Programação

Informações aprofundadas sobre o uso de JavaScript, C# e Query (SQL) no agilityflow.

- SQL Server (Query) - Dicas e Funções (Versões Mais Antigas do Agilityflow)
- Postgresql (Query) - Dicas e Funções (Versões Mais Recentes do Agilityflow)
- Programação em Html
- Programação em CSS
- Programação em Javascript
- Programação em C# - Na Regra de Negócio
- Programação em C# - Na API de integração (POST)
- Tabela de dados customizada através de programação: Query SQL, C#, HTML e CSS

SQL Server (Query) - Dicas e Funções (Versões Mais Antigas do Agilityflow)

> **IMPORTANTE**: Essa documentação é referente as versões antes de 2024 que são mais antigas do agilityflow e que utilizam o Banco de Dados **SQL SERVER.** O agilityflow nas últimas versões está utilizando **POSTGRESQL** .

No agilityflow você pode buscar os dados utilizando SQL, através de queries. As queries devem ser compatíveis com **SQL Server.** Além disso existem alguns padrões que você deve seguir para obter um melhor resultado no agilityflow. Abaixo estão alguns detalhes importantes.

Tratamento de Registros Deletados

> **IMPORTANTE:**
> É de extrema importância o tratamento de registros deletados, usando as regras abaixo.
>
> Já está no nosso Roadmap o tratamento automático dos deletados, enquanto isso, o tratamento se torna obrigatório pelo desenvolvedor.

Para manter a integridade dos dados, o agilityflow nunca apaga um registro de uma tabela. Apenas o marca como deletado. Portanto, para qualquer query, é preciso excluir os deletados da lista.

Essa marcação é feita com campo deletado. Se o valor for 0, significa que ele não foi deletado e está visível no sistema. Caso esteja 1, significa que o registro já foi excluído e não é possível mais vê-lo na listagem principal do sistema.

Exemplo 1, sem o tratamento de deletados:

```
select usu_nome, usu_email from tbl_usuario
```

Resultado:

Exemplo 2, com o tratamento de deletados:

```
select usu_nome, usu_email from tbl_usuario
where deletado = 0
```

Resultado:

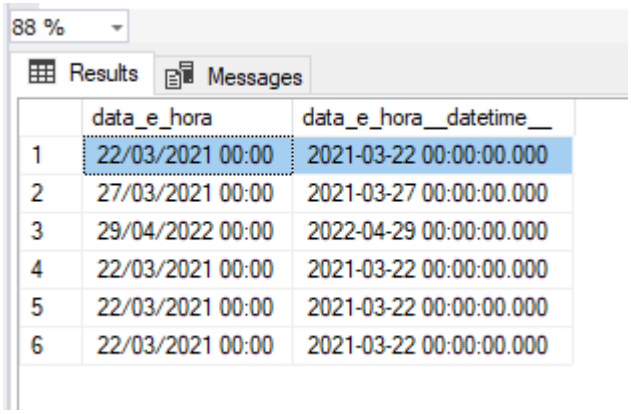
Campo do tipo Datetime (Data e Hora)

Sempre que um campo de texto com máscara do tipo data é criado, o sistema cria uma cópia com o mesmo nome seguido de "**__datetime__**" essa informação é gravada em uma coluna no banco de dados do tipo **datetime**, ao invés de varchar. Esses campos são criados para facilitar o uso tipado do dado nas queries

Exemplo de campo Datetime

Por exemplo, se o campo se chama "data_e_hora", haverá um outro chamado "data_e_hora__datetime__" e o conteúdo estará no formato padrão do sql server, **YYYY-MM-DD hh:mm:ss**

O campo "data_e_hora" estará sempre como varchar e o "data_e_hora__datetime__" estará como datetime



	data_e_hora	data_e_hora__datetime__
1	22/03/2021 00:00	2021-03-22 00:00:00.000
2	27/03/2021 00:00	2021-03-27 00:00:00.000
3	29/04/2022 00:00	2022-04-29 00:00:00.000
4	22/03/2021 00:00	2021-03-22 00:00:00.000
5	22/03/2021 00:00	2021-03-22 00:00:00.000
6	22/03/2021 00:00	2021-03-22 00:00:00.000

