

# API Architecture

[Author Name]

Université de Toulon, LIS UMR CNRS 7020

2026-10-14

## Table des matières

|                                        |   |
|----------------------------------------|---|
| 1 Purpose of this Document .....       | 1 |
| 2 API Strategy Overview .....          | 1 |
| 3 API Types and Usage .....            | 1 |
| 4 API Governance .....                 | 2 |
| 5 API Gateway Architecture .....       | 2 |
| 6 API Gateway Components .....         | 3 |
| 7 API Design Standards .....           | 3 |
| 7.1 REST API Standards .....           | 3 |
| 7.2 GraphQL Standards .....            | 4 |
| 7.3 gRPC Standards .....               | 4 |
| 8 API Security .....                   | 5 |
| 9 API Management .....                 | 5 |
| 10 API Operations .....                | 6 |
| 11 API Analytics .....                 | 6 |
| 12 API Deployment .....                | 7 |
| 13 Example Implementation .....        | 7 |
| 13.1 REST API Endpoint .....           | 7 |
| 13.2 GraphQL Resolver .....            | 7 |
| 14 Template Validation Checklist ..... | 8 |
| 15 Related Views .....                 | 8 |

## 1 Purpose of this Document

This document defines the API architecture strategy for [System Name], including API design principles, patterns, governance, and implementation details.

## 2 API Strategy Overview

API Philosophy: [Approach to API design]

### Primary Goals:

- [Goal 1, e.g., "Enable partner integration"]
- [Goal 2, e.g., "Support mobile applications"]
- [Goal 3, e.g., "Internal service communication"]

### Key Principles:

- [Principle 1, e.g., "API-first design"]
- [Principle 2, e.g., "Contract-driven development"]
- [Principle 3, e.g., "Consistent error handling"]

## 3 API Types and Usage

### API Types:

#### REST:

Style: [REST/HAL/JSON-API/OData]

Version Strategy: [URI/Header/

Content-Type]

Use Cases: [When to use REST]

Constraints: [Limitations or considerations]

#### GraphQL:

Schema Design: [Approach to schema design]

Federation: [Approach to federated schemas]

Use Cases: [When to use GraphQL]  
Constraints: [Limitations or considerations]

gRPC:

Proto Design: [Approach to Proto definitions]

Streaming: [Strategy for streaming data]

Use Cases: [When to use gRPC]

Constraints: [Limitations or considerations]

Event-Based:

Protocol: [Webhook/SSE/WebSockets]

Format: [Message format]

Use Cases: [When to use events]

Constraints: [Limitations or considerations]

## 4 API Governance

API Lifecycle:

Design: [Process for designing APIs]

Review: [Review process]

Versioning: [Approach to versioning]

Deprecation: [Process for deprecating APIs]

Retirement: [Process for retiring APIs]

Standards:

Naming: [Naming conventions]

Documentation: [Documentation requirements]

Security: [Security requirements]

Performance: [Performance requirements]

Monitoring:

Metrics: [Key metrics to track]

Alerting: [When to alert]

Reporting: [Required reports]

## 5 API Gateway Architecture

graph TD

subgraph "API Consumers"

A[Mobile Apps]

B[Web Apps]

C[Partner Systems]

D[Internal Services]

end

subgraph "API Gateway Layer"

E[API Gateway]

F[Authentication]

G[Rate Limiting]

H[Request Routing]

I[Response Transformation]

J[Monitoring]

end

subgraph "Backend Services"

K[Service 1]

L[Service 2]

M[Service 3]

end

A --> E

B --> E

C --> E

D --> E

E --> F

E --> G

E --> H

E --> I

E --> J

H --> K

H --> L

H --> M

## 6 API Gateway Components

```
@startuml
package "API Gateway" {
    [Rate Limiting] as RL
    [Authentication] as Auth
    [Authorization] as Authz
    [Routing] as Route
    [Transformation] as Trans
    [Caching] as Cache
    [Monitoring] as Mon
}

cloud "Clients" {
    [Mobile] as Mobile
    [Web] as Web
    [Partners] as Partners
}

cloud "Backend" {
    [Microservices] as MS
    [Legacy Systems] as Legacy
    [External Services] as Ext
}

Mobile --> RL
Web --> RL
Partners --> RL

RL --> Auth
Auth --> Authz
Authz --> Route
Route --> Trans
Trans --> Cache

Route --> MS
Route --> Legacy
Route --> Ext

Auth ..> Mon
Route ..> Mon
Cache ..> Mon
@enduml
```

