



MINISTÉRIO DA
CIÊNCIA, TECNOLOGIA
E INOVAÇÃO





Recapitulando Aula Passada

Django Admin

- Configurar o Admin para melhor Experiência e Usabilidade → Django Admin (ModelAdmin)
- Personalizar a exibição de dados (list_display) .
- Criar filtros laterais e de busca (list_filter, search_fields)
- Configurar variáveis editáveis e imutáveis (list_editable, readonly_fields)
- Propriedades Globais:
 - site_header: Texto no topo de todas as páginas (ex: "Sistema de Gestão Bolsa Futuro").
 - site_title: Título na aba do navegador.
 - index_title: Título da Dashboard principal.





Agenda da Aula

Engenharia de Protocolos Web e Formulários Django

- Configurar o TabularInline e as "Actions" (ERP style).
- Métodos HTTP (GET [QueryDict], POST [Body])
- Criação de Formulários (ModelForms e Widgets)
- Validação Básica de Formulários
- CSRF Token (ataque Cross-Site Request Forgery e por que o token do Django é vital.).



Transição de Contexto:

- Aula Anterior: Backoffice (Ambiente Controlado/Intranet).
- Aula Atual: Frontend Público (Ambiente Hostil/Internet)



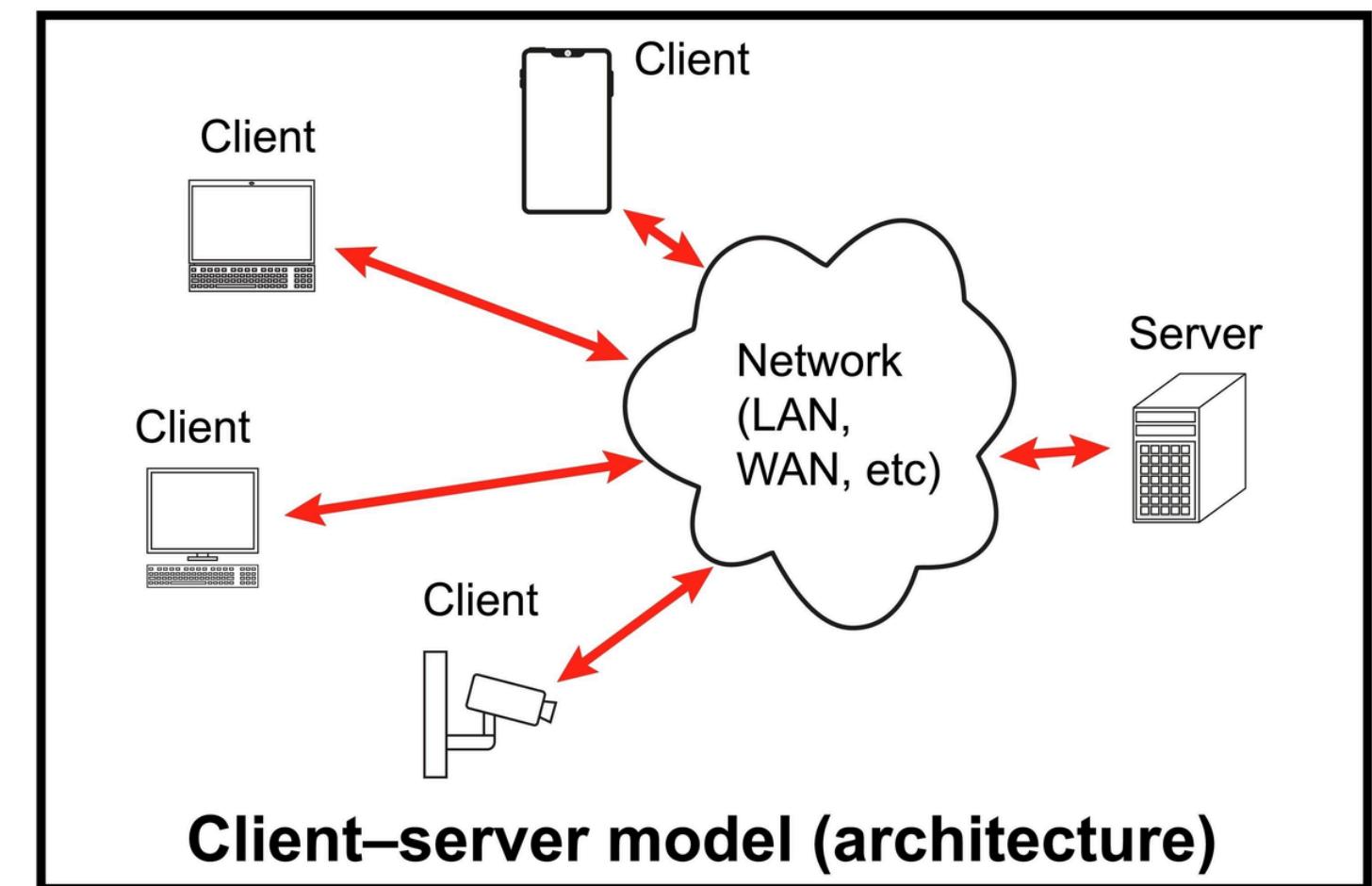
COMUNICAÇÃO WEB

Arquitetural: Client-Side vs. Server-Side

Aplicações web modernas não são apenas repositórios de leitura (Read-Only). Elas exigem Mutação de Estado, onde o cliente (Navegador) envia dados estruturados para alterar a persistência no servidor (Banco de Dados).

