SERVENT SSR

Manual Oficial da Extensão VS Code

Extensão v3.0 · Plataforma Servent-SSR v7.0

Fastify · EJS · Clean Architecture · MPA · Node.js

Geração de Código · Padronização · Produtividade · Testes · Arquitetura

Eduardo de Freitas Arantes

1. O Que é o Servent-SSR

Servent-SSR é uma plataforma arquitetural para aplicações web MPA (Multi-Page Application) construídas com Node.js, Fastify e EJS. A extensão VS Code é o braço operacional desta plataforma: ela transforma convenções arquiteturais em código concreto, eliminando o trabalho manual de criação de arquivos, a inconsistência entre projetos e o tempo gasto em tarefas repetitivas.

Em termos práticos: a extensão sabe exatamente como um projeto Servent-SSR deve ser organizado, como cada artefato deve ser nomeado, como as camadas devem se comunicar e como os componentes de interface devem ser encapsulados. Ao invés de o desenvolvedor decorar ou consultar a documentação para cada novo arquivo, a extensão gera o código correto com um único comando.

1.1 O Problema que a Extensão Resolve

Sem a extensão, criar um novo módulo em um projeto Servent-SSR significa:

- Criar manualmente 8 a 12 arquivos com estrutura específica
- Lembrar a convenção de nomenclatura de cada camada
- Escrever o boilerplate de Plugin Factory, Controller, Service e Repository do zero
- Garantir que o prefixo CSS está correto em cada arquivo style.css
- Garantir que o namespace window.Servent está correto em cada script.js
- Criar os testes unitários e E2E com a estrutura adequada
- Registrar o módulo no server.js sem esquecer nenhuma linha

Com a extensão, todo esse trabalho é reduzido a um wizard de 3 passos que leva menos de 30 segundos.

> A extensão não apenas gera código — ela é a arquitetura Servent-SSR executável. Cada arquivo gerado está em conformidade com o padrão, desde o primeiro caractere.

1.2 Cenário: Antes e Depois

Situação	Sem a Extensão	Com a Extensão
Novo módulo completo	30 a 60 minutos	< 1 minuto (wizard)
Novo component modal	15 a 20 minutos	< 10 segundos
Nova page com CSS/JS	10 a 15 minutos	< 10 segundos
Migration de banco	5 a 10 minutos	< 30 segundos (snippet)
Teste E2E de component	20 a 40 minutos	Automático (gerado)
Onboarding de novo dev	1 a 2 semanas	1 a 2 dias
Descobrir violação CSS	Revisão manual	Sublinhado automático
Navegar pelo projeto	Explorer do SO	Servent Explorer (painel)

1.3 Filosofia da Plataforma

A filosofia do Servent-SSR é: cada decisão arquitetural deve ser tomada uma única vez e depois executada automaticamente. A estrutura de pastas, a nomenclatura de classes CSS, o namespace JavaScript, a separação de camadas — tudo isso é definido no padrão e executado pela extensão. O desenvolvedor foca na lógica de negócio, não em infraestrutura de código.

- Convenção sobre configuração: a extensão conhece as regras; o desenvolvedor não precisa memorizá-las.
- Locality of Behavior: cada arquivo tem uma responsabilidade única, e a extensão garante isso na criação.
- Separação de camadas: Controller, Service, Repository e View nunca se misturam — a extensão gera cada um no lugar correto.
- SSR como padrão primário: o HTML gerado pelo servidor é completo, sem hydration e sem dependência de JavaScript no cliente para a renderização inicial.

2. Principais Benefícios

2.1 Velocidade de Desenvolvimento

A extensão elimina todo o boilerplate. Um módulo que normalmente leva uma hora de setup está pronto em menos de um minuto, com todos os arquivos no lugar correto, todos os nomes corretos e todo o código de infraestrutura escrito. O desenvolvedor começa imediatamente na lógica de negócio.

> **Geração em milissegundos**
> Plugin Factory, Controller SSR, Controller API, Service, Repository, Domain, Page, Testes unitários — tudo gerado com um único wizard.

2.2 Padronização Absoluta entre Projetos

Qualquer desenvolvedor que conheça a extensão pode abrir qualquer projeto Servent-SSR e navegar imediatamente. A estrutura de pastas é idêntica, a nomenclatura é idêntica, os padrões de código são idênticos. Não há projetos com "estilo do desenvolvedor X" — há projetos Servent-SSR.

> **Convenções automáticas**
> Prefixo CSS .srv-[categoria]-[nome] derivado do caminho. Namespace window.Servent.[Categoria].[Nome] derivado do caminho. Zero decisões manuais.

2.3 Redução de Erros Arquiteturais

A extensão não apenas gera código — ela também o valida. O validador automático sublinha em tempo real qualquer violação do padrão: classe CSS com prefixo errado, JavaScript exposto fora do namespace window.Servent, IIFE ausente, component sem documentação de variáveis. Erros que antes chegavam em code review agora são capturados enquanto o código é escrito.

> **Validação em tempo real**
> Diagnósticos automáticos ao salvar. Code Actions (lâmpada) com correção automática de uma única tecla.

