Backend Integration Patterns – React, Next.js, Spring Boot, GraphQL, NoSQL

# Pattern 1: Direct GraphQL

📌 Flow:  
UI (React/Next.js) → Domain GraphQL → Domain NoSQL DB

📘 Description:  
Used for lightweight, CRUD-only UIs. No business logic in backend, only schema-to-DB mapping.

✅ Use Case:  
Admin panels, Settings UI, User management CRUD

🔍 Pros:  
Fast to implement, minimal code, perfect for no-business-logic apps

# Pattern 2: Azure API Gateway + Spring Boot GraphQL Microservices

📌 Flow:  
UI → Azure Gateway → Spring Boot GraphQL APIs → NoSQL

📘 Description:  
Azure API Gateway handles auth, routing, throttling. Spring Boot services expose GraphQL endpoints, each with its own DB.

✅ Use Case:  
PBM Dashboards, Secure Microservices Architecture

🔍 Pros:  
Centralized API governance, scalable services

# Pattern 3: Backend-for-Frontend (BFF) with REST

📌 Flow:  
UI → Spring Boot BFF → Domain Services → NoSQL

📘 Description:  
Custom API layer per frontend app. UI-specific formatting and aggregation from multiple domains.

✅ Use Case:  
Tailored dashboards, mobile-first APIs

🔍 Pros:  
UI flexibility, avoids tight coupling with core services

# Pattern 4: Reusable Domain Services (REST/GraphQL)

📌 Flow:  
UI → API Gateway → Domain Service A/B → NoSQL

📘 Description:  
Well-defined domain boundaries with each service handling its own logic and persistence.

✅ Use Case:  
Multi-team development, modular APIs

🔍 Pros:  
Clear separation of concerns, reusable logic

# Pattern 5: Next.js Edge API → Spring Boot Proxy

📌 Flow:  
Next.js API Route → Spring Boot API → NoSQL

📘 Description:  
Edge functions for SSR, security or session orchestration before hitting core backend.

✅ Use Case:  
Secure SSR pages, cookie handling

🔍 Pros:  
Low latency, simplified frontend/backend handshake

# Pattern 6: Event-Driven Write / Query-Optimized Read (CQRS)

📌 Flow:  
UI → Command API → Kafka → DB → Query API → UI

📘 Description:  
Command-write and query-read are decoupled. Supports scaling and read optimization.

✅ Use Case:  
Analytics, Audit Logging, Read-heavy apps

🔍 Pros:  
Event sourcing ready, async and scalable

# Pattern 7: Hybrid REST + GraphQL APIs

📌 Flow:  
UI → GraphQL (reads), REST (writes) → Spring Boot → NoSQL

📘 Description:  
Reads via GraphQL for frontend flexibility. Writes via REST for validation and control.

✅ Use Case:  
PBM grids, dashboards, complex UI logic

🔍 Pros:  
Best of both API worlds, efficient data exchange

# Pattern 8: Template-Driven Config APIs

📌 Flow:  
UI → Config API (.pbmgrid.json) → Execution Engine → Result DB

📘 Description:  
UI defines templates/rules. Backend executes and stores outputs. Supports user-defined logic.

✅ Use Case:  
Smart models, formula templates, PBM simulations

🔍 Pros:  
Decouples UI config from backend logic, LLM-compatible