Fluxo de Dados (Data Flow):

- Interface (DOM): Captura do input do usuário.
- Transporte (HTTP): Encapsulamento dos dados.
- Processamento (View): Validação e Lógica de Negócio.
- Persistência (Model): Gravação no SGBD ou arquivo.db.

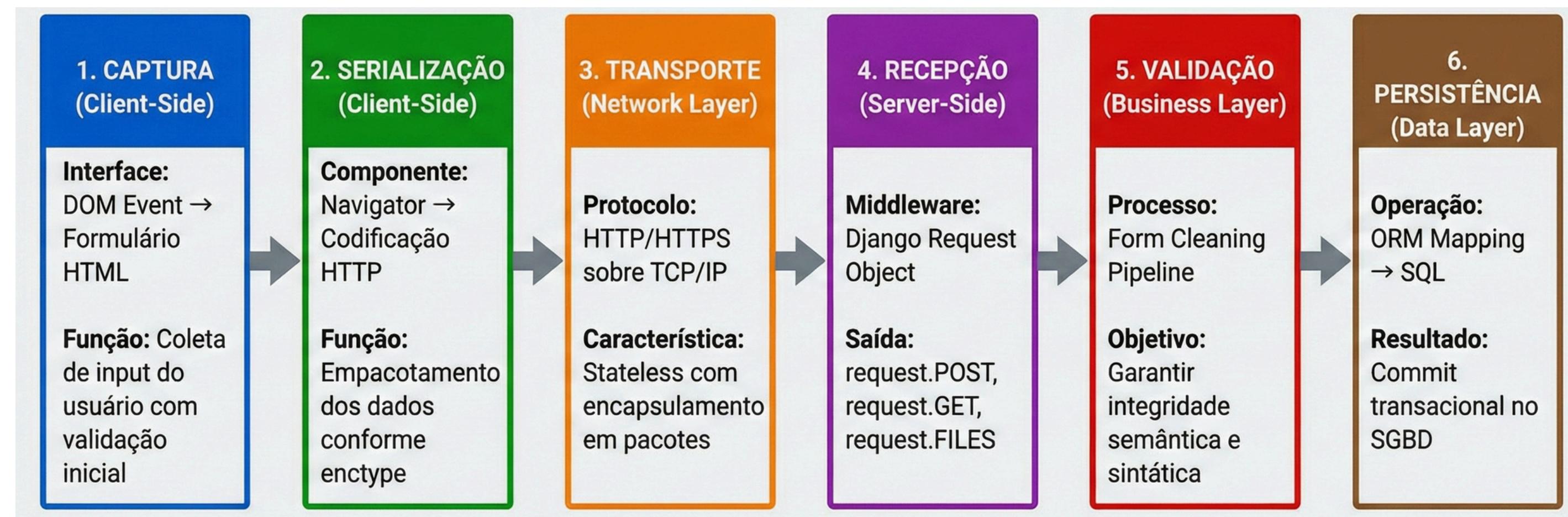




COMUNICAÇÃO WEB

Arquitetural: Client-Side vs. Server-Side

Aplicações web modernas não são apenas repositórios de leitura (Read-Only). Elas exigem Mutação de Estado, onde o cliente (Navegador) envia dados estruturados para alterar a persistência no servidor (Banco de Dados).

