Setup IDE

First Steps

Guia: https://forcedotcom.github.io/salesforcedx-vscode/articles/getting-started/install

Trailhead: https://trailhead.salesforce.com/content/learn/trails/sfdx_get_started

1. Ao instalar o Salesforce Extension Pack, instalar cada extensão individualmente, e finalmente o Pack e reiniciar o VSC.

Ligar à organização

1. Todos os comandos podem ser executados carregando nas teclas CTRL+SHIFT+P.
2. Criar um projecto com o comando: SFDX: Create Project ou SFDX: Create Project with Manifest.
3. Autorizar a organização destino com o comando: SFDX: Authorize an Org. Ao executar este comando irá ser aberto no browser o login salesforce. Este comando poderá ter de ser executado novamente ao reabrir o VSC.
4. Caso seja necessário puxar a informação da organização é necessário existir o manifesto.
    a. Tem que existir o ficheiro "/manifest/package.xml" com os metadados que irão ser trazidos. Caso este path ainda não exista terá de ser criado. Executando o segundo comando da linha 1, irá criar automaticamente.
5. Ao abrir o ficheiro "/manifest/package.xml" e clicando com o botão do lado direito, poderemos evocar o comando SFDX: Retrieve Source in Manifest from Org.
6. O comando SFDX: Deploy Source in Manifest to Org envia as pastas correspondentes para fazer push para a organização.
7. Para evocar todos os objetos da nossa organização deverá ser executado o comando: SFDX: Refresh SObject Definitions

Criar ficheiros

Existem vários comandos para criar os diferentes tipos de ficheiros que podem ser criados numa organização Salesforce:

- SFDX: Create Apex Trigger
- SFDX: Create Apex Class
- SFDX: Create Visualforce Page
- SFDX: Create Visualforce Component
- SFDX: Create Lightning App
- SFDX: Create Lightning Component
- SFDX: Create Lightning Event
- SFDX: Create Lightning Interface
- SFDX: Create Lightning Web Component

Alterações a ficheiros

Após ser feito o pull dos recursos da organização, estes podem ser editados no VSC.

1. Após abrir um ficheiro e clicando com o botão do lado direito do rato temos três opções:
    a. SFDX: Delete This from Project and Org: que irá apagar o recurso;
    b. SFDX: Deploy This Source to Org: faz o push para a organização;

c. SFDX: Retrive This Source from Org: faz o pull da organização.

Analisar Debug Logs

Para iniciar uma sessão de registo de logs executar o comando:

- SFDX: Turn On Apex Debug Log for Replay Debugger

Após ter sido ligado irão ser registados os debug logs do user que autorizou a organização. Estes logs poderão ser chamados através do comando:

- SFDX: Get Apex Debug Logs

Ao clicar com o botão do lado direito temos a opção:

- SFDX: Launch Apex Replay Debugger with Current File

Este comando irá iniciar o processo da extensão de debug do Salesforce Extension Pack

Executar classes de teste

Ao ser criada uma classe de teste esta irá aparecer numa Tab especifica do lado esquerdo do VSC com este icon:

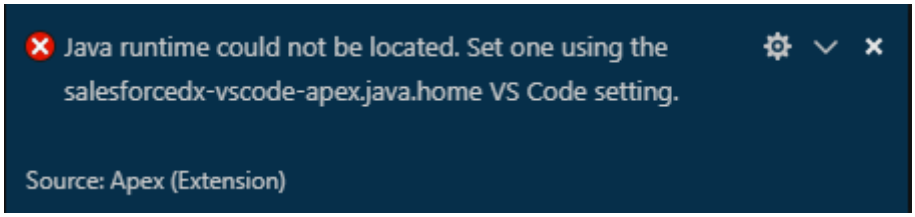
Aqui irão aparecer todas as classes de teste carregadas no nosso projecto.

Podem ser executadas todas as classe de teste ou apenas uma especifica carregando no respectivo botão de play:

