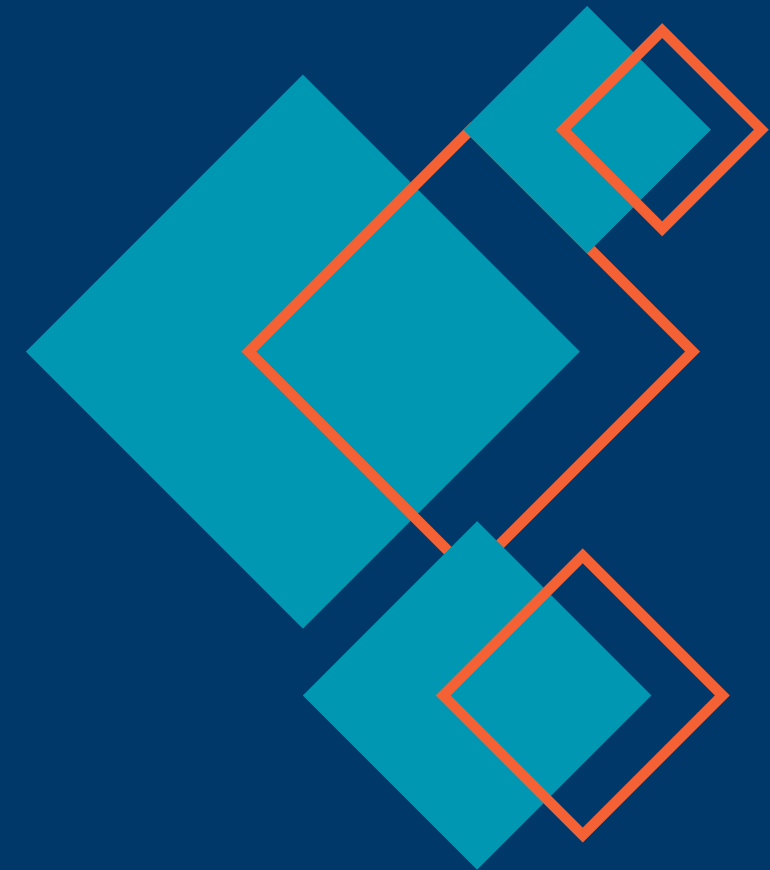




System Design

API ডিজাইন



API আর্কিটেকচারের ধরন

API Architecture প্রধানত ৩ ধরনের – এগুলোর মধ্যে বাস্তবে আরও কিছু variation আছে, তবে মৌলিক ক্যাটাগরি নিচের মতো:

1. REST API (Representational State Transfer)
2. GraphQL API
3. gRPC (Google Remote Procedure Call)

আরও কিছু জনপ্রিয় APPROACH






1. WebSocket API: Real-time data communication (chat apps, live update)
2. SOAP API: পুরাতন, XML-based, bank/enterprise systems এ এখনও ব্যবহৃত।
3. Async API (Message Driven): Kafka, RabbitMQ-এর মাধ্যমে asynchronous communication।

REST API কী?

REST API (Representational State Transfer) মানে হলো – এমন একটি API design যেখানে HTTP method (GET, POST, PUT, DELETE) ব্যবহার করে server-এর data CRUD operation করা হয়।

REST মূলত Resource Based Architecture – মানে প্রতিটি entity (User, Product, Order) আলাদা resource হিসাবে access করা যায়।

কেন REST API দরকার হয়?

কারণ	ব্যাখ্যা
 Cross-platform Communication	Backend আলাদা, frontend আলাদা হলেও REST API দিয়ে সহজে connect করা যায়। (mobile, web, desktop সব এক API use করে)
 Stateless Design	প্রতিটি request independent → system scale করতে সহজ।
 Simple & Scalable	Simple URL system, standard HTTP method → দ্রুত development হয়।
 Security Friendly	Token-based security (JWT), rate limiting সহজ।
 Modular System Design	Backend ফ্রেমওয়ার্ক বা language আলাদা হলেও frontend এর সাথে communicate করা যায়।

REST API কোথায় বানাবো বা কখন আমাদের লাগবে ?

