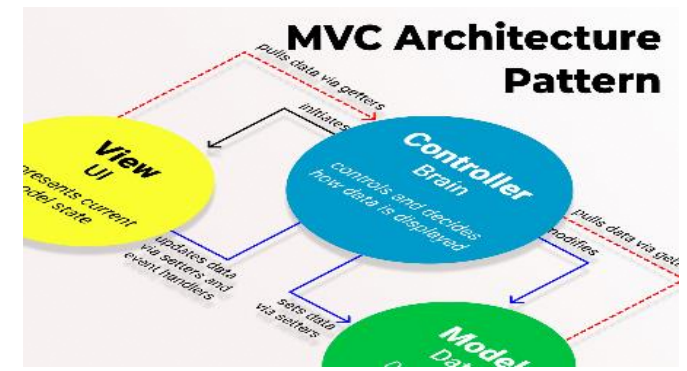




Ciência da **Computação**

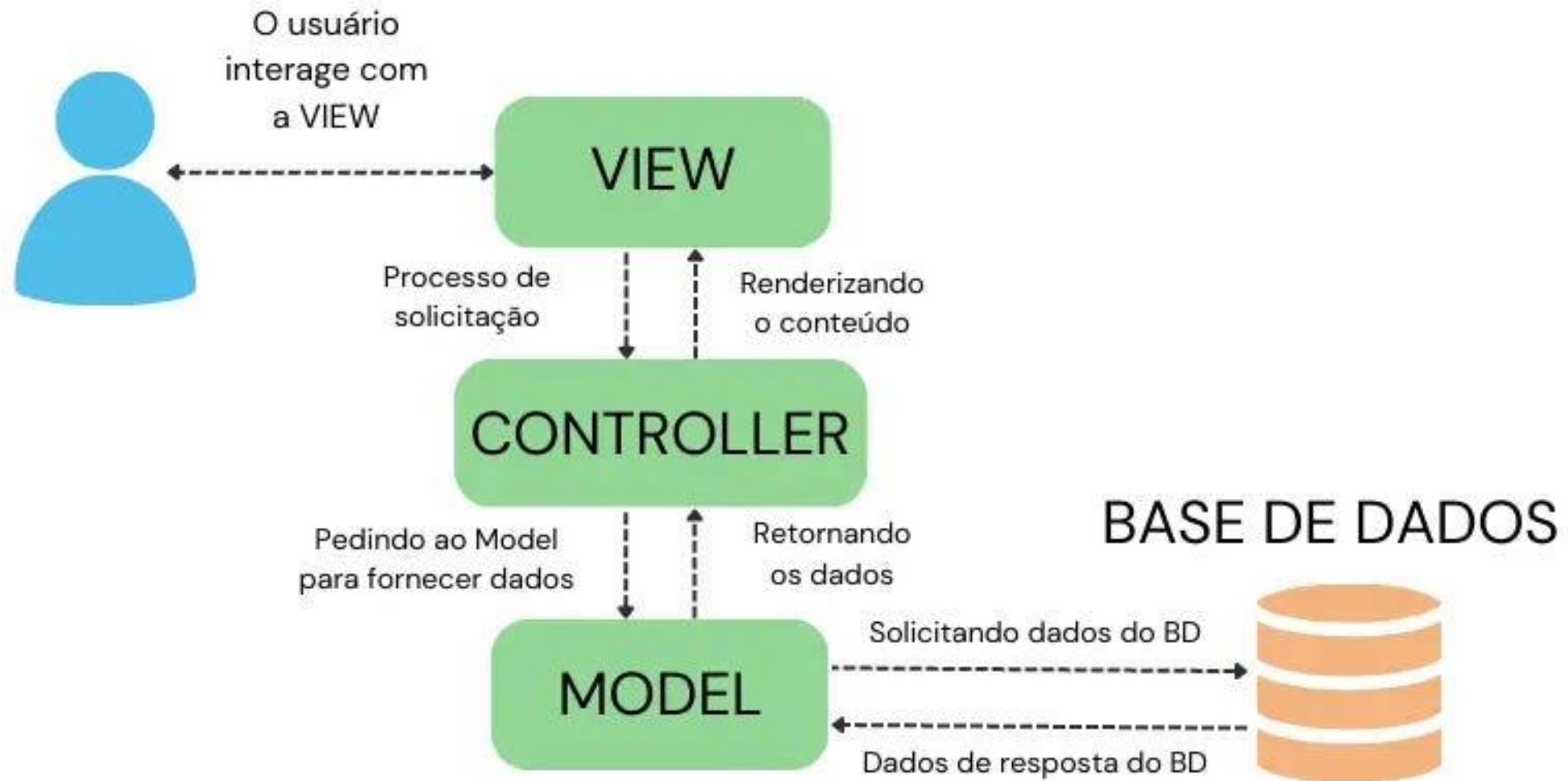
Programação Orientada a Objetos Avançado
Prof. Luciano Rodrigo Ferretto

