LLIÈGE

Orientações para Desenvolvimento e Deploy

Vision e Sist. Social

FEVEREIRO/2025

Versão 1.3

LLIÈGE

Sumário

Histórico de Revisão

Autor	Descrição	Data	Versão
Ezequiel Engunga	Criação Deploy Vision	01/03/2024	1.0
Emerson Soares da Silva	Inclusão de nova ação na abertura do chamado para deploy	05/02/2025	1.2
Emerson Soares da Silva	Nova ação para área de desenvolvimento	05/02/2025	1.2
Emerson Soares da Silva	Ajuste de Layout do documento, inclusão de sumario e histórico de revisão	05/02/2025	1.2
Emerson Soares da Silva	Inclusão de informações de Bibliotecas	05/02/2025	1.2
Emerson Soares da Silva	Período para abertura de Chamados de Deploy	05/02/2025	1.2
Emerson Soares da Silva	Inclusão gitflow Sist. Social	05/02/2025	1.2
Emerson Soares da Silva	Inclusão de informações fluxo git vision e Responsabilidades	09/02/2025	1.3
Emerson Soares da Silva	Inclusão de informação de instalação e atualização de itens pós deploy	14/02/2025	1.3

LLIÈGE

Introdução

Este documento visa auxiliar a equipe de desenvolvimento na realização de atualizações das aplicações **Vision e Sist. Social,** garantindo rastreabilidade, qualidade, e um fluxo padronizado.

1. Abertura do Chamado para Deploy

O chamado para deploy deve ser aberto através do módulo **ITSM** no Vision Lliege e deve conter obrigatoriamente os seguintes itens:

- **Categoria e Tipo:** Devem corresponder à área responsável pela aplicação do deploy, que é a **Infraestrutura**.

- **Descrição:**

  Clientes: Indicar quais clientes receberão a atualização.

  Aplicação: Especificar qual aplicação será atualizada.

  Data e Hora: Informar o dia e horário programados para o deploy.

- **Gestão Ágil:**

  · Na aba de gestão ágil, as tarefas a serem executadas durante o deploy devem estar vinculadas para garantir o registro e a validação das entregas.

- **Deploy`s do Vision :**

  · Listar os módulos envolvidos e as ações a serem realizadas, especificando caso seja uma **instalação** ou **atualização**.
  · Deve-se anexar ao chamado um arquivo .txt por cliente.
  · O arquivo .txt deve ser nomeado com o nome do cliente.
  · No arquivo .txt deve ser informado o nome técnico do módulo e se o módulo deve ser *instalado (i) ou atualizado (u),* segue exemplo:

  software_project_extends i
  lliege_extends_user_dashboard u

- **Deploy`s do Sist. Social :**

  · Listar clientes no chamado

· Caso tenha alterações em banco enviar a query em arquivo .txt

- **Importante para qualquer aplicação:**
  - **Banco de Dados**: Caso haja necessidade de atualização no banco de dados, o script correspondente deve ser anexado e informado no chamado.
  - **Bibliotecas**: Caso tenha Bibliotecas a serem instaladas, deve ser aberto chamado para infraestrutura no momento da aprovação da atividade por produtos em homologação do cliente, para que a infraestrutura inclua tal biblioteca no devido ambiente.

2. Prazo para abertura de chamado

Para fins de controle, o chamado deve ser aberto até as 12:00 do dia anterior ao Deploy.

