

PROCEDIMENTO OPERACIONAL

MÉTODO DE DESENVOLVIMENTO DE SITES

SOP-007

Portar Capacidade da Base para Projeto Legado

VERSÃO

v1.1 FINAL

DATA

11/09/2026

STATUS

FINAL — APROVADO

PROCEDIMENTO OPERACIONAL

SOP-007

Portar Capacidade da Base para Projeto Legado

VERSÃO	v1.1 FINAL
DATA	11/09/2026
STATUS	FINAL — APROVADO
MANUAL PROPRIETÁRIO	Manual 4 — Projeto-Base, Admin e Funcionalidades
MANUAIS RELACIONADOS	Manual 5 — Desenvolvimento com Codex, Lovable e GitHub; Manual 6 — Testes, QA, Debugging, Segurança e SEO; Manual 8 — Boas Práticas, Anti-Patterns, Práticas Depreciadas e Casos Reais
ESCOPO	AS-IS — portabilidade incremental e adaptativa de capacidades da Base Admin para projetos existentes/legados

NAVEGAÇÃO DO DOCUMENTO

Sumário

Estrutura do documento canônico v1.1 FINAL.

1. Finalidade	7
2. Gatilho — quando usar	7
3. Quando NÃO usar	8
4. Papéis	8
5. Pré-condições	9
6. Inputs	9
7. Ferramentas	10
PARTE I — DEFINIR O RECORTE	11
8. Passo 1 — Nomear a capacidade-alvo	12
9. Passo 2 — Descrever o resultado funcional esperado	12
10. Passo 3 — Delimitar o que está fora do recorte	12
PARTE II — INVENTARIAR O DESTINO	14
11. Passo 4 — Ler o projeto de destino antes de editar	15
12. Passo 5 — Identificar a geração técnica	15
13. Passo 6 — Inventariar o que o destino já possui	15
PARTE III — INVENTARIAR A ORIGEM	17
14. Passo 7 — Identificar a implementação correspondente na Base	18
15. Passo 8 — Separar capacidade de implementação	18
PARTE IV — MATRIZ ORIGEM × DESTINO	19
16. Passo 9 — Classificar equivalências	20
17. Equivalente	20
18. Parcialmente equivalente	20
19. Ausente no destino	20
20. Melhor no destino	20
21. Incompatível	21
22. Nova funcionalidade	21

PARTE V — SEPARAR PARIDADE DE FEATURE NOVA	22
23. Passo 10 — Perguntar “isso é delta ou produto novo?”	23
24. Caso real — Leads do Guilherme Maia	23
25. Caso real — “Jornada do paciente”	23
PARTE VI — MAPEAR DEPENDÊNCIAS	24
26. Passo 11 — Mapear dependências da capacidade	25
27. Passo 12 — Classificar dependências	25
28. Regra AS-IS	26
PARTE VII — BANCO, MIGRATIONS E DADOS	27
29. Passo 13 — Verificar impacto em banco/dados	28
30. AUSÊNCIA ATUAL DE PADRONIZAÇÃO — migrations	28
31. Dados existentes	28
32. Passo 14 — Separar migration de alteração frontend-only	29
PARTE VIII — AUTENTICAÇÃO, ROLES E PERMISSÕES	30
33. Passo 15 — Verificar se a capacidade depende de autorização	31
34. Não portar apenas a UI de permissão	31
35. Diferenças entre gerações	31
PARTE IX — DEFINIR O DELTA APROVADO	32
36. Passo 16 — Consolidar o delta	33
37. Stop intermediário	33
PARTE X — IMPLEMENTAR DEPENDÊNCIAS	34
38. Passo 17 — Portar/adaptar dependências necessárias	35
39. Regra de contenção	35
40. Validar dependência antes da camada seguinte	35
PARTE XI — IMPLEMENTAR A CAPACIDADE	36
41. Passo 18 — Adaptar a capacidade ao destino	37
42. Não perseguir identidade de código	37
43. Preservar comportamento local melhor	37
PARTE XII — ALTERAÇÕES FRONTEND-ONLY	38
44. Quando o delta é apenas frontend	39
45. Não criar migration sem necessidade	39

