Metodologia de Trabalho

Inicie uma nova tarefa

Em primeiro lugar, antes de colocar as mãos no código, devemos ter certeza de que a tarefa que nos foi atribuída e com a qual vamos trabalhar se encontra na coluna 'Desenvolvimento' em nosso ambiente de Trabalho em Equipe. Isso permite que toda a equipe saiba no que todos estão trabalhando no momento e ajuda a manter tudo organizado.

Feito isso, vamos ao nosso editor de código e abrimos um terminal para realizar todas as etapas para iniciar o fluxo de trabalho git e vtex.

1. Abrimos nossa pasta de projetos e nos certificamos de que estamos no ramo de develop. Em seguida, executamos o git pull ou git pull origin develop. Com isso trazemos todas as alterações que foram feitas no repositório por um membro da equipe. **Esta etapa é fundamental**, caso contrário começaríamos a trabalhar com uma versão desatualizada do projeto.

1.1. Caso tenhamos feito alterações, a próxima coisa que devemos fazer é executar o comando npm run sass para compilar as alterações nos arquivos scss. Caso contrário, não veríamos a atualização de estilos.

2. Assim que tivermos a certeza de que temos tudo atualizado e compilado, vamos iniciar um novo recurso para realizar nossa tarefa. Para fazer isso, executamos git flow feature start [nameOfTheFeature]. É importante que o nome do recurso seja consistente com o tipo de tarefa que iremos executar. Exemplo: git flow feature start settingStylesHome

3. Já dentro do nosso recurso, começamos a executar os comandos vtex para visualizar um ambiente de desenvolvimento e avançar com nossa correção. A primeira coisa que temos que fazer é entrar na conta do projeto: vtex login [conta]

4. Agora vamos criar um espaço de trabalho de desenvolvimento para que possamos trabalhar sem nos preocupar em modificar algo que possa afetar a loja em produção. Para isso, executamos o vtex use [workspacename]. O nome da área de trabalho deve estar em letras minúsculas e é recomendado que seja o mesmo nome que foi usado

para o recurso (ou algo muito semelhante) para ser consistente.
Exemplo: vtex use ajusteestiloshome
5. A partir deste ponto você pode começar a trabalhar nos arquivos necessários para realizar a correção / desenvolvimento ou o que quer que a tarefa em questão implique. Os dois últimos comandos que precisaremos para poder visualizar nossas mudanças e verificar os ajustes que estamos fazendo são:
5.1. Vtex link para renderizar nosso espaço de trabalho e poder visualizá-lo no navegador.
5,2 npm run sass para compilar as mudanças que fazemos nas folhas de estilo do sass.
5.3. É importante mencionar que ambos os comandos atuam como vigilantes e permanecem em execução até que um corte sua execução manualmente no terminal. A vantagem é que podemos fazer alterações nos arquivos, salvá-los e vê-los refletidos automaticamente no navegador, sem a necessidade de atualizar a página. Mas, infelizmente, também tem uma desvantagem: nem sempre funcionam como esperado. Geralmente acontece que quando temos os dois comandos em execução e fazemos alterações, estes não se refletem no navegador, por isso é fundamental que de vez em quando cortem a execução e a executem novamente, como se para fazer uma espécie de reset e ser capaz de ver realmente quais mudanças foram aplicadas na área de trabalho e quais não. **Acima de tudo, é fundamental fazê-lo antes de enviar a url do espaço de trabalho ao cliente para revisão**, caso

contrário corre-se o risco de enviar uma versão diferente daquela que estamos a ver.

