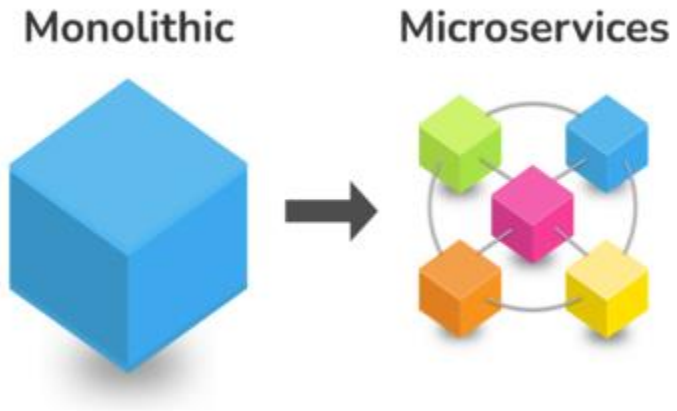
COTI INFORMÁTICA
ESCOLA DE NERDS

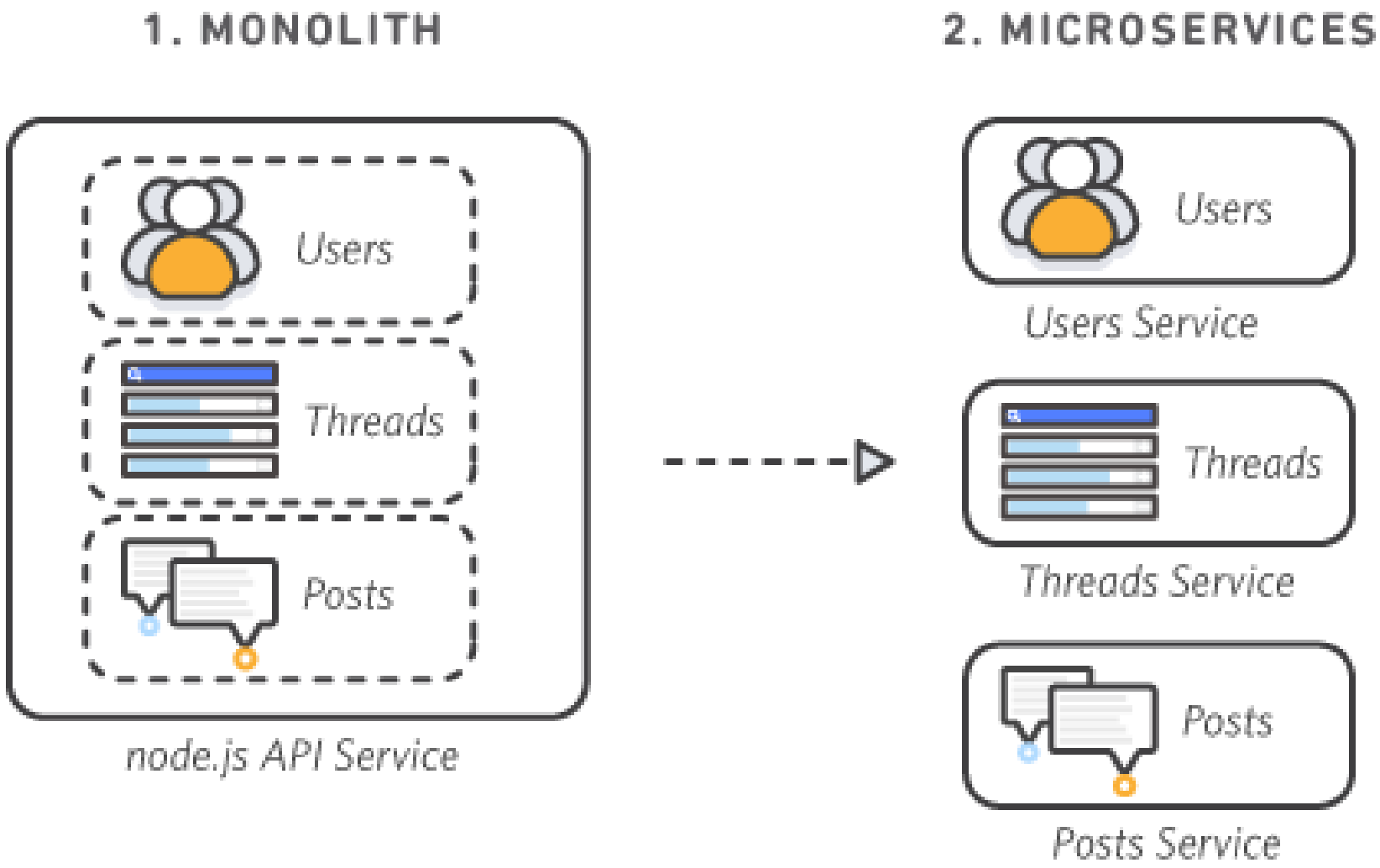
Java WebDeveloper
Formação FullStack

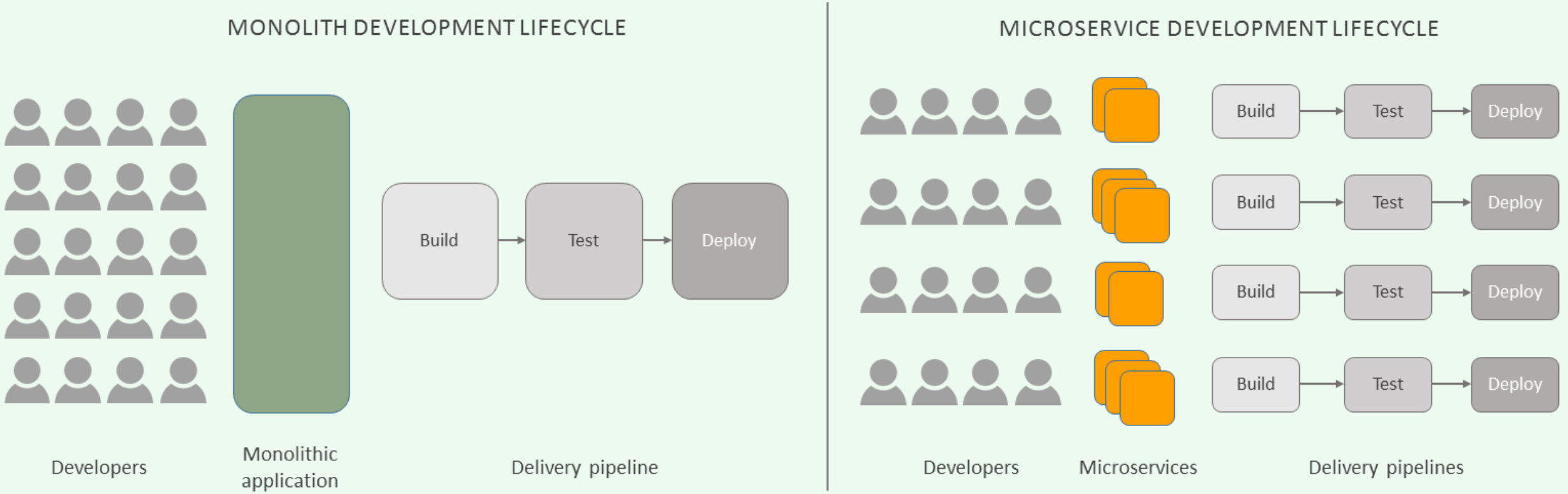
Professor Sergio Mendes

Aula 15 (27/02/23)

O que são **Microserviços?**




1. MONOLITH
2. MICROSERVICES
Users
Threads
Posts
node.js API Service
Users
Users Service
Threads
Threads Service
Posts
Posts Service

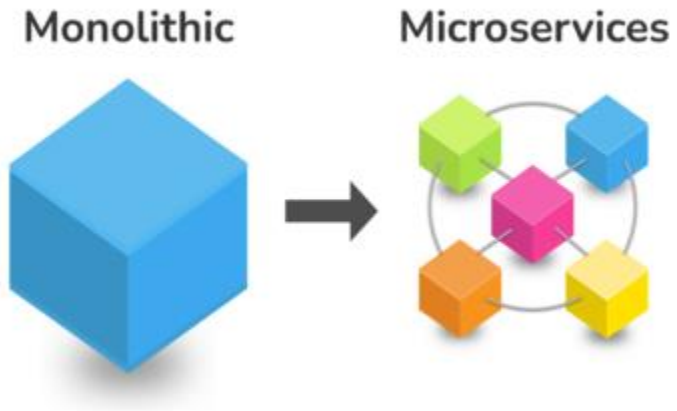

MONOLITH DEVELOPMENT LIFECYCLE
Build
Test
Deploy
Developers
Monolithic application
Delivery pipeline
MICROSERVICE DEVELOPMENT LIFECYCLE
Build
Test
Deploy
Build
Test
Deploy
Build
Test
Deploy
Build
Test
Deploy
Developers
Microservices
Delivery pipelines

Microsserviços são uma abordagem arquitetônica e organizacional do desenvolvimento de software na qual o software consiste em pequenos serviços independentes que se comunicam usando APIs bem definidas.

