Project Name: Unified GraphQL Data Platform (UGDP)

Project Objectives:

1. **Build a modular data access layer**: Develop a GraphQL API that connects various backend services while ensuring high performance, reliability, and modularity.
2. **Schema management**: Create a global schema that enables efficient data access and intuitive modeling for developers.
3. **Tenant experience improvements**: Design tooling and observability features to allow tenant teams to optimize performance and cost autonomously.
4. **Enable observability and monitoring**: Include logging, metrics, and tracing to support debugging and performance tuning.
5. **Scalability and extensibility**: Design an architecture that allows for easy upgrades and extensions.

Key Features:

1. **GraphQL API**:
    - A unified schema supporting multiple data sources like relational databases, NoSQL stores, and external APIs.
    - Query optimization for performance and cost efficiency.
    - Support for dynamic schema extensions.
2. **Observability**:
    - Integrated tools for performance monitoring (e.g., Micrometer, OpenTelemetry).
    - Dashboard for tracking API usage, query performance, and error rates.
3. **Tenant Developer Tools**:
    - Schema validation tools to detect inefficiencies during development.
    - Code templates and libraries for easier onboarding.
    - Self-service cost and performance analysis tools.
4. **Performance Optimization**:
    - Caching mechanisms for frequently accessed data.
    - Support for asynchronous and batched queries.
5. **Scalable Architecture**:
    - Built with **Kotlin** for backend development.
    - Modular microservice architecture with containerization (Docker) and orchestration (Kubernetes).
    - CI/CD pipelines for automated deployments.

Tech Stack:

1. **Backend**:
    - Language: **Kotlin**
    - Framework: **Spring Boot** for GraphQL APIs
    - Data Sources: MySQL, PostgreSQL, MongoDB
    - Query Language: GraphQL
2. **Observability**:
    - **OpenTelemetry** for tracing.
    - **Prometheus** and **Grafana** for metrics and visualization.
    - **ELK Stack** (Elasticsearch, Logstash, Kibana) for centralized logging.
3. **Deployment**:
    - Containerization: **Docker**
    - Orchestration: **Kubernetes**
    - CI/CD: **GitLab** pipelines or **Jenkins**
4. **Front-End (Optional)**:
    - Client: A lightweight React dashboard for tenant developers to manage schemas and view analytics.

Example Work Breakdown:

1. **Backend Development**:
    - Implement GraphQL schema with resolvers for key queries and mutations.
    - Integrate multiple data sources and enable caching.
2. **Observability**:
    - Add tracing, metrics, and logging in critical parts of the application.
    - Build performance dashboards.
3. **Tooling**:
    - Create developer utilities for schema validation.
    - Provide sample code and documentation for tenant teams.
4. **Testing & QA**:
    - Unit tests and integration tests for GraphQL queries.
    - Load testing to measure performance under heavy usage.

Deliverables:

1. A working GraphQL-based backend service with an extensible schema.
2. Observability dashboards displaying key metrics like query latencies, error rates, and request volumes.
3. Tenant-facing tools for schema validation and cost/performance analysis.
4. Documentation detailing the architecture, setup instructions, and usage guidelines.

Extensions for Differentiation:

- **AI/ML Integration**: Use AI to suggest optimizations for queries and schemas based on historical data.
- **Dynamic Authorization**: Implement a rule-based system to control access to sensitive data dynamically.

Phase 1: Project Initialization

1. Define Project Requirements:

- Gather functional and non-functional requirements, including scalability, modularity, and observability needs.
- Define the scope of the GraphQL API, data sources, and observability tools.

2. Set Up Development Environment:

- Install required tools: Kotlin, IntelliJ IDEA, Docker, Kubernetes, and Prometheus.
- Set up version control: Initialize a Git repository.

3. Select Frameworks and Libraries:

- Use **Spring Boot GraphQL Starter** for the GraphQL implementation.
- Include libraries for observability:
    - **Micrometer** for metrics.
    - **OpenTelemetry** for tracing.
- Include dependencies for database access:
    - **Spring Data JPA** for MySQL/PostgreSQL.
    - **MongoDB driver** for NoSQL integration.