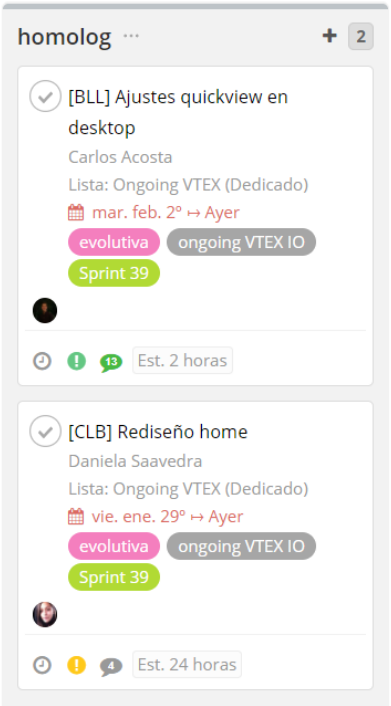
6. Assim que terminarmos de fazer os ajustes de nossa tarefa e estivermos prontos para enviá-la para revisão, primeiro devemos salvar nosso recurso no repositório. Para fazer isso, executamos `git add .` e `git commit -m "[breve descrição das alterações]"`. Finalmente, com `git push -u origin feature/ [featureName]`, deixamos nosso recurso carregado no repositório para que qualquer membro da equipe possa acessá-lo, se necessário. Seguindo o exemplo em que estávamos trabalhando, seria: `git push -u origin feature/ AdjustEstilosHome`

6.1. O envio para a origem é feito apenas uma vez. Então, se eles fizerem alterações no recurso novamente, eles as carregam diretamente com um `git push`

6,2 Os commits e o push de nosso recurso não precisam ser feitos uma vez no final de toda a tarefa. É recomendado que eles façam commits à medida que avançam com as correções e que também enviem para ter um backup na nuvem do que estiveram trabalhando. Você nunca sabe quando sua internet, energia ou pc podem ficar desligados, então ter o hábito de salvar seu trabalho remotamente regularmente é uma prática muito boa que beneficia toda a equipe. Em uma emergência, qualquer um pode pegar seu branch e continuar com seu trabalho, mas se eles não fizerem o upload para o repositório, eles terão que iniciar sua tarefa do zero.

7. Por último, mas não menos importante, após comprometer e enviar nossas alterações, notificaremos o cliente de que a correção está

pronta. Deixamos um comentário sobre a tarefa, passamos a url do nosso espaço de trabalho, carregamos o tempo que demorou para fazer todo esse processo e finalmente movemos a tarefa para a coluna Homologação.



Até aqui chega a primeira fase do ciclo de vida de uma tarefa. Agora temos que esperar que o cliente analise o ajuste e nos dê um retorno. Duas coisas podem acontecer: Se tudo estiver ok, iremos carregá-lo para produção. Se algo não deu certo ou eles perceberam que faltavam ajustes, continuamos com a correção. Para isso basta voltar do passo 5 em diante: Abrimos o projeto, entramos em nosso recurso, modificamos os arquivos, fazemos o link e rodamos o sass; Fazemos os commits e push do recurso, e quando terminamos notificamos o cliente novamente, recarregamos as horas e esperamos pela resposta novamente.

Subindo uma tarefa para produção

Quando o cliente confirmar que a tarefa está ok e que é hora de carregá-la para produção, realizaremos as seguintes etapas:

1. Abrimos o projeto e paramos no recurso em que realizamos o ajuste. A primeira coisa que faremos é verificar se há atualizações no repositório (como iniciar uma nova tarefa). Para fazer isso, executamos `git pull origin develop`. (Pode acontecer que, ao trazer alterações para nosso branch, tenhamos algum conflito git, por isso é importante estar atento nesta instância para resolvê-los e seguir em frente. Normalmente são apenas questões de espaço, basta aceitar as duas alterações (recebido e o seu) e está resolvido. Mas pode ser que um dos dois precise ser substituído, por isso é essencial que você esteja em comunicação com a equipe para avaliá-lo conforme apropriado.)