Esses serviços pertencem a pequenas equipes autossuficientes.
As arquiteturas de microsserviços facilitam a escalabilidade e agilizam o desenvolvimento de aplicativos, habilitando a inovação e acelerando o tempo de introdução de novos recursos no mercado.

COTI INFORMÁTICA
ESCOLA DE NERDS

Quais são as características dos **Microserviços**?



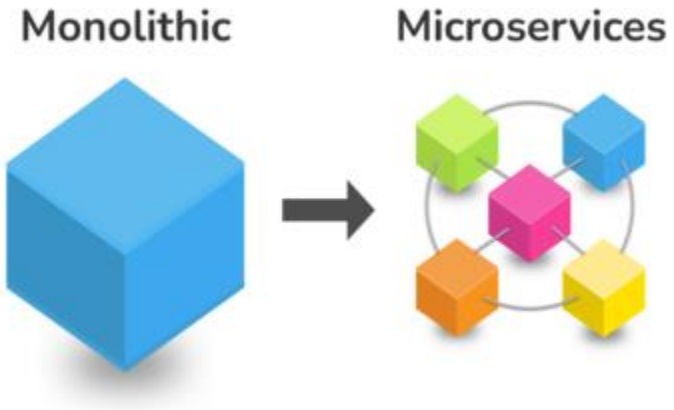
Autônomos

Cada serviço do componente de uma arquitetura de microsserviços pode ser desenvolvido, implantado, operado e escalado sem afetar o funcionamento de outros serviços. Os serviços não precisam compartilhar nenhum código ou implementação com os outros serviços. Todas as comunicações entre componentes individuais ocorrem por meio de APIs bem definidas.

Especializados

Cada serviço é projetado para ter um conjunto de recursos e é dedicado à solução de um problema específico. Se os desenvolvedores acrescentarem mais código a um serviço ao longo do tempo, aumentando sua complexidade, ele poderá ser dividido em serviços menores.

Quais são os benefícios dos **Microserviços**?



Agilidade

Os microsserviços promovem uma organização de equipes pequenas e independentes que são proprietárias de seus serviços. As equipes atuam dentro de um contexto pequeno e claramente compreendido e têm autonomia para trabalhar de forma mais independente e rápida. O resultado é a aceleração dos ciclos de desenvolvimento.

Escalabilidade flexível

Os microsserviços permitem que cada serviço seja escalado de forma independente para atender à demanda do recurso de aplicativo oferecido por esse serviço. Isso permite que as equipes dimensionem corretamente as necessidades de infraestrutura, meçam com precisão o custo de um recurso e mantenham a disponibilidade quando um serviço experimenta um pico de demanda.

Fácil implantação

Os microsserviços permitem a integração e a entrega contínuas, o que facilita o teste de novas ideias e sua reversão caso algo não funcione corretamente. O baixo custo de falha permite a experimentação, facilita a atualização do código e acelera o tempo de introdução de novos recursos no mercado.

Liberdade tecnológica

As arquiteturas de microsserviços não seguem uma abordagem generalista. As equipes são livres para escolher a melhor ferramenta para resolver problemas específicos. O resultado é que as equipes que criam microsserviços podem optar pela melhor ferramenta para cada tarefa.

Código reutilizável

A divisão do software em módulos pequenos e bem definidos permite que as equipes usem funções para várias finalidades. Um serviço criado para uma determinada função pode ser usado como componente básico para outro recurso. Isso permite que os aplicativos sejam reutilizados, pois os desenvolvedores podem criar recursos sem precisar escrever código.

Resiliência

A independência do serviço aumenta a resistência a falhas do aplicativo. Em uma arquitetura monolítica, a falha de um único componente poderá causar a falha de todo o aplicativo. Com os microsserviços, os aplicativos lidam com a falha total do serviço degradando a funcionalidade, sem interromper todo o aplicativo.

COTI INFORMÁTICA
ESCOLA DE NERDS

O que é o **Spring Boot**?

O **Spring Boot** é um framework que torna fácil a criação de aplicações Spring autossuficientes e robustas, possibilitando a execução imediata. Contudo isso só é possível por conta da abordagem opinativa sobre a plataforma Spring e bibliotecas de terceiros, que permite ao desenvolvedor gastar o mínimo de tempo possível configurando o projeto, e sim codificando suas regras de negócio.

Para cumprir com esse propósito, o framework se baseia em quatro princípios centrais:

1. Oferecer uma experiência de início de projeto rápida e direta;
2. Apresentar uma visão opinativa e flexível sobre o modo como os projetos Spring devem ser configurados;
3. Fornecer requisitos não funcionais pré-configurados;
4. Não prover geração de código e zerar a necessidade de arquivos XML.