2.4 Onboarding Acelerado

Um desenvolvedor novo em um projeto Servent-SSR com a extensão instalada tem acesso imediato a: Servent Explorer (mapa visual do projeto), snippets documentados para cada padrão e comandos de geração que ensinam a estrutura ao serem executados. O tempo para o primeiro commit produtivo cai de semanas para dias.

2.5 SSR de Alta Performance

O padrão gerado pela extensão produz HTML completo no servidor sem hydration. O browser recebe um documento pronto — CSS e JavaScript de cada componente já injetados inline via EJS. Não há carregamento assíncrono de estilos, não há flash de conteúdo não estilizado, não há JavaScript de runtime de framework. A performance de carregamento é máxima por design.

3. Instalação

3.1 Requisitos

Componente	Versão mínima	Observação
VS Code	1.85.0+	Qualquer edição (OSS, Insiders, Cursor)
Node.js	18.0.0+	Para executar a aplicação gerada
npm	9.0.0+	Para instalar dependências
nvm	Qualquer	Recomendado para gerenciar versões Node

3.2 Instalação via VSIX

A extensão é distribuída como arquivo .vsix. Para instalar:

1. Baixe o arquivo servent-ssr-3.0.0.vsix
2. No VS Code, abra o menu de extensões (Ctrl+Shift+X)
3. Clique nos três pontos (···) no topo do painel de extensões
4. Selecione "Install from VSIX..."
5. Escolha o arquivo servent-ssr-3.0.0.vsix
6. Recarregue o VS Code quando solicitado

Ou via terminal:

```
code --install-extension servent-ssr-3.0.0.vsix
```

3.3 Instalação para Publicar Atualizações

Para desenvolvedores que mantêm a extensão:

```
npm install -g @vscode/vsce
nvm use 20
cd servent-ssr-vscode
vsce package          # gera o .vsix
vsce publish          # publica no Marketplace
```

3.4 Verificação da Instalação

Após instalar, verifique:

- Um novo ícone do Servent-SSR aparece na Activity Bar (barra lateral esquerda do VS Code)
- Ctrl+Shift+P exibe comandos com o prefixo "Servent-SSR:"
- Arquivos .ejs exibem sintaxe colorida com tags EJS destacadas
- Em qualquer .ejs, digitar srv- exibe sugestões de snippet

4. Primeiros Passos — Tutorial Completo

4.1 Inicializando um Projeto do Zero

Com uma pasta vazia aberta no VS Code:

7. Abra a Paleta de Comandos: Ctrl+Shift+P
8. Digite e selecione: Servent-SSR: Inicializar Projeto
9. Confirme com "Sim" na janela de confirmação

A extensão gera imediatamente a estrutura completa:

```
meu-projeto/
├── server.js                    # Fastify configurado e pronto
├── package.json                 # Dependências completas
├── .env                         # Variáveis + keys geradas
├── .gitignore
├── public/
│   ├── css/app.css
│   ├── js/app.js
│   └── vendor/                  # Espaço para libs externas
├── src/
│   ├── infrastructure/
│   │   ├── database/
│   │   │   ├── Repository.js    # Classe base Sequelize
│   │   │   ├── connection.js
│   │   │   ├── migrator.js      # Umzug configurado
│   │   │   ├── models/user.model.js
│   │   │   └── migrations/001-create-users.js
│   │   └── plugins/             # 6 plugins Fastify prontos
│   ├── modules/
│   │   └── auth/                # Módulo de autenticação completo
│   └── views/
│       ├── index.ejs            # Shell HTML
│       ├── pages/auth/login/    # Page de login com CSS e JS
│       ├── components/          # menus/ modals/ sidebars/
│       └── partials/            # 4 partials prontos
└── tests/
    ├── unit/ integration/ e2e/
```

> A inicialização já inclui um módulo de autenticação completo com JWT, sessão, guard de rotas, model de usuário, migration, seed e página de login. O projeto está pronto para rodar.

4.2 Instalando Dependências e Rodando

```
npm install
npm run dev      # nodemon server.js
```

O projeto sobe em http://localhost:3000. A página /auth/login já está funcional com usuário admin criado automaticamente pelo seeder.

> Credenciais do admin gerado: admin@servent.local / Admin@1234 — Troque a senha antes do primeiro deploy em produção.

4.3 Criando o Primeiro Módulo

Com o projeto inicializado, crie o primeiro módulo de negócio:

10. Ctrl+Shift+P → Servent-SSR: Novo Módulo Completo ✨
11. Passo 1: informe o nome do módulo — ex: operators
12. Passo 2: selecione as pages — ex: list, detail
13. Passo 3: selecione os components — ex: modal-detail, sidebar-new
14. Confirme: "Criar tudo"

Resultado imediato:

```
src/modules/operators/
├── operators.plugin.js           # Two-Plugin Factory registrado
├── controllers/
│   ├── operators.controller.js   # Handler SSR: reply.render()
│   └── operators.api.controller.js # Handler JSON: reply.send()
├── services/operators.service.js # Lógica de negócio
├── repositories/operators.repository.js
└── domain/operators.domain.js

src/views/pages/operators/
├── list/   index.ejs + style.css + script.js
└── detail/ index.ejs + style.css + script.js

src/views/components/
├── modals/detail/  index.ejs + style.css + script.js
└── sidebars/new/   index.ejs + style.css + script.js

tests/unit/operators/operators.service.test.js
```

O trecho de registro em server.js é copiado automaticamente para o clipboard. Cole no server.js e o módulo está funcionando.