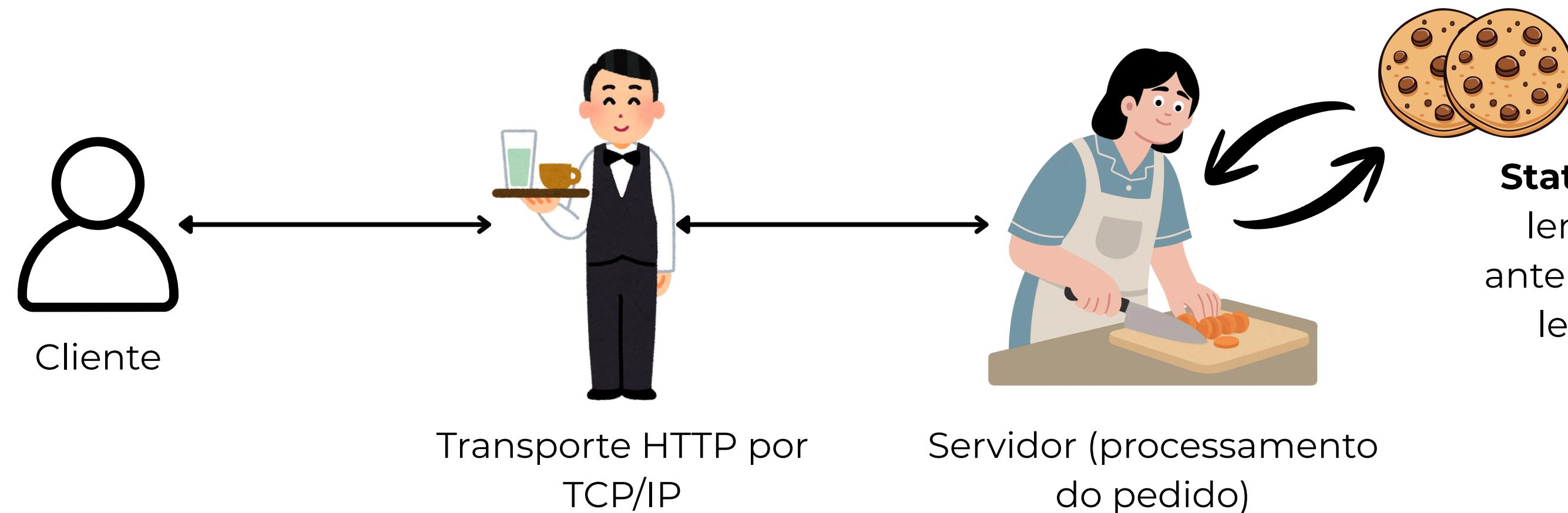




COMUNICAÇÃO WEB

Protocolo HTTP

O Hypertext Transfer Protocol (HTTP) é um protocolo de camada de aplicação para sistemas de informação distribuídos e colaborativos. Ele opera no modelo Requisição-Resposta e é, por definição, Stateless (não mantém estado entre conexões).

