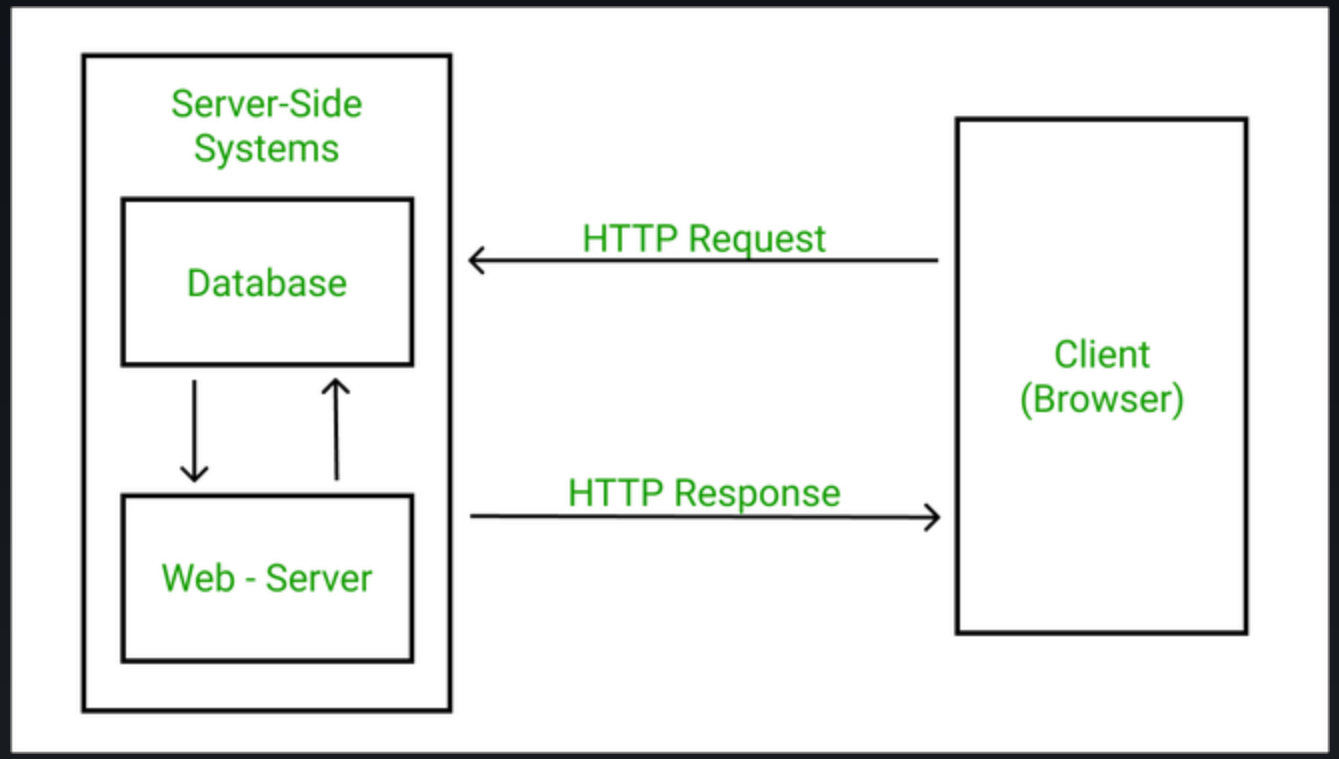
O que são APIs

O acrônimo API significa Application Programming Interface (em português, Interface de Programação de Aplicações). Uma API é um conjunto de **padrões e regras documentadas** para que uma aplicação X possa utilizar funcionalidades de uma aplicação Y sem precisar conhecer os detalhes da implementação dessa aplicação X.

Vamos imaginar o dia-a-dia de uma empresa que possui um e-commerce. Os desenvolvedores que trabalham na solução tem como objetivo criar a dinâmica da loja, como criar, atualizar, deletar os produtos internamente e mostrar os produtos para os clientes. Essas funcionalidades podem ser criadas em uma aplicação do lado do servidor como se fosse uma API, de forma que o site do e-commerce possa usar essas informações.