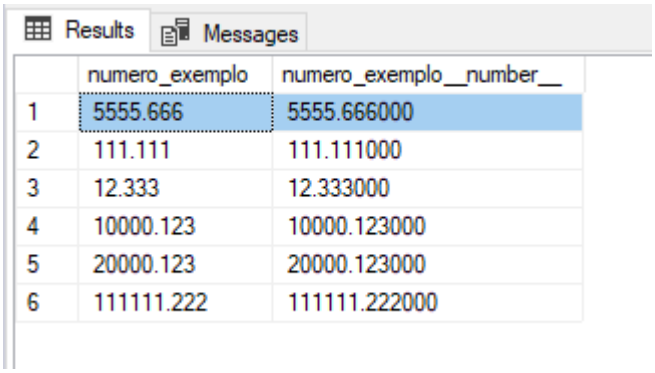
Campo do tipo Numérico (decimal, float, double, int, number)

Sempre que um campo de texto com máscara do tipo número é criado, o sistema cria uma cópia com o mesmo nome seguido de "**__number__**" essa informação é gravada em uma coluna no banco de dados do tipo **numeric(18,6),** ao invés de varchar. Esses campos são criados para facilitar o uso tipado do dado nas queries

Exemplo de campo numérico

Por exemplo, se o campo se chama "numero_exemplo", haverá um outro chamado "numero_exemplo__number__" e o conteúdo estará no formato padrão do sql server.

O campo "numero_exemplo" estará sempre como varchar e o "numero_exemplo__number__" estará como numeric(18,6)



	numero_exemplo	numero_exemplo__number__
1	5555.666	5555.666000
2	111.111	111.111000
3	12.333	12.333000
4	10000.123	10000.123000
5	20000.123	20000.123000
6	111111.222	111111.222000

Formatação de números

Nome da Função

`format(number,format,idioma)`

Parâmetro: **number**

Pode ser um número inteiro ou um decimal

Parâmetro: **format**

Aqui você passa a formatação que deve ser retornada:

'n0' = retorna o número sem nenhuma casa decimal
'n1' = retorna o número com 1 casa decimal
'n2' = retorna o número com 2 casas decimais
'n3' = retorna o número com 3 casas decimais
'n4' = retorna o número com 4 casas decimais
'n5' = retorna o número com 5 casas decimais
'n6' = retorna o número com 6 casas decimais

Parâmetro: **idioma** - **@sysCurrentLanguage**

Aqui você precisa passar o idioma do usuário logado, pois alguns idiomas invertem o . (ponto) e a , (virgula) do número.
O agilityflow guarda o idioma do usuário dentro da variável @sysCurrentLanguage então apenas passe no parâmetro esse variável.

Exemplo de utilização

Abaixo o número será formato para 5 casas decimais, no padrão do idioma do usuário logado.

```
select FORMAT(5800000.888, 'n5', @sysCurrentLanguage)
```

Retorno para o usuário que está logado no agilityflow em português ou espanhol

```
5.800.000,88800
```

Retorno para o usuário que está logado no agilityflow usando em inglês

```
5,800,000.88800
```

Funções de data no SQL Server

As principais funções para manipular datas são: **GETDATE**, **DATEPART**, **DATEADD** e **DATEDIFF**.

Um detalhe importante é que as funções de data trabalham referenciando unidades de data. As mais comuns são:

- year(ano);
- month(mês);
- day(dia);

GETDATE ( )

A função **GETDATE** retorna a data e a hora atuais do sistema.

```
SELECT GETDATE()
```

DATEPART (unidade, data)

A função **DATEPART** retorna a parte especificada de uma data como um inteiro. Observe os exemplos:

```
SELECT DATEPART ( YEAR , '2024-02-01' )
```

Reposta: 2024

```
SELECT DATEPART ( MONTH, '2024-02-01' )
```

Reposta: 2

```
SELECT DATEPART ( DAY, '2024-02-01' )
```

Reposta: 1

DATEADD (unidade, numero_unid, data)

A função **DATEADD** retorna uma nova data através da soma do número de unidades especificadas pelo valor *unidade* a uma data. Observe os exemplos:

```
SELECT DATEADD ( DAY ,6, '2024-02-01' )
```

Reposta: 2024-02-07

```
SELECT DATEADD ( MONTH ,6, '2024-02-01' )
```

Reposta: 2024-08-01

```
SELECT DATEADD ( YEAR ,6, '2024-02-01' )
```

Reposta: 2030-02-01

DATEDIFF (unidade, data1, data2)