PARTE XIII — ALTERAÇÕES COM BACKEND/BANCO	40
46. Quando o delta exige backend	41
47. Quando exige banco	41
PARTE XIV — ESTABILIZAÇÃO	42
48. Passo 19 — Executar a capacidade no destino	43
49. Passo 20 — Corrigir incompatibilidades	43
50. Não “resolver” regressão apagando comportamento local	43
PARTE XV — VALIDAÇÃO FUNCIONAL	44
51. Passo 21 — Testar a capacidade portada	45
52. Passo 22 — Testar o que foi preservado	45
53. Passo 23 — Testar dependências	45
PARTE XVI — REGRESSÃO	46
54. Regra AS-IS	47
55. Passo 24 — Comparar antes x depois	47
56. QA completo	47
PARTE XVII — ROLLBACK E CONTINGÊNCIA	49
57. AUSÊNCIA ATUAL DE PADRONIZAÇÃO — rollback específico de portabilidade	50
58. Recursos observados no método geral	50
59. Quando parar	50
PARTE XVIII — PROJETOS COM ARQUITETURA ANTIGA	51
60. Legado não precisa virar projeto atual inteiro	52
61. Norma técnica vigente	52
PARTE XIX — CASO GUILHERME MAIA	53
62. Papel do caso	54
63. O que o caso ensina	54
64. Leads como exemplo	54
65. Blog como exemplo	55
66. O que NÃO generalizar do caso Maia	55
PARTE XX — STOP CONDITION	56
67. Condição de parada	57
68. Checklist mínimo	57
69. O que NÃO é necessário para encerrar	57

PARTE XXI — CONTINGÊNCIAS	59
70. A Base tem algo que o destino já faz melhor	60
71. O destino já tem a capacidade	60
72. A capacidade depende de algo ausente	60
73. A dependência exigiria mudança ampla	60
74. A comparação revela feature nova	60
75. A migration ameaça dados existentes	60
76. A nova capacidade quebra comportamento local	61
77. A UI funciona, mas backend não	61
78. A portabilidade exige trocar framework	61
PARTE XXII — AUSÊNCIAS ATUAIS DE PADRONIZAÇÃO	62
79. Não inventar	63
PARTE XXIII — NORMA TÉCNICA VIGENTE	64
80. Normas continuam aplicáveis em camada própria	65
PARTE XXIV — ANTI-PATTERNS	66
81. “Copiar o Admin novo por cima”	67
82. Comparar arquivos sem comparar função	67
83. Refazer capacidade equivalente	67
84. Portar UI sem dependência	67
85. Chamar feature nova de “paridade”	67
86. Apagar particularidade local porque não existe na Base	67
87. Assumir migration idêntica	67
88. Transformar atualização seletiva em migração de stack	68
PARTE XXV — OUTPUT	69
89. Saída esperada	70
PARTE XXVI — PRÓXIMO PASSO	71
90. Encaminhamento	72
91. Referências	72

1. Finalidade

Portar para um projeto existente somente a capacidade realmente necessária da Base Admin atual, preservando o que o destino já possui e evitando substituição indiscriminada do painel ou da arquitetura do projeto.

Fluxo principal:

```
necessidade de evolução no legado
→ definir capacidade-alvo
→ comparar Base atual × destino
→ inventariar equivalências
→ identificar delta real
→ separar paridade de feature nova
→ identificar dependências
→ avaliar impacto em dados/auth/backend
→ portar dependências necessárias
→ adaptar a capacidade ao destino
→ estabilizar
→ validar capacidade
→ revalidar dependências/regressões
→ concluir delta
```

Princípio central:

A Base é referência funcional reutilizável, não autoridade absoluta para sobrescrever o projeto existente.

2. Gatilho — quando usar

Usar quando um site/projeto existente precisa receber:

- uma capacidade que evoluiu na Base;
- parte de um módulo atual;
- melhoria funcional já existente na Base;
- comportamento equivalente ao padrão atual;
- atualização seletiva de Admin/Blog/Leads/Chat/CTA ou outra capacidade reutilizável.

O projeto de destino já existe e possui implementação própria.

3. Quando NÃO usar

Não usar:

- para criar projeto novo;
- para remixar a Base em novo cliente;
- para substituir automaticamente todo o Admin;
- para redesenhar a arquitetura completa do legado;
- para incluir funcionalidade nova apenas porque “seria melhor”;
- para executar modernização tecnológica geral;
- para converter todo projeto antigo para a stack mais recente;
- para executar QA geral sem uma portabilidade específica.