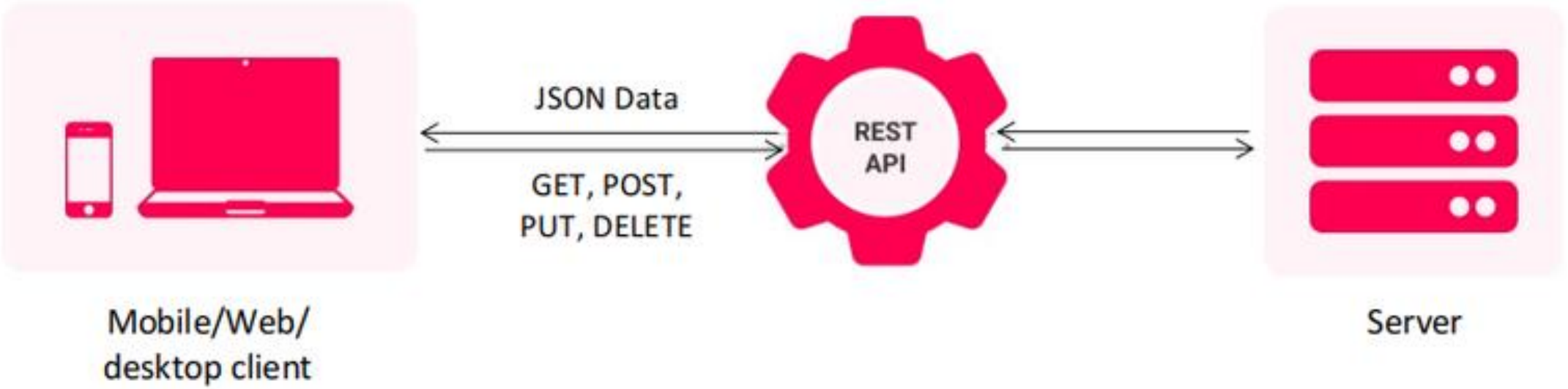
COTI INFORMÁTICA
ESCOLA DE NERDS

APIs **REST**

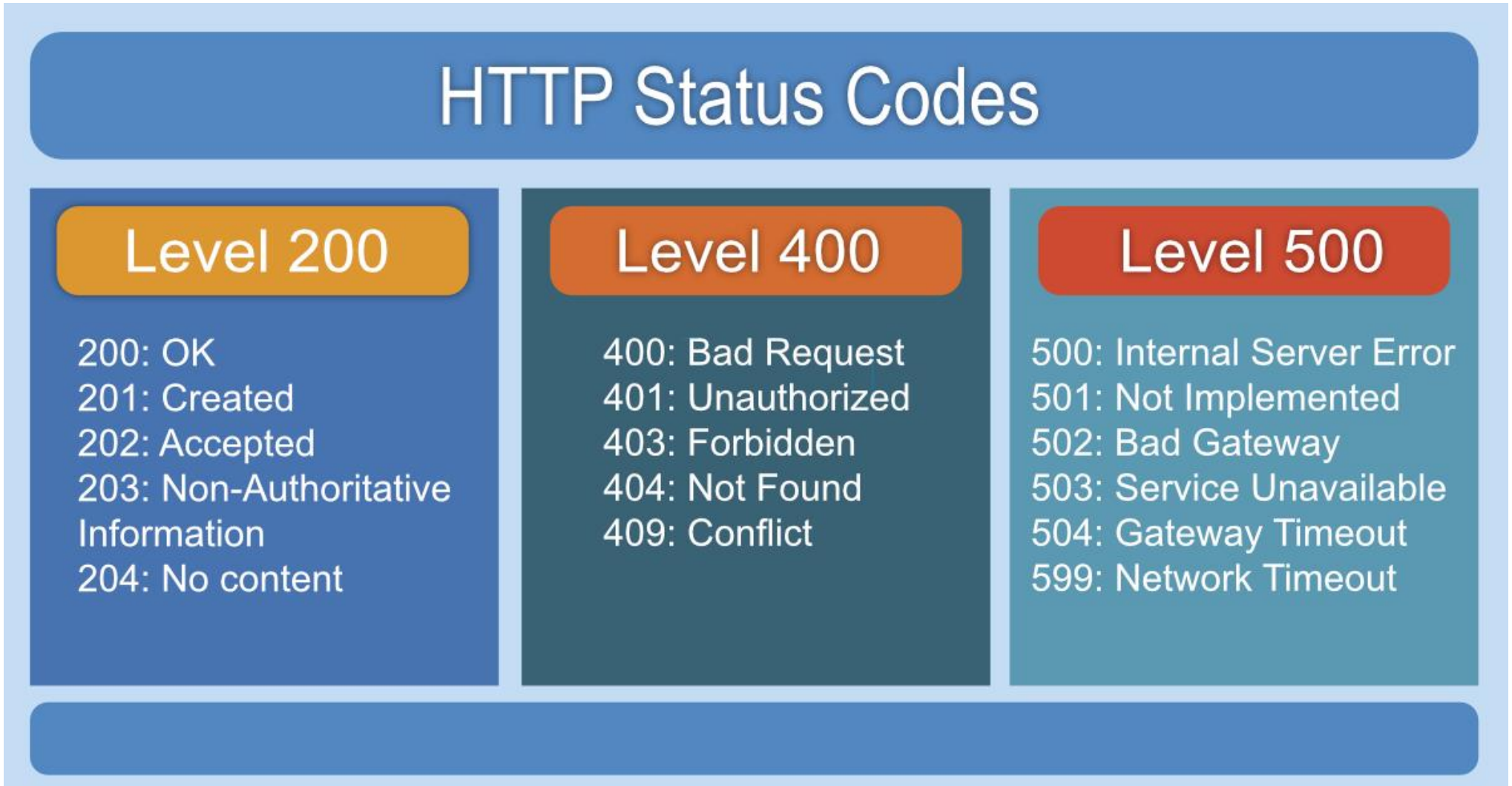
Java WebDeveloper - Formação FullStack| Professor: Sergio Mendes

