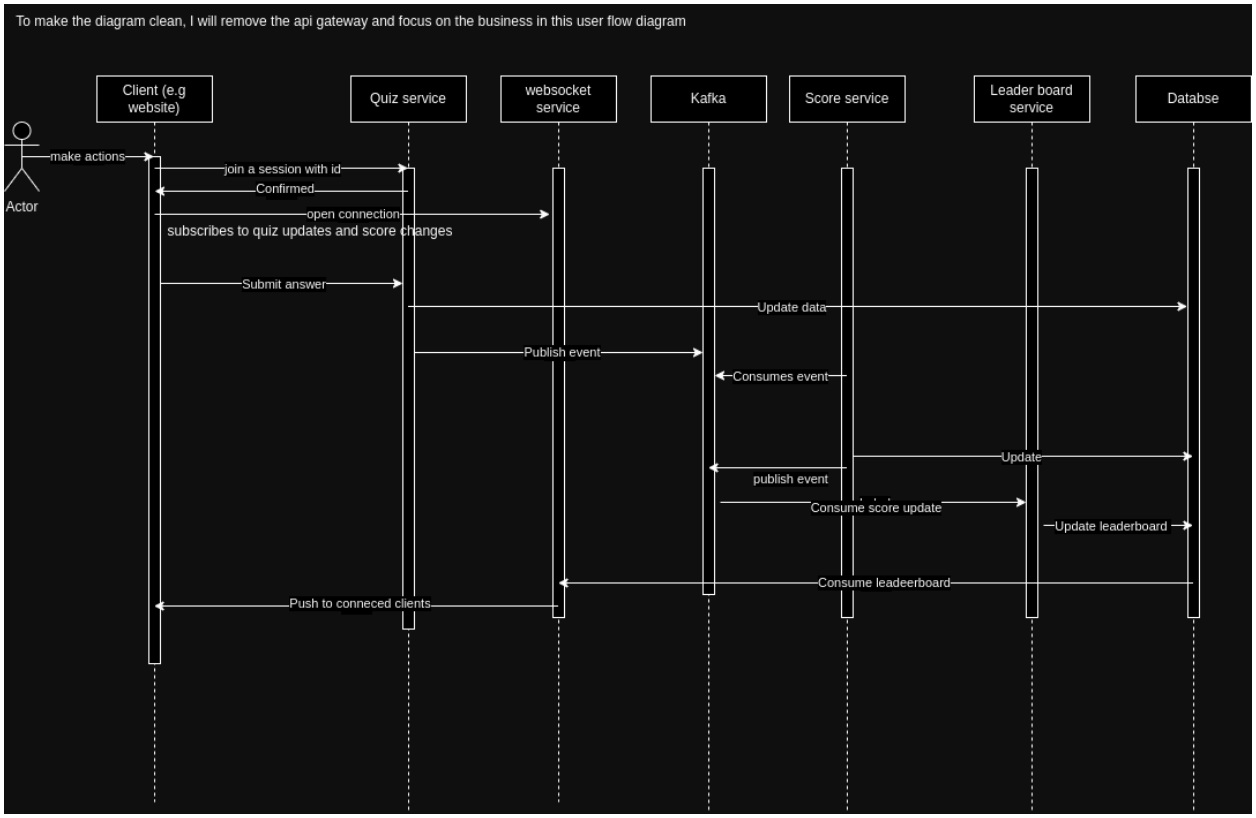
1. Architecture Overview Diagram



2. Component Description

- **Client Application**: Users can join quizzes, submit answers, and view the leaderboard.
- **Server-side services (Quiz, Leaderboard, and Score)**: Handles business logic, processes quiz answers, updates scores, and manages quiz sessions.
- **Database (MongoDB)**: Persists data related to users, quizzes, and scores.
- **Apache Kafka**: Asynchronous communication between services.
- **WebSocket Service**: Ensures real-time updates of scores and leaderboard.
- **External Services**: Provides additional functionalities like authentication, monitoring, etc.