A função **DATEDIFF** calcula a diferença entre as datas *data2 e data1*, retornando o resultado como um inteiro, cuja unidade é definida pelo valor *unidade*. Observe os exemplos:

```
SELECT DATEDIFF ( DAY , '2024-02-01' , '2024-05-25' )
```

Reposta: 114 (dias)

```
SELECT DATEDIFF ( MONTH , '2024-02-01' , '2024-05-25' )
```

Reposta: 3 (meses)

```
SELECT DATEDIFF ( YEAR , '2024-02-01' , '2026-05-25' ) )
```

Reposta: 2 (anos)

Sabendo disso, para por exemplo saber o número de dias vivido por você até hoje é:

```
SELECT DATEDIFF(DAY, suadata, GETDATE())
```

suadata deve ser substituída pela sua data de nascimento.

Outro exemplo interessante é mostrado através do código SQL abaixo. Usando funções de data, exibimos, para cada cliente, a idade em dias, meses e em anos (idade do cliente na data atual e

em 31 de dezembro).

Observe a lógica utilizada no comando **CASE**. Neste caso, é testado se o cliente já fez aniversário, comparando o mês em que ele nasceu com o mês corrente e comparando o dia em que ele nasceu com o dia corrente. Se essa comparação for verdadeira, basta diminuir o ano atual do ano de nascimento do cliente. Caso contrário (o cliente ainda não fez aniversário), temos que diminuir 1 do valor anterior.

```
SELECT NOME, NASCIMENTO,
  DATEDIFF(DAY,NASCIMENTO,GETDATE())AS DIASVIVIDOS,
  DATEDIFF(MONTH,NASCIMENTO,GETDATE()) AS MESESVIVIDOS,
  CASE WHEN
    DATEPART(MONTH,NASCIMENTO)<= DATEPART(MONTH,GETDATE()) AND
    DATEPART(DAY,NASCIMENTO)<= DATEPART(DAY,GETDATE())
  THEN
      (DATEDIFF(YEAR,NASCIMENTO,GETDATE()))
  ELSE
      (DATEDIFF(YEAR,NASCIMENTO,GETDATE()))- 1
      END AS IDADEATUAL,
  DATEDIFF(YEAR,NASCIMENTO,GETDATE())AS IDADE3112
FROM CLIENTE
```

26-05pic.JPG

Relatório

Relatório - Filtro

Sempre que um relatório possuir um filtro, é necessário incluir esse filtro na query que gera os dados do relatório.

Não importa como é a query, ela deve incluir a cláusula where, como a chamada da função Filter. Essa função, possui 3 parâmetros:

```
Filter (variável_filtro, tabela, campo)
```

- variável_filtro: é a variável que o agilityflow criou, após a configuração do filtro.

- tabela: nome da tabela (definido nos Dados Técnicos) onde se encontra o dado a ser filtrado.
- campo: é o nome do campo (conforme informado no campo Coluna Banco de Dados SQL) onde a informação que o filtro utiliza se encontra.

No exemplo abaixo, a query lista os campos *usu_nome*, *usu_email e usu_sexo* da tabela do sistema de usuário (tbl_usuario) e se inclui a função Filter, usando a variável @sexo aplicada no campo *usu_sexo*.

```
select usu_nome, usu_email, usu_sexo from tbl_usuario
where Filter(@sexo, tbl_usuario, usu_sexo)
```

Exemplo de relatório sem filtro

Baseado na query acima, mas sem filtro configurado no relatório.

```
select usu_nome, usu_email, usu_sexo from tbl_usuario
```

Exemplo de relatório com filtro

O mesmo exemplo acima, com o filtro no campo sexo.

```
select usu_nome, usu_email, usu_sexo from tbl_usuario
where Filter(@sexo, tbl_usuario, usu_sexo)
```

relatorio_filtro.gif

Caso seja criado um novo filtro, basta colocar "*and*" depois do primeiro Filter, e colocar o segundo Filter. Não importa a quantidade de filtros, desde que sejam adicionados todos os Filter e os *and*.

```
select usu_nome, usu_email, usu_sexo from tbl_usuario
where Filter(@sexo, tbl_usuario, usu_sexo) and
    Filter(@nome, tbl_usuario, usu_nome) and
    Filter(@email, tbl_usuario, usu_email)
```

