

GRPC

Microservices with gRPC [Java + Spring Boot + Protobuf]



Contents

[1. Code Smells 2](#_Toc127976312)

[1.1. Rigidity 2](#_Toc127976313)

[1.2. Fragility 2](#_Toc127976314)

[1.3. Immobility 2](#_Toc127976315)

[1.4. Viscosity 2](#_Toc127976316)

[1.5. Needless Complexity 2](#_Toc127976317)

[1.6. Needless Repetition 2](#_Toc127976318)

[1.7. Opacity 2](#_Toc127976319)

[2. EMS (Employee Management System) 2](#_Toc127976320)

[3. SRP (Single Responsibility Principle) 2](#_Toc127976321)

[3.1. Cohesion 2](#_Toc127976322)

[3.2. Coupling 3](#_Toc127976323)

[4. OCP 3](#_Toc127976324)

[5. LSP 3](#_Toc127976325)

[6. ISP 3](#_Toc127976326)

[7. DIP 3](#_Toc127976327)

# HTTP2 vs HTTP1

HTTP2 and HTTP1 are different versions of the Hypertext Transfer Protocol (HTTP), which is the protocol used for communication between web browsers and servers. Here are some key differences between HTTP2 and HTTP1:

## Protocol Structure

HTTP1 uses a textual format where requests and responses are sent as plain text messages. In contrast, HTTP2 utilizes a binary framing layer, which allows for more efficient and optimized transmission of data

## Multiplexing

One of the significant improvements in HTTP2 is the support for multiplexing. In HTTP1, browsers are limited to making a limited number of simultaneous requests to a server, which can cause delays and inefficiencies. HTTP2, however, allows multiple requests and responses to be sent over a single TCP connection concurrently, improving performance and reducing latency.

## Server Push

HTTP2 introduces a server push feature, which allows the server to proactively send additional resources (such as images, scripts, or stylesheets) to the client without waiting for a request. This can lead to faster page load times and improved performance.

## Header Compression

HTTP2 includes header compression mechanisms, which significantly reduce the overhead of transmitting headers in each request and response. This helps to improve efficiency and decrease network latency.

## Prioritization

HTTP2 introduces the ability to assign priority levels to different requests, enabling the server and client to prioritize which resources should be loaded first. This can enhance the user experience by ensuring critical resources are delivered promptly.

## Backward Compatibility

HTTP2 is designed to be fully backward compatible with HTTP1. This means that if a client or server doesn't support HTTP2, the communication can still happen using HTTP1.

Overall, HTTP/2 offers several performance enhancements over HTTP/1, such as faster page load times, reduced latency, improved bandwidth utilization, and better handling of concurrent requests. However, adoption of HTTP/2 requires support from both the web server and the client (typically the browser), so its full benefits may not be realized on older systems or outdated software.

# gRPC VS REST

gRPC and REST are two different approaches for implementing client-server communication in distributed systems. Here are some key differences between gRPC and REST:

## Protocol and Message Format

gRPC uses the Protocol Buffers (protobuf) protocol for defining services and message types. Protobuf is a language-agnostic binary serialization format that offers efficient and compact data representation. REST, on the other hand, typically uses JSON or XML for message formatting, which is a textual format.

## Communication Style

gRPC is based on the Remote Procedure Call (RPC) communication model. It allows clients to directly call methods or functions on the server, making it easier to define and work with APIs. REST, in contrast, follows the principles of Representational State Transfer and uses standard HTTP methods (GET, POST, PUT, DELETE) to perform actions on resources.

## Transport Protocol

gRPC uses HTTP/2 as its underlying transport protocol, which brings several performance benefits such as multiplexing, header compression, and server push. REST can use either HTTP/1.1 or HTTP/2 as the transport protocol

## Language Support

gRPC provides support for multiple programming languages, including popular ones like C++, Java, Python, Go, and more. REST, being based on HTTP, is language-agnostic and can be used with any programming language that can make HTTP requests.

## Payload Size and Efficiency

gRPC typically sends binary data using protobuf serialization, which results in smaller payload sizes compared to REST with JSON or XML. This can be advantageous in scenarios where bandwidth is limited or network efficiency is critical.

