TOP

7 REST API DESIGN PATTERNS FOR SCALABILITY

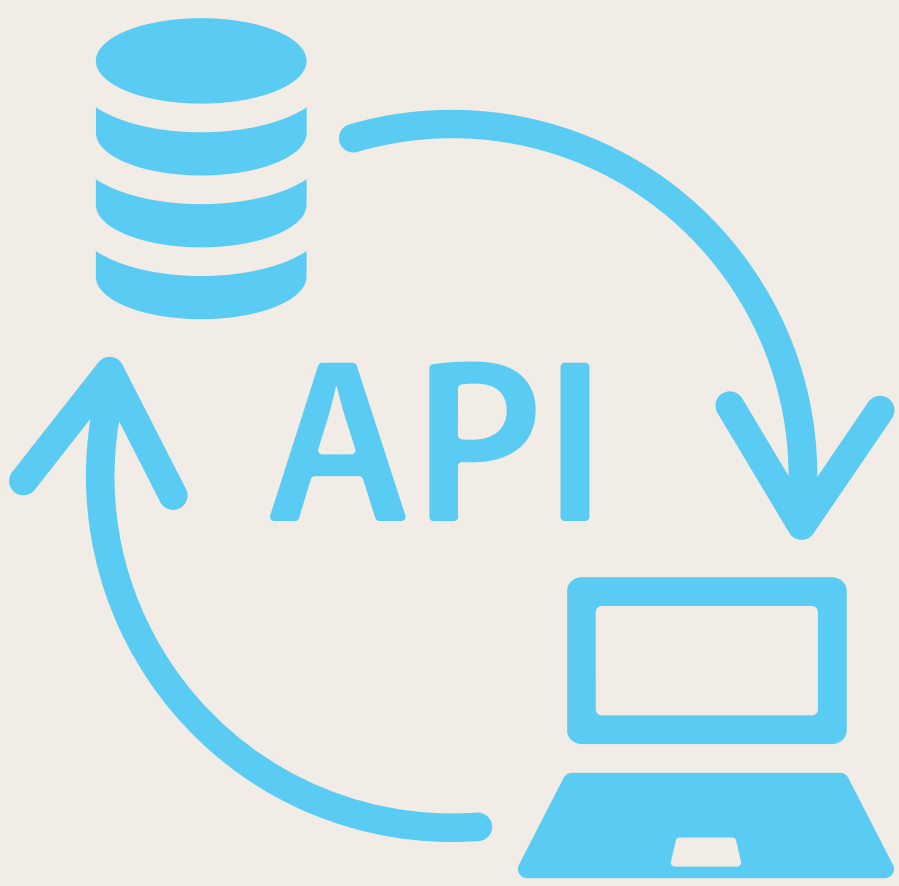
BY ALI AHMAD

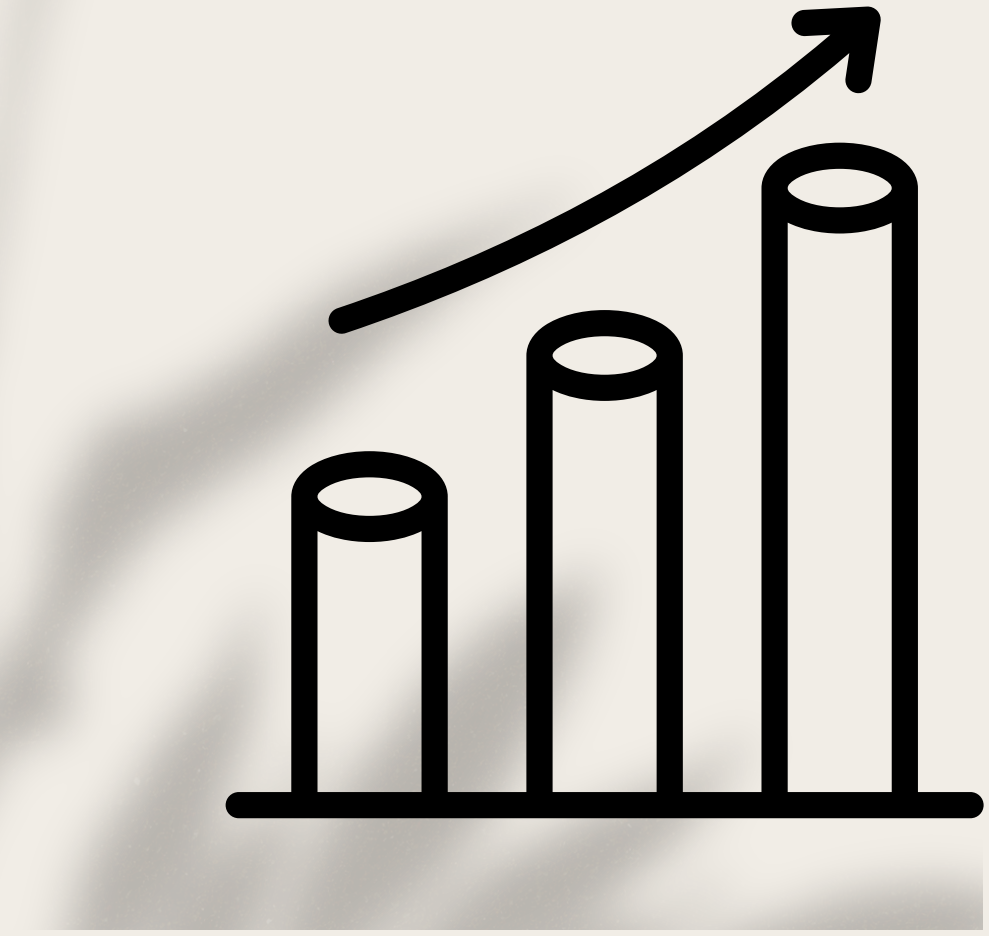
ALI AHMAD
@aliahmad14

1

Introduction

Building scalable and secure REST APIs is crucial for modern applications. Here are **7 patterns** that can help you design better APIs!