1.1. Se notarmos que o pull nos traz modificações, imediatamente depois mudamos para o branch `develop` e executamos `git pull` `y/o` `git pull origin develop`. Então rodamos um `npm run sass` para compilar o css novamente. (Isso é necessário se as alterações que recebemos incluem ajustes nas folhas de estilo. Nem sempre é o caso, mas é mais fácil e seguro executá-lo, desde que seja para verificar o que foi modificado)

1.2. Agora, voltamos ao nosso recurso e executamos o `npm run sass` mais uma vez.

1.3. (OPCIONAL) Por fim, vamos revisar nosso espaço de trabalho novamente e verificar se as alterações que trouxemos não afetam a correção que fizemos anteriormente. Para isso temos que fazer o `vtex login [cuenta]`, e `vtex use [workspace]` e finalmente `vtex link`
1.4. (OPCIONAL) Caso algo se estrague, obviamente devemos corrigir neste caso e deixar como mostramos ao cliente. Caso 1,5. (OPCIONAL) caso contrário, se tudo correr bem seguimos em frente. 2. Agora que verificamos que o repositório está atualizado no desenvolvimento, **voltamos a nossa feature**, vamos fechar a nossa feature. Para fazer isso, executamos o `git flow feature finish`. Se você não realizou as etapas anteriores, assumindo que temos pulls pendentes em `develop`, este comando retornará um erro porque a feature não pode ser fechado se o `develop` não estiver atualizado.
3. Assim que a feature for fechada, o `develop` será interrompido e executaremos um `npm run sass`. pela última vez. Isso é para compilar o css de nossa feature, caso contrário, faríamos upload do antigo css de `develop` para produção.
4. Agora fazemos um `git push` para enviar nossas alterações para o repositório. Até agora o fluxo do git vai, só temos que continuar com os comandos do vtex.
5. Se não o fizemos antes, a primeira coisa é estar logado na conta do projeto, então executamos o `vtex login [cuenta]` **E se já estávamos logados na conta, é importante verificar se estamos parados no ambiente** `master` e não em uma área de trabalho. Para isso podemos executar o `vtex whoami` que nos dirá em que conta e

em que ambiente estamos. Se estivéssemos em um espaço de trabalho, simplesmente executamos o `vtex use master`. (**Importante,** não deve ser confundido com o branch master do github.)

6. Agora que estamos logados e no ambiente master, vamos lançar uma nova versão de nossa loja. Para fazer isso, executamos o `vtex release patch stable`. Isso compilará todo o nosso código e publicará a nova versão e, no processo, nos pedirá duas confirmações que devemos aceitar.

7. Se tudo correr bem, veremos a confirmação da publicação da nova versão e o que falta fazer é `vtex install` para que as alterações que fizemos impactem a loja em produção.

7.1. Nesse momento, devemos revisar a loja em produção e nos certificar de que as alterações que carregamos estão sendo exibidas, bem como verificar se nossa correção não impactou negativamente qualquer outro componente ou seção da página.

7,2 Caso encontremos algum problema, podemos fazer um rollback e voltar para a versão anterior que a loja tinha. Para fazer isso, devemos executar um `vtex install [vendor].[app]@[version]`. Por se tratar da loja completa, nosso fornecedor será a conta do projeto e o aplicativo será o nome do aplicativo. Esses dados podem ser vistos fazendo uma lista `vtex list`

```
Installed Apps in merrellcl at workspace master
corebiz.nosto-integration     1.3.15
mercadopago.mercadopago-app   2.0.6
merrellcl.store-theme         1.6.58
```