3. Responsabilidades

Matriz de Responsabilidades R: responsável por realizar a tarefa (entregável); A: indivíduo encarregado da aprovação da atividade, que responderá pela demanda. Também chamado de autoridade; C: pessoa que é consultada sobre o processo; I: aquele que deve ser informado sobre a tarefa, por meio de avisos sobre a conclusão de um fluxo de trabalho.		Area/Departamento				
	Tarefas	Infraestrutura	Desenvolvimento	Produtos	Projetos/Operacoes	Cliente interno ou externo
1	Cria Branch de trabalho		R			
2	Disponibiliza em ambiente de teste Lliege		R	I		
3	Atualiza a branch no GIT com as últimas alterações.		R			
4	Publica em Homologação Cliente		R	I	I	I
5	Prepara Pacote para deploy		R	C	C	
6	Abre o chamado no módulo ITSM com todas as informações detalhadas.	I	R	I	I	
7	Atualiza ou Instala itens que foram feitos deploy	I	R	I		
8	Valida e acompanha deploy em produção	I	R	I	I	
9	Instala Bibliotecas em ambiente de teste e homologacao	I	R			
10	Atualiza banco de dados em ambiente de homologacao e testes	I	R	I		
11	Solicita ao Desenvolvimento criação, alteração ou correção		I	R		
12	Valida alteração, criação ou correção em ambiente de teste Lliege		I	R	I	
13	Valida alteração, criação ou correção em ambiente de homologação cliente		I	R	I	
14	Em caso de melhorias não solicitadas pelo cliente, deve-se informar projetos e operações sobre a melhoria, correção ou criação			R	I	
15	Fornece ao desenvolvimento relação de tarefas que serão implementadas durante o deploy.		I	R		
16	Após a execução do deploy, realiza a validação para garantir que tudo está funcionando corretamente.		I	R	I	
17	Agenda a janela de deploy em coordenação com o cliente, garantindo que o horário escolhido minimize o impacto nas operações.		I	I	R	
18	Caso tenha gerado demanda valida em homologação cliente		I	I	R	I
19	Caso tenha gerado a demanda valida em produção cliente		I	I	R	I
20	Informar cliente da "subida" de atualizações, correções ou criações			I	R	I
21	Instala Bibliotecas	R	I			
22	Roda script em banco de dados	R	I			
23	Executa o deploy conforme o cronograma estabelecido no chamado.	R	I	I		

LLIÈGE

4. Fluxo de Trabalho

1. **Desenvolvimento**:
    - Cria Branch de trabalho
    - Disponibiliza em ambiente de teste Lliege
    - Atualiza a branch no GIT com as últimas alterações.
    - Publica em Homologação Cliente
    - Prepara Pacote para deploy
    - Abre o chamado no módulo ITSM com todas as informações detalhadas.
    - Instala ou atualiza módulo em ambiente de produção
    - Valida e acompanha deploy em produção
2. **Produtos**:
    - Solicita ao Desenvolvimento criação, alteração ou correção
    - Valida alteração, criação ou correção em ambiente de teste Lliege
    - Valida alteração, criação ou correção em ambiente de homologação cliente
    - Em caso de melhorias não solicitadas pelo cliente, deve-se informar projetos e operações sobre a melhoria, correção ou criação
    - Fornece ao desenvolvimento relação de tarefas que serão implementadas durante o deploy.
    - Após a execução do deploy, realiza a validação para garantir que tudo está funcionando corretamente.
3. **Projetos e Operações**:
    - Agenda a janela de deploy em coordenação com o cliente, garantindo que o horário escolhido minimize o impacto nas operações.
    - Caso tenha gerado demanda válida em homologação cliente
    - Caso tenha gerado a demanda válida em produção cliente
    - Informar cliente da "subida" de atualizações, correções ou criações

4. **Infraestrutura**:
    - Instala Bibliotecas
    - Roda script em banco de dados
    - Executa o deploy conforme o cronograma estabelecido no chamado.

LLIÈGE

5. Anexos e Documentação

- **Scripts de Banco de Dados**: Devem ser anexados ao chamado caso sejam necessárias atualizações no banco de dados.
- **Documentação de Módulos**: Incluir qualquer documentação relevante sobre os módulos que serão instalados ou atualizados.
- **Relatórios de Validação**: Após a validação do deploy, os relatórios devem ser anexados na atividade para registro.