Agora que já sabemos o que é uma API e para que utilizamos uma, precisamos entender os protocolos que são utilizadas para a comunicação entre as aplicações e como os dados trafegados podem ser representados.



HTTP e REST são a mesma coisa?

O principal protocolo de comunicação na *Web* é o **HTTP**. Ele funciona como um protocolo de requisição-resposta em um modelo que chamamos de cliente-servidor. No exemplo do e-commerce, o navegador que é usado para acessar o *site* seria o cliente e o computador ou máquina virtual em algum serviço de *cloud* em que a **API** está hospedada é o servidor.

Assim, o cliente manda uma **requisição HTTP** para o servidor e o servidor, com recursos e conteúdos próprios, retorna uma mensagem de **resposta** para o cliente.

O protocolo HTTP tem sido usado desde 1990 e a versão atual do protocolo é o HTTP/3. O protocolo define oito métodos que determinam ações a serem efetuadas no momento da requisição de algum recurso ao servidor.

De 8 métodos 4 deles são os mais utilizados:

- **GET**: método utilizado para ler e recuperar dados. Requisita uma representação do recurso especificado e retorna essa representação.
- **POST:** método utilizado para criar um novo recurso. Envia dados ao servidor. O tipo do corpo da solicitação é indicado pelo cabeçalho `[Content-Type](https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Headers/Content-Type)`.

- **PUT:** cria um novo recurso ou substitui uma representação do recurso de destino com os novos dados. A diferença entre `PUT` e `POST` é que `PUT` é idempotente: ao chamá-lo uma ou várias vezes sucessivamente o efeito é o mesmo, enquanto se chamar o `POST` repetidamente pode ter efeitos adicionais.

- **PUT**

Por exemplo, se criarmos um produto com `POST`, se a URL definida na API for chamada 20 vezes, 20 produtos serão criados e cada um deles terá um ID diferente. Já o com o `PUT` se você executar a URL definida na API 20 vezes, o resultado **tem que ser** o mesmo: o mesmo produto atualizado 20 vezes.

- **DELETE:** exclui o recurso.

Códigos de resposta

Baseado nesses métodos , o servidor deve processar cada uma das requisições e retornar uma resposta adequada. O conteúdo da resposta pode estar no formato **XML**, **JSON**, **YAML**, ****texto, ****dentre outros. E essas as respostas são separadas em cinco grupos

- **1XX** — Informações Gerais
- **2XX** — Sucesso
- **3XX** — Redirecionamento
- **4XX** — Erro no cliente
- **5XX** — Erro no servidor

Status codes e seus detalhes
https://http.dog/
https://http.cat/

Mas isso não é **REST! REST** é acrônimo de **Representational State Transfer**, é uma abstração dessa arquitetura que detalhamos acima. É um estilo de arquitetura de software que define uma série de restrições para a criação de *web services* (serviços *Web*), ou seja, restringe como seus componentes devem interagir entre si.

Artigo Fonte: https://mari-azevedo.medium.com/construindo-uma-api-restful-com-java-e-spring-framework-46b74371d107

Link para estudo: https://www.redhat.com/pt-br/topics/api/what-is-a-rest-api

CRUD

CRUD é uma sigla criada para as operações:

- CREATE
- READ
- UPDATE
- DELETE

Essas operações podem ser feitas em memória, em arquivos ou em um banco de dados.
A ideia é que as rotas de nossa API chamem os comandos específicos para as ações que desejamos executar

Objetivos da disciplina

- Entender o modelo de requisição e resposta
- Criar nosso próprio servidor e nossa API
- Realizar a integração da nossa aplicação com uma base de dados
- Aprender o que é e como funciona o Node.js
- Aprender o que é o paradigma assincrono
- Aprender a criar uma aplicação com Node.js e Typescript sem frameworks
- Consumir uma API externa
- Como tratar nossos dados e métodos de manipulação de Arrays
- Como estruturar nossa aplicação no Modelo MVC
- Recriar nossa aplicação utilizando Express e realizar os devidos testes