Scenario	REST API দরকার কি?
Simple Web Application	✓ হ্যাঁ, basic CRUD site এ REST খুব ভালো কাজ করে।
Mobile App Backend	✓ হ্যাঁ, mobile data efficiency জন্য REST widely used.
Admin Panel + Customer Panel আলাদা	✓ হ্যাঁ, REST দিয়ে আলাদা frontend connect হয়।
Microservices Architecture	✗ Partial → Microservices এ gRPC বা Event Driven architecture better হয়।
Real-time Application (Chat, Live Streaming)	✗ WebSocket API better (REST latency বেশি)।
Complex Data Relationship (GraphQL উপযুক্ত)	✗ Better to use GraphQL.

GRAPHQL কি?

GRAPHQL হচ্ছে একটি API QUERY LANGUAGE যেটা CLIENT কে FULL CONTROL দেয় সে ঠিক কী DATA লাগবে সেটা ঠিক করতে।

- SINGLE ENDPOINT → /GRAPHQL
- CLIENT SIDE থেকে QUERY পাঠালে SERVER ঠিক ওই DATA RETURN করে।

কেন GRAPHQL দরকার?

1. REST এ অনেক DATA আসে যা লাগে না, GRAPHQL-এ শুধু প্রয়োজনীয় DATA আসে।
2. REST API তে কখনও MULTIPLE API CALL করতে হয়। GRAPHQL-এ ১টা CALL-এ MULTIPLE RESOURCE আনা যায়।
3. LOW BANDWIDTH APP এর জন্য পারফেক্ট কারণ শুধুমাত্র লাগবে এমন DATA আনা যায়।

GRAPHQL EXAMPLE USE CASE:

1. 🖱️ ইউজার DASHBOARD: USER INFO, ORDERS, PRODUCT SUMMARY → সব এক CALL-এ
2. 🖱️ E-LEARNING PLATFORM: USER PROGRESS, COURSE LIST, INSTRUCTOR INFO → NESTED QUERY
3. 🖱️ SOCIAL MEDIA FEED: POST, COMMENTS, LIKES COUNT → EFFICIENT FETCH

কখন GraphQL ব্যবহার করবেন?

Scenario	GraphQL ব্যবহার করবেন?
✅ Complex relational data	হ্যাঁ, যেমন user → orders → products একসাথে লাগলে
✅ Mobile App with Limited Data Need	হ্যাঁ, যেমন mobile app এ শুধু ২টা ফিল্ড দরকার
✅ Frontend-driven Data Need	হ্যাঁ, React/Vue app এ Component অনুযায়ী data লাগলে
✅ Multiple UI Client (Web+Mobile)	হ্যাঁ, কারণ এক endpoint থেকে আলাদা আলাদা client customize করতে পারে

GraphQL vs REST সংক্ষিপ্ত তুলনা:

বিষয়	REST API	GraphQL
Endpoint	Multiple endpoint (/users , /products)	Single endpoint (/graphql)
Data Fetching	Fixed response	Client chooses
Over-fetching	Common	কম
Under-fetching	Common (multiple API call)	নাই
Learning Curve	Easy	একটু বেশি
Real-time	❌ WebSocket দরকার	✅ Subscription আছে
Best Use Case	CRUD, simple system	Complex data, nested resource

gRPC API কি?

gRPC হলো GOOGLE-এর বানানো REMOTE PROCEDURE CALL FRAMEWORK। HTTP/2 ব্যবহার করে এবং PROTOCOL BUFFERS (.PROTO) দিয়ে MESSAGE ENCODE হয়।

এক লাইনে বলা যায় –

☞ REST/GRAPHQL যেখানে RESOURCE-BASED, gRPC সেখানে FUNCTION CALL-BASED।

তাহলে RESOURCE-বেসড কি ?