Relatório - Tabela de Dados

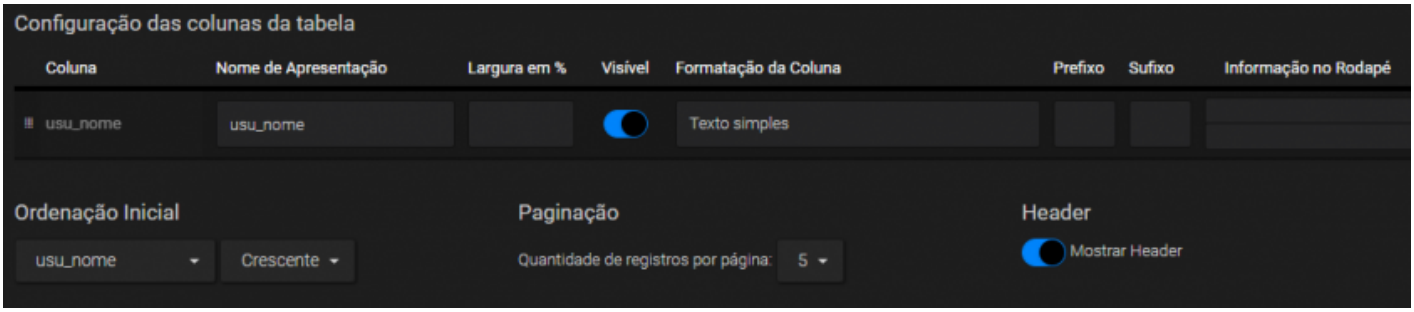
Para as tabelas de dados, existem algumas regrinhas para a customização via Query Sql

1 - Lembre-se de tratar os dados que já foram deletados, usando no where "...deletado = 0.."

2 - **Não** pode conter 'Order by' no select. Essa informação será controlada automaticamente pelo agilityflow e você poderá manipular essa informação na tela de configuração, como na imagem abaixo.

3 - **Não** pode conter o controle de 'Top' no select. Essa informação será controlada automaticamente pelo agilityflow e você poderá manipular essa informação na tela de configuração, como na imagem abaixo.

4 - **Não** pode conter regras de paginação. Essa informação será controlada automaticamente pelo agilityflow e você poderá manipular essa informação na tela de configuração, como na imagem abaixo.



Relatório - Gráfico

Para gráficos, existem algumas regrinhas para a customização via Query Sql

1 - Lembre-se de tratar os dados que já foram deletados, usando no where "...deletado = 0.."

2 - A query precisa iniciar com 'select **top** ' e o máximo do Top para essa query é **100**

3 - Diferentemente da query da tabela de dados, nessa query pode sim conter regras de 'Order by'

Relatório - Label

Para as labels do relatório, existem algumas regrinhas para a customização via Query Sql

1 - Lembre-se de tratar os dados que já foram deletados, usando no where "...deletado = 0.."

2 - A query precisa iniciar com 'select **top** ' e o máximo do Top para essa query é **1**

3 - Diferentemente da query da tabela de dados, nessa query pode sim conter regras de 'Order by'

Formulário

Lista Dinâmica - Regras

Os campos que são carregados com Lista dinâmica, podem ser customizados utilizando Query:

Algumas regras importantes:

1 - O resultado final da query precisa sempre ser os dados do formulário que você usou como base de dados

2 - Lembre-se de tratar os dados que já foram deletados, usando no where "...deletado = 0.."

3 - A query deve retornar duas colunas com os seguintes *alias*: "Name" e "Value". O "Name" deve ser a mesma coluna definida no campo de apresentação (Nome) e o "Value", deve ser a coluna "id" do formulário definido como base de dados (Cliente). Exemplo:

4 - Na query, o campo de apresentação "Name" pode ser um campo concatenado entre outras colunas.