## 7 API Design Standards

### 7.1 REST API Standards

- **Resource Naming:** Use plural nouns for collections (`/users`), singular for specific resources (`/users/{id}`)
- **HTTP Methods:**
  - GET: Retrieve resources
  - POST: Create resources
  - PUT: Replace resources
  - PATCH: Update resources partially
  - DELETE: Remove resources
- **Status Codes:**
  - 200: Success
  - 201: Created
  - 204: No Content
  - 400: Bad Request
  - 401: Unauthorized
  - 403: Forbidden
  - 404: Not Found
  - 409: Conflict
  - 422: Unprocessable Entity
  - 429: Too Many Requests
  - 500: Internal Server Error
- **Query Parameters:**
  - Filtering: `?status=active`
  - Sorting: `?sort=createdAt:desc`
  - Pagination: `?page=2&limit=10`
  - Fields selection: `?fields=id,name,email`
- **Response Format:**

```
{
  "data": {
    "id": "123",
    "type": "user",
    "attributes": {
      "name": "John Doe",
      "email": "john@example.com"
    },
    "relationships": {
      "orders": {
        "links": {
          "related": "/users/123/"
        }
      }
    }
  }
}
```

```

orders"
    }
  }
},
"meta": {
  "requestId": "abc-123"
}
}

```

## 7.2 GraphQL Standards

- **Schema Design:**
  - Use descriptive type names
  - Include descriptions for types and fields
  - Use interfaces for common fields
  - Implement connections for pagination
- **Operations:**
  - Queries: For fetching data
  - Mutations: For modifying data
  - Subscriptions: For real-time updates
- **Error Handling:**
  - Use standard GraphQL errors
  - Include error codes
  - Provide actionable error messages
- **Example Schema:**

```

"""
A user in the system
"""
type User {
  id: ID!
  name: String!
  email: String!
  createdAt: DateTime!
  orders(first: Int, after: String):
OrderConnection!
}

type OrderConnection {
  edges: [OrderEdge!]!
  pageInfo: PageInfo!
  totalCount: Int!
}

```

```

}

type OrderEdge {
  node: Order!
  cursor: String!
}

type PageInfo {
  hasNextPage: Boolean!
  hasPreviousPage: Boolean!
  startCursor: String
  endCursor: String
}

```

## 7.3 gRPC Standards

- **Proto Design:**
  - Use descriptive service and message names
  - Version proto files
  - Include field comments
  - Use appropriate field types
- **Service Types:**
  - Unary: Simple request/response
  - Server streaming: One request, multiple responses
  - Client streaming: Multiple requests, one response
  - Bidirectional streaming: Multiple requests and responses
- **Example Proto:**

```

syntax = "proto3";
package example.v1;

service UserService {
  // GetUser returns a single user
  by ID
  rpc GetUser(GetUserRequest) returns
(GetUserResponse);

  // ListUsers returns a paginated
list of users
}

```

```

    rpc ListUsers(ListUsersRequest)
    returns (ListUsersResponse);

    // CreateUser creates a new user
    rpc CreateUser(CreateUserRequest)
    returns (CreateUserResponse);

    // WatchUserUpdates streams user
    updates in real-time
    rpc WatchUserUpdates(WatchUserUpdatesReq
    returns (stream UserUpdate);
}

message GetUserRequest {
    string user_id = 1;
}

message GetUserResponse {
    User user = 1;
}

message User {
    string id = 1;
    string name = 2;
    string email = 3;
    google.protobuf.Timestamp
    created_at = 4;
}

```

## 8 API Security

Security Controls:

Authentication:

- OAuth 2.0:
  - Grant Types: [Authorization Code, Client Credentials, etc.]
  - Token Format: [JWT]
  - Token Lifetime: [Access token and refresh token lifetime]
- API Keys:
  - Usage: [When to use API keys]
  - Distribution: [How API keys

are distributed]

Rotation: [Key rotation policy]

- mTLS:
  - Usage: [When to use mTLS]
  - Certificate Management: [Process for managing certificates]
- Authorization:
  - RBAC:
    - Roles: [Key roles]
    - Permissions: [How permissions are managed]
  - Scopes:
    - Definition: [How OAuth scopes are defined]
    - Validation: [How scopes are validated]
  - Rate Limits:
    - Default: [Default limits]
    - Tiers: [Rate limit tiers]
    - Enforcement: [How limits are enforced]
    - Headers: [Response headers]
- Transport Security:
  - TLS Version: [Minimum TLS version]
  - Cipher Suites: [Allowed cipher suites]
  - HSTS: [HSTS configuration]
  - Certificate Management: [Process for managing certificates]

## 9 API Management

Developer Experience:

Documentation:

- API Reference: [Format and location]
- Tutorials: [How-to guides]
- Code Examples: [Example code in

key languages]

- SDKs: [Available SDKs]

Developer Portal:

- Self-service: [Registration and API key management]
- Analytics: [Usage reporting]
- Support: [How to get support]

Sandbox Environment:

- Access: [How to access]
- Data: [Sample data]
- Limitations: [Differences from production]

API Gateway Features:

Request Processing:

- Validation: [Schema validation]
- Transformation: [Request transformation]
- Enrichment: [Request enrichment]

Response Processing:

- Transformation: [Response transformation]
- Compression: [Compression settings]
- Caching: [Caching strategy]

Operational:

- Logging: [Logging configuration]
- Monitoring: [Metrics collected]
- Alerting: [Alert conditions]

## 10 API Operations

Monitoring:

Metrics:

- Availability: [SLA targets]
- Response Time: [Performance targets]
- Error Rate: [Error rate targets]
- Usage: [Usage tracking]

Logging:

- Request Logs: [What is logged]
- Error Logs: [Error details]
- Audit Logs: [Audited events]
  - Retention: [Log retention period]

Alerts:

- SLA Breaches: [Alert conditions]
- Error Spikes: [Alert conditions]
  - Usage Anomalies: [Alert conditions]
  - Security Events: [Alert conditions]

Troubleshooting:

Tools:

- Request Tracing: [Tracing tools]
- Log Analysis: [Log analysis tools]
- API Testing: [Testing tools]

Common Issues:

- Authentication Failures: [Troubleshooting steps]
- Rate Limiting: [Troubleshooting steps]
  - Performance Issues: [Troubleshooting steps]
  - Integration Errors: [Troubleshooting steps]

## 11 API Analytics

Usage Analytics:

Metrics:

- Calls by API: [Tracking API usage]
- Calls by Consumer: [Tracking consumer usage]
- Calls by Endpoint: [Tracking endpoint usage]

- Error Rates: [Tracking errors]
  - Performance: [Tracking performance]

Reporting:

- Dashboards: [Available dashboards]
- Alerts: [Alert thresholds]
- Exports: [Data export options]

Insights:

- Trend Analysis: [Usage trends]
- Anomaly Detection: [Unusual patterns]
- Consumer Behavior: [Usage patterns]

## 12 API Deployment

Deployment Strategy:

- CI/CD Pipeline: [Approach to automated deployment]
- Environment Promotion: [Dev → Test → Prod flow]
- Canary Releases: [Approach to gradual rollout]
- Feature Flags: [Runtime configuration]

API Versioning:

- Strategy: [URI/Header/Content-Type]
- Compatibility: [Backward compatibility policy]
- Deprecation: [Process for deprecating versions]
- Sunsetting: [Process for retiring versions]

## 13 Example Implementation

### 13.1 REST API Endpoint

```
@RestController
@RequestMapping("/api/v1/users")
public class UserController {

    @GetMapping("/{id}")
    public
    ResponseEntity<UserResponse>
    getUser(@PathVariable String id) {
        // Authorization check
        // Fetch user
        // Transform to response
        return
        ResponseEntity.ok(userResponse);
    }

    @PostMapping
    public
    ResponseEntity<UserResponse>
    createUser(@RequestBody @Valid
    UserRequest request) {
        // Validation
        // Create user
        // Return response with
        location header
        return ResponseEntity
            .created(URI.create("/
            api/v1/users/" + userId))
            .body(userResponse);
    }
}
```

### 13.2 GraphQL Resolver

```
const resolvers = {
  Query: {
    user: async (_, { id }, context)
  => {
    // Authorization check
    const user = await context.dataSources.users.getUser
    return user;
  }
}
```

```

    },
    users: async (_, { filter, page,
limit }, context) => {
    // Filter and paginate users
    const result = await context.dataSources.users.listUsers(filter,
page, limit);
    return {
        edges: result.items.map(user
=> ({
            node: user,
            cursor: encodeCursor(user.id)
        })),
        pageInfo: {
            hasNextPage: result.hasMore,
            endCursor:
result.items.length ? encodeCursor(result.items[result.items.length
- 1].id) : null
        },
        totalCount: result.totalCount
    };
    },
    Mutation: {
        createUser: async (_, { input },
context) => {
            // Create user
            const user = await context.dataSources.users.createUser(input);
            return { user };
        }
    }
};

```

- ☐ Deployment and versioning approach specified
- ☐ Example implementations provided
- ☐ Team review completed

## 15 Related Views

- System Context - Overall system context
- Integration Architecture - System integration interfaces
- Component Architecture - Service details

## 14 Template Validation Checklist

- ☐ API strategy defined
- ☐ API types and usage documented
- ☐ API design standards specified
- ☐ Security controls identified
- ☐ API gateway architecture defined
- ☐ Management approach described
- ☐ Monitoring and operations strategy outlined