LLIÈGE

6. Vision - Orientações para uso do GIT e Fluxo de desenvolvimento

1. Branches Principais:
   - **Produção (master-cliente):** Branch estável, onde o código é sempre funcional e pronto para ser liberado aos usuários finais.

   - **Homologação (homolog_lliege>**· validacao_cliente**):** Branch onde as mudanças são testadas antes de serem promovidas para produção. Essa branch reflete o ambiente onde o QA e os testes de aceitação são realizados.

   - **Desenvolvimento (T1,T2,T3 ....):** Branch onde as features aprovadas são integradas e testadas em conjunto antes de serem movidas para a homologação.

2. Branch de Features:
   - **Feature Branches (FBXXXX-NomeDaTarefa):** Criada a partir da branch de homologação. Essas branches são usadas para desenvolver novas funcionalidades ou corrigir bugs específicos. Cada feature deve ser isolada em sua própria branch.

3. Fluxo de Trabalho:
   - **Criação de Feature Branch:**Para iniciar uma nova funcionalidade ou correção, cria-se uma branch a partir de HOMOLOGAÇÃO.

   - **Nome da branch**: FBXXXX(código da tarefa)-NomeDaTarefa.

   - **Desenvolvimento da Feature**: O desenvolvimento ocorre na branch de feature criada. Testes unitários e revisões de código são realizados localmente ou em um ambiente isolado.

   - **Integração em Desenvolvimento:**Uma vez que a feature está pronta, é feito um pull request (PR) para integrar a branch de feature na branch develop. A integração em develop é revisada e testada. Se a feature passa nos testes e revisões, o PR é aceito e a feature é mergeada em develop.

   - **Teste em Homologação:** Após a aprovação em develop, a Feature é mergeada na branch de Homologação. A branch Homologação é então testada em um ambiente de homologação para verificar se todas as funcionalidades estão funcionando conforme o esperado e sem impactar outras partes do sistema. (Por Operações).

- **Deploy em Produção:** Quando todas as features e correções na branch Homologação foram aprovadas, é realizado um merge de Homologação para Produção, que reflete a versão final e estável do sistema.

4. Regras Gerais-
   - **Commit Messages:** As mensagens de commit devem ser claras e descritivas.

   - **Revisão de Código:** Nenhuma mudança deve ser mergeada sem uma revisão de código.

   - **Padrão para nome de técnico dos módulos**:
     i. Módulos novos (módulos que não são extensões): lliege_<cliente se externo>_<nomemodulo>
     ii. Módulos de extensão: lliege_extends_<cliente><*nome modulo estendido*>_<nome modulo>.

   - **Descrição Alterações:** Todas as alterações realizadas nos módulos devem ter suas informações preenchidas nos campos específicos de seus aplicativos, por exemplo:
     i. Aba informações:
        1. Categoria: Ferramentas Extras
        2. Nome técnico: lliege_activity_dashboard_mngmnt
        3. Licença: Affero GPL-3
        4. Versão mais recente: 17.0.1.0.0

     ii. **Aba Dados Técnicos:** Deve ser incluídas todas as dependências do módulo, para não termos que "*adivinhar*" quais módulos fazem parte de seu conjunto e evitarmos quebra.

     iii. **Aba Recursos Instalados:** Importante sempre incluirmos esta informação, pois ajudara a sabermos quais recursos fazem parte do módulo, exemplo:
        <u>Descritivo básico do módulo</u>
        <u>Menus criados</u>
        - Activity Management
        - Activity Management/Activity
        - Activity Management/Configuration
        - Activity Management/Configuration/Activity Tag
        - Activity Management/Configuration/Activity Type
        - Activity Management/Dashboard

        <u>Visualizações criadas</u>

* INHERIT
mail.activity.view.tree.inherit.activity.dashboard
.mngmnt
(tree)
activity.tag.view.form (form)
activity.tag.view.tree (tree)
mail.activity.view.form (form)