Exemplo simples:

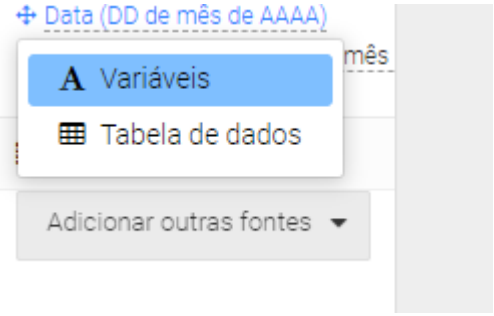
```
SELECT Id as Value, [CAMPO_DE_APRESENTACAO] as Name FROM [TABELA] where deletado = 0
```

Report Print - Relatório de Impressão

Para as labels do relatório, existem algumas regrinhas para a customização via Query Sql

Para acessar a customização, clique no botão "Adicionar Outras Fontes de Dados" e selecione o tipo:

1. **Variáveis:** para retornar uma query com apenas 1 linha e várias colunas, cada coluna se torna um campo separado para ser utilizado no relatório
2. **Tabela de Dados:** para retornar uma tabela com diversas colunas e diversas linhas e usa-las em conjunto em uma tabela



Algumas regras importantes:

1 - Lembre-se de tratar os dados que já foram deletados, usando no where "...deletado = 0.."

2 - Utilizar no "where" da query, o parâmetro **@formularioid** para filtrar pelo formulário que está sendo solicitada a impressão, caso contrário trará informações de formulários aleatórios.

3 - A query precisa iniciar com 'select **top** ' e o máximo do Top para essa query quando for uma query do tipo:

- **Tabela de Dados** é **500**,
- **Variáveis** é **1**

4 - Essa query pode sim conter regras de 'Order by'

Postgresql (Query) - Dicas e Funções (Versões Mais Recentes do Agilityflow)

> **IMPORTANTE**: Essa documentação é referente as versões pós 2023 do agilityflow que utilizam o Banco de Dados **Postgresql.** O agilityflow nas últimas versões está utilizando **POSTGRESQL** .

No agilityflow você pode buscar os dados utilizando SQL, através de queries. As queries devem ser compatíveis com **SQL Server.** Além disso existem alguns padrões que você deve seguir para obter um melhor resultado no agilityflow. Abaixo estão alguns detalhes importantes.

Tratamento de Registros Deletados

> **IMPORTANTE:**
> É de extrema importância o tratamento de registros deletados, usando as regras abaixo.
>
> Já está no nosso Roadmap o tratamento automático dos deletados, enquanto isso, o tratamento se torna obrigatório pelo desenvolvedor.

Para manter a integridade dos dados, o agilityflow nunca apaga um registro de uma tabela. Apenas o marca como deletado. Portanto, para qualquer query, é preciso excluir os deletados da lista.

Essa marcação é feita com campo deletado. Se o valor for **FALSE**, significa que ele não foi deletado e está visível no sistema. Caso esteja **TRUE**, significa que o registro já foi excluído e não é possível mais vê-lo na listagem principal do sistema.

Exemplo 1, sem o tratamento de deletados:

```
select usu_nome, usu_email from tbl_usuario
```

Resultado:

Exemplo 2, com o tratamento de deletados:

```
select usu_nome, usu_email from tbl_usuario
where deletado = false
```

Resultado:

Campo do tipo Date, Timestamp e Datetime (Data e Hora)

Sempre que um campo de texto com máscara do tipo data é criado, o sistema cria uma cópia com o mesmo nome seguido de "**__datetime__**" essa informação é gravada em uma coluna no banco de dados do tipo **"date"** quando for uma coluna sem hora ou **"timestamp without time zone"** quando for uma coluna com hora_, ao invés de varchar. Esses campos são criados para facilitar o uso tipado do dado nas queries

Exemplo de campo Datetime

Por exemplo, se o campo se chama "data_e_hora", haverá um outro chamado "data_e_hora__datetime__" e o conteúdo estará no formato padrão do sql server, **YYYY-MM-DD hh:mm:ss**

O campo "data_e_hora" estará sempre como varchar e o "data_e_hora__datetime__" estará como timestamp

data_e_hora	data_e_hora__datetime__
2027-01-31 23:5959	2027-01-31 23:5959
2027-01-31 23:5959	2027-01-31 23:5959
2027-01-31 23:5959	2027-01-31 23:5959

Campo do tipo Numérico (decimal, float, double, int, number)

Sempre que um campo de texto com máscara do tipo número é criado, o sistema cria uma cópia com o mesmo nome seguido de "**__number__**" essa informação é gravada em uma coluna no banco de dados do tipo **numeric(18,6),** ao invés de varchar. Esses campos são criados para facilitar o uso tipado do dado nas queries

Exemplo de campo numérico

Por exemplo, se o campo se chama "numero_exemplo", haverá um outro chamado "numero_exemplo__number__" e o conteúdo estará no formato padrão do sql server.