Arquitetura de Camadas MVC



- **MVC** é uma abreviação que significa "**Model-View-Controller**" (Modelo-Visão-Controlador, em português).
- É um padrão de arquitetura de software amplamente utilizado no desenvolvimento de aplicativos e sistemas, especialmente em aplicações web e de desktop.
- O objetivo do padrão MVC é separar as preocupações em um aplicativo, tornando-o mais organizado, modular e fácil de manter.

Model-View-Controller



Modelo (Model)

- O Modelo representa a camada de dados e a lógica de negócios do aplicativo.
- Ele lida com a manipulação, validação e armazenamento dos dados do aplicativo.

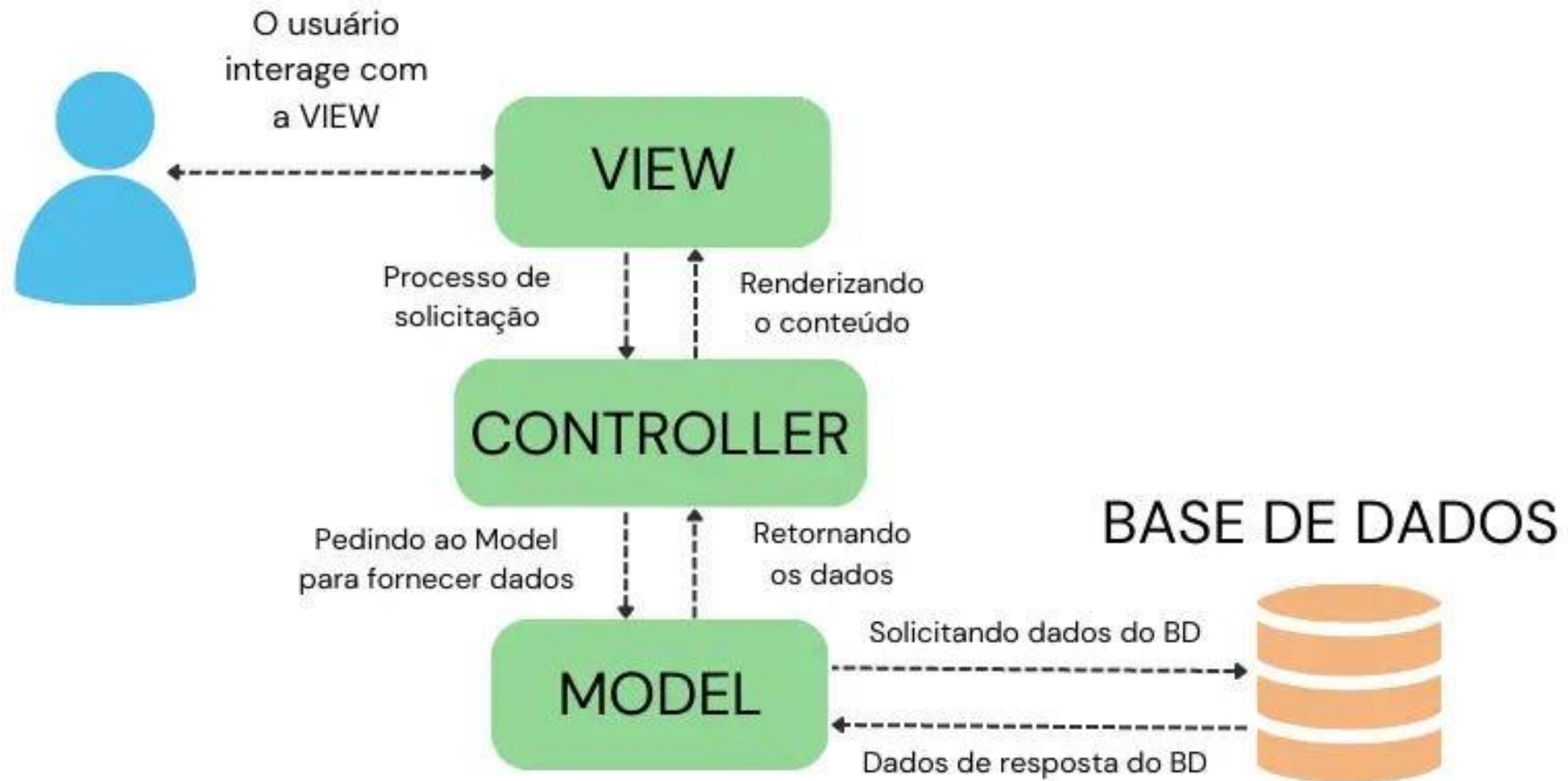
Visão (View)