Procedimentos relacionados

- projeto novo → SOP-002;
 - configuração normal da Base em cliente novo → SOP-005;
 - debugging → SOP-009;
 - QA → SOP-008.
-

4. Papéis

4.1 Desenvolvedor

Responsável por:

- comparar origem e destino;
- inventariar capacidade existente;
- identificar delta;
- identificar dependências;
- adaptar implementação;
- preservar comportamento local melhor;
- testar a capacidade;
- revalidar impacto.

4.2 WSS / Direção / Operações

Responsável por decidir quando houver:

- dúvida de escopo;
- funcionalidade nova;
- mudança comercial;
- risco relevante;
- necessidade de alteração ampla;
- conflito entre manter legado e evoluir capacidade.

4.3 IA / Codex / Lovable / Claude

Podem auxiliar em:

- inspeção;
- comparação;
- implementação;
- diagnóstico;
- testes.

Não decidem autonomamente:

- escopo;
 - perda aceitável de funcionalidade;
 - migração de dados;
 - remoção de comportamento local;
 - troca de arquitetura.
-

5. Pré-condições

Antes de iniciar:

1. projeto de destino identificado;
2. Base atual identificada;
3. capacidade-alvo conhecida;
4. acesso ao código/configuração necessária;
5. estado atual do destino compreendido;
6. origem e destino comparáveis no nível da capacidade;
7. responsável humano disponível para decisões de escopo quando necessário.

Não é pré-condição

O destino não precisa:

- usar a mesma stack;
 - ter a mesma estrutura;
 - ter o mesmo Admin;
 - estar na geração técnica atual.
-

6. Inputs

Obrigatórios

- capacidade-alvo na Base;
- implementação atual do destino;

- requisito/necessidade que motivou a portabilidade.

Conforme o caso

- componentes;
- hooks;
- rotas;
- tabelas;
- migrations;
- funções;
- policies;
- autenticação;
- roles/capabilities;
- configurações;
- variáveis;
- documentação;
- testes;
- dados existentes;
- comportamento público/Admin.

7. Ferramentas

Podem ser usadas:

- GitHub;
- Codex;
- ambiente local;
- Lovable;
- Supabase/Lovable Cloud;
- navegador;
- DevTools;
- Claude quando houver impacto visual.

Regra

A ferramenta é escolhida pela camada do problema.

A mecânica Git pertence ao Manual 5.

WSS · MÉTODO DE DESENVOLVIMENTO DE SITES

PARTE I — DEFINIR O RECORTE

8. Passo 1 — Nomear a capacidade-alvo

Antes de comparar repositórios inteiros, formular o objetivo em termos de capacidade.

Exemplos:

- melhorar KPIs de Leads;
- trazer ações em lote;
- portar uma melhoria do Blog Admin;
- incorporar comportamento atual de Chat;
- adicionar configuração presente na Base.

Evitar formulação ampla

“Atualizar o site antigo para ficar igual à Base.”

Essa formulação é grande demais e contradiz o padrão AS-IS.

9. Passo 2 — Descrever o resultado funcional esperado

Responder:

- o que a capacidade deve permitir;
- quem usa;
- onde aparece;
- qual problema resolve;
- qual comportamento atual precisa permanecer.

Objetivo

Comparar **função**, não apenas arquivos.

10. Passo 3 — Delimitar o que está fora do recorte

Registrar explicitamente o que não deve ser alterado.

Exemplos:

- institucional;
- módulos não relacionados;
- dados existentes;
- autenticação que já funciona;

- fluxos locais superiores;
 - arquitetura global.
-

WSS · MÉTODO DE DESENVOLVIMENTO DE SITES

PARTE II — INVENTARIAR O DESTINO

11. Passo 4 — Ler o projeto de destino antes de editar

Entender:

- stack;
- estrutura;
- Admin;
- backend;
- banco;
- autenticação;
- capacidade existente;
- particularidades locais.

Regra

Não assumir que “legado” significa “inferior”.

12. Passo 5 — Identificar a geração técnica

O projeto pode pertencer a geração diferente da Base atual.

Caso real Guilherme Maia:

- React;
- Vite;
- React Router;
- Supabase;
- Admin próprio já desenvolvido.

Projetos atuais podem utilizar stack diferente.

Consequência

Não copiar solução tecnicamente incompatível sem adaptação.