O campo "numero_exemplo" estará sempre como varchar e o "numero_exemplo__number__" estará como numeric(18,6)

numero_exemplo	numero_exemplo__number__
999.99	999.99
123.89	123.89

Formatação de números

Nome da Função

`format_number(number,format,idioma)`

Parâmetro: **number**

Pode ser um número inteiro ou um decimal

Parâmetro: **format**

Aqui você passa a formatação que deve ser retornada:

'0' = retorna o número sem nenhuma casa decimal
'1' = retorna o número com 1 casa decimal
'2' = retorna o número com 2 casas decimais
'3' = retorna o número com 3 casas decimais
'4' = retorna o número com 4 casas decimais
'5' = retorna o número com 5 casas decimais
'6' = retorna o número com 6 casas decimais

Parâmetro: **idioma** - **@sysCurrentLanguage**

Aqui você precisa passar o idioma do usuário logado, pois alguns idiomas invertem o . (ponto) e a , (virgula) do número.
O agilityflow guarda o idioma do usuário dentro da variável @sysCurrentLanguage então apenas passe no parâmetro esse variável.

Exemplo de utilização

Abaixo o número será formato para 5 casas decimais, no padrão do idioma do usuário logado.

```
select format_number(5800000.888, '5', @sysCurrentLanguage)
```

Retorno para o usuário que está logado no agilityflow em português ou espanhol

```
5.800.000,88800
```

Retorno para o usuário que está logado no agilityflow usando em inglês

```
5,800,000.88800
```

Funções de data no Postgresql

As principais funções para manipular datas são: **GETDATE**, **DATEPART**, **DATEADD** e **DATEDIFF**.

Um detalhe importante é que as funções de data trabalham referenciando unidades de data. As mais comuns são:

- year(ano);
- month(mês);
- day(dia);

CURRENT_TIMESTAMP)

A função CURRENT_TIMESTAMP retorna a data e a hora atuais do sistema.

```
SELECT CURRENT_TIMESTAMP
```

Extrair parte de uma data

A função EXTRACT retorna a parte especificada de uma data como um inteiro. Observe os exemplos:

```
SELECT EXTRACT(YEAR FROM '2024-02-01'::DATE)
```

Reposta: 2024

```
SELECT EXTRACT(MONTH FROM '2024-02-01'::DATE)
```

Reposta: 2

```
SELECT EXTRACT(DAY FROM '2024-02-01'::DATE)
```

Reposta: 1

Adicionar dias em uma data

A função **abaixo** retorna uma nova data através da soma do número de unidades especificadas pelo valor *unidade* a uma data. Observe os exemplos:

```
SELECT '2024-02-01'::DATE + INTERVAL '7 days';
```

Reposta: 2024-02-07

Calcular a diferença entre datas (date diff)

A função **abaixo** calcula a diferença entre as datas *data2 e data1*, retornando o resultado como um inteiro, cuja unidade é definida pelo valor *unidade*. Observe os exemplos:

```
SELECT EXTRACT(DAY FROM '2024-02-01'::DATE - '2023-10-10'::DATE);
```

Reposta: 114 (dias)

Cálculo de idade em anos, meses e dias

```
SELECT DATE_PART('year', AGE(NOW(), '1990-05-15')) AS idade_anos;
```

Relatório

Como fazer Filtro nos Relatórios?

Sempre que um relatório possuir um filtro, é necessário incluir esse filtro na query que gera os dados do relatório.

Não importa como é a query, ela deve incluir a cláusula where, como a chamada da função Filter. Essa função, possui 3 parâmetros:

`**Filter** (variável_filtro, tabela, campo)`

- variável_filtro: é a variável que o agilityflow criou, após a configuração do filtro.
- tabela: nome da tabela (definido nos Dados Técnicos) onde se encontra o dado a ser filtrado.
- campo: é o nome do campo (conforme informado no campo Coluna Banco de Dados SQL) onde a informação que o filtro utiliza se encontra.

