Conciliando uso de camadas

com microservices

Isaac Felisberto de Souza
Engenheiro de Software

...há “quase” 20 anos
no mundo de desenvolvimento

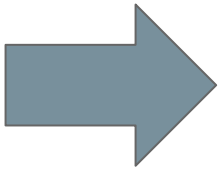
ResultadosDigitais

Já somos mais de 100 na área de desenvolvimento!

Vocês já sabem as vantagens de microservices!!?

E quem acha que existe desvantagens?

Monolito



Microservices

Porque ao fragmentar, algumas coisas NÃO melhoram?

Há vários motivos!

Muitas vezes fragmentamos,
mas não melhoramos
nossa forma de produzir código

Um dos temas é o uso:

camadas

Descarta ou faz mau uso de camadas

Mas... em software,

O que são camadas?

É usar MVC?

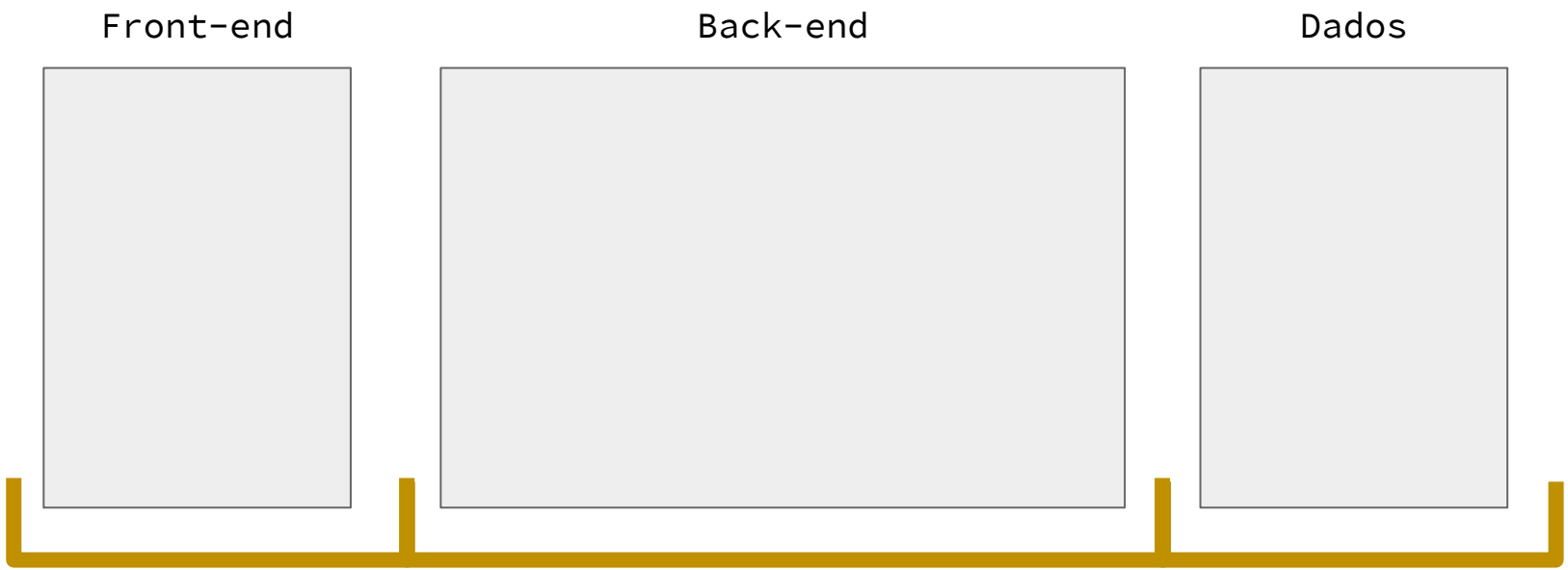
MVC é um Design Pattern

Que apresenta uma forma de usar camadas

2, 3, N...