13. Passo 6 — Inventariar o que o destino já possui

Para a capacidade-alvo, listar:

- telas;
- fluxos;
- dados;

- operações;
- filtros;
- ações;
- integrações;
- permissões;
- particularidades.

Objetivo

Evitar refazer algo que já funciona.

WSS · MÉTODO DE DESENVOLVIMENTO DE SITES

PARTE III — INVENTARIAR A ORIGEM

14. Passo 7 — Identificar a implementação correspondente na Base

Na Base atual, localizar:

- superfície;
- fluxo;
- componentes;
- backend;
- dados;
- permissões;
- dependências.

15. Passo 8 — Separar capacidade de implementação

Pergunta central:

o que é comportamento funcional reutilizável e o que é apenas detalhe da arquitetura atual da Base?

Exemplo conceitual

Uma capacidade pode existir em ambas as stacks, mas exigir implementações diferentes.

WSS · MÉTODO DE DESENVOLVIMENTO DE SITES

PARTE IV — MATRIZ ORIGEM × DESTINO

16. Passo 9 — Classificar equivalências

Para cada parte da capacidade, marcar como:

- equivalente;
 - parcialmente equivalente;
 - ausente no destino;
 - existente e melhor no destino;
 - incompatível;
 - nova funcionalidade.
-

17. Equivalente

Se origem e destino já resolvem a mesma necessidade:

não refazer por padrão.

18. Parcialmente equivalente

Identificar exatamente o delta.

Evitar trocar o módulo inteiro.

19. Ausente no destino

Pode ser candidato à portabilidade.

Ainda precisa passar por:

- dependências;
 - compatibilidade;
 - escopo;
 - risco.
-

20. Melhor no destino

Preservar a solução local, salvo decisão humana contrária.

Princípio

evolução não significa regressão para igualar a Base.

21. Incompatível

Se a implementação da Base depende de arquitetura inexistente no destino:

- não copiar diretamente;
- identificar o comportamento desejado;
- adaptar à arquitetura local;
- avaliar custo/risco.

22. Nova funcionalidade

Se algo existe na Base ou surge durante comparação, mas não corresponde ao objetivo aprovado:

não classificar como paridade.

Exige decisão de escopo.

WSS · MÉTODO DE DESENVOLVIMENTO DE SITES

PARTE V — SEPARAR PARIDADE DE FEATURE NOVA

23. Passo 10 — Perguntar “isso é delta ou produto novo?”

Para cada melhoria:

- já existe na Base?
- resolve capacidade já prevista?
- é necessária para equivalência?
- adiciona jornada ou regra nova?

24. Caso real — Leads do Guilherme Maia

O destino já possuía:

- CRUD;
- busca/filtros;
- score;
- urgência;
- ficha/drawer;
- timeline;
- exportação.

Esses itens não deveriam ser refeitos.

Deltas identificados incluíam, por exemplo:

- KPIs;
- gráficos;
- ações em lote.

25. Caso real — “Jornada do paciente”

Foi explicitamente separada da portabilidade porque representava:

funcionalidade nova, não paridade.

Regra

Não aproveitar uma portabilidade para ampliar silenciosamente o produto.

WSS · MÉTODO DE DESENVOLVIMENTO DE SITES

PARTE VI — MAPEAR DEPENDÊNCIAS

26. Passo 11 — Mapear dependências da capacidade

Antes de portar UI ou comportamento, investigar se depende de:

- tabela;
 - coluna;
 - função;
 - RPC;
 - policy;
 - role;
 - capability;
 - endpoint;
 - hook;
 - componente;
 - configuração;
 - variável;
 - package;
 - migration.
-

27. Passo 12 — Classificar dependências

Separar:

Já existe no destino

Reutilizar quando compatível.

Existe parcialmente

Adaptar.

Não existe

Avaliar necessidade de portabilidade.

É específica da Base

Não transportar automaticamente.

28. Regra AS-IS

Dependências estruturais necessárias devem ser tratadas antes da capacidade que depende delas.

WSS · MÉTODO DE DESENVOLVIMENTO DE SITES

PARTE VII — BANCO, MIGRATIONS E DADOS

29. Passo 13 — Verificar impacto em banco/dados

Se a capacidade exige mudança persistente, identificar:

- tabela envolvida;
 - estrutura atual;
 - dados existentes;
 - compatibilidade;
 - necessidade de migration.
-

30. AUSÊNCIA ATUAL DE PADRONIZAÇÃO — migrations

Não existe hoje um procedimento universal recorrente e formal para toda portabilidade contendo:

- modelo único de migration;
- ordem única;
- ferramenta única;
- checklist único.