5. Tudo o que a Extensão Faz

Esta seção documenta cada funcionalidade da extensão com seus detalhes, fluxo de uso e impacto no projeto.

5.1 Inicializar Projeto

> **Comando**
> Servent-SSR: Inicializar Projeto | Ctrl+Shift+P

Gera a estrutura completa de um projeto Servent-SSR do zero. É executado uma única vez por projeto e produz todos os artefatos de infraestrutura necessários para o desenvolvimento imediato.

O que é gerado

- server.js — Bootstrap Fastify com todos os plugins registrados na ordem correta
- package.json — Todas as dependências (Fastify, EJS, Sequelize, Umzug, bcrypt, jsonwebtoken, Jest, Playwright)
- .env — Variáveis de ambiente com SESSION_SECRET e JWT_SECRET gerados dinamicamente (48 chars base64url)
- 4 chaves de ambiente geradas dinamicamente: srvt_local_*, srvt_dev_*, srvt_hmg_*, srvt_prd_*
- 6 plugins Fastify: session, flash, locals (reply.render), static, view (EJS), jwt-auth
- Módulo auth completo: Plugin Factory, Controller SSR/API, Service com bcrypt/JWT, Domain, Guard de rotas
- Model User com Sequelize, migration Umzug, seeder com admin inicial
- Page de login funcional com CSS próprio e validação client-side
- Shell index.ejs, 3 categorias de components (menus/, modals/, sidebars/), 4 partials prontos
- Estrutura de testes (unit/, integration/, e2e/)

> Impacto: o projeto está pronto para npm install && npm run dev imediatamente após a inicialização. Zero configuração adicional necessária.

5.2 Novo Módulo Completo — Wizard ✨

> **Comando**
> Servent-SSR: Novo Módulo Completo ✨ | Ctrl+Shift+P ou botão no Servent Explorer

O comando mais poderoso da extensão. Um wizard guiado de 3 passos que gera um módulo de negócio completo com todas as suas camadas arquiteturais e artefatos de interface.

Fluxo do Wizard

15. Nome do módulo — ex: operators, documents, products
16. Quais pages criar — multi-select: list, detail, new, edit (com seleção múltipla)
17. Quais components criar — multi-select: modal-detail, modal-confirm, sidebar-new, sidebar-edit
18. Confirmação com resumo do que será criado

Artefatos gerados para o módulo "operators" com pages list+detail e modal-detail

```
src/modules/operators/
  operators.plugin.js             # Two-Plugin Factory com guard de
autenticação
  controllers/
    operators.controller.js        # GET /operators → reply.render(page,
dados)
    operators.api.controller.js    # GET /api/operators → reply.send(json)
  services/operators.service.js   # getAll() → { kind, status, body }
  repositories/operators.repository.js
  domain/operators.domain.js

src/views/pages/operators/
  list/   → index.ejs + style.css (.srv-page-operators-list) + script.js
  detail/ → index.ejs + style.css (.srv-page-operators-detail) + script.js

src/views/components/modals/detail/
  index.ejs  → modal acessível com backdrop, header, body, footer
  style.css  → .srv-modal-detail com posicionamento fixed
  script.js  → IIFE com window.Servent.Modals.Detail = { open, close }

tests/unit/operators/operators.service.test.js
```

> Ao finalizar, o trecho de registro no server.js é copiado automaticamente para o clipboard. Cole, salve e o módulo está ativo.

5.3 Novo Módulo Simples

> **Comando**
> Servent-SSR: Novo Módulo | Ctrl+Shift+P ou menu de contexto do Explorer

Cria as camadas de negócio de um módulo (Plugin Factory, Controller, Service, Repository, Domain, page list padrão e teste unitário) sem o fluxo de seleção de pages e components do wizard. Indicado quando o desenvolvedor quer criar o módulo e adicionar pages e components manualmente depois.

5.4 Nova Page

> **Comando**
> Servent-SSR: Nova Page | Ctrl+Shift+P ou menu de contexto

Cria os 3 arquivos de uma page com prefixo CSS e namespace JavaScript derivados automaticamente do caminho informado.

Exemplo: módulo operators, page reports

Entrada: módulo = operators, page = reports

Saída:

```
src/views/pages/operators/reports/
  index.ejs  → page com <%- include('./style.css') %> e <%-
include('./script.js') %>
               inclui _alert-flash.ejs e _page-header.ejs automaticamente
  style.css  → .srv-page-operators-reports { padding: 1.5rem; }
  script.js  → IIFE com window.Servent.Pages.OperatorsReports = { refresh: ...
}
```

> A page já inclui os partials _alert-flash.ejs e _page-header.ejs e está pronta para receber dados do Controller via locals.

