# Rotas

## Parâmetros opcionais e valores padrões

Os parâmetros são passados entre {}. Os parâmetros opcionais terminam com “?” e costumam ser inseridos no final. Sendo a ordem: os obrigatórios seguidos dos opcionais.



## Agrupando Rotas

Ao chamar a classe Route, ao invés de chamar o método get chamaremos o método prefix, e a seguir o método group com uma função de call-back.



## Nomeando Rotas

É possível definir um nome para a rota que pode ser utilizado dentro da aplicação como um alias:

Web.php



principal.blade.php



Este é um recurso que evita que os links tenham dependências diretas com as rotas.

OBS: ela não pode ser usada na URL

## Redirecionando Rotas

É possível usar o **Redirect** de algumas formas:

1ª do objeto Route



2ª da função de callback



## Rota de contingência (fallback)

**Fallback** é chamado quando a página não existe:



# Controladores

Encaminhando parâmetros da rota para o controlador

Os parâmetros para o controlador são passados da seguinte forma:

web.php



TesteController.php  


Note que os parâmetros devem ser passados na mesma ordem.

## Encaminhando parâmetros do controlador para visualização

TesteController.php

teste.blade.php

# Views

## Funções do blade

[Documentação do blade](https://laravel.com/docs/7.x/blade)

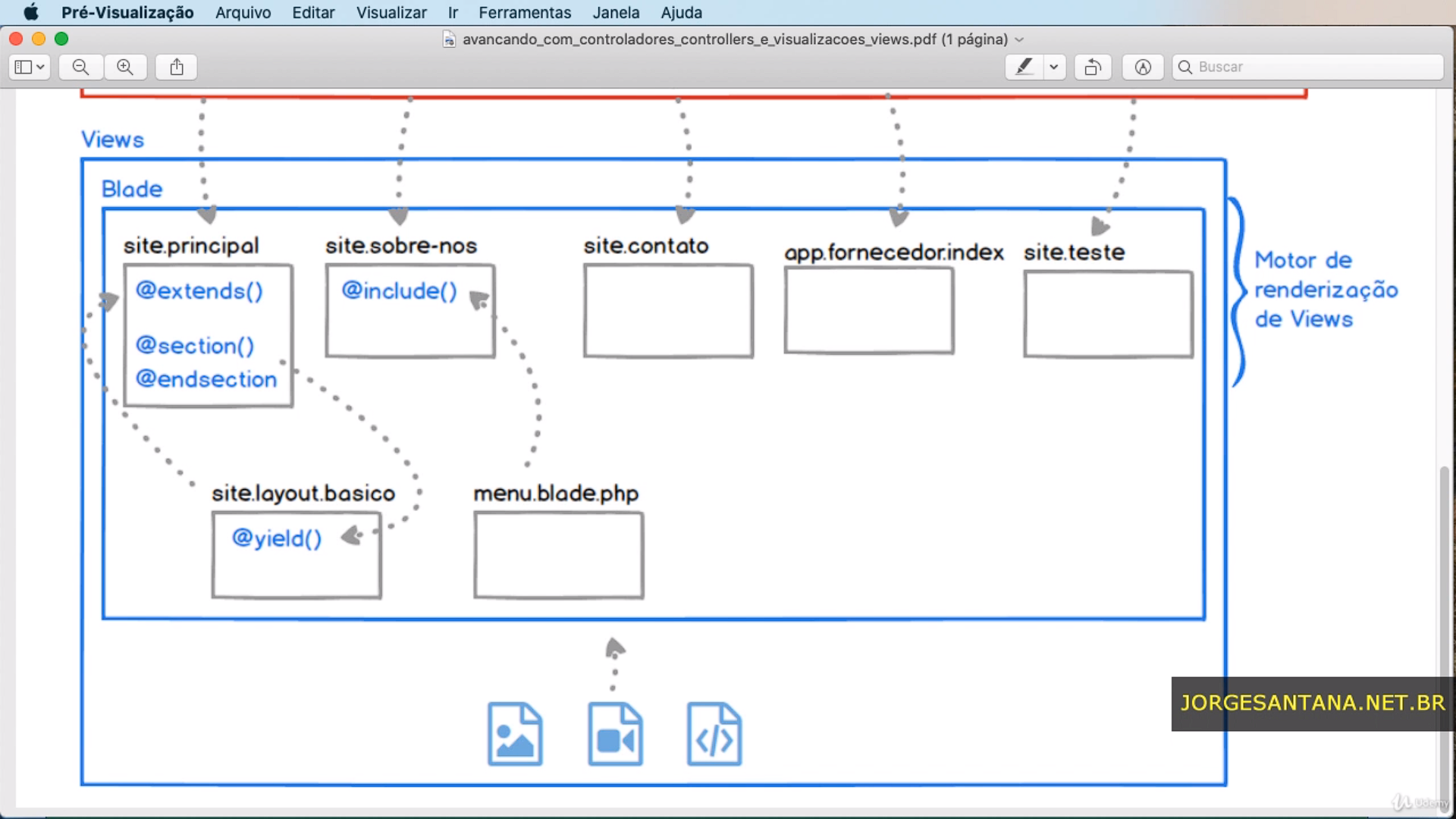
## Assets

É tudo aquilo que é utilizado para complementar o conteúdo do frontend das aplicações web. Ex: Midia, js, css.