3. Data Flow



1. **User Joins Quiz**:
    - A user sends a request to the Quiz Service to join a quiz using a unique quiz ID.
    - The Quiz Service verifies the quiz ID and responds to the client confirming the user's participation.
2. **Real-time Quiz Participation**:
    - The client opens a WebSocket connection to the WebSocket Service.
    - The service keeps track of connected users in Redis
    - The client subscribes to quiz updates and score changes via this connection.
3. **User Submits Answer**:
    - The client sends the user's answer to the Quiz Service via an HTTP request.
    - The Quiz Service publishes the answer event to Kafka.
    - Keep track of persistent data in the database
4. **Score Update**:
    - The Score Service consumes the answer event from Kafka.
    - The Score Service processes the answer, updates the score in the MongoDB database, and publishes a score update event to Kafka.
5. **Leaderboard Update**:
    - The Leaderboard Service consumes the score update event from Kafka.

- The Leaderboard Service retrieves the latest scores and updates the leaderboard in the MongoDB database.
- The Leaderboard Service publishes the updated leaderboard event to Kafka.

6. **Real-time Leaderboard**:
    - The WebSocket Service consumes the leaderboard update event from Kafka.
    - The WebSocket Service broadcasts the updated leaderboard to all connected clients (get from Redis)

4. Technologies and Tools

1. **Service side (Leader board, Score, Quiz, and WebSocket Service): Scala + Akka HTTP**
    a. **Real-Time Communication**: WebSockets are essential for real-time bidirectional communication, allowing the server to push updates to the client as soon as they happen.
    b. **Scalability**: Akka HTTP, a highly performant library for building HTTP-based services in Scala, integrates well with Akka Streams, making it a good choice for handling numerous WebSocket connections efficiently.
    c. **Concurrency Handling**: Akka HTTP leverages the actor model provided by Akka, which is ideal for managing concurrent connections and messages without the complexity of traditional thread-based approaches.
    d. **Performance**: Scala and Akka are well known for high-performance
    e. **Reliability**: A strong-type, functional language is very reliable.
    f. **Last, the ecosystem:** The JVM stack is mature, rich, and stable, especially back-compatibility when upgraded so maintainability is easy and quite cheap.
2. **Client-side technologies**
    a. Web: any framework (Angular) or library (vuejs, react) will be ok
    b. Mobile: native android and iOS
3. **Apache Kafka as the backbone for service communication**
    a. Decoupling Services
    b. Scalability
    c. Fault Tolerance
4. **Database (MongoDB):** Persists data related to users, quizzes, and scores with high-performance, easy to scale with heavily loaded.
5. **Caching mechanism**: Redis
6. **Authentication mechanism:** Any 3rd party tool will be fine, e.g Auth0
7. **Monitoring and Observability (Prometheus and Grafana)**
    a. **Metrics Collection**: Prometheus is a powerful monitoring and alerting toolkit that can collect metrics from your application, including custom metrics from your Scala services.

    b. **Alerting**: Prometheus integrates well with alerting systems, allowing you to set up alerts based on specific thresholds or anomalies in your application's performance.
    c. **Visualization**: Grafana is a popular open-source platform for monitoring and observability that works seamlessly with Prometheus. It provides customizable dashboards to visualize metrics in real time.
    d. **Scalability**: Both Prometheus and Grafana are designed to handle high volumes of metrics and data, making them suitable for applications with significant load.
8. **Logging (ELK Stack: Elasticsearch, Logstash, Kibana)**
    a. **Centralized Logging**: The ELK stack allows you to collect, process, and visualize logs from various sources in one centralized location.
    b. **Search and Analysis**: Elasticsearch provides powerful full-text search capabilities, making it easy to query and analyze logs.
    c. **Data Processing**: Logstash allows you to ingest, transform, and ship logs from multiple sources to Elasticsearch.
    d. **Visualization**: Kibana offers rich visualization options, enabling you to create dashboards that provide insights into your application's logs.
    e. **Scalability**: The ELK stack can handle large volumes of log data, making it suitable for applications with extensive logging requirements.
9. **Containerization and Orchestration (Docker and Kubernetes)**
    a. **Consistency**: Docker ensures that your application runs consistently across different environments by packaging it with all its dependencies.
    b. **Isolation**: Docker containers provide process isolation, improving security and resource management.
    c. **Scalability**: Kubernetes automates the deployment, scaling, and management of containerized applications. It helps ensure the high availability and resilience of your services.
    d. **Orchestration**: Kubernetes provides advanced orchestration features, such as automated rollouts and rollbacks, service discovery, and load balancing, which are essential for managing microservices in production.