Consequência

Não inventar sequência universal.

A migration deve ser tratada conforme:

- projeto;
 - stack;
 - banco;
 - risco;
 - dados existentes.
-

31. Dados existentes

A descoberta sustenta o princípio de **preservar o que o destino já possui**.

Por isso:

- não apagar dados por conveniência;
- não substituir estrutura útil apenas para igualar a Base;
- não presumir que migration destrutiva é aceitável.

DECISÃO HUMANA NECESSÁRIA

Quando houver risco real de perda/transformação de dados.

32. Passo 14 — Separar migration de alteração frontend-only

Comparações reais já distinguiram:

- fases apenas frontend;
- fases que exigiam migration;
- integrações que exigiriam desenho adicional.

Essa distinção deve ser feita antes de implementar.

WSS · MÉTODO DE DESENVOLVIMENTO DE SITES

PARTE VII — AUTENTICAÇÃO, ROLES E PERMISSÕES

33. Passo 15 — Verificar se a capacidade depende de autorização

Identificar:

- quem deve acessar;
 - quem não deve acessar;
 - role/capability;
 - regra server-side;
 - RLS quando aplicável.
-

34. Não portar apenas a UI de permissão

Se a capacidade depende de autorização:

- preservar regra efetiva;
- não considerar ocultação visual como autorização suficiente.

Ver

Manual 6 / validação de segurança.

35. Diferenças entre gerações

Destino e Base podem ter modelos de permissão diferentes.

Não substituir a estrutura local sem análise.

Adaptar o requisito funcional ao modelo compatível.

WSS · MÉTODO DE DESENVOLVIMENTO DE SITES

PARTE IX — DEFINIR O DELTA APROVADO

36. Passo 16 — Consolidar o delta

Antes de editar, produzir uma lista enxuta:

Preservar

O que o destino já faz.

Portar

O que falta e está aprovado.

Adaptar

O que precisa mudar para ser compatível.

Excluir do recorte

Feature nova ou item sem ganho.

Decidir

Questões que dependem de humano.

37. Stop intermediário

Se o delta ainda não consegue ser descrito claramente:

não iniciar portabilidade.

Voltar à comparação.

WSS · MÉTODO DE DESENVOLVIMENTO DE SITES

PARTE X — IMPLEMENTAR DEPENDÊNCIAS

38. Passo 17 — Portar/adaptar dependências necessárias

Tratar primeiro o suporte requerido.

Pode incluir, conforme caso:

- banco;
 - funções;
 - permissões;
 - componentes base;
 - packages;
 - configuração.
-

39. Regra de contenção

Portar apenas dependência necessária.

Não importar subsistema inteiro apenas porque veio junto na Base.

40. Validar dependência antes da camada seguinte

Quando possível:

- confirmar que a dependência existe;
 - confirmar compatibilidade;
 - confirmar que não quebrou comportamento existente.
-

WSS · MÉTODO DE DESENVOLVIMENTO DE SITES

PARTE XI — IMPLEMENTAR A CAPACIDADE

41. Passo 18 — Adaptar a capacidade ao destino

Implementar o delta aprovado respeitando:

- stack do destino;
- componentes locais;
- autenticação local;
- dados existentes;
- identidade;
- fluxos existentes.

42. Não perseguir identidade de código

O objetivo é:

equivalência/ganho funcional compatível.

Não é:

arquivos idênticos à Base.

43. Preservar comportamento local melhor

Se o destino resolve uma parte melhor:

- manter;
- integrar o novo delta ao redor;
- evitar regressão funcional.

WSS · MÉTODO DE DESENVOLVIMENTO DE SITES

PARTE XII — ALTERAÇÕES FRONTEND-ONLY

44. Quando o delta é apenas frontend

Se a capacidade não exige mudança de dados/backend:

- limitar a alteração ao frontend;
 - preservar backend;
 - validar interação e regressão.
-

45. Não criar migration sem necessidade

A simples existência de migration na Base não significa que o destino precise da mesma migration.

Comparar estrutura real.