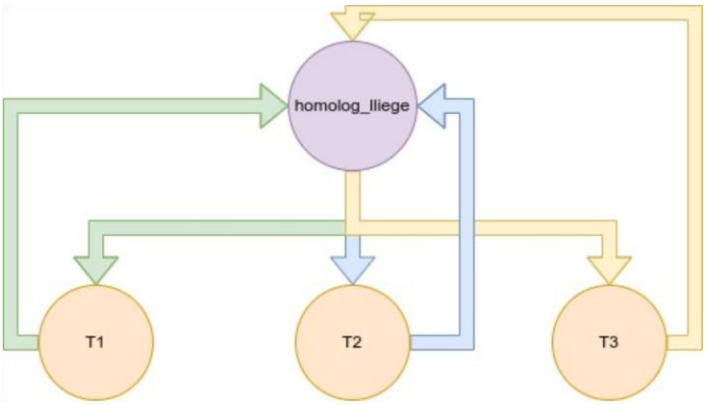
- **Testes:** Todas as funcionalidades devem ser acompanhadas de testes (unitários, de integração e de aceitação).

- **Qualidade:** Os testes de qualidade, devem ser feitos de maneira assertiva respeitando-se os critérios de aceite da solicitação, ao final da validação, deve ser informado ao desenvolvedor para preparar o pacote de deploy.

Importante: Evite a todo custo manter itens desenvolvidos somente em ambiente de homologação do cliente, desta forma evitamos que alterações quebrem em produção e que alterações se percam, assim evitamos o famoso - ***"mais isso não tinha sido feito? - Ou, porque o novo deploy retirou uma funcionalidade que já havia sido implementada?".***

5. Fluxograma
   - **homolog_lillege**: Este é o branch principal onde as alterações dos outros ramos (T1, T2, T3) são centralizadas.

   - **Ramos T1, T2 e T3**: T1, T2 e T3 são branches de trabalho que recebem e enviam alterações para "homolog_lille ge".



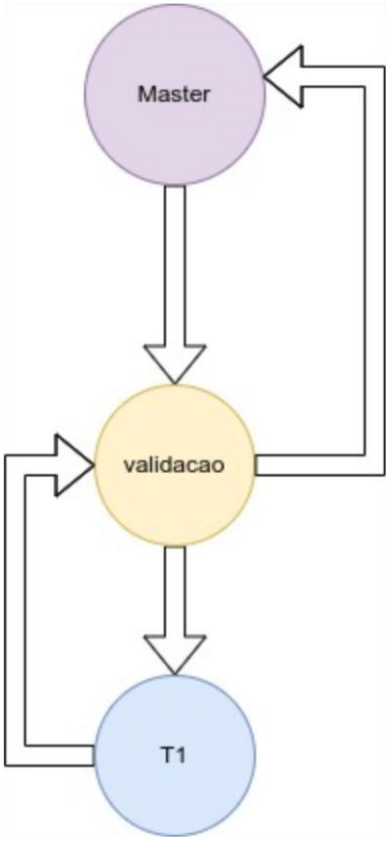
Após a aprovação da tarefa na branch de homologação Dev, ela deve ser transferida para a branch de homologação do cliente. Caso a tarefa tenha sido desenvolvida

diretamente na branch do cliente, as alterações devem ser enviadas para a homologação Dev para manter o versionamento do módulo sincronizado.

Para realizar o deploy, é necessário solicitar o merge da branch de homologação do cliente para a branch de produção do cliente.

Exemplo de fluxo:

· validacao_ibiuna > master · validacao_cotia > master_cotia · validacao_chavantes > master_chavantes



LLIÈGE

7. Sist. Social - Orientações para uso do GIT e fluxo de Desenvolvimento

1. Branches Principais:
   - **Produção (master-sistsocial):** Branch estável, onde o código é sempre funcional e pronto para ser liberado aos usuários finais.