7

1.Rate Limiting

Rate Limiting helps control the number of API requests a client can make in a given time frame. This protects your API from abuse and ensures fair usage.

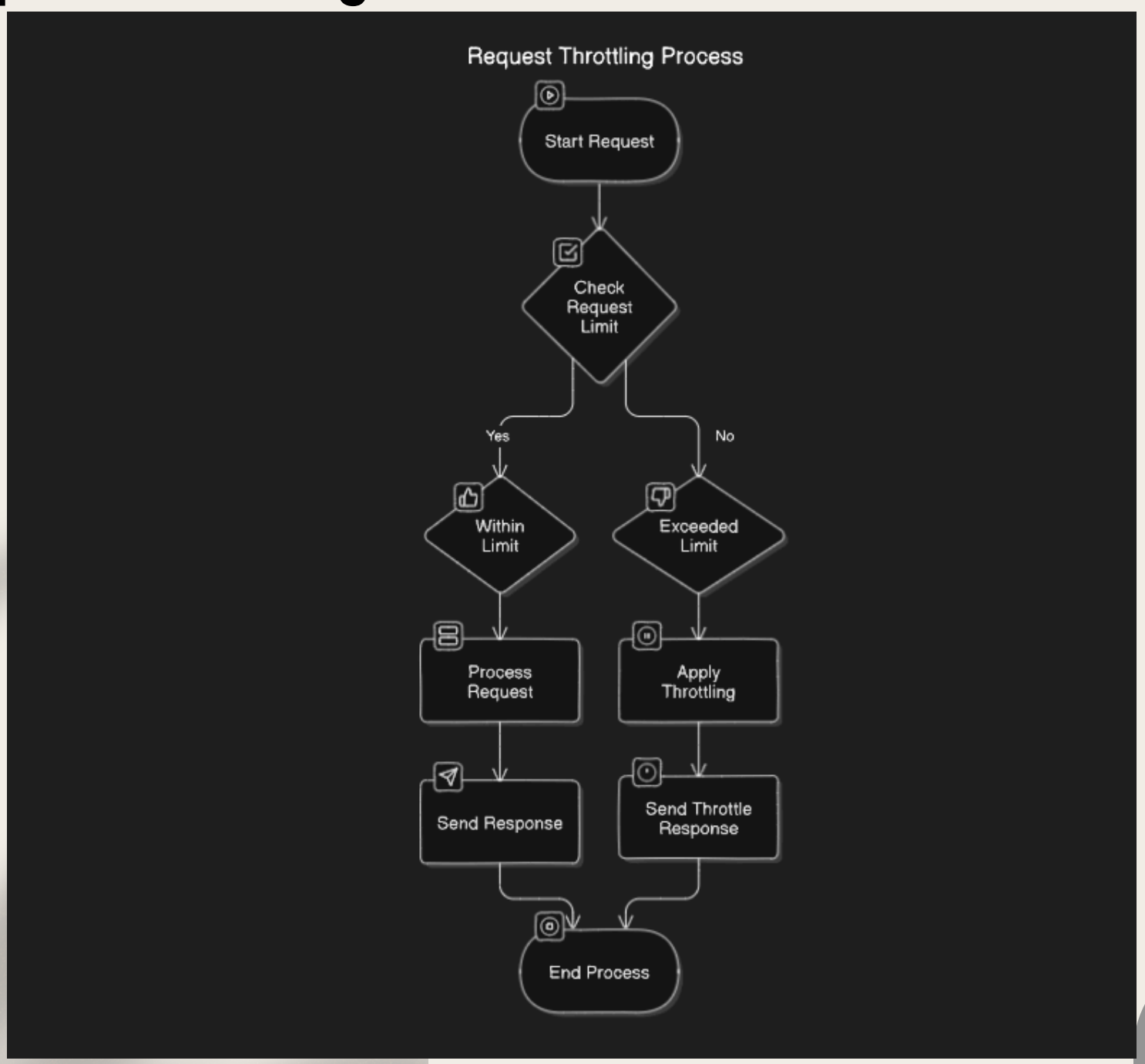
Use Case: Preventing DoS (Denial of Service) attacks and managing traffic spikes.