- A Visão é a camada responsável pela apresentação e interação com o usuário.
- Ela exibe os dados do Modelo aos usuários e coleta entrada do usuário, como cliques de botão ou entradas de formulário.
- A Visão não deve ser a responsável pela lógica de negócios; em vez disso, ela simplesmente apresenta os dados e as interações aos usuários.
- A camada de Visão corresponde a camada Cliente em uma arquitetura Cliente-Servidor, ou seja, o **frontend**.

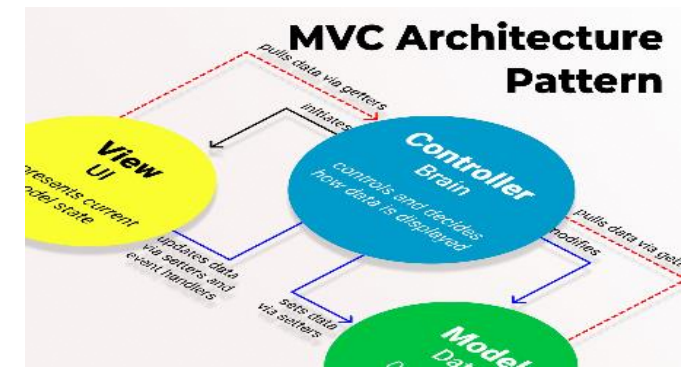
Controlador (Controller)

- O Controlador age como um intermediário entre o Modelo e a Visão.
- Ele recebe as entradas do usuário da Visão e processa essas entradas, interagindo com o Modelo conforme necessário.
- O Controlador é responsável por controlar o fluxo de dados.

Model-View-Controller



Arquitetura MVC



- A separação clara dessas três responsabilidades (Modelo, Visão e Controlador) no padrão MVC torna o código mais modular e fácil de manter.
- Isso facilita o desenvolvimento de aplicativos escaláveis, pois as alterações em uma camada não afetam necessariamente as outras.
- Além disso, o padrão MVC promove a reutilização de código, o que pode economizar tempo e esforço no desenvolvimento de software.