Os assets por padrão se encontram no diretório ‘public’.

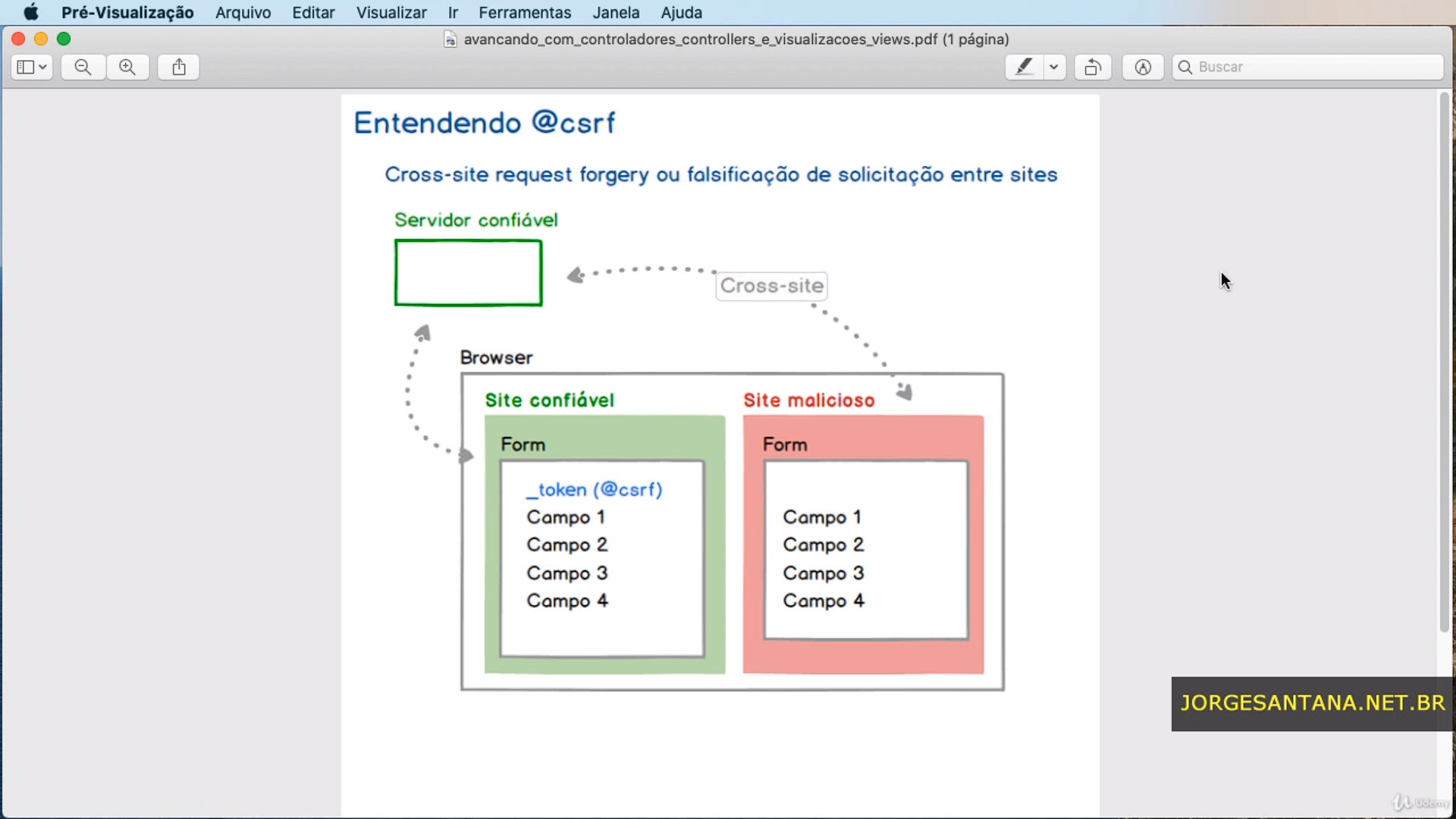


## @extends, @section, @yield e @include



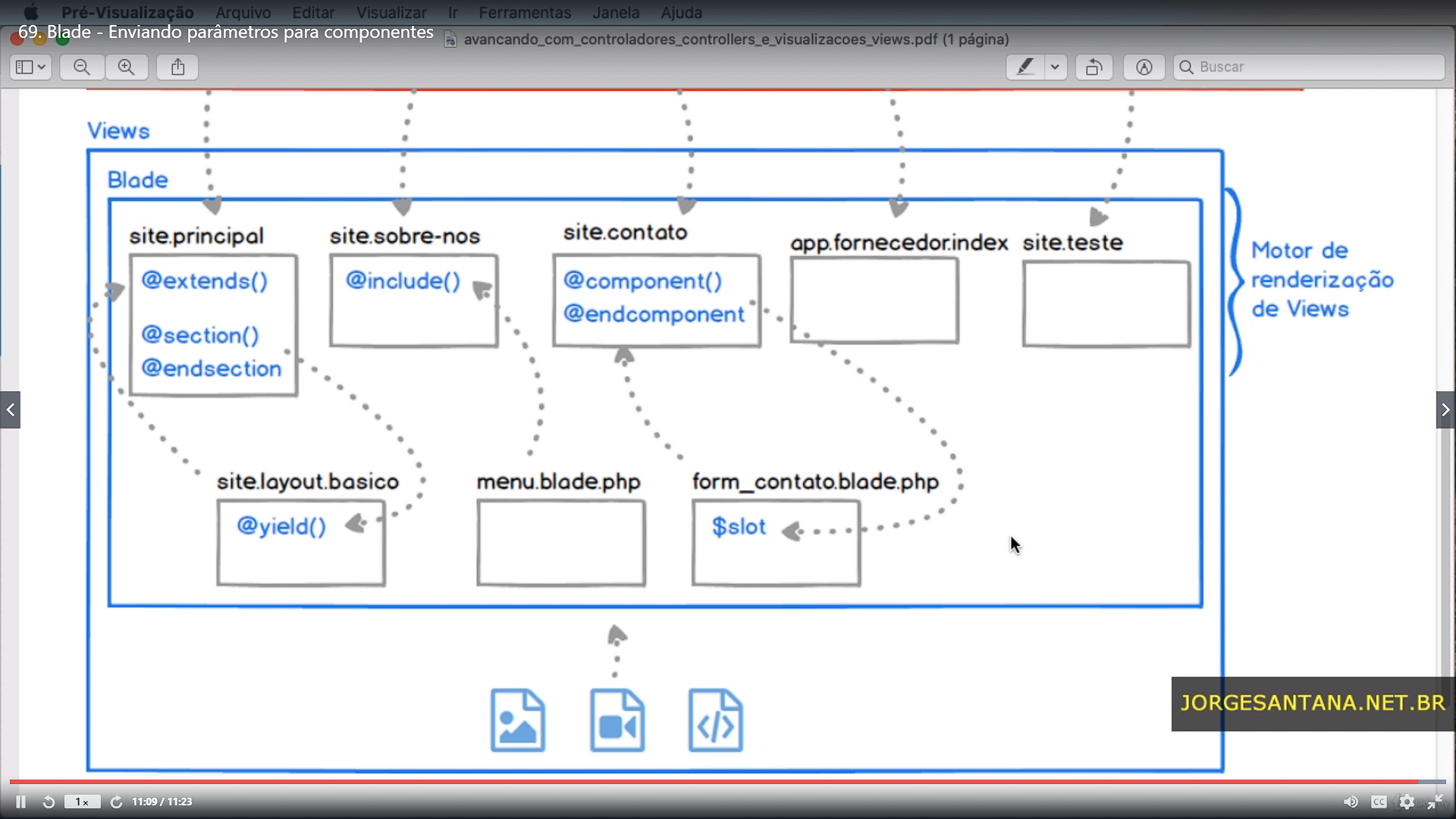
## @csrf

O csrf (cross-site request forgery ou Falsificação de solicitação entre sites) é um token é adicionado ao formulário da pagina legítima ao realizar uma requisição para exibição de um formulário de transferencia de valores, o servidor retorna a pagina aplicando ao formulario um token exclusivo transformando o formulario em algo legitimo.



## @component

**@component** e **@endcomponent** é parecido com ‘include’, porém é possível adicionar informações adicionais para o componente e mostrar o conteúdo enviado pela view para o componente através da variável **$slot** criadaautomaticamente pelo component .



View

Componente



Ou então passando mais variáveis após a chamada do componente como array associativo:

View

Componente



# Models

# Migrations

[Documentação migrations](https://laravel.com/docs/7.x/migrations)

Migrations são como controle de versão para seu banco de dados, permitindo uma equipe de modificar e compartilhar o schema da aplicação facilmente. Fazem toda a parte de DDL do banco de dados.

## Criando uma migration

php artisan make:migration create\_produtos\_table

Por padrão esse comando cria um arquivo no diretório /dataqbase/migrations contendo uma classe com o nome da migration criada com dois métodos up() e down() vazios que são usados, respectivamente, para execução da migration e reversão da migration.

Se o arquivo começar com create\_[tabela]\_table, é chamado automaticamente o método create da classe Schema dentro do método up() e o método dropIfExists da classe Schema dentro do método down() .

## Comandos

php artisan migrate:status

atalho para listagem das migrations que já foram executadas. E além disso podemos ver se a migration já foi executada.

php artisan migrate:reset

reverte todas as migrações do banco, ou seja, executa o rollback de todas as migrações independente do batch/lote.

php artisan migrate:refresh

reverte todas as migrações do banco, ou seja, executa o rollback de todas as migrações independente do batch/lote. Após isso executa novamente as migrations.

php artisan migrate:fresh

dropa todas as tabelas. Após isso executa novamente as migrations.