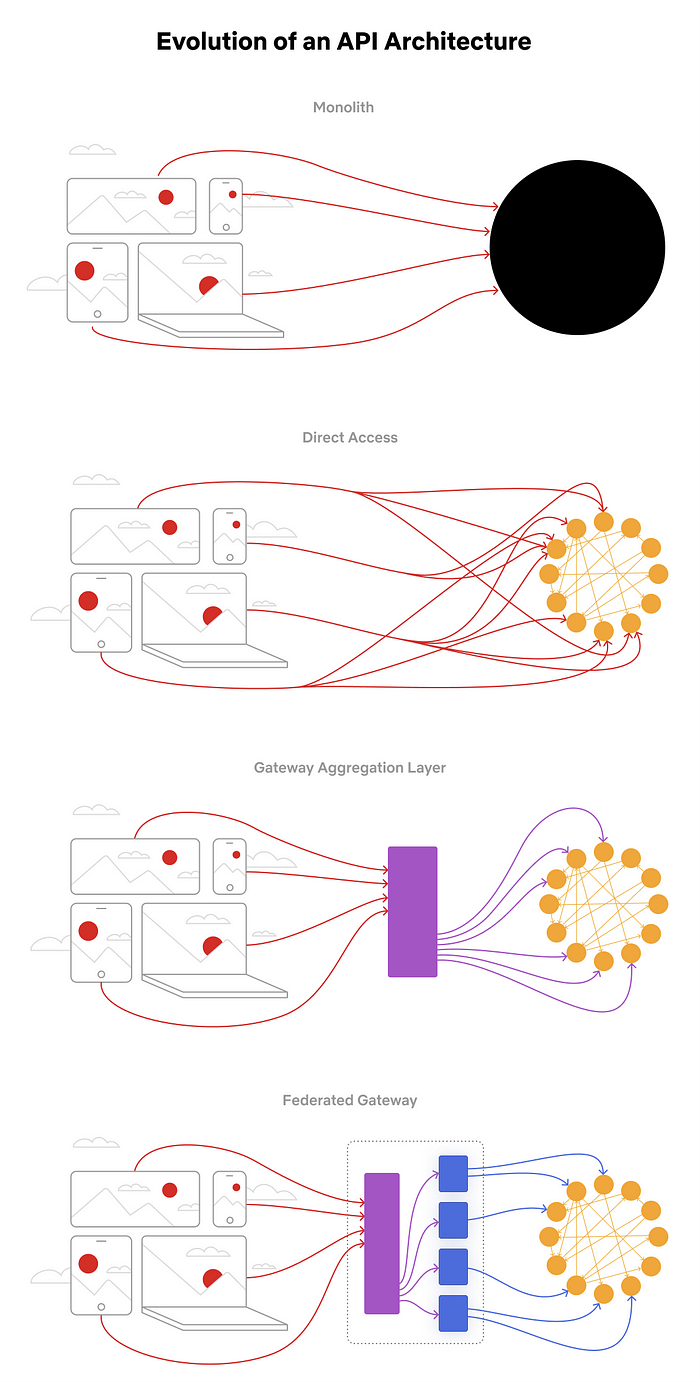