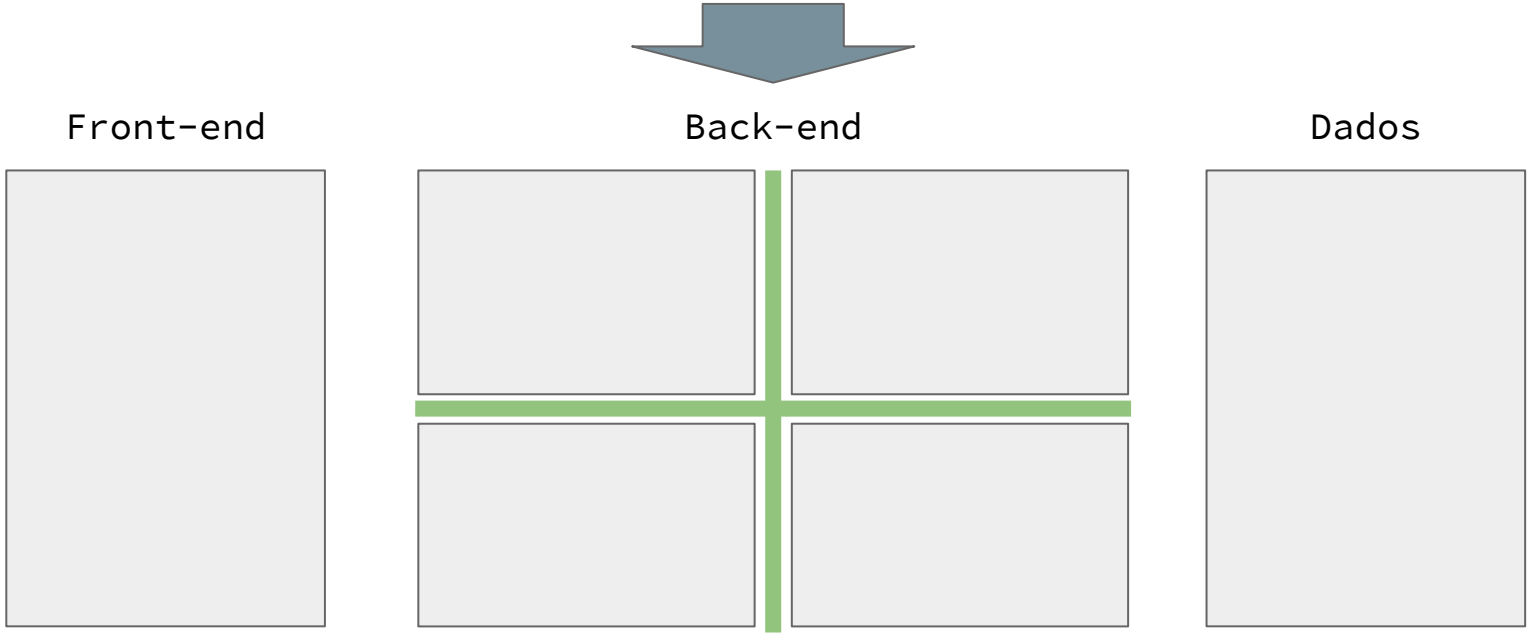
camadas

Existem camadas “físicas” e “lógicas”