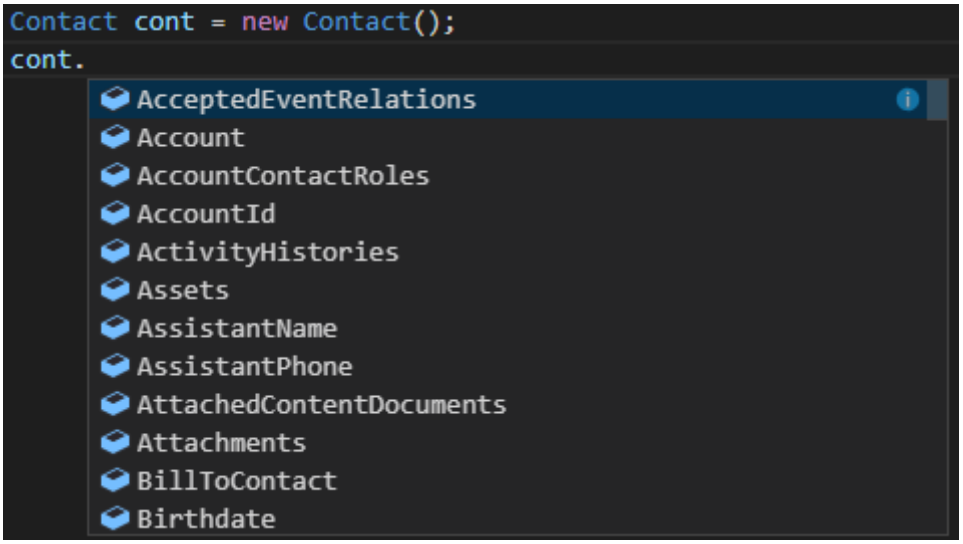
Após a executada a classe de testes, é devolvida uma resposta e impressa no output do VSC.

Plugin de auto-complete

1. Instalar extensão do VSC, “Salesforce Language Support”, e reiniciar o VSC.
    a. Caso apareça o seguinte erro:



    b. Ir ao ficheiro “/.vscode/settings.json” e adicionar a linha:
    c. "salesforcedx-vscode-apex.java.home": "C:/Program Files/Java/jdk1.8.0_201",
    d. Que aponta para a pasta onde foi instalado o Java Development Kit
2. Resultado final deverá ser assim:



Plugin de indentação

Guia: https://medium.com/@taxnexus/salesforce-apex-beautified-in-vs-code-with-uncrustify-7d0cbfc5ea1a

1. Fazer download do ficheiro presente em: Sourceforge e correr o exe.
2. Instalar extensão do VSC, “Uncrustify”, e reiniciar o VSC.
3. Após instalação executar o comando:
    a. >Uncrustify: Create default config file
4. Abrir o ficheiro criado, uncrustify.cfg criado na root.
    a. Em Indenting options:
        i. Alterar indent_class para true;
        ii. Alterar indent_col1_comment para true.
5. Podem gravar estas configurações de indentação num preset carregando no botão SAVE PRESET.
6. Após feitas estas alterações a macro SHIFT+ALT+F deverá fazer indentação correta do documento aberto. Ou clicando no botão do lado direito do rato e carregando na opção Format Document ou Format Selection.
7. Resultado final deverá ser:

```
1   public with sharing class c_ExampleClass {
2   public c_ExampleClass() {
3   if (true) {
4   //comment1
5   for (Integer i = 0; i < 1; i++) {
6   //comment2
7   }
8   }
9   }
10  public void method1(){
11  //comment3
12  }
13  }
```

8. Deverá passar a:

```apex
public with sharing class c_ExampleClass {
    public c_ExampleClass() {
        if (true) {
            //comment1
            for (Integer i = 0; i < 1; i++) {
                //comment2
            }
        }
    }
    public void method1(){
        //comment3
    }
}
```

Apex best practices

Recomendações gerais:

- Consultar limites de Salesforce: https://resources.docs.salesforce.com/206/latest/en-us/sfdc/pdf/salesforce_app_limits_cheatsheet.pdf
- Trigger framework guide: https://github.com/kevinohara80/sfdc-trigger-framework
- SOAP API (SOAP UI) : https://www.soapui.org/
- REST API (Postman) : https://www.getpostman.com/
- Nomenclatura:
    - Todos os ficheiros deverão começar com prefixo seguindo a seguinte tabela:

Tipo	Prefixo
Trigger	Trigger_
Trigger Handler	TriggerHandler_
Trigger Logic	TriggerLogic_
Controller class	Controller_
Visualforce page	Page_
Batchable class	Batch_
Schedulable class	Sch_
SOAP Webservice class	Soap_
REST Webservice class	Rest_
Imported WSDL	Wsdl_
Utilities class	Util_
Interface class	i_
Structure class	N/A
Test class	Test_
Lightning component	LC_
Lightning application	LA_
Lightning interface	LI_
Lightning web component	LWC_
Lightning event	LE_
Visualforce component	Component_

    - Nomes de classes deverão seguir a norma CamelCase;
    - Nomes de variáveis estáticas e/ou finais deverão ser escritas em UPPERCASE e as palavras separadas por underscores (snake_case);

