Backend Developer Roadtrip

Random Collection of Knowledge

EDUARDO CARDENAZ

Part I: Basics

1 APIs

Application Programming Interface (API) An API is basically an intermediary that allows two applications to talk to each other.

It's useful to think of API communication in terms of requests and responses between a client and a server. The application submitting the request is the client, and the server provides the response.

API Specification Is a document or standard that describes how to build or how to use an API. A system that meets this standard its said to be *implementing* or *exposing* an API. The term API can refer to both the implementation and the specification.

1.1 RESTful APIs

"The design rationale behind the Web architecture can be described by an architectural style consisting of the set of constraints applied to elements within the architecture." source

To understand REST, we'll be expanding and building on top of each constraint that composes this architecture.

Violating any constraint other than Code on Demand means that the service is not strictly RESTful.

1.1.1 Uniform Interface

Resource Based Individual resources are identified using URIs as resource identifiers. The resources themselves are conceptually different from the *representation* that is returned to the client. For example, the server doesn't return its database but rather some HTML, JSON or XML that represents some database records.

Manipulation of Resources Through Representations When a client holds a representation of a resource given by the server, including any metadata attached to it, it should have enough information to modify or delete said resource on the server, given it has the right permissions.

Self-Descriptive Messages Each message includes all the necessary information to describe how to handle that message.

1.1.2 Client-Server

Separation of Concerns is the principle behind this constraint. By separating the user interface (UI) concerns from the data storage concerns, we improve the portability of the UI across multiple platforms.

1.1.3 Stateless

This constraint stablish that the interaction between the client and the server must be stateless in nature, by that, meaning that any request made by the client must contain all the necessary information so that the server can understand the request. It shall not take advantage of any stored context.

Stateless Imagine that you're buying coffee at a shop. Each time you want to order a coffee you need to tell the cashier exactly what you want and how you want it. The cashier doesn't remember you or what you ordered the last time you went there. Each visit is like starting from scratch.

Stateful The cashier remembers you and remember what you usually ask for as you're a frequent client. So, you could say 'the usual' and they'd know exactly what you want.

1.1.4 Cacheable

Clients should can cache responses. Responses must, implicitly or explicitly, define themselves as cacheable or not. Clients should be able to negotiate wether to cache or not to prevent reusing stale or inappropriate data in response to further requests.

1.1.5 Layered System

A client cannot tell whether its connected to the main server or an intermediary along the way. The layered system style allows an application to be composed of hierarchical layers by constraining component behavior such that each component can't see beyond the immediate layer with which they are interacting.

1.1.6 Code on Demand

This is a kind-of unique thing about RESTful systems. Code on Demand is optional. Basically means that a server can temporarily extend functionality to a client by transferring logic to the client. As an example, a request can return client-side scripts such as Javascript code.

1.2 JSON APIs

1.3 SOAP APIs

1.4 GraphQL APIs

1.5 gRPC APIs

1.6 Authentication

1.6.1 JWT

1.6.2 Basic Auth

1.6.3 Token Auth

1.6.4 OAuth

1.6.5 Cookie Based

1.6.6 OpenID & SAML

2 Caching

2.1 Client Side

2.2 Server Side

2.3 Content Delivery Network (CDN)

2.4 Redis

2.5 Memcached

3 Web Security

3.1 Hashing Algorithms

3.2 API Security Best Practices

4 Testing

4.1 Unit Testing

4.2 Integration Testing

4.3 Functional Testing

5 CI/CD

6 Databases

7 Software Design & Architecture

7.1 Design and Development Principles

7.1.1 Separation of Concerns

7.1.2 Reusability

7.1.3 Keep It Simple Stupid (KISS)

7.1.4 Don't Repit Yourself (DRY)

7.1.5 Scalability

7.1.6 Security

7.2 GOF Design Patterns

7.3 Domain Driven Design

7.4 CQRS

7.5 Event Sourcing

8 Architectural Patterns

9 Message Brokers

9.1 RabbitMQ

9.2 Kafka

10 Containerization Vs Virtualization

10.1 LXC

10.2 Docker

10.3 Kubernetes

10.4 Elasticsearch

10.5 Solr

11 Web Servers

11.1 Server Sent Events

11.2 WebSockets

11.3 Long Polling

11.4 Short Polling

12 GraphQL

12.1 Apollo

13 NoSQL Databases

13.1 Document DBs - MongoDB

13.2 Time Series - InfluxDB

13.3 Realtime - Firebase

13.4 Column DBs - Cassandra

13.5 Key Value - Redis

13.6 Graph DBs - Neo4j

14 Building for Scale

14.1 Difference + Usage

14.2 Mitigation Strategies

Part II: Infraestructure Knowledge

15 Go Programming Language

15.1

15.2

16 Networking and Protocols

16.1

16.2

17 Docker

17.1

17.2

18 Amazon Web Services

18.1

18.2

19 Terraform

19.1

19.2

20 Ansible

20.1

20.2

21 Github Actions

21.1

21.2