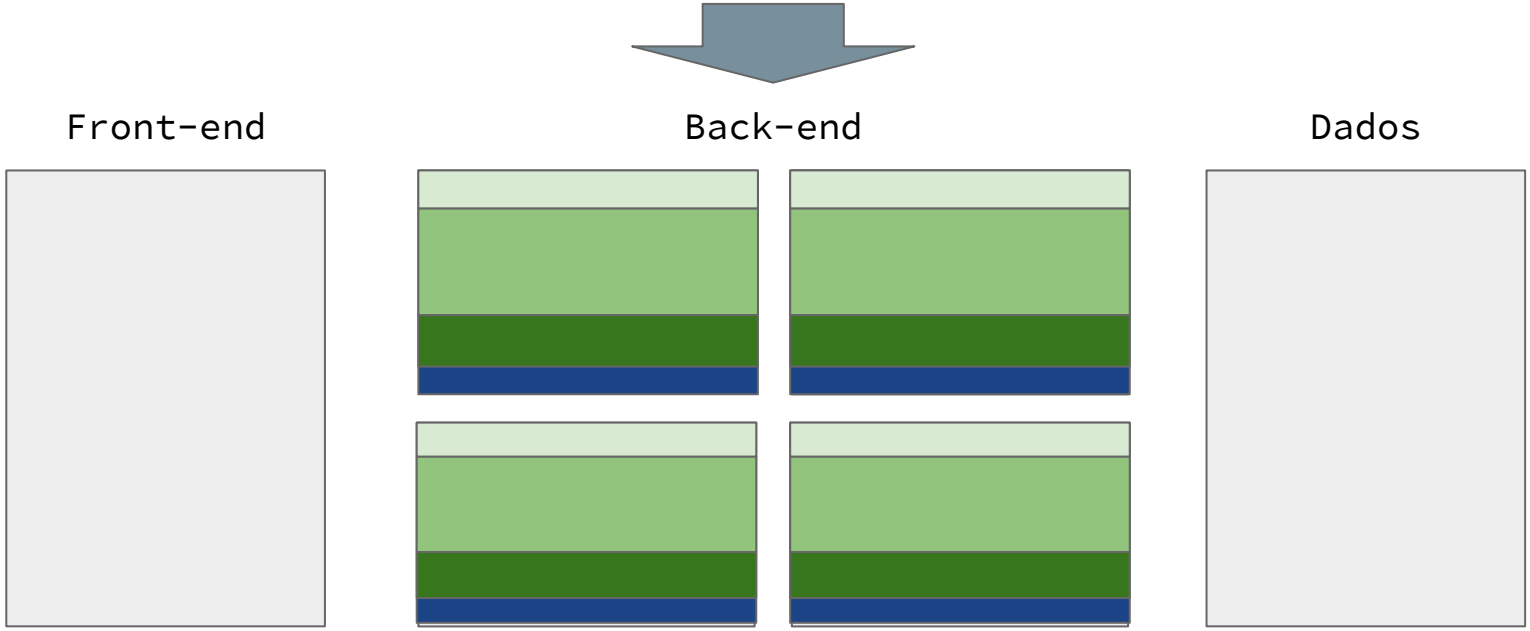


camadas “físicas”

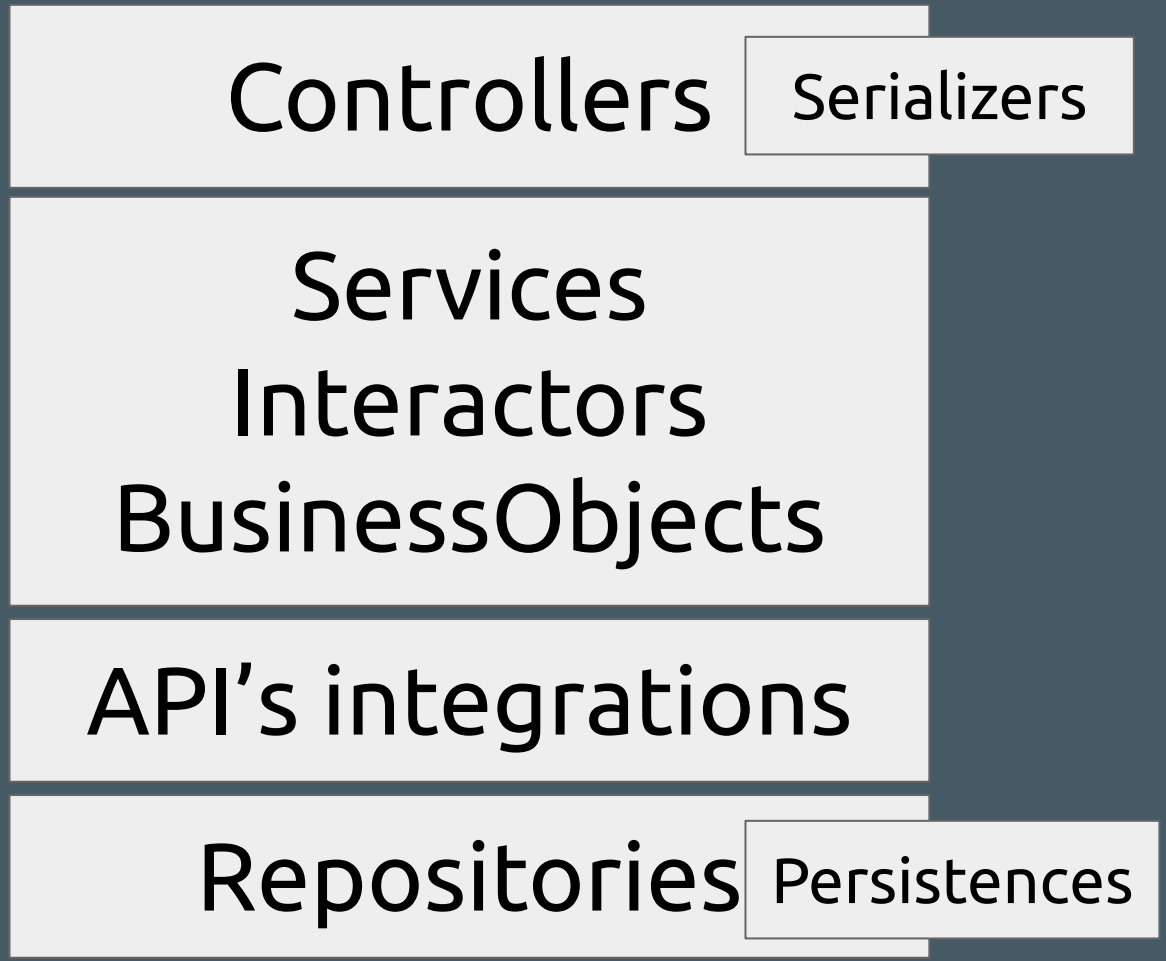
O que chamamos de microservice



O que chamamos de microservice



camadas “lógicas”