    - Nomes de variáveis não estáticas deverão seguir a norma pascalCase;
    - Nomes de métodos deverão ser verbos e seguir a norma pascalCase;
    - Nomes de variáveis booleanas deverão começar por "is" + objectivo, e deverão ser true se o objetivo for positivo.
    - Schedulable/Batchable interfaces deverão estar sempre em classes separadas.
- Não deverão existir valores string gravados no código.
- Não poderão existir **Id's** gravados no código.
- Não poderá existir SOQL/SOSL/DML/@future dentro de loops.
- Todos os métodos para tratamento de informação deverão existir com um parâmetro de entrada em lista, estes métodos poderão posteriormente chamar um método unitário para tratar cada registo.
- Batchables, Triggers, Webservices e Controller classes são apenas interfaces que evocam classes auxiliares utilitárias para executar a sua lógica.
- Validar sempre as variáveis antes de fazer alguma operação com as mesmas, por exemplo:
    - Lista deverá sempre ter um if(lista.isEmpty());
    - Decimais e Strings que sejam usadas numa operação deverão sempre ter uma validação de que os seus valores não serão null
- Cada classe deverá ter a sua respetiva classe de testes que deverá:
    - Existir um testmethod para cada método/para cada use case da funcionalidade;
    - Deverão ser feitos asserts para validar os valores esperados;
    - Não poderá ter secções de try/catch;
    - Deverão ser usadas fábricas de registos para testar funcionalidades que deverão ser escaláveis, pelo menos 200 registos deverão ser criados (ajustável ao tamanho máximo do lote).
- Cada classe deverá conter comentários de:
    - Autor da classe;
    - Data de criação;
    - Breve descrição da classe
    - Histórico de modificações;

```
/*
   @Author: Developer1
   @CreatedDate: 01/01/2019
   @Description: Breve descrição da classe
   Update History:
   Developer1 - 01/01/2019 - Criação da classe
 */
```

- Cada método deverá conter comentários de:
    - Descrição do método;
    - Descrição dos parâmetros de entrada;
    - Descrição do return caso exista;

```
/*
   @Description: Breve descrição do método
   @param1: Descrição do parâmetro
   @param2: Descrição do parâmetro
   @return: Descrição da variável de retorno
 */
```

- Caso exista a necessidade de copiar código, um método deverá ser feito numa classe utilitária.
- Deverá existir apenas um trigger por objeto e deverá seguir a lógica do TriggerHandler
- Comentários deverão ser escritos em apenas uma língua.

Triggers

Triggers deverão ser construídos com a seguinte lógica:

- O trigger deverá apenas conter as operações em que terá de despoletar, e o run do seu respetivo TriggerHandler;
- Para cada uma das operações o nosso handler deverá conter um método para evocar a lógica necessária.
- Para cada acção necessária em cada operação do trigger, será evocado um método da classe TriggerLogic respetiva.

PAD (to be developed)

PAD (Parallel Apex Development) Framework é uma ferramenta de configuração de bypasses que poderão existir ao código. Para configurar é necessário criar uma tabela auxiliar (metadados ou custom settings) para guardar os registos de configuração.

Esta configuração poderá ser utilizada para:

- Ativar e inativar triggers
- Ativar e inativar chamadas a serviços
- Ativar e inativar determinadas funcionalidas
- Não executar bocados de código ainda em desenvolvimento para não interferir com testes de outros developers/clientes

Classes de teste

Todas as classes de teste deverão estar preparadas para serem executadas sem existir a necessidade de desligar ou modificar qualquer configuração de negócio da plataforma.

Classes de teste servem para mostrar programaticamente que o código faz o que é esperado, e devolve o necessário.

Boas práticas:

- Classes de teste deverão cobrir pelo menos 80% do código (recomendado pela Salesforce);

- Deverá existir uma classe de testes para cada classe implementada.
- Cada testmethod deverá corresponder a um método da classe a testar ou a cada use case de negócio.
- Anotação (@seeAllData=true) é estritamente proibido em qualquer desenvolvimento.
- Para executar código com determinado utilizador terá de ser utilizado o método System.runAs()
- Utilização de System.assert valida que o nosso código está a devolver valores esperados com os inputs fornecidos.
- Test.startTest() e Test.stopTest() podem ser utilizados para que quando necessários alguns ajustes aos dados dentro do cenário que estamos a testar, os limites sejam reiniciados antes de efectivamente iniciar o teste.
- Utilizar a anotação @isSetup para criar os dados para o cenário de teste.