Uma **API** (**Interface de Programação de Aplicações**, na sigla em inglês), é um conjunto de padrões e protocolos que integram um usuário a uma aplicação, permitindo que ele acesse e faça uso das funcionalidades do *software* em questão. Uma API funciona como um mediador, ou comunicador, entre o usuário e o sistema. Deste modo, ela facilita o acesso e o desenvolvimento de aplicações para a internet.

A abreviatura **REST** se refere a ***Representational State Transfer*** (Transferência de Estado Representacional) e é um tipo de arquitetura de *software*. Uma REST indica então um conjunto de restrições que devem ser seguidas no desenvolvimento de uma aplicação na internet. Estas regras permitem o desenvolvimento de uma aplicação com interface bem definida, com rotinas padronizadas e facilmente representadas, que facilitam a comunicação entre máquinas e usuários.



Em termos de nomenclatura, é importante sabermos a diferença entre os conceitos de REST e RESTful. Como já definimos anteriormente, REST é um conjunto de princípios e restrições de arquitetura de *softwares*.
Uma API RESTful é aquela que está em conformidade com os critérios estabelecidos pela Transferência de Estado Representacional (REST).


HTTP Status Codes
Level 200
200: OK
201: Created
202: Accepted
203: Non-Authoritative Information
204: No content
Level 400
400: Bad Request
401: Unauthorized
403: Forbidden
404: Not Found
409: Conflict
Level 500
500: Internal Server Error
501: Not Implemented
502: Bad Gateway
503: Service Unavailable
504: Gateway Timeout
599: Network Timeout

COTI INFORMÁTICA
ESCOLA DE NERDS

Para que serve o **Swagger**?

Open API
Specification

Swagger

O **Swagger** é um framework composto por diversas ferramentas que, independente da linguagem, auxilia a descrição, consumo e visualização de serviços de uma API REST.

A **especificação** da API consiste em determinar os modelos de dados que serão entendidos pela API e as funcionalidades presentes na mesma. Para cada funcionalidade, é preciso especificar o seu nome, os parâmetros que devem ser passados no momento de sua invocação e os valores que irão ser retornados aos usuários da API.

Com o Swagger UI, a partir da especificação da API, podemos criar documentações elegantes e acessíveis ao usuário, permitindo assim uma compreensão maior da API, pois além de poder ver os endpoints e modelos das entidades com seus atributos e respectivos tipos, o módulo de UI possibilita que os usuários da API interajam intuitivamente com a API

COTI INFORMÁTICA
ESCOLA DE NERDS

Reduzindo a verbosidade do código com o **Lombok**

O **Lombok** é um framework para Java que permite escrever código eliminando a verbosidade. Seu uso permite gerar em tempo de compilação os métodos getters e setters, métodos construtores e muito mais.

O Projeto Lombok é uma ferramenta de biblioteca Java usada para minimizar / remover o código clichê e economizar o tempo precioso dos desenvolvedores durante o desenvolvimento, usando apenas algumas anotações. Além disso, também aumenta a legibilidade do código-fonte e economiza espaço. Mas você deve estar pensando que hoje em dia todo mundo usa IDEs que oferecem uma opção para gerar esses códigos clichê, então qual é o uso do Lombok.