## Error Handling

gRPC uses a rich status code system that allows for detailed error reporting, including specific error messages and error metadata. REST, on the other hand, often relies on HTTP status codes and textual error messages.

## Tooling and Ecosystem

REST has been widely adopted and has a mature ecosystem with numerous tools, libraries, and frameworks available. gRPC is newer but growing in popularity and also has a growing ecosystem, although it may not have the same breadth of options as REST.

The choice between gRPC and REST depends on various factors such as the specific requirements of the project, performance considerations, interoperability needs, and existing infrastructure. REST is more established and fits well with the web architecture, while gRPC offers better performance and efficiency but may require additional setup and adoption efforts.

# SOLID

All principles are closely connected and interdependent. They are most effective when they combined together.

SOLIC principles complement each other and work together in unison, to achieve the common purpose of well-designed software.

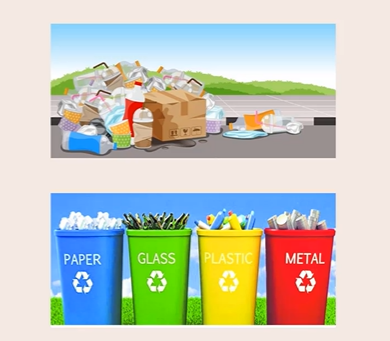
# SRP (Single Responsibility Principle)

Every software component should have one and only one responsibility (reason to change).

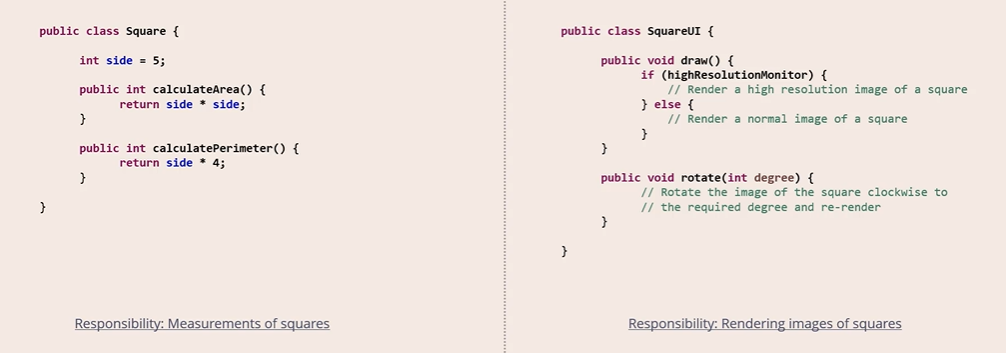
Component can be a class, a method or a module.

## Cohesion

Is the degree to which the various parts of a software component are related.



**Higher Cohesion helps achieve better adherence to the SRP.**



## Coupling

Is defined as the level of inter dependency between various software components.

**Loose Coupling helps achieve better adherence to the SRP.**

****

# OCP (Open Closed Principle)

Software component should be closed for modification but open for extension.

Software component should be extendable to add new feature or behaviour but without Modifying existing code. Sometimes we have redesign or revamp component as well.

Very easy to add new features. And required minimum development and testing as only new feature is required to test only as it is not changing to existing code and features.

OCP mostly required decoupling, which automatically follow SRP.

Do not follow blindly OCP sometime for resolving bug, it is better to made existing code changes over the revamp code design. if only if same type of bug occurrence then go with revamp if it is suitable otherwise we will have a lot of classes that complicate overall design.

It is subjective decision rather than objective to apply OCP.

# LSP (Liskov Substitution Principle)

Object should be replaceable with their subtypes without affecting the correctness of the program.

Car is interface and hatchback is a car implementation. But racing car does not fully implement car interface so it not **replaceable**. as racing car does not have cabin as it has cockpit only. So better create new class Vehicle who is parent of both car and racing car. By rule **Break the hierarchy** if it fails the substitution test.

Second Rule is ‘**Tell, Don’t Ask’**

# ISP (Interface Segregation Principle)

No client should not be forced to depend on methods it does not use.

Avoid below because they lead to violation of ISP  
Fat interfaces

Interfaces with low cohesion

Empty Method Implementation

# DIP (Dependency Inversion Principle)

High-level Modules should not depend on low-level modules. Both should depend on abstractions.

Abstractions should not depend on details. Details should depend on abstractions.