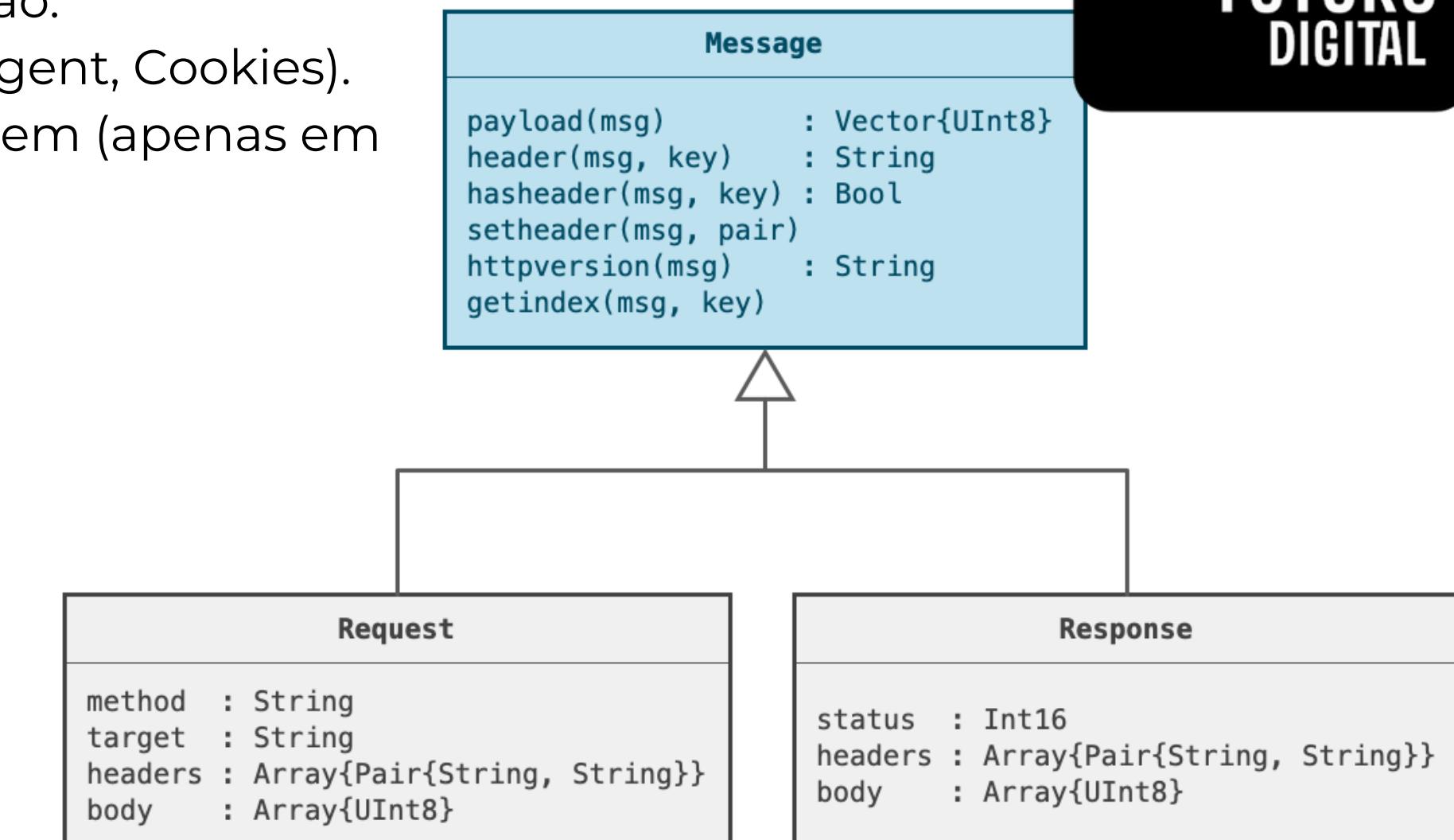
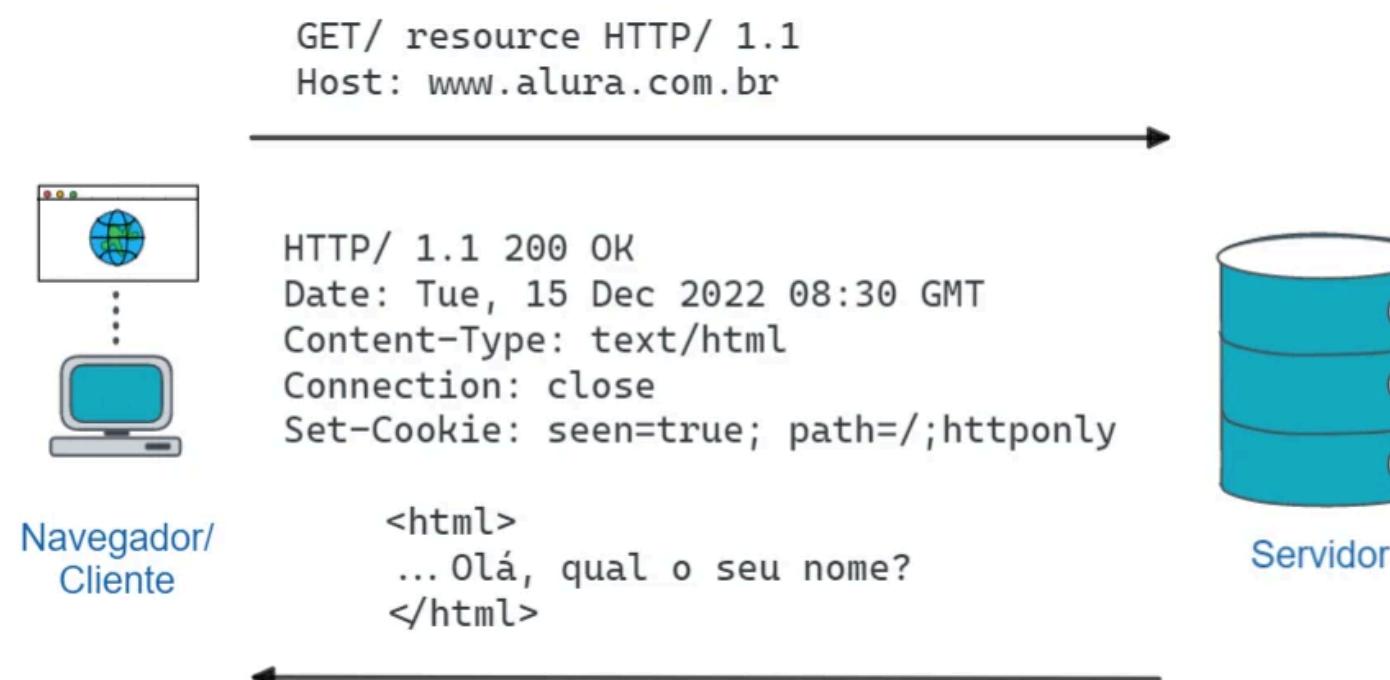




COMUNICAÇÃO WEB

Estrutura da Mensagem

- Start-Line: Método (GET/POST) + URI + Versão.
- Headers: Metadados (Content-Type, User-Agent, Cookies).
- Body (Payload): O conteúdo útil da mensagem (apenas em métodos de escrita).