WSS · MÉTODO DE DESENVOLVIMENTO DE SITES

PARTE XIII — ALTERAÇÕES COM BACKEND/ BANCO

46. Quando o delta exige backend

Verificar:

- compatibilidade de função;
 - contrato;
 - payload;
 - autenticação;
 - retorno;
 - dependências.
-

47. Quando exige banco

Tratar projeto-especificamente:

- estrutura;
- dados;
- migration;
- autorização;
- regressão.

Ausência

Não existe receita única formal do AS-IS.

WSS · MÉTODO DE DESENVOLVIMENTO DE SITES

PARTE XIV — ESTABILIZAÇÃO

48. Passo 19 — Executar a capacidade no destino

Depois da implementação:

- abrir;
 - navegar;
 - executar;
 - observar;
 - corrigir falhas locais.
-

49. Passo 20 — Corrigir incompatibilidades

Problemas podem surgir por:

- stack diferente;
- dependência diferente;
- dado diferente;
- autenticação;
- componente local;
- expectativa herdada da Base.

Corrigir preservando o destino.

50. Não “resolver” regressão apagando comportamento local

Se o conflito decorre de uma particularidade importante do projeto:

reavaliar a adaptação.

WSS · MÉTODO DE DESENVOLVIMENTO DE SITES

PARTE XV — VALIDAÇÃO FUNCIONAL

51. Passo 21 — Testar a capacidade portada

Executar o fluxo principal da capacidade.

Exemplo genérico:

```
abrir módulo  
→ executar ação  
→ persistir/consultar quando aplicável  
→ confirmar resultado
```

52. Passo 22 — Testar o que foi preservado

Revalidar as funções que já existiam no destino e não deveriam ter sido alteradas.

53. Passo 23 — Testar dependências

Se houve alteração em:

- banco;
- autenticação;
- API;
- componente compartilhado;

revalidar as áreas dependentes.

WSS · MÉTODO DE DESENVOLVIMENTO DE SITES

PARTE XVI — REGRESSÃO

54. Regra AS-IS

Depois da portabilidade, o destino passa por:

- estabilização;
- correções;
- QA.

Não basta “a nova tela abriu”.

55. Passo 24 — Comparar antes x depois

Perguntar:

- o delta apareceu?
- o que já funcionava continua funcionando?
- dados foram preservados?
- permissões permanecem corretas?
- particularidades locais foram mantidas?

56. QA completo

O QA geral pertence ao SOP-008.

Este SOP exige a validação dirigida da:

- capacidade;
- dependências;
- áreas diretamente afetadas.

56.1 Retorno controlado de melhoria/correção à Base

Se a portabilidade ou a execução real no projeto revelar uma melhoria/correção potencialmente reutilizável, avaliar o fluxo inverso antes de encerrar a rodada:

```
melhoria/correção validada no projeto
→ verificar se é reutilizável
→ remover dados e regras específicas do cliente
→ comparar com a Base vigente
→ avaliar risco/regressão
→ incorporar à Base somente quando aprovado
→ validar a Base
```

Esse retorno é controlado e humano. Não constitui sincronização automática Base ↔ projetos e não autoriza incorporar à Base toda particularidade do cliente.

WSS · MÉTODO DE DESENVOLVIMENTO DE SITES

PARTE XVII — ROLLBACK E CONTINGÊNCIA

57. AUSÊNCIA ATUAL DE PADRONIZAÇÃO — rollback específico de portabilidade

Não existe hoje um ritual universal formal para rollback de toda evolução Base → legado.

58. Recursos observados no método geral

Conforme o projeto, recuperação pode usar:

- histórico Git;
- commit anterior;
- histórico/plataforma quando disponível;
- restauração orientada da versão funcional.

Limite

Este SOP não cria uma política nova de rollback.

59. Quando parar

Se a portabilidade:

- ameaça dados;
- quebra funções importantes;
- exige mudança arquitetural ampla não aprovada;
- virou feature nova;
- ultrapassou o delta definido;

parar e retornar à decisão humana.

WSS · MÉTODO DE DESENVOLVIMENTO DE SITES

PARTE XVIII — PROJETOS COM ARQUITETURA ANTIGA

60. Legado não precisa virar projeto atual inteiro

O fato de a Base atual usar geração tecnológica diferente não implica:

- refatorar tudo;
- migrar framework;
- converter toda renderização;
- substituir roteamento;
- reconstruir Admin.

61. Norma técnica vigente

Pode existir norma atual aplicável a novos projetos ou a determinadas superfícies.