   - **Homologação (hml):** Branch onde as mudanças são testadas antes de serem promovidas para produção. Essa branch reflete o ambiente onde o QA e os testes de aceitação são realizados.

   - **Desenvolvimento (dev):** Branch onde as features aprovadas são integradas e testadas em conjunto antes de serem movidas para a homologação.

2. Branch de Features:
   - **Feature Branches (FBXXXX-NomeDaTarefa):** Criada a partir da branch de homologação (hml). Essas branches são usadas para desenvolver novas funcionalidades ou corrigir bugs específicos. Cada feature deve ser isolada em sua própria branch.

3. Fluxo de Trabalho:
   - **Criação de Feature Branch:**Para iniciar uma nova funcionalidade ou correção, cria-se uma branch a partir de HOMOLOGAÇÃO.

   - **Nome da branch**: FBXXXX(código da tarefa)-NomeDaTarefa.

   - **Desenvolvimento da Feature**: O desenvolvimento ocorre na branch de feature criada. Testes unitários e revisões de código são realizados localmente ou em um ambiente isolado.

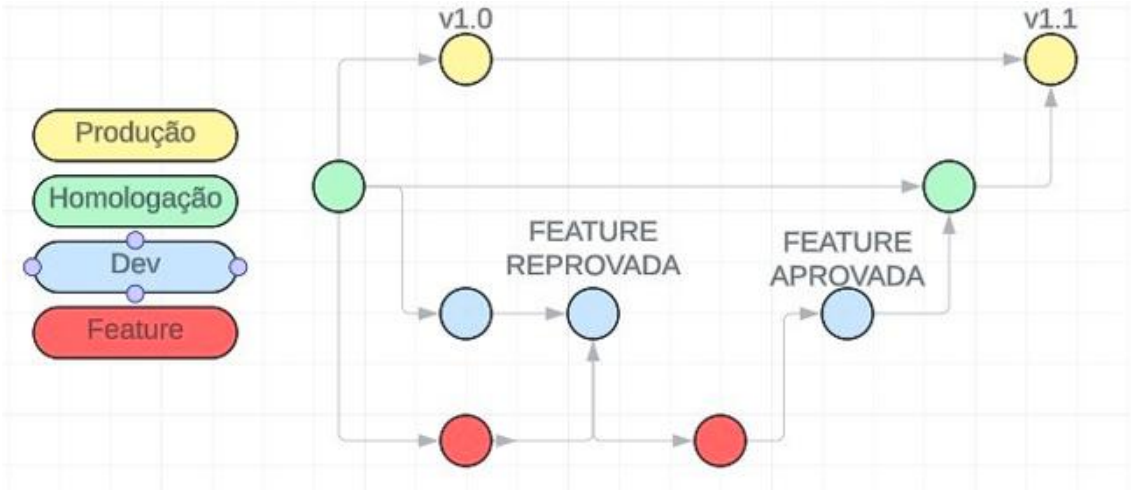
   - **Integração em Desenvolvimento:**Uma vez que a feature está pronta, é feito um pull request (PR) para integrar a branch de feature na branch develop.A integração em develop é revisada e testada. Se a feature passa nos testes e revisões, o PR é aceito e a feature é mergeada em develop.

   - **Teste em Homologação:**Após a aprovação em develop, a Feature é mergeada na branch de Homologação.A branch Homologação é então testada em um ambiente de homologação para verificar se todas as funcionalidades estão funcionando conforme o esperado e sem impactar outras partes do sistema. (Por Operações).

- **Deploy em Produção:** Quando todas as features e correções na branch Homologação foram aprovadas, é realizado um merge de Homologação para Produção, que reflete a versão final e estável do sistema.

4. Regras Gerais-
   - **Commit Messages:** As mensagens de commit devem ser claras e descritivas.
   - **Revisão de Código:** Nenhuma mudança é mergeada sem uma revisão de código.
   - **Testes:** Todas as funcionalidades devem ser acompanhadas de testes (unitários, de integração e de aceitação).