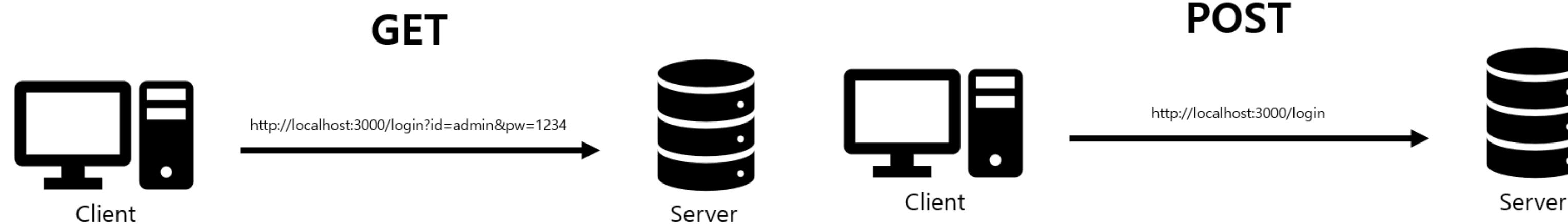




COMUNICAÇÃO WEB

Semântica dos Métodos: GET vs. POST

- Método GET (Safe & Idempotent):
 - Solicita a representação de um recurso específico.
 - Transporte: Os dados são anexados à URI como Query String Parameters (`?chave=valor&id=1`).
 - Django: `request.GET` (Instância de `QueryDict` imutável).
- Método POST (Unsafe & Non-Idempotent):
 - Envia uma entidade para o recurso especificado, causando uma mudança de estado.
 - Transporte: Os dados viajam no Message Body (Corpo da Requisição), separados dos cabeçalhos.
 - Django: `request.POST` (Dicionário com dados do formulário).





COMUNICAÇÃO WEB

Análise Comparativa e Segurança da Informação



Característica	GET	POST	Impacto
Visibilidade	Exposto na URL	Oculto no Body	GET: Vulnerável a shoulder surfing, logs
Armazenamento	Histórico browser, logs server, proxy, bookmarks	Apenas no corpo (temporário)	GET: Rastreamento permanente de dados sensíveis
Cacheabilidade	Sim (Browser, CDN, Proxy)	Não (por padrão)	GET: Pode vaziar dados cacheados em estações compartilhadas
Limite de Tamanho	~2048-4096 chars	Ilimitado (teórico)	GET: Rejeição silenciosa de dados truncados
CSRF Exposure	Menor risco (dados visíveis)	Maior risco (oculto)	POST: Alvo principal de ataques CSRF



COMUNICAÇÃO WEB

Diretrizes de Segurança OWASP

- Regra: "Jamais utilize GET para dados sensíveis"
- Cenários de Vazamento com GET:
 - 1. Logs de Servidor Web

```
192.168.1.1 - - [10/Nov/2023:10:15:30] "GET /login?token=abc123... HTTP/1.1"
```

- 2. Histórico do Navegador: Acessível por qualquer usuário físico da máquina (sicronizado na nuvem)
 - Seu navegador guarda todas as URLs que você visita
- 3. Referer Headers (cabeçalho referenciado)
 - Vazado para sites terceiros via links
 - Exemplo: Referer: https://bank.com/transfer?valor=1000&conta=attacker





COMUNICAÇÃO WEB

Boas Práticas Comprovadas

- Dados Sensíveis → SEMPRE POST + HTTPS
 - Credenciais (senhas, tokens)
 - Dados PII (CPF, RG, endereço)
 - Informações financeiras (cartão de crédito, saldo)
- GET Apenas para:
 - Identificadores de recurso (/produtores/123)
 - Parâmetros de navegação (?page=2)
 - Filtros não sensíveis (?categoria=hortifruti)





ENGENHARIA DE FORMULÁRIOS NO DJANGO

Padrão de Projeto: ModelForm

Uma abstração de alto nível que implementa o padrão Data Mapper, vinculando automaticamente os campos de um Modelo ORM (models.py) aos campos de um Formulário HTML.



(DRY - Don't Repeat Yourself): Elimina a redundância de definir tipos de dados duas vezes (uma no banco, outra no HTML).

```
# forms.py
from django import forms
from .models import Produtor

class ProdutorForm(forms.ModelForm):
    class Meta:
        model = Produtor # Vínculo com o ORM
        fields = ['nome', 'cpf', 'documento_pdf'] # Whitelist de campos
```



ENGENHARIA DE FORMULÁRIOS NO DJANGO

Fluxo de Transformação Automática:

Model Field (`models.py`)

```
CharField(max_length=100)  
EmailField()  
DateField()  
BooleanField()  
x"  
FileField()  
ForeignKey(Produto)
```

→ Form Field (`forms.py`)

```
→ CharField(max_length=100)  
→ EmailField()  
→ DateField()  
→ BooleanField()  
  
→ FileField()  
→ ModelChoiceField(Produto)
```

→ HTML Render.

```
→ <input type="text">  
→ <input type="email">  
→ <input type="date">  
→ <input type="checkbox">  
  
→ <input type="file">  
→ <select>...</select>
```





ENGENHARIA DE FORMULÁRIOS NO DJANGO

Widgets e Renderização de Interface

O Widget é a classe responsável pela renderização do HTML (<input>, <textarea>, <select>) e pela extração de dados do dicionário de submissão.