5.5 Novo Component

> **Comando**
> Servent-SSR: Novo Componente | Ctrl+Shift+P ou menu de contexto

Cria um component encapsulado com os 3 arquivos. A extensão pergunta a categoria (menus, modals, sidebars ou outra) e o nome. A partir disso, o prefixo CSS e o namespace JavaScript são derivados automaticamente — sem decisão manual do desenvolvedor.

Exemplo: category = modals, name = confirmation

```
src/views/components/modals/confirmation/
  index.ejs  → modal com backdrop, header, body, footer, botões de ação
  style.css  → .srv-modal-confirmation { position: fixed; inset: 0; ... }
               .srv-modal-confirmation[hidden] { display: none; }
  script.js  → IIFE com window.Servent.Modals.Confirmation = { open, close }
               + listeners de Escape e backdrop
```

Para sidebars (offcanvas), a extensão gera um component com panel lateral, backdrop, header e botão de fechar — estrutura completa para offcanvas acessível.

5.6 Novo Partial

> **Comando**
> Servent-SSR: Novo Partial | Ctrl+Shift+P ou menu de contexto

Cria um fragmento EJS puro em src/views/partials/. O arquivo recebe o prefixo _ automaticamente e contém um cabeçalho de documentação indicando que não deve ter CSS ou JS próprios.

```
src/views/partials/_nome-do-partial.ejs
```

> Distinção fundamental: Partials não têm style.css nem script.js. São fragmentos de markup reutilizáveis que herdam CSS e contexto de quem os inclui. Se precisar de comportamento, use um Component.

5.7 Servent Explorer — Painel Lateral

> **Acesso**
> Ícone Servent-SSR na Activity Bar | Atualização automática ao salvar

Um painel dedicado na barra lateral do VS Code que apresenta uma visão arquitetural do projeto. Substitui a navegação cega pelo Explorer de arquivos genérico.

O painel exibe três seções expansíveis:

- □ Pages — lista por módulo → por tela → arquivos (index.ejs, style.css, script.js clicáveis)
- □ Components — lista por categoria (menus/, modals/, sidebars/) → por nome → arquivos
- □ Partials — lista direta de todos os arquivos _nome.ejs

Cada arquivo listado é clicável e abre diretamente no editor. O painel exibe a descrição arquitetural de cada item (prefixo CSS da page, categoria do component).

No header do painel há dois botões: um para refresh manual e um para abrir o Wizard de novo módulo completo — permitindo criar módulos sem sair do painel.

> O Servent Explorer atualiza automaticamente quando qualquer arquivo é salvo, criado ou deletado no projeto.

5.8 Validador em Tempo Real

> **Ativação**
> Automático ao abrir e salvar | Manual: Servent-SSR: Validar Arquivo

O validador analisa cada arquivo Servent-SSR automaticamente e exibe diagnósticos diretamente no editor — os mesmos sublinhados coloridos que o VS Code usa para erros de sintaxe.

Regras validadas por tipo de arquivo

Arquivo	Severidade	Regra verificada
style.css (component/page)	□ Aviso	Classes .srv-* com prefixo diferente do esperado para o caminho
script.js	□ Erro	Ausência de IIFE — código exposto ao escopo global
script.js	□ Aviso	window.X atribuído fora de window.Servent.*
index.ejs (component)	□ Aviso	Falta <%- include('./style.css') %>
index.ejs (component)	□ Aviso	Falta <script><%- include('./script.js') %></script>
index.ejs (component)	□ Informação	Sem bloco de documentação de variáveis esperadas

Arquivo	Severidade	Regra verificada
_partial.ejs	□ Aviso	Presença de <style> ou <script> próprios

5.9 Code Actions — Correções Automáticas (□)

> **Ativação**
> Clique na lâmpada □ ao lado de qualquer diagnóstico Servent-SSR

Para cada diagnóstico gerado pelo validador, a extensão oferece uma correção automática aplicável com um único clique. Não é necessário editar o arquivo manualmente.

Diagnóstico	Correção Automática Oferecida
`Falta include('./style.css')`	Insere <style><%- include('./style.css') %></style> no topo
`Falta include('./script.js')`	Insere <script><%- include('./script.js') %></script> no final
`Sem documentação de variáveis`	Insere bloco <%-- Variáveis esperadas: ... --%> completo no topo
`script.js sem IIFE`	Envolve todo o conteúdo em (function() { 'use strict'; ... })()
`window.MinhaVar = ...`	Renomeia para window.Servent.MinhaVar automaticamente

5.10 Highlight de Sintaxe EJS

> **Ativação**
> Automático em qualquer arquivo .ejs — sem configuração

A extensão registra o EJS como linguagem própria com uma gramática TextMate completa. O VS Code passa a colorir corretamente todos os arquivos .ejs sem depender de extensões externas ou configurações adicionais.

Tag EJS	Significado	Cor no tema
<% %>	Execução — lógica JS sem output	keyword (laranja/amarelo do tema)
<%= %>	Output escapado — exibe string	keyword escaped (verde)
<%- %>	Output não-escapado — exibe HTML	keyword unescaped (azul)
<%# %>	Comentário — não vai para o HTML	comment (cinza)

O código JavaScript dentro das tags EJS tem highlight completo de JavaScript — incluindo palavras-chave, strings, números e funções. O HTML fora das tags tem highlight completo de HTML.