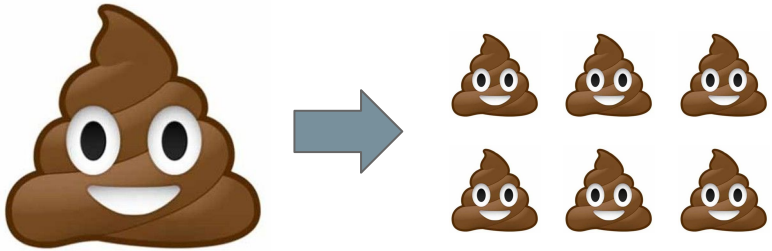

Controllers
Serializers
Services
Interactors
BusinessObjects
API’s integrations
Repositories
Persistences

Organização

Responsabilidades

Reutilização

Sem isso,
o que ocorre?



Vamos ver exemplos!

Caso de uso: Atualizar dados de um autor

- Validar se está ativo. Autor inativo não pode ser atualizado
- Atualizar dados do autor
- Gera histórico com usuário que atualizou

Controllers

app ▸ controllers ▸ authors_controller.rb

```ruby
class AuthorsController < ApplicationController
  before_action :set_author, :validate_active_author, only: [:update]

  def update
    respond_to do |format|
      if @author.update(author_params)
        History.create("Author #{@author.name} updated by #{@user.name}")
        format.html { redirect_to @author, notice: 'Author was successfully updated.' }
        format.json { render :show, status: :ok, location: @author }
      else
        format.html { render :edit }
        format.json { render json: @author.errors, status: :unprocessable_entity }
      end
    end
  end

  private

    def set_author
      @author = Author.find(params[:id])
    end

    def author_params
      params.require(:author).permit(:name, :last_name)
    end

    def validate_active_author
      head :precondition_failed unless @author.active?
    end
end
```

Validação?

Mais regras de negócio?

Dados de saída

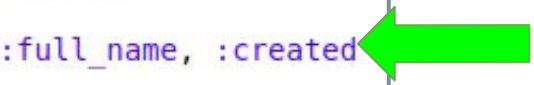
Recuperar dados

Dados de entrada

Serializers

app ▸ serializers ▸ author_serializer.rb

```ruby
class AuthorSerializer
  include FastJsonapi::ObjectSerializer

  attributes :name, :last_name, :full_name, :created

  def name
    unless object.active?
      "#{name} (Inactive Author)"
    else
      name
    end
  end

  def full_name
    "#{object.name} #{object.last_name}"
  end

  def created
    object.created_at
  end
end
```



Services

app ▸ services ▸ author ▸ update.rb

```ruby
module Services::Author
  class Update < Services::Application

    def call(context, author_id, attributes)
      author = Author.find(author_id)
      if author.update(attributes)
        History.create("Author #{author.name} updated by #{context.user.name}")
        return {
          status: :ok,
          content: { author: { name: author.name, last_name: author.last_name } }
        }
      else
        return { status: :error }
      end
    end

  end
end
```

Regras de negócio!

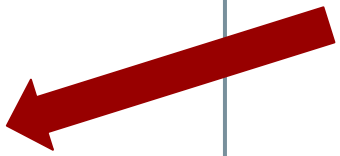
Definindo aqui dados de saída?

Só status error? Como a camada acima vai tratar?

Models

app ▸ models ▸ author.rb

```ruby
class Author < ApplicationRecord

  before_update :create_history

  def create_history
    History.create("Author #{name} updated by #{@user.name}")
  end

  def changed_by_user(user)
    @user = user
  end

  def to_hash
    { name: name, last_name: last_name }
  end

  def full_name
    "#{name} #{last_name}"
  end
end
```



Regras de negócio?

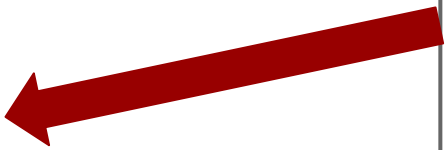
Definindo aqui dados de saída?