- Customização de Atributos (DOM): Podemos injetar classes CSS, Placeholders e atributos HTML5 via dicionário attrs.

```
widgets = {
    'observacao': forms.Textarea(attrs={'class': 'form-control', 'rows': 3}),
    'data_nascimento': forms.DateInput(attrs={'type': 'date'}), # HTML5 Date
}
```

O Django converte automaticamente models.CharField → forms.TextInput.





ENGENHARIA DE FORMULÁRIOS NO DJANGO

Widget ≠ Field

- Field: Define a lógica (validação, tipo de dado)
- Widget: Define a apresentação (HTML, atributos visuais)

```
forms.Widget (Base)
└── forms.TextInput      → <input type="text">
└── forms.PasswordInput  → <input type="password">
└── forms.EmailInput     → <input type="email">
└── forms.NumberInput    → <input type="number">
└── forms.DateInput      → <input type="date">*
└── forms.DateTimeInput  → <input type="datetime-local">
└── forms.Textarea        → <textarea></textarea>
└── forms.Select          → <select><option>...
└── forms.CheckboxInput   → <input type="checkbox">
└── forms.FileInput       → <input type="file">
```

```
class ProdutorForm(forms.ModelForm):
    class Meta:
        model = Produtor
        fields = ['nome', 'cpf', 'data_nascimento',
                  'categoria', 'foto_perfil']

    # DICIONÁRIO DE WIDGETS - Customização precisa
    widgets = {
        # Campo: Widget(attrs={atributos HTML})
        'nome': forms.TextInput(attrs={
            'class': 'form-control form-control-lg',
            'placeholder': 'Digite seu nome completo',
            'autofocus': True, # Foco automático
            'aria-label': 'Nome do produtor',
            'data-toggle': 'tooltip',
            'title': 'Nome completo conforme documentos'
        }),
    }
```





ENGENHARIA DE FORMULÁRIOS NO DJANGO

Injeção de Contexto e Template Engine

O formulário instanciado na View é passado como variável de contexto para o Template.

- Estratégias de Renderização (DTL - Django Template Language):

- {{ form.as_p }}: Renderização rápida (prototipagem) encapsulando campos em <p>.
- {{ form.as_div }}: Padrão moderno (Django 4.0+) usando <div>.
- Renderização Manual: Iteração sobre { % for field in form % } para controle granular de CSS.





ENGENHARIA DE FORMULÁRIOS NO DJANGO

Injeção de Contexto e Template Engine

```
# views.py
def criar_produto(request):
    form = ProdutoForm() # Instancia formulário
    return render(request, 'form.html', {'form': form})
```

A tag <form method="POST"> deve encapsular a renderização.

```
<!-- form.html -->
<form method="POST">
    {% csrf_token %} <!-- PROTEÇÃO OBRIGATÓRIA -->

    <!-- Opção 1: Automático (rápido) -->
    {{ form.as_p }}

    <!-- Opção 2: Manual (controle total) -->
    <div class="mb-3">
        {{ form.nome.label_tag }}
        {{ form.nome }}
        {% if form.nome.errors %}
            <div class="text-danger">{{ form.nome.errors }}</div>
        {% endif %}
    </div>

    <button type="submit">Salvar</button>
</form>
```





ENGENHARIA DE FORMULÁRIOS NO DJANGO

Protocolo de Transferência de Arquivos (MIME Types)

O envio de binários (PDF, Imagens) exige alteração na codificação da mensagem HTTP (requisito PI 1 - Certificação).



- Tipos de Codificação (enctype):
 - application/x-www-form-urlencoded (Default): Codifica caracteres em ASCII. Inapto para binários.
 - multipart/form-data (documentos): Divide a requisição em múltiplas partes, permitindo envio de metadados + stream de arquivos binários.

```
<form method="POST" enctype="multipart/form-data">  
    <!-- ↑ ESSENCIAL para arquivos -->  
    {% csrf_token %}  
    <input type="file" name="documento">  
    <button>Enviar</button>  
</form>
```



ENGENHARIA DE FORMULÁRIOS NO DJANGO

Protocolo de Transferência de Arquivos (MIME Types)

```
# views.py - Recebendo arquivos
def upload(request):
    if request.method == 'POST':
        # request.FILES contém os arquivos
        arquivo = request.FILES['documento']

        print(f"Nome: {arquivo.name}")
        print(f"Tamanho: {arquivo.size} bytes")
        print(f"Tipo: {arquivo.content_type}")

    # Salvar no Model com FileField
    documento = Documento(arquivo=arquivo)
    documento.save()
```

- Implementação:
 - Frontend: <form enctype="multipart/form-data" ...>
 - Backend: A view deve receber request.FILES além de request.POST.





ENGENHARIA DE FORMULÁRIOS NO DJANGO

Pipeline de Validação e Sanitização

O processo de garantir que os dados de entrada conformam-se às restrições de tipo, formato e lógica de negócio antes da persistência.

- Método `form.is_valid()`: Dispara o ciclo de validação.
 - a. Field Cleaning: Validação sintática (ex: Email válido? Inteiro?).
 - b. Form Cleaning: Validação semântica e inter-campos.
 - c. `cleaned_data`: dicionário com os dados do formulário após validação.