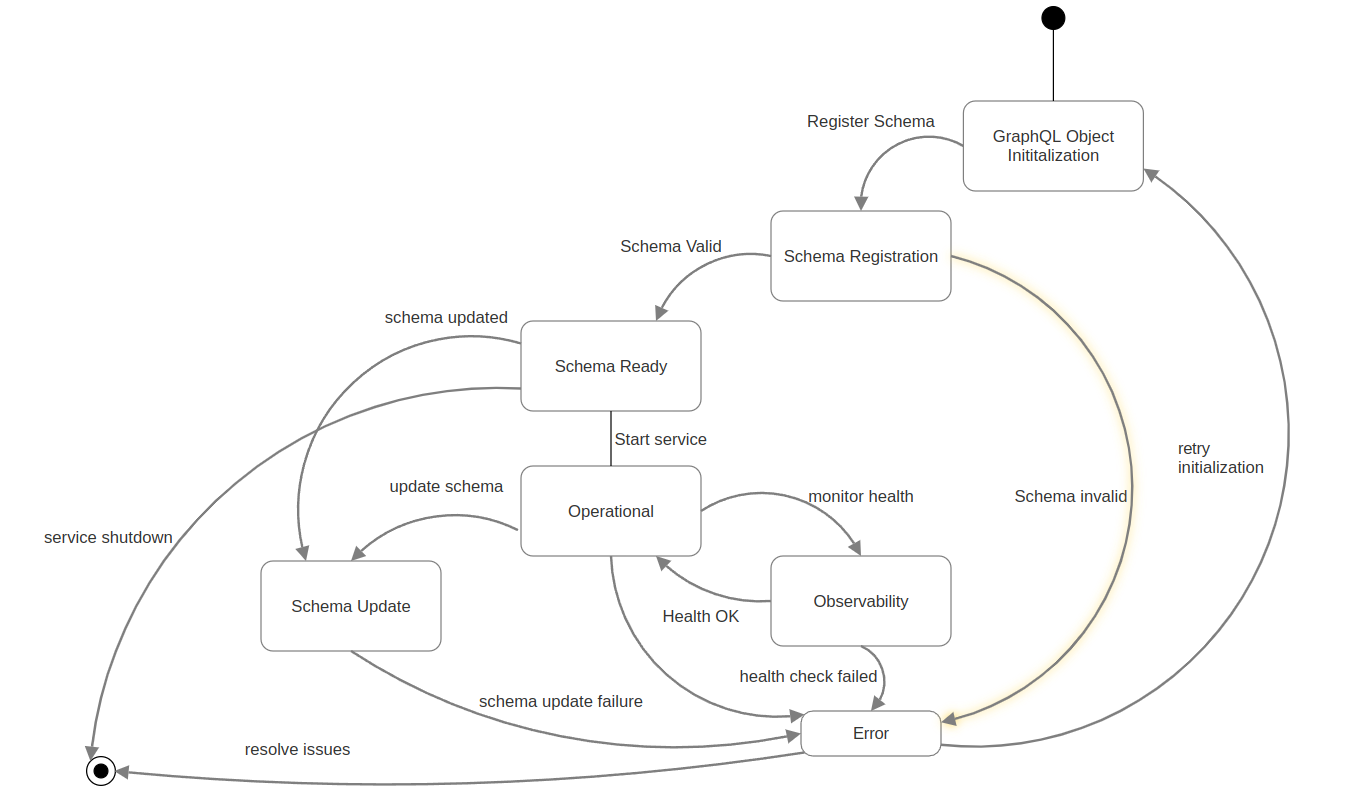
SW3 ASS2