Camada Model (Modelo)

- No contexto do padrão MVC, a camada Modelo pode ser subdividida em várias partes, dependendo da complexidade do aplicativo e das boas práticas de arquitetura de software. Algumas das subdivisões comuns incluem:

Entidades (Entities)

- As entidades representam objetos de negócios essenciais que são armazenados e recuperados do banco de dados.
- Elas geralmente mapeiam diretamente para tabelas em um banco de dados relacional em sistemas que usam banco de dados relacional.



User

- id
- name
- email
- password
- type



Register

- id
- latitude
- longitude
- description
- **user {}**

Repositórios (Repositories)

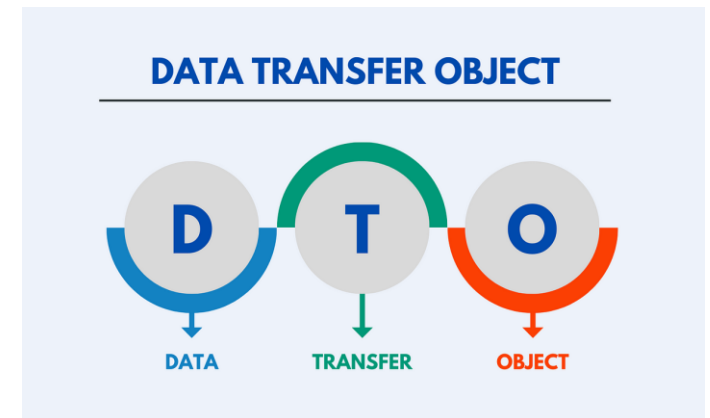
- Os repositórios são responsáveis pela comunicação com o banco de dados ou outra fonte de dados.
- Eles encapsulam as operações de leitura, gravação, atualização e exclusão de entidades.
- Os repositórios fornecem uma interface de programação consistente para interagir com os dados, o que facilita a manutenção e a troca de fontes de dados.

Serviços (Services)

- Os serviços contêm a lógica de negócios do aplicativo que não pertence diretamente às entidades ou aos repositórios.
- Eles são responsáveis por coordenar várias operações e podem chamar métodos em entidades e repositórios para realizar tarefas específicas.
- Os serviços podem ser usados para implementar a lógica de alto nível do aplicativo.

DTO (Data Transfer Objects)

- Os DTOs são usados para transferir dados entre o Modelo e a Visão, geralmente em formato serializado.
- Eles são especialmente úteis quando a estrutura dos dados exibidos na Visão não coincide diretamente com a estrutura das entidades no banco de dados.
- Em alguns projetos são chamados de **Payloads**