Sempre que usamos IDEs para gerar esses códigos clichê, apenas evitamos escrever todos esses códigos, mas na verdade eles estão presentes em nosso código-fonte e aumentam o LOC (linhas de código) e reduzem a capacidade de manutenção e a legibilidade. Por outro lado, o Lombok adiciona todos esses códigos clichê no tempo de compilação no arquivo “.class” e não em nosso código-fonte.

COTI INFORMÁTICA
ESCOLA DE NERDS

Mapeamento ORM com **Spring Data**

Spring Data

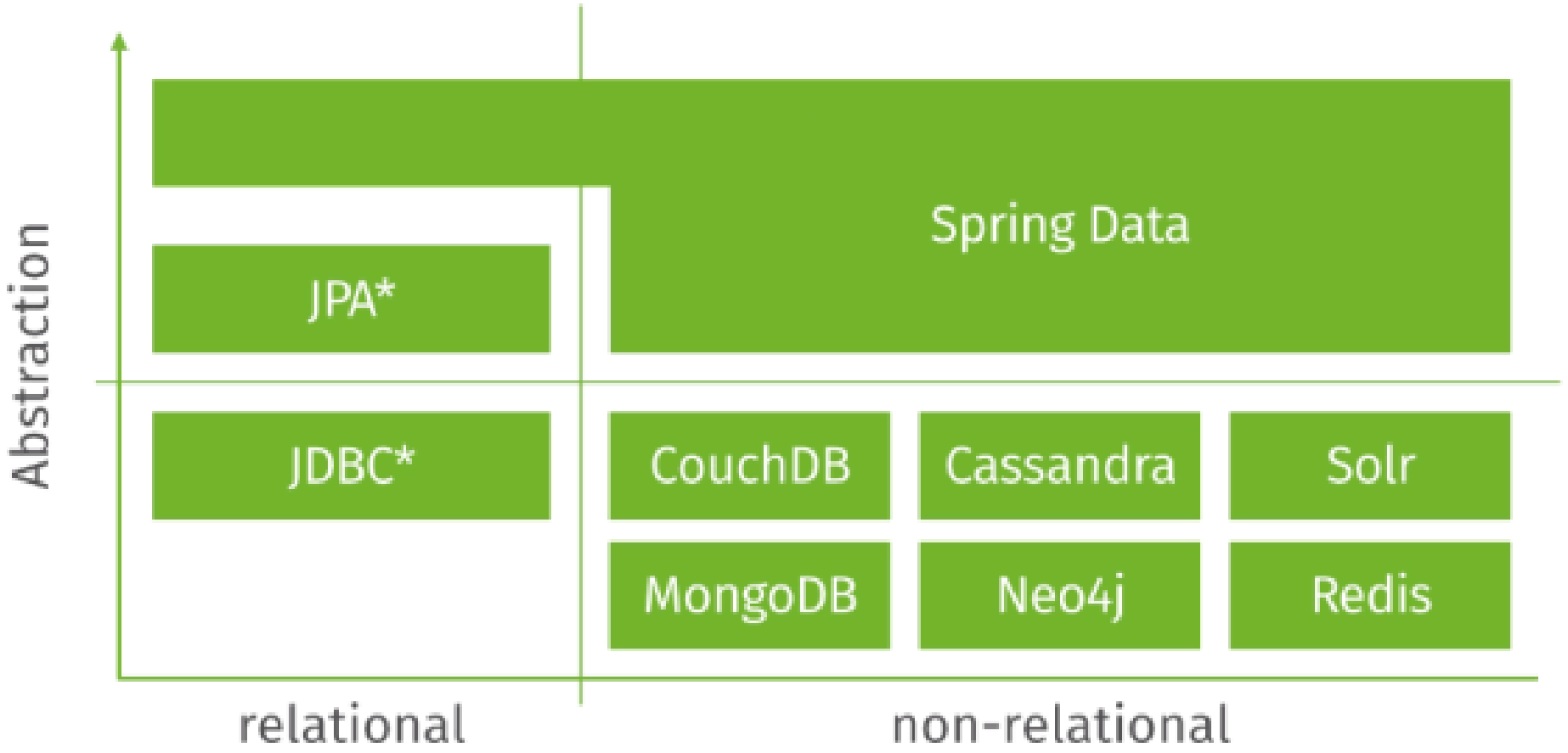
O Spring Data é o modelo de programação dentro do **Spring Framework** para acesso e manipulação de dados.

Chegando com a intenção de facilitar a configuração e utilização com o seu banco de dados, seja ele relacional ou não, o Spring Data traz vários recursos bacanas para acelerar o nosso desenvolvimento. Por exemplo, a configuração padronizada, onde devemos colocar apenas algumas propriedades e ele já vai saber o que fazer.