5. Fluxograma



6. Orientações Gerais e pontos de Atenção

Git e Fluxo de Atualizações

- Os branchs para desenvolvimento e correções devem ser criados a partir do branch hml.
  1. As correções e implementações devem ser adicionadas ao branch dev (Homologação Llège).
  2. Após a validação na homologação da Llège, as correções devem ser adicionadas ao branch hml (Homologação dos clientes).
  3. Após testes na homologação dos clientes, o branch hml deve ser mesclado ao branch master-sistsocial.

Antes de fazer o merge do hml com o master-sistsocial:

- Todos os chamados contidos no hml devem ser validados.
- Chamados não aprovados no hml devem ser devolvidos para produtos e, se necessário, ao desenvolvedor.

- Chamados não aprovados no hml e sem tempo hábil para correção antes do deploy em produção devem ser retirados do hml (revert). Esse caso deve ser evitado para que o branch de correção não perca a referência de sua origem (hml).

BUGs impeditivos após deploy em produção

- Bugs impeditivos, quando complexos ou de correção demorada, podem levar ao revert do merge do hml com master-sistsocial.
- **Por isso, é importante manter o link do merge salvo** para acessá-lo rapidamente, caso seja necessário desfazê-lo. O revert é feito diretamente na interface web do Git. O link do revert deve ser salvo para acessá-lo rapidamente após a correção do bug.

- Após a correção do bug, o revert deve ser desfeito (revert do revert). Um revert pode ser desfeito várias vezes. O fluxo da correção do bug deve seguir o processo normal:
  1. Criação do branch de correção (a partir do hml) ou uso de um branch existente.
  2. Aplicação da correção nos branchs dev e hml, respectivamente.
  3. Revert do revert anterior do hml com o master-sistsocial.

CUIDADOS NO MERGE

- Ao fazer o merge de branchs (com dev e hml), sempre verificar a quantidade de alterações. Uma quantidade excessiva pode indicar um erro na criação do branch de trabalho. Nesses casos, deve-se recriar o branch de trabalho, garantindo que ele seja originado do hml.
- Resolver conflitos pela interface online do Git (não recomendado) pode ocasionar alterações inesperadas no branch de trabalho.
- Se no momento do merge aparecer a mensagem "nenhuma alteração a ser mergeada", isso indica que o branch de trabalho não tem relação com o branch de destino. Nesse caso, o branch deve ser recriado a partir do hml.
- Merges não concluídos permanecem abertos e devem ser fechados antes de uma nova tentativa.

CUIDADOS NA CRIAÇÃO DOS BRANCHS

- Os branchs devem ser criados a partir do hml.
- **A interface do Git pode alterar a origem do branch para master-sistsocial após a primeira escolha.** Esse detalhe deve ser observado e corrigido antes de gerar o branch.

RESOLUÇÃO DE CONFLITOS

- Conflitos não devem ser resolvidos pela interface do Git. O método recomendado é:
  1. Na máquina do desenvolvedor, fazer checkout no branch onde ocorreu o conflito (dev ou hml).
  2. Atualizar o branch (git pull).
  3. Realizar o merge com o branch de correção (git merge <branch_da_correção>).
  4. Resolver os conflitos manualmente.
  5. Realizar o commit com a mensagem "Resolução de conflitos" (git commit -m 'Resolução de conflitos').
  6. Enviar as alterações (git push).
     O último comando irá submeter o merge, atualizando os branchs dev ou hml conforme necessário.

8. Considerações Finais

Seguir este processo garante que as atualizações nas aplicações sejam realizadas de maneira organizada, eficiente e com mínima interrupção para os clientes.
"A colaboração e a comunicação entre as diferentes áreas envolvidas são fundamentais para qualidade, rastreabilidade e o sucesso da publicação dos itens desenvolvidos."