Regra

Não usar este SOP para declarar automaticamente que todo legado precisa ser convertido integralmente à arquitetura normativa atual.

Quando norma vigente exigir intervenção no legado:

tratar como decisão/projeto próprio, com escopo explícito.

WSS · MÉTODO DE DESENVOLVIMENTO DE SITES

PARTE XIX — CASO GUILHERME MAIA

62. Papel do caso

`maia-uro-shine` é a principal evidência real de projeto antigo usado para comparação com a Base.

63. O que o caso ensina

O projeto já possuía Admin desenvolvido.

As comparações não partiram de:

“substituir pelo Admin novo”.

Partiram de:

“o que falta aqui que vale portar?”

64. Leads como exemplo

Preservar:

- CRUD;
- busca/filtros;
- score;
- urgência;
- ficha;
- timeline;
- exportação.

Avaliar como delta:

- KPIs;
- gráficos;
- ações em lote.

Excluir do recorte quando não aprovado:

- nova jornada funcional.
-

65. Blog como exemplo

Comparações reais distinguiram:

- capacidades existentes;
- lacunas;
- frontend-only;
- migration;
- integrações que exigiriam desenho adicional.

Lição

O delta pode ter naturezas técnicas diferentes dentro do mesmo módulo.

66. O que NÃO generalizar do caso Maia

Não generalizar:

- stack como definição de todo legado;
- módulos específicos como obrigatórios;
- deltas de Leads como padrão para todo projeto;
- migrations específicas;
- estrutura concreta do Admin.

Generalizar apenas o princípio de portabilidade incremental/adaptativa.

WSS · MÉTODO DE DESENVOLVIMENTO DE SITES

PARTE XX — STOP CONDITION

67. Condição de parada

O SOP está concluído quando:

o delta funcional aprovado foi portado/adaptado ao projeto de destino, as dependências necessárias foram tratadas, as capacidades locais que deveriam ser preservadas continuam funcionando e a alteração passou por validação dirigida e regressão proporcional nas áreas afetadas.

68. Checklist mínimo

Antes de encerrar:

- ☐ capacidade-alvo definida;
- ☐ destino inventariado;
- ☐ Base inventariada no recorte;
- ☐ equivalências identificadas;
- ☐ delta real identificado;
- ☐ feature nova separada;
- ☐ dependências mapeadas;
- ☐ impacto em banco/dados identificado;
- ☐ impacto em autenticação/permisões identificado;
- ☐ itens a preservar registrados;
- ☐ delta aprovado implementado;
- ☐ dependências tratadas;
- ☐ capacidade testada;
- ☐ comportamento local preservado testado;
- ☐ regressão dirigida executada;
- ☐ pendências registradas.

69. O que NÃO é necessário para encerrar

Não exigir:

- igualdade integral com a Base;
- mesma stack;
- mesma arquitetura;
- substituição de todo Admin;
- migração completa do projeto;

- Design System idêntico;
 - eliminação de toda divergência histórica;
 - modernização tecnológica geral.
-

WSS · MÉTODO DE DESENVOLVIMENTO DE SITES

PARTE XXI — CONTINGÊNCIAS

70. A Base tem algo que o destino já faz melhor

Preservar o destino.

Portar somente o ganho real.

71. O destino já tem a capacidade

Não refazer.

Verificar se existe delta útil.

72. A capacidade depende de algo ausente

Portar/adaptar a dependência necessária antes.

Não implementar só a camada superficial.

73. A dependência exigiria mudança ampla

Parar e avaliar escopo/risco.

74. A comparação revela feature nova

Não implementar automaticamente.

Retornar à gestão.

75. A migration ameaça dados existentes

Não prosseguir por inferência.

DECISÃO HUMANA NECESSÁRIA.

76. A nova capacidade quebra comportamento local

Reverter/corrigir o delta.

Não considerar a portabilidade concluída.

77. A UI funciona, mas backend não

Capacidade incompleta.

Seguir debugging por camada.

78. A portabilidade exige trocar framework

Isso deixou de ser uma portabilidade incremental simples.

Retornar à decisão de escopo.