Outro recurso importante é a criação de *query* pela assinatura do método, falando desse jeito pode ficar um pouco confuso, mas na prática acaba sendo muito simples de utilizar.

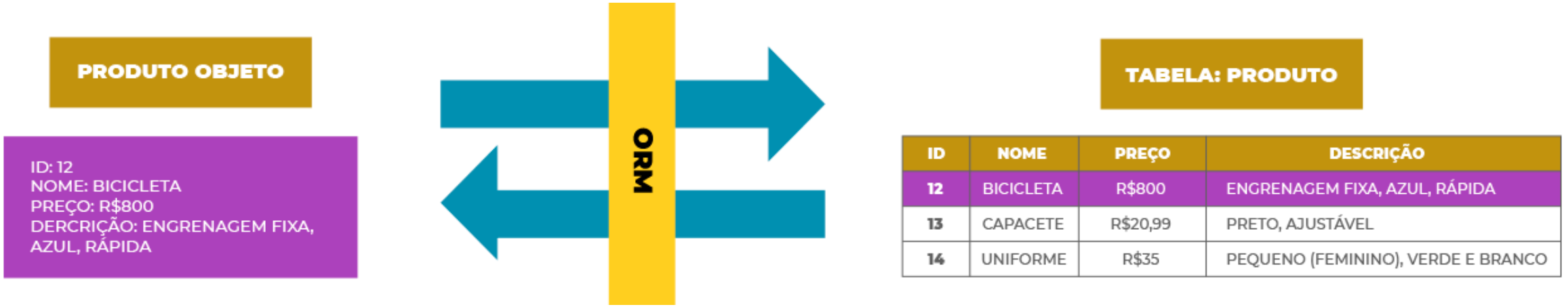
O Spring Data é composto de 3 subprojetos:
- Spring Data JPA;
- Spring MongoDB;
- Spring Data Redis.

Spring Data


Abstraction
Spring Data
JPA*
JDBC*
CouchDB
Cassandra
Solr
MongoDB
Neo4j
Redis
relational
non-relational

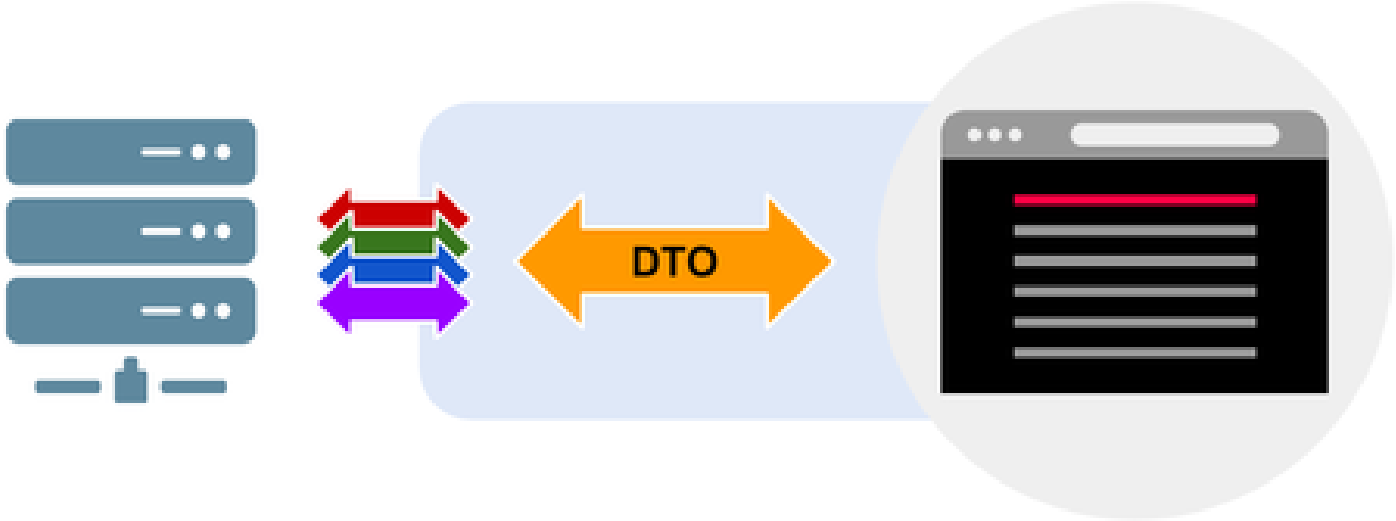
ORM – Mapeamento Objeto Relacional

Object-Relational Mapping (ORM), em português, mapeamento objeto-relacional, é uma técnica para aproximar o paradigma de desenvolvimento de aplicações orientadas a objetos ao paradigma do banco de dados relacional. O uso da técnica de mapeamento objeto-relacional é realizado através de um mapeador objeto-relacional que geralmente é a biblioteca ou framework que ajuda no mapeamento e uso do banco de dados.