1.1. Así como también en el `manifest.json`

```
"vendor": "merrellcl",
"name": "store-theme",
"version": "1.6.58",
```

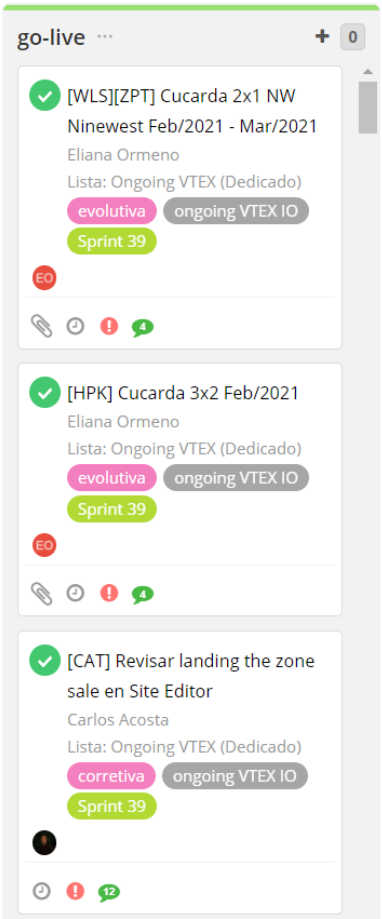
Como queremos instalar uma versão anterior à atual para reverter a mudança, logicamente devemos subtrair um número da versão quando executamos a instalação. Exemplo: `vtex install merrellcl.store-theme@1.6.57`

7.3. Depois de instalar a versão anterior, devemos iniciar todo o processo de uma tarefa novamente para fazer os ajustes necessários. Por isso **é muito importante** que antes de passarmos para a produção, tenhamos feito todos os testes e verificações necessários em nosso espaço de trabalho para evitar esse tipo de situação.

8. Assim que verificamos na produção que nossa mudança está correta e que toda a loja parece boa, ficamos com a etapa final de todo o processo, que é o `vtex deploy`. Isso informa ao vtex que a nova versão que instalamos foi testada e aprovada para permanecer como a versão final, caso contrário, se não executarmos este comando, após algumas horas o vtex retorna automaticamente para a última versão que foi implantada.

8,1 Deve-se notar que para fazer `vtex deploy`, pelo menos 7 minutos devem decorrer após `vtex install`. Caso contrário, obteremos um erro no terminal nos avisando para esperar.

9. Assim que implantamos, encerramos o ciclo de vida de nossa tarefa. Dizemos ao cliente que a correção está em produção, carregamos as horas correspondentes e passamos a tarefa para a coluna “Go Live”.


go-live
0
[WLS][ZPT] Cucarda 2x1 NW Ninewest Feb/2021 - Mar/2021
Eliana Ormeno
Lista: Ongoing VTEX (Dedicado)
evolutiva
ongoing VTEX IO
Sprint 39
EO
4
[HPK] Cucarda 3x2 Feb/2021
Eliana Ormeno
Lista: Ongoing VTEX (Dedicado)
evolutiva
ongoing VTEX IO
Sprint 39
EO
4
[CAT] Revisar landing the zone sale en Site Editor
Carlos Acosta
Lista: Ongoing VTEX (Dedicado)
corretiva
ongoing VTEX IO
Sprint 39
12

Isso completa nosso fluxo de trabalho para cada tarefa.

Listar etapa de comando para produção de espaços de trabalho dev

1 se não estivermos no branch, paramos em nosso branch (git checkout feature / mirama)

2 **git add.** (adicionamos as alterações para fazer o upload)

3 **git commit -m** "a mudança que fiz" (comentamos as mudanças feitas da forma mais específica e curta possível)

4 **git pull origin develop** (no branch em que estamos trabalhando para atualizar o branch com o que está no git)

5 **git push origin feature/mirama** (publicamos as alterações remotas no git)

6 **git checkout develop** y **git pull origin develop** (para atualizar, desenvolver para desenvolver a partir de git)

7 **git checkout feature/mirama** (voltamos ao nosso branch)

Finalização do recurso

8 **git flow feature finish** (fechamos o branch e ele automaticamente vai para o desenvolvimento)

9 no desenvolvimento npm, execute o sass (executamos o sass caso alguns estilos não sejam compilados e paremos o sass ctrl c)

10 **git push origin develop**

11 **vtex login cuenta**

12 **vtex whoami** (verificar que estamos en master)

13 **vtex release patch stable**

14 **vtex install**

15 verificamos se as mudanças estão corretas na loja em produção

16 esperamos 7 minutos e **vtex deploy**