O autoclose de tags EJS está configurado: ao digitar <%, o VS Code fecha automaticamente com %>. O mesmo vale para <%= e <%-.

5.11 EJS Formatter

> **Ativação**
> Shift+Alt+F | Format Document | Format on Save

O Prettier não tem suporte oficial para EJS e frequentemente destrói as tags <% %> ao formatar. A extensão resolve isso com um formatter dedicado para .ejs que entende a sintaxe do padrão.

O que o formatter faz

- Indenta HTML com base no nível de nesting de tags (div, section, main, table, etc.)
- Preserva tags EJS (<%...%>) exatamente como estão — nunca as quebra, move ou reformata
- Não toca no conteúdo de <script> e <style> — respeita o código inline
- Remove espaços em branco à direita de cada linha
- Normaliza linhas em branco consecutivas — máximo de 1 linha em branco seguida

> Para ativar Format on Save em .ejs: abra settings.json e adicione: "[ejs]": { "editor.formatOnSave": true }

5.12 Hover com Documentação Inline

> **Ativação**
> Passe o mouse sobre qualquer include() em .ejs ou window.Servent.* em .js

Ao passar o mouse sobre um include de component em um arquivo .ejs, a extensão lê o bloco de documentação do index.ejs correspondente e exibe um tooltip com todas as informações do component.

Hover em include EJS exibe:

- Nome do component e caminho
- Prefixo CSS obrigatório (.srv-categoria-nome)
- Namespace JavaScript (window.Servent.Categoria.Nome)
- Lista completa de variáveis esperadas (lidas do bloco <%-- Variáveis esperadas: --%>)
- Se não houver documentação: template sugerido para adicionar ao topo do index.ejs

Hover em window.Servent.X.Y em .js exibe:

- Nome completo do namespace
- Métodos da API pública extraídos automaticamente do script.js (open(), close(), refresh(), etc.)
- Link clicável para abrir o script.js do component

5.13 Go to Definition — Ctrl+Click

> **Ativação**
> Ctrl+Click em include() em .ejs | Ctrl+Click em window.Servent.* em .js

O VS Code nativamente não resolve caminhos de include EJS. A extensão adiciona suporte a Go to Definition para os dois contextos mais frequentes:

Contexto	Comportamento do Ctrl+Click
`<%- include('./components/modals/detail/index.ejs') %>`	Abre o arquivo incluído diretamente no editor
`<%- include('./partials/_page-header.ejs') %>`	Abre o partial no editor
`window.Servent.Modals.Detail`	Abre o script.js do component na linha da API pública
`window.Servent.Sidebars.UserNew`	Localiza e abre o script.js da sidebar correspondente

5.14 Gerador de Teste E2E

> **Comando**

> Servent-SSR: Gerar Teste E2E □ | Ctrl+Shift+P com o index.ejs aberto

Com o index.ejs de qualquer component ou page aberto no editor, este comando analisa o template e gera automaticamente um arquivo de teste Playwright completo e adaptado ao tipo do artefato.

O que a extensão extrai do template

- □ IDs de elementos (id="valor") → locators #valor
- □ data-attributes (data-open-detail, data-close) → locators [data-*]
- □ Classes CSS srv-* → locators .srv-*
- □ Campos de formulário (name="email") → page.fill()
- □ Botões de submit → page.click('[type=submit]')
- □ Namespace window.Servent (do bloco de documentação) → chamadas diretas no teste

Testes gerados por tipo de artefato

Tipo	Localização de saída	Casos de teste gerados
Modal	tests/e2e/components/modals/detail.test.js	Oculto por padrão, open(), close(), ESC, clique no backdrop
Sidebar	tests/e2e/components/sidebars/new.test.js	Oculto por padrão, open(), close(), ESC, botão de fechar
Page	tests/e2e/pages/operators/list.test.js	Renderização, submit de formulário, clique em item de lista
Component	tests/e2e/components/categoria/nome.test.js	Presença no DOM com todos os seletores comentados

O arquivo gerado é aberto automaticamente no editor após a criação.

5.15 Snippets EJS

> **Ativação**
> Digite o prefixo em qualquer arquivo .ejs e pressione Tab

Prefixo	Arquivo	O que gera
`srv-shell`	.ejs	Shell HTML completo com head, body, CSS/JS globais e include(pageComponent)

Prefixo	Arquivo	O que gera
`srv-page`	.ejs	Page com include de style.css e script.js, div com prefixo correto
`srv-component`	.ejs	Component encapsulado com role e aria-label
`srv-modal`	.ejs	Modal completo: backdrop, container, header, body, footer, botões de fechar
`srv-sidebar`	.ejs	Sidebar/offcanvas: backdrop, panel, header, body, botão de fechar
`srv-include`	.ejs	<%- include('caminho', { chave: valor }) %> com placeholders
`srv-partial`	.ejs	<%- include('partials/_nome', { dados }) %> com placeholders
`srv-each`	.ejs	<% items.forEach(function(item) { %> ... <% }); %>
`srv-if`	.ejs	<% if (condition) { %> ... <% } else { %> ... <% } %>
`srv-local`	.ejs	<% if (locals.variavel) { %> ... <% } %>
`srv-flash`	.ejs	Bloco completo de flash messages (success, error, info)
`srv-empty`	.ejs	Condicional de empty state com include do partial
`srv-pagination`	.ejs	Include do partial de paginação com currentPage e totalPages