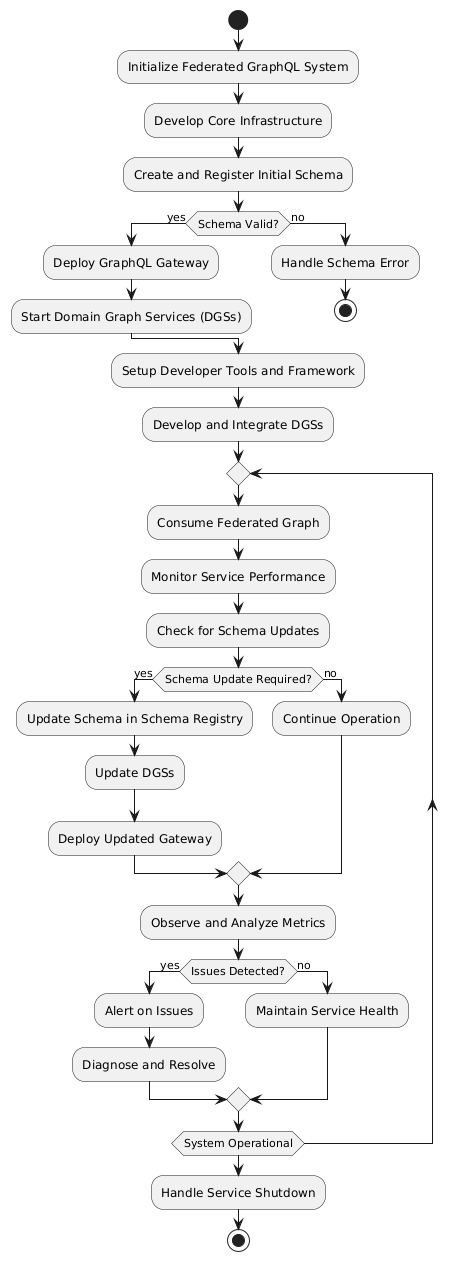


STATE DIAGRAM



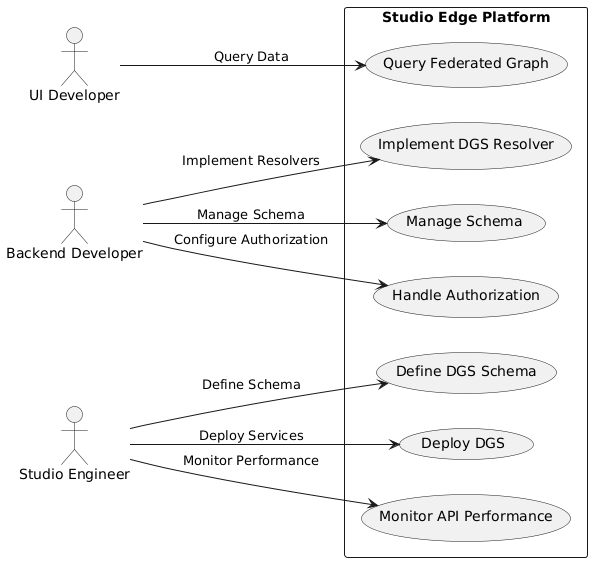
1. Monolith: The initial architecture was a monolithic system with a single deployment unit. This approach was inefficient and difficult to scale.
2. Direct Access: Netflix transitioned to a microservices-based architecture where clients could directly access microservices. However, this led to challenges like inefficiency and difficulty in managing multiple microservices.
3. Gateway Aggregation Layer: A gateway layer was introduced to address the challenges of direct access. This layer unified the API, provided data abstraction, and simplified client access.
4. Federated Gateway: The final evolution involved a federated gateway that provides a federated API and domain isolation. This approach offers improved scalability, security, and flexibility.

ACTIVITY DIAGRAM



1. The initial monolithic approach had a single deployment unit that handled all API requests, leading to potential inefficiencies.
2. The architecture then evolved to a direct access approach where microservices were directly accessible by clients, however this created client access and inefficiency issues.
3. This evolved into a gateway aggregation layer, which provided unified API access and data abstraction.
4. Finally, the architecture evolved to a federated gateway approach, which offers a federated API, allowing for domain isolation and better scalability.

USE CASE DIAGRAM



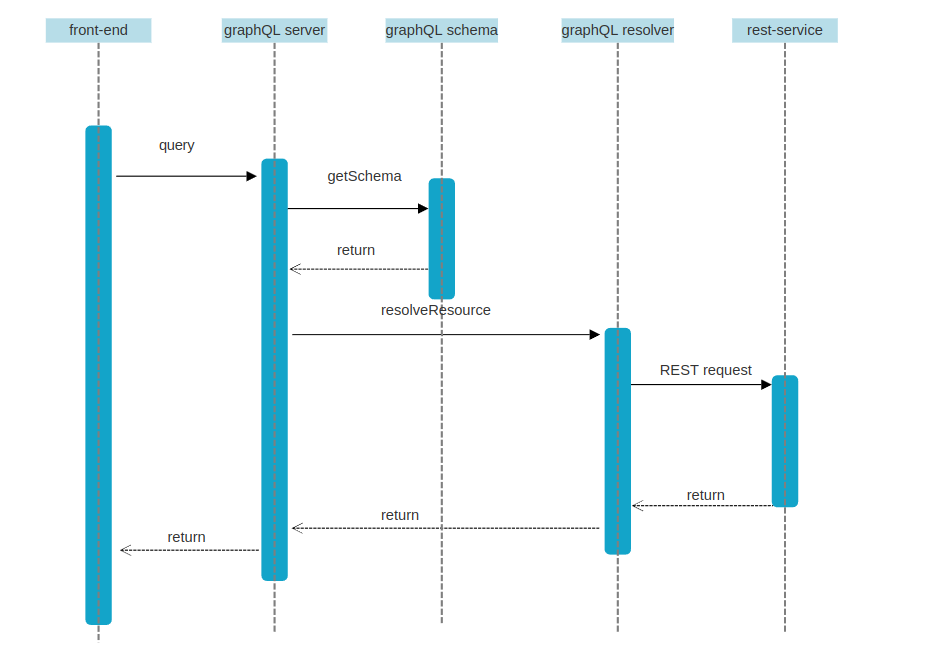
Actors:

* 1. UI Developer: Interacts with the system by querying the federated graph.
  2. Backend Developer: manages schema and authorization.
  3. Studio Engineer: Defines and deploys Domain Graph Services (DGS) and monitors the system.

Use Cases:

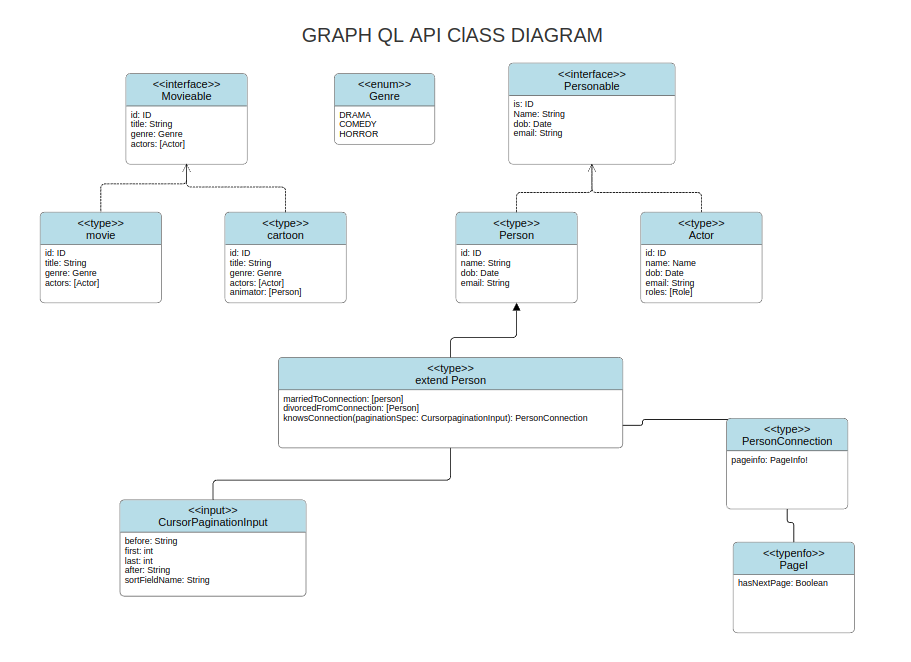
* 1. Define DGS Schema: Define the schema for a new or existing DGS.
  2. Implement DGS Resolver: Implement resolvers for the DGS.
  3. Deploy DGS: Deploy the defined DGS services.
  4. Query Federated Graph: UI developers query the unified federated GraphQL API.
  5. Manage Schema: Backend developers manage and evolve the GraphQL schema.
  6. Monitor API Performance: Studio engineers and developers monitor the performance of the GraphQL API.
  7. Handle Authorization: Backend developers configure and manage authorization.

SEQUENCE DIAGRAM



|  |  |
| --- | --- |
| **Architecture** | **Key Features** |
| Monolith | Single Deployment Unit |
| Direct Access | Client Access Microservices, Inefficiency |
| Gateway Aggregation Layer | Unified API, Data Abstraction |
| Federated Gateway | Federated API, Domain Isolation |

CLASS DIAGRAM



How This Might Be Used in GraphQL

Querying a Movie: A user might query for a specific movie by title, and then request its actors.

Retrieving Actors: You might want to fetch a list of all actors, possibly using pagination. The PersonConnection type would be used to represent the connection between actors, allowing you to request multiple pages of results.

Creating a New Movie: Creating a new movie would require providing a title, genre, release date, and assigning it to specific directors and actors.