WSS · MÉTODO DE DESENVOLVIMENTO DE SITES

PARTE XXII — AUSÊNCIAS ATUAIS DE PADRONIZAÇÃO

79. Não inventar

O AS-IS não possui:

- sincronização automática contínua Base ↔ projetos/sites;
 - substituição integral obrigatória do Admin;
 - algoritmo automático de delta;
 - matriz universal de compatibilidade;
 - regra única de migrations;
 - procedimento único de preservação de dados;
 - ritual único de rollback;
 - obrigação de atualizar stack do legado;
 - regra universal de branch para portabilidade;
 - threshold numérico de risco;
 - obrigação de portar toda melhoria nova da Base.
-

WSS · MÉTODO DE DESENVOLVIMENTO DE SITES

PARTE XXII — NORMA TÉCNICA VIGENTE

80. Normas continuam aplicáveis em camada própria

Quando uma alteração toca área sujeita a norma vigente:

- segurança;
- SEO;
- arquitetura pública;
- privacidade;
- autenticação;

a norma deve ser considerada.

Para SEO/AEO/GEO, a autoridade vigente é:

- WSS_Padrao_Tecnico_Operacional_SEO_AEO_GEO_v0.2_FINAL .

Princípio de não regressão

- divergência do legado não reduz a norma vigente;
- retrofit integral do projeto legado não é automático;
- risco, impacto, custo, contrato, prioridade e ciclo podem orientar a estratégia e o momento de remediação.

Isso não altera o princípio central deste SOP: a portabilidade permanece incremental e adaptativa.

Limite

Norma atual não deve ser usada para reescrever retroativamente o AS-IS do projeto legado sem decisão explícita.

WSS · MÉTODO DE DESENVOLVIMENTO DE SITES

PARTE XXIV — ANTI-PATTERNS

81. “Copiar o Admin novo por cima”

Contradiz o padrão observado.

82. Comparar arquivos sem comparar função

Pode produzir falsa divergência.

83. Refazer capacidade equivalente

Gera custo e risco sem ganho.

84. Portar UI sem dependência

Cria módulo visualmente presente, mas funcionalmente incompleto.

85. Chamar feature nova de “paridade”

Mascaramento de escopo.

86. Apagar particularidade local porque não existe na Base

A Base não é autoridade absoluta sobre o destino.

87. Assumir migration idêntica

Estrutura/dados podem ser diferentes.

88. Transformar atualização seletiva em migração de stack

Escopo diferente.

WSS · MÉTODO DE DESENVOLVIMENTO DE SITES

PARTE XXV — OUTPUT

89. Saída esperada

Ao final:

Comparação

- origem e destino compreendidos;
- equivalências conhecidas;
- delta delimitado.

Preservação

- capacidades locais úteis mantidas;
- dados relevantes preservados;
- particularidades necessárias preservadas.

Portabilidade

- dependências necessárias tratadas;
- capacidade adaptada ao destino;
- feature nova excluída ou decidida separadamente.

Validação

- capacidade funciona;
 - dependências funcionam;
 - regressão dirigida concluída;
 - pendências registradas.
-

WSS · MÉTODO DE DESENVOLVIMENTO DE SITES

PARTE XXVI — PRÓXIMO PASSO

90. Encaminhamento

Depois do SOP-007:

- executar **SOP-008 — Executar QA Visual, Funcional e Responsivo** quando a alteração integrar uma rodada maior;
 - usar **SOP-009 — Diagnosticar e Corrigir Bug por Camada** se houver falha durante estabilização.
-

91. Referências

Manual proprietário

- `WSS_Manual_04_Projeto_Base_Admin_Funcionalidades_v1.1_FINAL.md`

Manuais relacionados

- `WSS_Manual_05_Development_Codex_Lovable_GitHub_v1.1_FINAL.md`;
- `WSS_Manual_06_Testes_QA_Debugging_Seguranca_SEO_v1.1_FINAL.md`;
- `WSS_Manual_08_Boas_Praticas_AntiPatterns_Depreciadas_Casos_Reais_v1.1_FINAL.md`.

Baseline

- `WSS_Arqueologia_Tecnica_ASIS_v0.3.md`;
- `WSS_Matriz_Cobertura_ASIS_P0_P1_P2_v0.3_FINAL.md`;
- `WSS_Handoff_Consolidacao_Manuais_SOPs_Skills_v0.2.md`.

Evidência técnica complementar

- `WSS_VAL_004_Troubleshooting_Estudos_de_Caso_v0.1.md`.

Caso real principal

- Dr. Guilherme Maia — `maia-uro-shine`.
-

Status: FINAL — APROVADO