5.16 Snippets JavaScript

> **Ativação**
> Digite o prefixo em qualquer arquivo .js e pressione Tab

Prefixo	Arquivo	O que gera
`srv-iife`	.js	IIFE base com window.Servent, getElementById, funções privadas e API pública
`srv-iife-modal`	.js	IIFE completa de modal: open(data), close(), ESC, backdrop, CustomEvents
`srv-iife-sidebar`	.js	IIFE completa de sidebar: open(), close(), ESC, botões, CustomEvents
`srv-iife-page`	.js	IIFE base de page com window.Servent.Pages.[Nome] e refresh()
`srv-dispatch`	.js	document.dispatchEvent(new CustomEvent('componente:acao', { detail }))
`srv-listen`	.js	document.addEventListener('componente:acao', function(e) { })
`srv-service`	.js	Método async com try/catch e retorno { kind, status, body }
`srv-ok`	.js	return { kind: 'json', status: 200, body: { data } }
`srv-err`	.js	return { kind: 'json', status: 4xx/5xx, body: { erro } } com seleção de status
`srv-controller`	.js	Handler SSR Fastify com out.status check, flash e reply.render()
`srv-api`	.js	Handler JSON Fastify com reply.status(out.status).send(out.body)

Prefixo	Arquivo	O que gera
`srv-plugin`	.js	Two-Plugin Factory completo: viewPlugin + apiPlugin + guard hook
`srv-repo`	.js	Método de Repository com Sequelize findAll, where e order
`srv-migration`	.js	Migration Umzug com up/down e seleção de operação SQL
`srv-migration-table`	.js	Migration createTable completa: id, coluna, created_at, updated_at
`srv-migration-col`	.js	Migration addColumn com tipo, allowNull, defaultValue e removeColumn no down
`srv-fetch-modal`	.js	fetch() + window.Servent.Modals.[Nome].open() com tratamento de erro
`srv-e2e`	.js	Teste E2E Playwright base com beforeAll, afterAll e it()
`srv-e2e-modal`	.js	Teste E2E completo de modal: oculto, open(), ESC, com casos reais

6. Experiência do Desenvolvedor

6.1 Referência de Comandos

Comando	Atalho / Acesso	Quando usar
Inicializar Projeto	Ctrl+Shift+P	Uma vez, no início de cada projeto
Novo Módulo Completo ✨	Ctrl+Shift+P ou Explorer	Toda nova feature de negócio
Novo Módulo	Ctrl+Shift+P ou Explorer	Quando quer criar camadas sem pages
Nova Page	Ctrl+Shift+P ou Explorer	Para adicionar telas a um módulo
Novo Component	Ctrl+Shift+P ou Explorer	Para adicionar modal, sidebar, etc.
Novo Partial	Ctrl+Shift+P ou Explorer	Para fragmentos de markup reutilizáveis
Validar Arquivo	Ctrl+Shift+P	Verificação manual antes de commit
Gerar Teste E2E □	Ctrl+Shift+P	Com index.ejs de component/page aberto
Atualizar Explorer	Botão no painel	Após operações externas ao VS Code

6.2 Menu de Contexto do Explorer

Ao clicar com o botão direito em qualquer pasta no Explorer do VS Code, o menu exibe os comandos Servent-SSR:

- Servent-SSR: Novo Módulo Completo ✨(primeira opção — grupo servent@0)
- Servent-SSR: Novo Módulo
- Servent-SSR: Novo Componente
- Servent-SSR: Nova Page
- Servent-SSR: Novo Partial

6.3 Configurações da Extensão

A extensão tem uma configuração disponível em File → Preferences → Settings → Servent-SSR:

Configuração	Valores e comportamento
`servent-ssr.framework`	fastify (padrão) | express — determina o framework HTTP nos templates gerados

6.4 Fluxo de Trabalho Diário Recomendado

Fluxo para adicionar uma nova funcionalidade completa ao projeto:

19. Ctrl+Shift+P → Novo Módulo Completo ✨→ preencher o wizard
20. Colar o trecho de registro no server.js (já está no clipboard)
21. Abrir o service gerado e implementar a lógica de negócio
22. Abrir o repository gerado e adicionar as queries específicas
23. Abrir a page gerada e incluir os components necessários
24. Com o index.ejs aberto, Ctrl+Shift+P → Gerar Teste E2E □
25. Rodar npm test para verificar os testes unitários
26. Rodar playwright test para os E2E

> O validador está sempre ativo. Qualquer violação de padrão (prefixo CSS errado, window.X fora do namespace) é sublinhada em tempo real — corrija antes de salvar.

7. Arquitetura Reforçada pela Extensão

7.1 As Quatro Camadas da View

A extensão conhece e reforça a separação em quatro camadas distintas. Cada comando gera artefatos na camada correta, com as convenções corretas, sem que o desenvolvedor precise tomar essa decisão.