Sanitização (`cleaned_data`): Se válido, o Django retorna um dicionário de dados higienizados (convertidos para tipos Python nativos), protegidos contra injeção de código malicioso.





ENGENHARIA DE FORMULÁRIOS NO DJANGO

Pipeline de Validação e Sanitização

```
# forms.py
class ProdutorForm(forms.ModelForm):
    class Meta:
        model = Produtor
        fields = ['cpf', 'email']

    def clean_cpf(self):
        cpf = self.cleaned_data['cpf']
        if not validar_cpf(cpf): # Sua validação
            raise forms.ValidationError("CPF inválido")
        return cpf

    def clean(self):
        # Validação entre campos
        data = super().clean()
        if data.get('idade') < 18 and data.get('salario') > 5000:
            raise forms.ValidationError("Idade e salário incompatíveis")
        return data
```

```
# views.py - Usando validação
def salvar(request):
    if request.method == 'POST':
        form = ProdutorForm(request.POST)
        if form.is_valid(): # Dispara todo o pipeline
            dados_limpos = form.cleaned_data # Dados sanitizados
            form.save() # Salva no banco
            return redirect('sucesso')
        else:
            # Mostra erros no template
            return render(request, 'form.html', {'form': form})
```





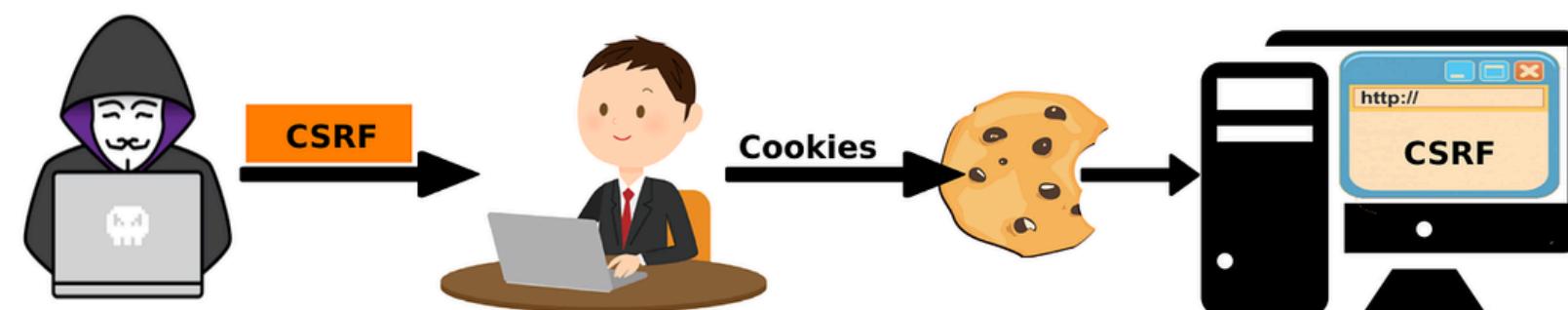
SEGURANÇA DA INFORMAÇÃO

Vetor de Ataque: CSRF (Cross-Site Request Forgery)

Vulnerabilidade que permite a um atacante induzir usuários autenticados a executarem ações indesejadas em uma aplicação web na qual confiam.

- Mecanismo do Ataque:
 - a. Usuário faz login em banco.com → Cookie de sessão é armazenado
 - b. Usuário visita site-malicioso.com
 - c. Site malicioso envia requisição oculta para banco.com/transferir
 - d. Navegador anexa AUTOMATICAMENTE o cookie de sessão
 - e. Banco processa como se fosse ação legítima do usuário

Transferências bancárias, mudança de senha, exclusão de dados.





SEGURANÇA DA INFORMAÇÃO

Contramedida: Synchronizer Token Pattern

Padrão de segurança onde o servidor emite um token (hash aleatório) único para cada sessão/formulário e o valida na submissão.

- Implementação no Django:
 - Middleware: CsrfViewMiddleware verifica a presença do token em métodos inseguros (POST, PUT, DELETE).
 - Template Tag: {% csrf_token %} injeta um <input type="hidden"> com o hash.

Consequência: Se o token estiver ausente ou incorreto, o servidor rejeita a conexão com HTTP 403 Forbidden.