“atributo” derivado!

Em Java...

Regras de negócio!

```java
public class AuthorRepository extends AbstractRepository<Responsavel, Long>{

    @PersistenceContext
    EntityManager entityManager;

    public void save(Responsavel responsavel) {
        entityManager.persist(responsavel)
        history = new History(responsavel, getUser())
        entityManager.persist(history)
    }
```



Regras de negócio!

```java
@Entity
@SequenceGenerator(name = "responsavelSeq", sequenceName = "RESPONSAVEL_SEQ", allocationSize = 1)
public class Responsavel implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "responsavelSeq")
    private Long id;

    @Column()
    @NotNull
    @Size(max = 150)
    @JsonGetter("nomeCompleto")
    private String nome;

    @Column()
    @NaturalId
    @NotNull
    @CPF
    private String cpf;
```

Definindo aqui dados de saída!

Boas práticas:

- Controllers
- Serializers
- Regra de negócio
- Persistência
- Erros de Design de código
- Clean Architecture !

Princípios e Padrões de Projeto

- SOLID
  - Single responsibility principle
- GoF
  - Business delegate, Facade
- GRASP
  - High cohesion, Low Coupling
- MVC: A API é a VIEW

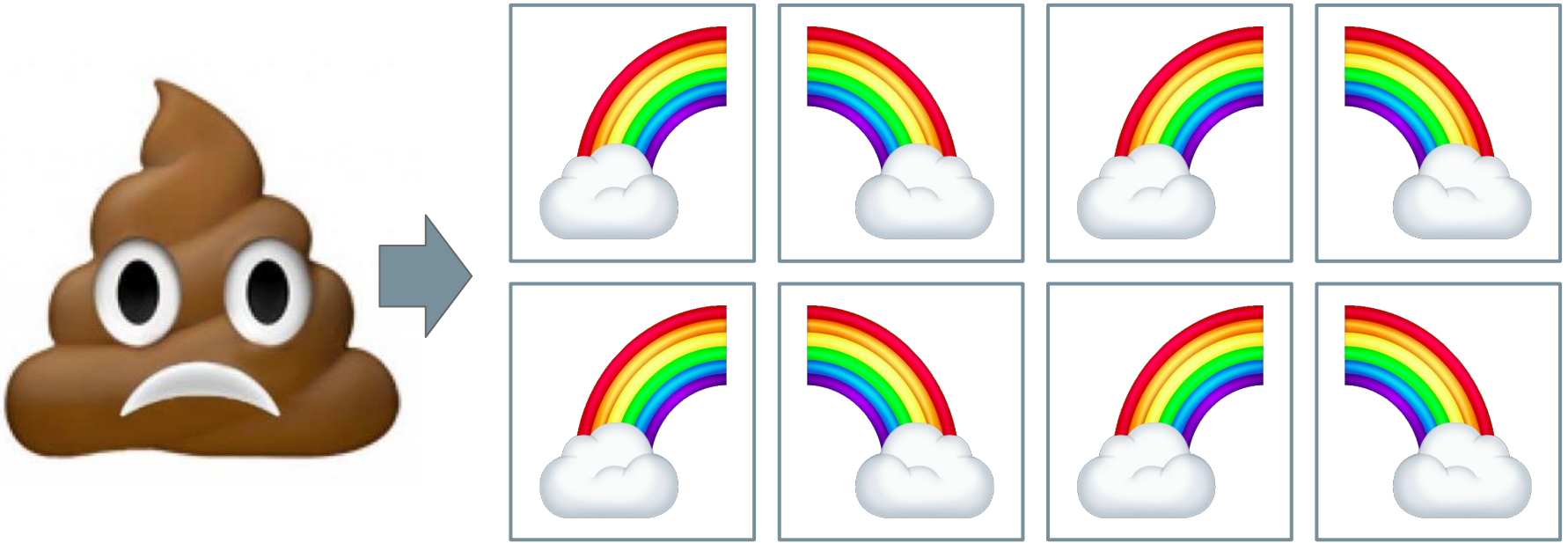
Quantos arquivos, classes, métodos serão modificados?

Seria possível trocar uma camada de forma fácil?

A principal mensagem é...

É possível conciliar uso de Camadas com Microservices

Camadas, é **um** dos temas na caminhada para...



Obrigado!

Dúvidas?

Isaac Felisberto de Souza
isaacsouza@gmail.com

linkedin.com/in/isaacfsouza