Camada	Localização	Criada por	Característica
Shell	src/views/index.ejs	Inicializar Projeto	1 por projeto. Assets globais de public/. Inclui pageComponent.
Page	src/views/pages/[mod]/[pag]/	Nova Page / Wizard	3 arquivos. CSS .srv-page-*. JS window.Servent.Pages.*
Component	src/views/components/[cat]/[nome]/	Novo Componente / Wizard	3 arquivos. CSS .srv-[cat]-*. JS window.Servent.[Cat].*
Partial	src/views/partials/_nome.ejs	Novo Partial	EJS puro. Sem CSS/JS próprios. Herda contexto.

7.2 Clean Architecture — As Camadas de Negócio

A extensão gera cada camada no lugar correto e com a responsabilidade correta. O desenvolvedor não precisa decidir "onde coloco isso" — a arquitetura já está expressa na geração.

Camada	Responsabilidade	Regra reforçada pela extensão
Plugin Factory	Registra rotas e guard hooks	Nunca contém lógica de negócio
Controller SSR	Chama Service → reply.render()	Único arquivo que referencia paths de view
Controller API	Chama Service → reply.send()	Apenas { kind, status, body } do Service
Service	Lógica de negócio pura	Sem req, res, EJS ou paths de arquivo
Repository	Queries Sequelize	Único ponto de acesso ao banco de dados
Domain	Entidades e invariantes	Sem imports de infraestrutura

7.3 Convenções Derivadas Automaticamente

A extensão nunca pede ao desenvolvedor que defina prefixo CSS ou namespace JavaScript. Eles são derivados deterministicamente do caminho da pasta — garantindo unicidade e rastreabilidade.

Pasta do artefato	Derivação automática
`components/modals/detail/`	CSS: .srv-modal-detail | JS: window.Servent.Modals.Detail
`components/sidebars/user-new/`	CSS: .srv-sidebar-user-new | JS: window.Servent.Sidebars.UserNew
`components/menus/left/`	CSS: .srv-menu-left | JS: window.Servent.Menus.Left
`pages/operators/dashboard/`	CSS: .srv-page-operators-dashboard | JS: window.Servent.Pages.OperatorsDashboard
`pages/operators/documents/list/`	CSS: .srv-page-operators-documents-list | JS: window.Servent.Pages.OperatorsDocumentsList

8. Casos de Uso por Tipo de Projeto

8.1 Sistema Administrativo / Back-office

Cenário: painel de gestão com múltiplos módulos (usuários, pedidos, relatórios), controle de acesso por role e dashboards com dados tabulares.

Como a extensão ajuda

- Inicializar Projeto gera o módulo de autenticação com controle de sessão e JWT já integrado
- Wizard cria cada módulo (usuarios, pedidos, relatorios) com Controller SSR + API + Service + Repository em menos de 1 minuto cada
- O guard isAuthenticatedHook e requireRole() gerado são prontos para uso — basta configurar as roles
- Pages list com empty state, paginação e alert flash geradas automaticamente
- Modals de detalhe e sidebars de edição gerados com comportamento completo
- Testes E2E gerados automaticamente para cada modal e página de formulário

8.2 Portal SSR Corporativo / Intranet

Cenário: portal com conteúdo estruturado, busca, autenticação corporativa e múltiplas seções de conteúdo.

Como a extensão ajuda

- Shell único com menu lateral gerado pela extensão — component menus/left com collapse/expand
- Partials _page-header.ejs e _pagination.ejs reutilizados em todas as pages sem duplicação
- Cada seção do portal é um módulo Servent-SSR — criado em 30 segundos com o wizard
- O validador garante consistência de nomenclatura CSS em toda a base de código
- O Servent Explorer permite navegar pela estrutura do portal rapidamente

8.3 E-commerce Tradicional

Cenário: loja com catálogo de produtos, carrinho, checkout e painel do vendedor.

Como a extensão ajuda

- Módulo products com pages list, detail, new geradas pelo wizard
- Modal de detalhe do produto gerado com open(data) pronto para fetch de dados
- Sidebar de filtros gerada como component sidebar com estado encapsulado na IIFE
- Módulo cart e módulo checkout gerados com suas próprias camadas isoladas
- CustomEvents (cart:item-added, cart:updated) gerados via snippet srv-dispatch para comunicação entre components

8.4 API Enterprise com Rotas SSR e JSON

Cenário: plataforma com interface web e API REST consumida por mobile ou parceiros.

Como a extensão ajuda

- O Two-Plugin Factory gerado por padrão separa rotas SSR de rotas API no mesmo módulo
- Controller SSR (reply.render) e Controller API (reply.send) gerados como arquivos separados
- O contrato { kind, status, body } do Service garante que a mesma lógica serve ambos os controllers
- snippet srv-api gera handlers JSON em segundos com o padrão correto
- Testes unitários do Service cobrem tanto o fluxo SSR quanto o API

8.5 Sistema Governamental / Plataforma Operacional

Cenário: sistema com múltiplos perfis de usuário (admin, operador, analista), fluxos de aprovação e requisitos de acessibilidade.