REST বা GRAPHQL-এ API মানে হল কোনো RESOURCE বা OBJECT কে REPRESENT করা – যেমন USER, PRODUCT, ORDER এখানে প্রতিটি ENDPOINT RESOURCE/ENTITY কে REPRESENT করছে।

gRPC FUNCTION-CALL BASED কি?

gRPC API মানে হল – আমরা সরাসরি একটি FUNCTION/METHOD CALL করি, যেনো আমরা REMOTE COMPUTER-এ FUNCTION CALL করতে পারি

EXAMPLE (gRPC .PROTO FILE):

```
service UserService {  
    rpc GetUser (UserRequest) returns (UserResponse);  
    rpc CreateOrder (OrderRequest) returns (OrderResponse);  
}
```

এখানে /GetUser বা /CreateOrder resource না, বরং function।

gRPC এর সুবিধা :

PROTOCOL BUFFERS BINARY FORMAT হয়ে থাকে তাই – JSON থেকে অনেক ফাস্ট হয়।

DATA COMPRESS হয়ে আসে → তাই কম BANDWIDTH USE হয়।

INTERNAL SERVICE COMMUNICATION এ BEST LIKE : MICROSERVICE।

PROTO FILE দিয়ে API STRICTLY DEFINED থাকে → AUTO-CODE-GEN করা যায়।

কেন gRPC দরকার?

- স্পিড বাকিগুলোর তুলনায় বেশ ভালো।
- BI-DIRECTIONAL STREAM মানে, CLIENT এবং SERVER একই সময়ে একে অপরকে ডেটা পাঠাতে এবং রিসিভ করতে পারে, একই CONNECTION-এর মধ্যে।
- SCALABLE এর দিক থেকে এইটি HIGH।
- ইন্টারনাল কমিউনিকেশন এর জন্য gRPC বেশ ভালো

Real Life Example:

- UBER → MICROSERVICES COMMUNICATION → gRPC
- NETFLIX → REAL-TIME DATA FLOW → gRPC
- STRIPE → PUBLIC API → REST, INTERNAL BILLING → gRPC
- CLOUD PROVIDERS (GOOGLE CLOUD) → INTERNAL gRPC, EXTERNAL REST

gRPC কখন ব্যবহার করবেন?

- ✓ MICROSERVICES COMMUNICATION (INTERNAL SERVICES FAST COMMUNICATE করে)
- ✓ REAL-TIME STREAMING SERVICE (ভিডিও/চ্যাট DATA STREAM করার জন্য)
- ✓ HIGH-FREQUENCY REQUESTS (IOT) (SMALL PACKET FAST পাঠানোর জন্য)
- ✓ ML MODEL COMMUNICATION (HEAVY BACKEND TASK PROCESS করার জন্য)
- ✓ PAYMENT GATEWAY INTERNAL NODE (INTERNAL GRPC, PUBLIC REST)

WebSocket API

WEBSOCKET হলো REAL-TIME COMMUNICATION SYSTEM। CLIENT ↔ SERVER একই সময় DATA পাঠায় ও পায়, CONNECTION OPEN থাকে।

- LIVE CHAT SYSTEM
- GAMING REAL-TIME UPDATE
- LIVE STOCK/PRICE UPDATE

SOAP API

SOAP (SIMPLE OBJECT ACCESS PROTOCOL) পুরানো, XML-BASED API। খুব STRICT এবং HEAVY WEIGHT।

- > BANKING SYSTEM (STRICT SECURITY দরকার)
- > ENTERPRISE ERP

Async API (Asynchronous Message Driven Architecture)

ASYNC API মানে আপনি REQUEST করেন → SERVER সাথে সাথে RESPONSE দেয় না, পরে EVENT TRIGGER করে RESPONSE দেয়।
➡ MESSAGE QUEUE (RABBITMQ, KAFKA) USE করে।