SignupDTO

- name
- email
- password

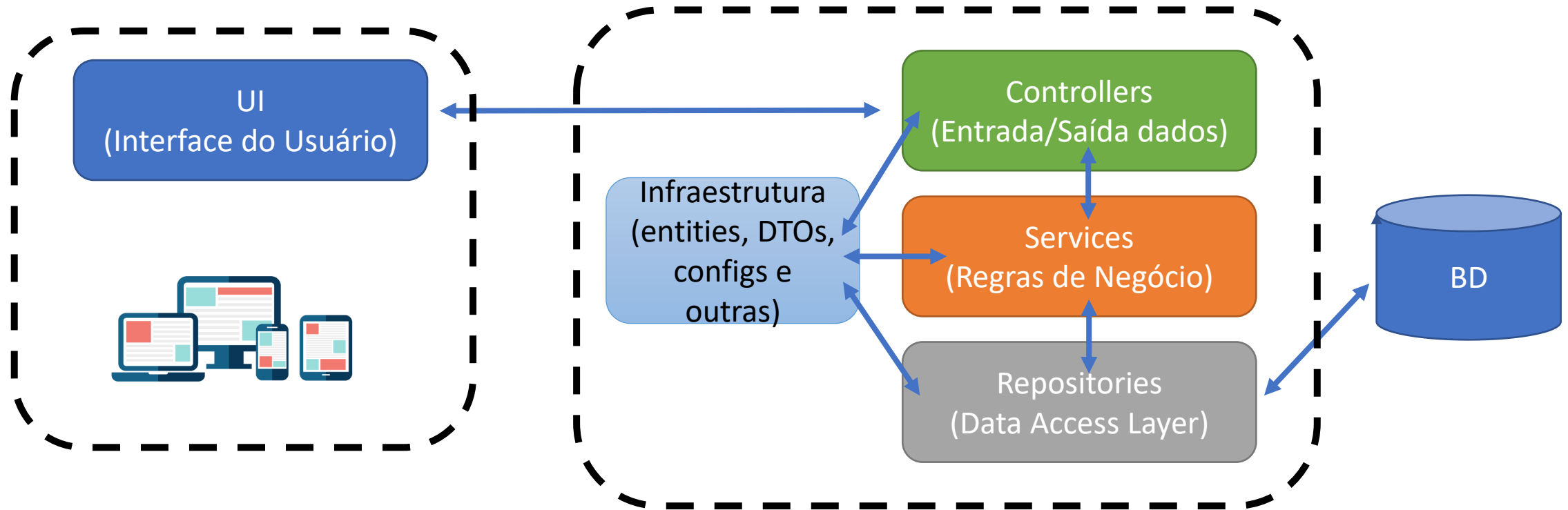


SigninDTO

- email
- password

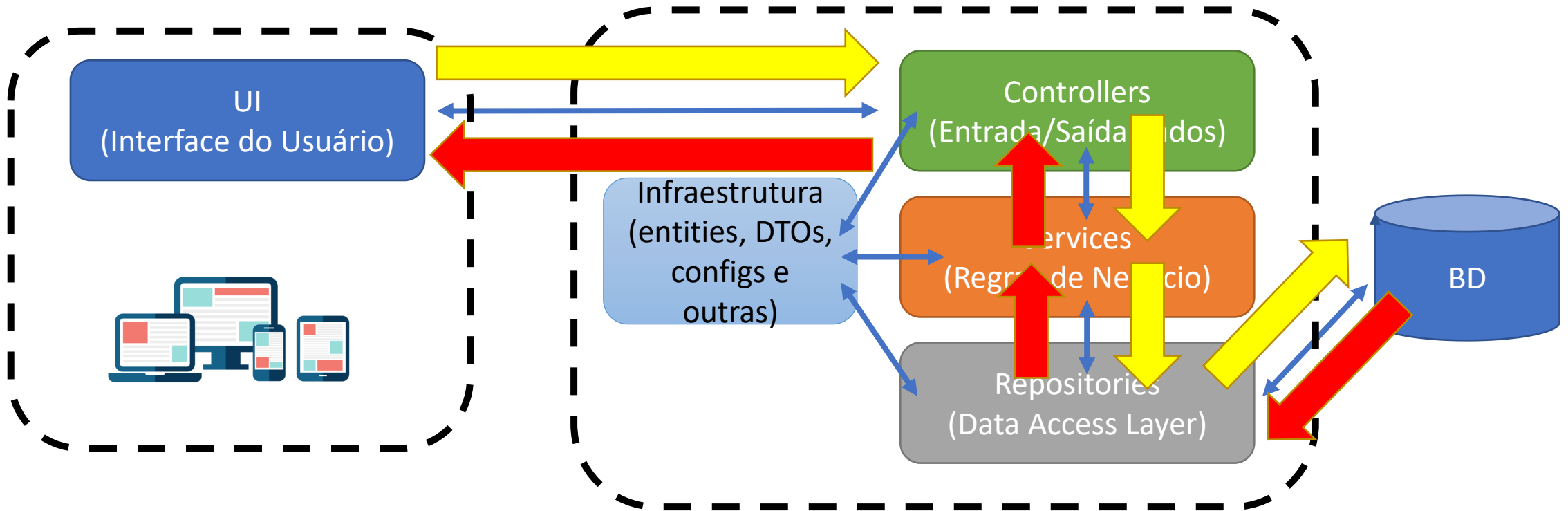
E o nosso projeto??? Como vai ser???