SEGURANÇA DA INFORMAÇÃO

Contramedida: Synchronizer Token Pattern

```
# settings.py - Já vem ativado por padrão
MIDDLEWARE = [
    'django.middleware.csrf.CsrfViewMiddleware', # ← VERIFICA TOKENS
]
```

```
<form method="POST">
    {% csrf_token %} 
    <input name=" dado" type="text">
    <button>Enviar</button>
</form>
```

- GET: Servidor envia formulário + token único
- POST: Cliente devolve dados + token
- Django compara: token recebido == token esperado?
- Diferente → Bloqueia (403) | Igual → Processa





SEGURANÇA DA INFORMAÇÃO

View Function - Padrão GET/POST

Utilização de Views Baseadas em Função (FBV) para gerenciar o fluxo GET/POST.

```
from django.shortcuts import render, redirect

def criar_produto(request):
    if request.method == 'POST': # USUÁRIO ENVIOU DADOS
        form = ProdutoForm(request.POST, request.FILES)
        if form.is_valid(): # VALIDAÇÃO
            form.save() # PERSISTÊNCIA
            return redirect('lista') # POST-REDIRECT-GET
    else: # PRIMEIRO ACESSO (GET)
        form = ProdutoForm()

    return render(request, 'form.html', {'form': form})
```

Por que POST-REDIRECT-GET?

- Problema: Recarregar página após POST reenvia dados
- Solução: Redirect após sucesso → Nova requisição GET
- Benefício: Evita duplicação ao recarregar (F5)





SEGURANÇA DA INFORMAÇÃO

Feedback de Interface (Flash Messages)

O sistema deve fornecer feedback imediato sobre o resultado das ações do usuário (Heurística de Nielsen).



- Django Messages Framework: Armazenamento temporário de mensagens na sessão/cookie que são consumidas na próxima requisição.
- Níveis de Log:
 - messages.debug() - Depuração
 - messages.info() - Informação
 - messages.success() - Sucesso ✓
 - messages.warning() - Aviso !
 - messages.error() - Erro X
- Código: `messages.success(request, "Operação realizada com sucesso!")`.



SEGURANÇA DA INFORMAÇÃO

Feedback de Interface (Flash Messages)

```
# Sucesso
```

```
messages.success(request, 'Arquivo enviado com sucesso!')
```

```
# Erro
```

```
messages.error(request, 'Senha incorreta. Tente novamente.')
```

```
# Aviso
```

```
messages.warning(request, 'Sua sessão expirará em 5 minutos.')
```





SEGURANÇA DA INFORMAÇÃO

Checklist de Segurança Django

1. CSRF Protection:

```
MIDDLEWARE = [  
    'django.middleware.csrf.CsrfViewMiddleware', # ✓ ATIVADO  
]
```

2. Todos os Forms:

```
<form method="POST">  
    {% csrf_token %} <!-- ✓ OBRIGATÓRIO -->  
    <!-- campos -->  
</form>
```

3. Views Seguras:

```
def view_segura(request):  
    if request.method == 'POST':  
        form = MeuForm(request.POST)  
        if form.is_valid(): # ✓ VALIDAÇÃO  
            form.save() # ✓ CLEANED_DATA  
        return redirect('sucesso') # ✓ POST-REDIRECT-GET
```





SEGURANÇA DA INFORMAÇÃO

PROJETOS INTEGRADORES

- Aplicação no PI 1 (Certificação e Comércio Justo): Submissão de Documentação
- Requisitos Técnicos:
 - Campos Textuais: Nome, CPF (com máscara no frontend), Endereço.
 - Campo Binário: FileField para upload do "Certificado de Propriedade" (PDF).
 - Validação Customizada: Implementar método clean_documento() para rejeitar arquivos maiores que 5MB ou extensões não permitidas (.exe, .bat).
- Aplicação no PI 2 (Sistema PDV): Interface de Frente de Caixa.
- Requisitos de Usabilidade:
 - Latência Mínima: O formulário deve ser otimizado para digitação contínua





Atividade Prática Guiada

Atividade Prática

A GastroTech precisa de uma Página Inicial (Landing Page) pública onde o Cliente Final possa fazer um "Pedido Rápido" sem precisar de login.



- Implementar:
 - Usa POST e CSRF (o cliente enviando dados).
 - Usa ModelForm (baseado no modelo Pedido).
 - Usa Widgets (para estilizar os inputs e o select).
 - Conecta o "Frontend Público" com o "Backend Administrativo"



Atividade Prática Guiada

Entrega 5: PROJETO INTEGRADOR (Sprint 4)

Controle de Acesso e Interface (foco em Backend e Frontend)



- Semana 8: Planejamento da Sprint 4
- ○ Objetivo: Planejar e Desenvolver as permissões de acesso do usuário e desenvolver a interface.
- ○ Rituais Scrum: Sprint Planning 4.
- ○ Atividades:
 - Sprint Backlog Atualizado: Tarefas da Sprint 4 no quadro (Trello/GitHub)
 - Implementação de Login, Logout e Grupos de Permissão (caso houver).
 - Desenvolver o Frontend básico para Interação e Testes.
 - Desenvolver a integração do Front com Backend.
- ○ Artefatos:
 - Código-fonte funcional (4º incremento) no GitHub: rotas protegidas e interface amigável.
 - Teste Funcional: comunicação frontend e backend.
 - Sprint Review (Demo) e Atualização do Quadro Trello



MINISTÉRIO DA
CIÊNCIA, TECNOLOGIA
E INOVAÇÃO