API Type	Use For	Speed	Real-time?	Best Use
WebSocket	Real-time, Live Update	Fast	✓	Chat, Games
SOAP API	Enterprise, Banking	Slow	✗	Legacy Secure Apps
Async API	Background Process	Delayed	✗	Heavy Processing, Queue System

API ডিজাইন করার Best Practices

1. **RESOURCE NAMING:** সবসময় PLURAL NOUN ব্যবহার করা উচিত /USERS, /PRODUCTS
2. **VERSIONING:** API VERSIONING অবশ্যই রাখতে হবে /API/V1/
3. **PAGINATION:** ?PAGE=1&LIMIT=20
4. **FILTERING & SORTING:** ?SORT=CREATED_AT&ORDER=DESC
5. **STATUS CODE:** PROPER HTTP STATUS CODE ব্যবহার করতে হবে: 200 OK, 201 CREATED, 400 BAD REQUEST, 404 NOT FOUND, 500 SERVER ERROR
6. **IDEMPOTENCY:** একই API CALL MULTIPLE বার করলে একই RESULT আসবে (POST ছাড়া)
7. **ERROR HANDLING STRUCTURE**

```
{  
  "ERROR": TRUE,  
  "MESSAGE": "INVALID INPUT",  
  "CODE": 400  
}
```

Authentication এবং Authorization

1. API KEY: SIMPLE AUTHENTICATION কিন্তু কম SECURE
2. JWT (JSON WEB TOKEN): STATELESS, TOKEN-এর ভিতরেই USER INFORMATION থাকে
3. OAUTH2: THIRD-PARTY AUTHENTICATION এর জন্য BEST (GOOGLE, FACEBOOK LOGIN)
4. RATE LIMITING: EXCESSIVE API CALLS ব্লক করার জন্য (যেমন প্রতি মিনিটে ১০০ কল)
5. API GATEWAY: সব API CALL এক জায়গা থেকে MANAGE করা হয় (KONG, NGINX)

API Scalability

1. LOAD BALANCER: REQUESTS MULTIPLE SERVER এ DISTRIBUTE করা (NGINX, AWS ALB)
2. HORIZONTAL SCALING: INSTANCE বাড়িয়ে SCALABILITY বাড়ানো
3. RATE LIMITING & THROTTLING: ABUSE কমানোর জন্য ব্যবহার
4. CACHING: REDIS, CDN (CLOUDFLARE) ব্যবহার করে RESPONSE দ্রুত পাঠানো
5. RETRY MECHANISM: EXPONENTIAL BACKOFF POLICY
6. CIRCUIT BREAKER: একটি API FAIL করলে পুরো সিস্টেম যেন না পড়ে যায়

Monitoring এবং Observability

1. LOGGING: CENTRALIZED LOGGING (ELK STACK, GRAYLOG)
2. API METRICS: LATENCY, ERROR RATE, REQUEST COUNT MONITOR করা (PROMETHEUS, GRAFANA)
3. TRACING: DISTRIBUTED TRACING TOOL (JAEGER, ZIPKIN)

API Documentation এবং Developer Experience (DX)

1. OPENAPI SPECIFICATION (SWAGGER): অটোমেটিক API DOCUMENTATION
2. POSTMAN COLLECTIONS: API সহজে টেস্ট করার জন্য
3. SDK ও CODE SAMPLES: DEVELOPER FRIENDLY করার জন্য
4. MOCK API SERVERS: FRONTEND DEVELOPER দ্রুত কাজ করতে পারে

শেষ কথা:

পরবর্তীতে আমরা আরো বেশ কিছু অ্যাডভান্সড টপিকস নিয়েও আলোচনা করবো।

1. API Gateway (Kong, AWS API Gateway)
2. Backend for Frontend (BFF Pattern)
3. Webhook design এবং Event Driven API
4. Async API architecture