Code Snippet: Using **express-rate-limit** in Node.js

```
const rateLimit = require('express-rate-limit');
const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100 // limit each IP to 100 requests per windowMs
});
app.use(limiter);

```

Request Throttling Process



Tools: Kong, Express middleware

3

2.Circuit Breakers

Circuit Breakers prevent your system from making repeated calls to a failing service, improving resilience and reducing latency.

Use Case: Handling unreliable services or network issues gracefully

Code Snippet: Using **opossum** library in Node.js

```
const CircuitBreaker = require('opossum');

const breaker = new CircuitBreaker(yourFunction, {
  timeout: 3000,// If our function takes longer than 3 seconds, trigger a failure
  errorThresholdPercentage: 50,
  resetTimeout: 30000 // 30 seconds
});
breaker.fire()
  .then(response => console.log(response))
  .catch(err => console.error(err));
```

TIP 💡**:**Implement circuit breakers to avoid cascading failures in microservices architectures

Tools: You can use these tools for Circuit Breaker.
Opossum, Hystrix

3.API Gateway

An API Gateway acts as a single entry point, handling requests and routing them to appropriate microservices. It can also manage authentication, rate limiting, and logging.

Use Case: Essential in microservices architecture to manage service communication.

Code Snippet:

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: api-gateway
spec:
  rules:
  - host: example.com
    http:
      paths:
      - path: /service1
        backend:
          serviceName: service1
          servicePort: 80
      - path: /service2
        backend:
          serviceName: service2
          servicePort: 80

```

TIP 💡**:**Use API Gateways to simplify and centralize authentication for multiple services.

Tools: You can use these tools for API Gateway.
Kong, AWS API Gateway

5

4.Versioning

API Versioning helps you manage changes and updates without breaking existing clients. Common methods include URI versioning, query parameters, and custom headers.

Use Case: Managing backward compatibility when updating APIs

Code Snippet: Versioning with Express

```
const express = require('express');

  const app = express();

// Example 1
app.use('/api/v1', v1Routes);
app.use('/api/v2', v2Routes);


// Example 2
app.get('/api/v1/resource', (req, res) => {
  res.send('This is version 1 of the resource');
});

app.get('/api/v2/resource', (req, res) => {
  res.send('This is version 2 of the resource');
});
```

TIP 💡**:**Always communicate deprecation timelines clearly when sunsetting old API versions.

Tools: You can use these tools for Versioning.
Express middleware

5.Caching

Caching can significantly reduce the load on your server by storing frequently accessed data closer to the user.

Use Case: Improving response times for read-heavy APIs.

Code Snippet: Caching in Node.js with Redis

```
const redis = require('redis');
const client = redis.createClient();
app.get('/data', (req, res) => {
  client.get('key', (err, data) => {
    if (data) return res.send(data);
    // Fetch data and cache it
  });
});
```

TIP 💡**:**Leverage caching for static and infrequently updated data to reduce database load

Tools: You can use these tools for Caching. **Redis**

6.Authentication & Authorization

Securing APIs with OAuth2, JWTs, or API keys ensures that only authorized users access your services

Use Case: Protecting sensitive data and preventing unauthorized access.

Code Snippet: Using JWT with Express

```
1  import  jwt  from "jsonwebtoken";
2
3  const generateToken =(id)=>{
4      return jwt.sign({id } , process.env.JWT_SECRET , {
5          expiresIn:"30d"
6      })
7  }
8  export default generateToken
```

TIP 💡**:**Use short-lived JWT tokens and refresh tokens to enhance security

Tools: You can use these tools for Auth. **JWT , oAuth**

7.Pagination

Pagination helps break down large sets of data into manageable chunks, improving API performance and user experience

Use Case: When returning lists of data like user records, search results, etc

Code Snippet: Pagination with MongoDB

```
app.get('/users', async (req, res) => {
    const page = req.query.page || 1;
    const limit = req.query.limit || 10;
    const users = await User.find()
      .skip((page - 1) * limit)
      .limit(limit);
    res.json(users);
  });
```

TIP 💡**:**Provide total records count along with paginated data for a better client experience.

Tools: You can use these tools for Pagination.
Express middleware

9

Call to Action

- Implementing these patterns can take your REST API's scalability and security to the next level. Start with one and see the difference!
- Comment below with your favorite pattern or the one you find most challenging!

Resources & Tools

Tools to help you implement these patterns: **Kong, Express Middleware, Redis, JWT , AWS API Gateway ,Opossum** and **Hystrix** libraries.

SHARE THIS INSIGHT NOW

ALI AHMAD
@aliahmad14