- Arquitetura Cliente-Servidor em conjunto com Arquitetura de Camadas (MVC)
 - Somente o lado do Servidor (**Backend**)

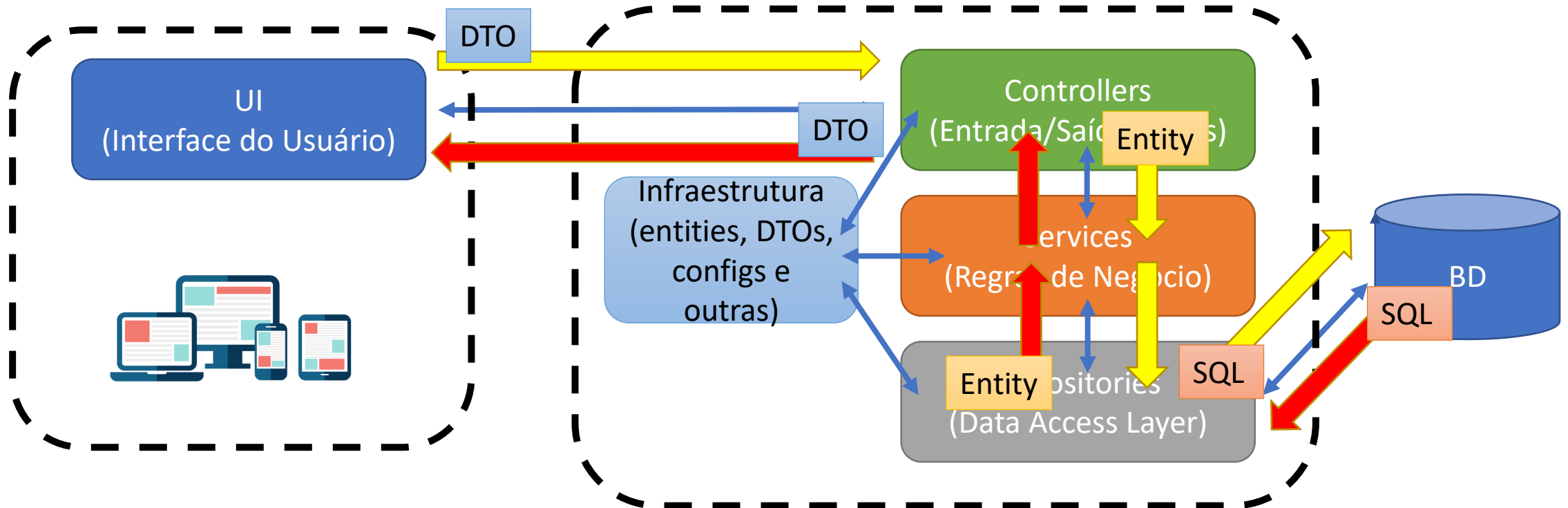


O Caminho dos dados!!!

- Como nossa Arquitetura será Cliente-Servidor, então o início de qualquer comunicação partirá do **Cliente** com direção o **Servidor**, ou seja, da **UI** para as **Controllers**, e o caminho seguirá as setas “**amarelas**” tendo o retorno no sentido das setas “**vermelhas**”.



O Caminho dos dados!!!



Em alguns casos NÃO há a necessidade de DTOs, então as Entidades são usadas para a comunicação entre o Frontend e Backend

E o nosso projeto??? Como vai ser???

- Arquitetura Cliente-Servidor
 - Somente o lado do Servidor
 - Seguindo os princípios KISS

Como será essa comunicação interna?

- Como? Através da invocação de métodos
- Tipo de dados? Objetos Java