Como a extensão ajuda

- requireRole('admin', 'operador') gerado no guard hook de cada módulo — controle de acesso por feature
- Modals e sidebars gerados com atributos ARIA completos (role, aria-modal, aria-labelledby, aria-label)

- Botões de fechar com aria-label='Fechar' e suporte a ESC gerados automaticamente
- Chaves de ambiente srvt_local_*, srvt_dev_*, srvt_hmg_*, srvt_prd_* geradas para separação total entre ambientes
- Migrations Umzug versionadas garantem rastreabilidade de alterações de schema

9. Impacto Operacional e Ganhos Mensuráveis

9.1 Redução de Código Repetitivo

A tabela abaixo compara a quantidade de código que o desenvolvedor escreve manualmente versus o que a extensão gera:

Artefato	Linhas geradas pela extensão	O que o desenvolvedor escreve
Plugin Factory completo	~35 linhas	Nome do módulo (1 input)
Controller SSR	~25 linhas	Lógica específica da rota
Controller API	~10 linhas	Nada — o padrão é fixo
Service com try/catch	~15 linhas	A lógica de negócio
Modal completo (3 arquivos)	~80 linhas	O markup do corpo do modal
Sidebar completa (3 arq.)	~70 linhas	O conteúdo do painel
Page com CSS e JS	~30 linhas	O markup da tela
Teste E2E de modal	~45 linhas	Ajuste de URL e validações
Migration createTable	~20 linhas	As colunas específicas

9.2 Impacto no Onboarding

Sem a extensão, um desenvolvedor novo em um projeto Servent-SSR precisa:

- Ler a documentação arquitetural completa para entender as convenções
- Memorizar as regras de nomenclatura de CSS e JavaScript
- Estudar exemplos de cada tipo de artefato antes de criar o primeiro
- Ter o código revisado nas primeiras semanas para garantir conformidade

Com a extensão:

- O Servent Explorer mostra a estrutura do projeto de forma visual imediata
- Os snippets sv-* são auto-documentados — o desenvolvedor vê o padrão ao usá-los
- O validador corrige em tempo real, ensinando as convenções sem intervenção humana
- O wizard guia o desenvolvedor pelo processo correto de criação de módulo

> O onboarding de um desenvolvedor novo em um projeto Servent-SSR com a extensão instalada leva de 1 a 2 dias para o primeiro módulo funcional em produção.

9.3 Consistência em Times Distribuídos

Em times com múltiplos desenvolvedores trabalhando em paralelo, a extensão atua como árbitro de padrões:

- Cada desenvolvedor usa o mesmo wizard para criar módulos — o resultado é idêntico independentemente de quem executou
- O validador detecta divergências do padrão antes do code review — menos comentários de revisão sobre estilo
- O Servent Explorer permite que qualquer desenvolvedor entenda imediatamente a estrutura do trabalho de outro
- As chaves de ambiente geradas dinamicamente garantem que cada desenvolvedor tem credenciais únicas no ambiente local

10. Por Que Usar o Servent-SSR

Esta seção apresenta uma síntese dos ganhos arquiteturais e operacionais da plataforma.

10.1 Comparativo de Abordagens

Critério	Servent-SSR + Extensão	Abordagem Manual (qualquer stack)
Novo módulo de negócio	**< 1 minuto, wizard guiado**	30–60 minutos, criação manual
Conformidade arquitetural	**Automática, validada em tempo real**	Dependente de revisão humana
Onboarding de dev novo	**1–2 dias para primeiro deploy**	1–2 semanas
Consistência CSS/JS	**Garantida por convenção derivada**	Garantida por disciplina individual
Testes E2E	**Gerados automaticamente do template**	Escritos do zero por desenvolvedor
Navegação no projeto	**Servent Explorer arquitetural**	Explorer genérico de arquivos
Erros de padrão	**Detectados enquanto o código é escrito**	Detectados em code review
Documentação inline	**Hover com assinatura e variáveis**	Consulta à documentação externa

10.2 Quando o Servent-SSR é a Escolha Certa

- Sistemas administrativos, dashboards, portais corporativos, intranets e plataformas operacionais
- E-commerces com fluxo tradicional de navegação entre páginas
- Times com diferentes níveis de senioridade que precisam de padronização automática
- Projetos que exigem SSR real (SEO, primeiro carregamento rápido, sem JavaScript de framework no cliente)
- Projetos Node.js com Fastify onde Clean Architecture é um requisito arquitetural
- Aplicações onde manutenibilidade e onboarding de longo prazo são prioridade

10.3 Quando Outras Abordagens São Mais Indicadas

- Aplicações com estado complexo distribuído entre muitos componentes em tempo real
- Editores colaborativos, ferramentas de design baseadas em browser ou jogos de browser
- SPA com roteamento client-side como requisito central da experiência do usuário

10.4 O Compromisso da Plataforma

O Servent-SSR é uma plataforma que evolui com o projeto. A extensão garante que cada artefato criado hoje — seja por um desenvolvedor sênior ou júnior — está em conformidade com o padrão. Daqui a seis meses, quando outro desenvolvedor precisar manter o código, a estrutura será imediatamente legível porque é sempre a mesma estrutura.

Esse é o ganho mais difícil de quantificar e o mais valioso a longo prazo: a certeza de que o projeto vai envelhecer bem.