COTI INFORMÁTICA
ESCOLA DE NERDS

Utilizando o padrão
DTO Data Transfer Object



Data Transfer Object (DTO) ou simplesmente ***Transfer Object*** é um padrão bastante usado em Java para o transporte de dados entre diferentes componentes de um sistema, diferentes instâncias ou processos de um sistema distribuído ou diferentes sistemas via serialização.

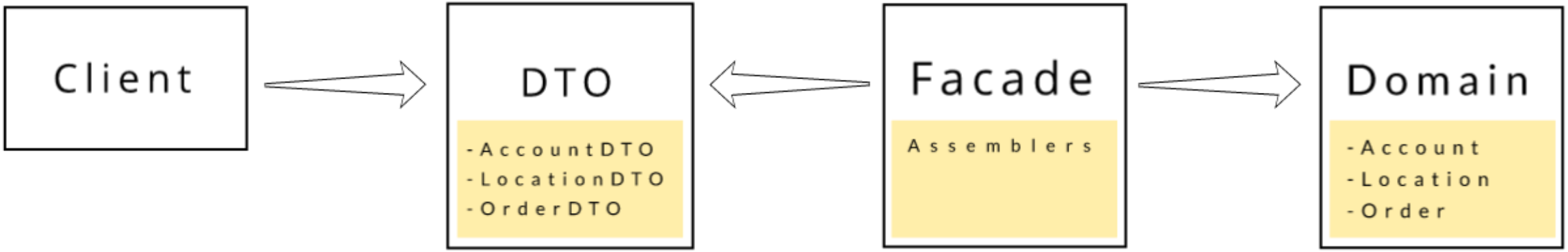
A ideia consiste basicamente em agrupar um conjunto de atributos numa classe simples de forma a otimizar a comunicação.

Numa chamada remota, seria ineficiente passar cada atributo individualmente. Da mesma forma seria ineficiente ou até causaria erros passar uma entidade mais complexa.

Além disso, muitas vezes os dados usados na comunicação não refletem exatamente os atributos do seu modelo. Então, um DTO seria uma classe que provê exatamente aquilo que é necessário para um determinado processo.

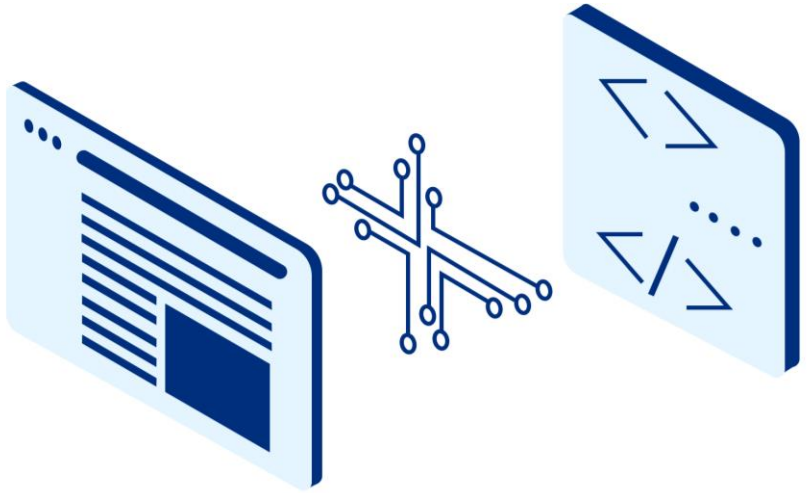
O padrão de projeto DTO é muito útil tanto para receber dados quanto para enviá-los, pois podemos manipular da forma que quisermos tais dados para facilitar a comunicação entre o servidor e o cliente.

Quando se está criando uma API é de extrema importância não apenas pensar como um usuário regular irá interagir, mas também se defender contra usuários maliciosos para que eles não consigam causar nenhum prejuízo para a sua aplicação e para seus usuários.



O padrão *DTO* busca otimizar a comunicação entre as camadas cliente e servidor da *API*, de modo que a Deserialização dos dados se faça por uma maneira simples concisa.

Com intuito de utilizar o padrão *DTO* para transferência de dados de uma forma desacoplada e de fácil interação no SpringBoot é interessante fazer uso do ModelMapper, que é um framework que realiza o mapeamento de modelos de forma simples e genérica.



{ COTI INFORMÁTICA
ESCOLA DE NERDS }