No exemplo abaixo, a query lista os campos *usu_nome*, *usu_email e usu_sexo* da tabela do sistema de usuário (tbl_usuario) e se inclui a função Filter, usando a variável @sexo aplicada no campo *usu_sexo*.

```
select usu_nome, usu_email, usu_sexo from tbl_usuario
where Filter(@sexo, tbl_usuario, usu_sexo)
```

Exemplo de relatório sem filtro

Baseado na query acima, mas sem filtro configurado no relatório.

```
select usu_nome, usu_email, usu_sexo from tbl_usuario
```

Exemplo de relatório com filtro

O mesmo exemplo acima, com o filtro no campo sexo.

```
select usu_nome, usu_email, usu_sexo from tbl_usuario
where Filter(@sexo, tbl_usuario, usu_sexo)
```

relatorio_filtro.gif

Caso seja criado um novo filtro, basta colocar "*and*" depois do primeiro Filter, e colocar o segundo Filter. Não importa a quantidade de filtros, desde que sejam adicionados todos os Filter e os *and*.

```
select usu_nome, usu_email, usu_sexo from tbl_usuario
where Filter(@sexo, tbl_usuario, usu_sexo) and
    Filter(@nome, tbl_usuario, usu_nome) and
    Filter(@email, tbl_usuario, usu_email)
```

Relatório - Tabela de Dados

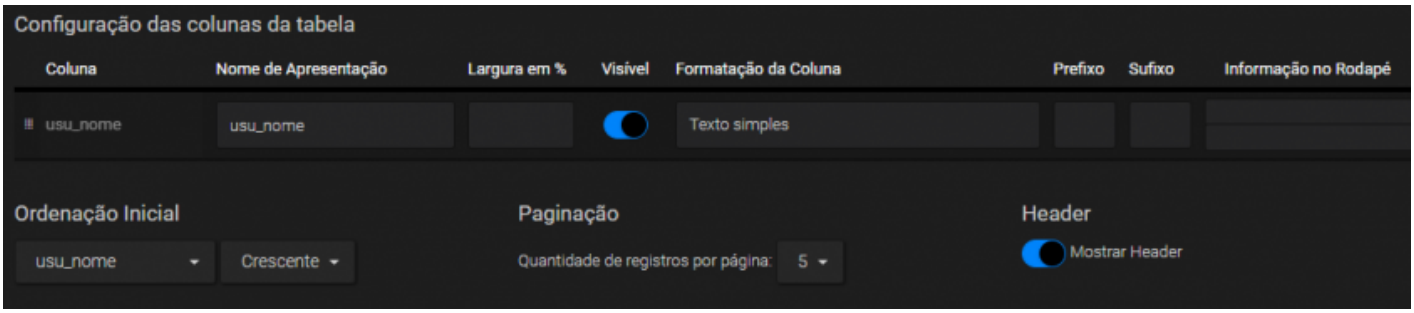
Para as tabelas de dados, existem algumas regrinhas para a customização via Query Sql

1 - Lembre-se de tratar os dados que já foram deletados, usando no where "...deletado = false.."

2 - **Não** pode conter 'Order by' no select. Essa informação será controlada automaticamente pelo agilityflow e você poderá manipular essa informação na tela de configuração, como na imagem abaixo.

3 - **Não** pode conter o controle de 'Limit' no select. Essa informação será controlada automaticamente pelo agilityflow e você poderá manipular essa informação na tela de configuração, como na imagem abaixo.

4 - **Não** pode conter regras de paginação. Essa informação será controlada automaticamente pelo agilityflow e você poderá manipular essa informação na tela de configuração, como na imagem abaixo.



Relatório - Gráfico

Para gráficos, existem algumas regrinhas para a customização via Query Sql

1 - Lembre-se de tratar os dados que já foram deletados, usando no where "...deletado = false.."

2 - A query precisa conter '**limit** ' e o máximo do limit para essa query é **100**

3 - Diferentemente da query da tabela de dados, nessa query pode sim conter regras de 'Order by'

Relatório - Label

Para as labels do relatório, existem algumas regrinhas para a customização via Query Sql

1 - Lembre-se de tratar os dados que já foram deletados, usando no where "...deletado = false.."

2 - A query precisa conter '**limit** ' e o máximo do limit para essa query é **1**

3 - Diferentemente da query da tabela de dados, nessa query pode sim conter regras de 'Order by'

Formulário

Lista Dinâmica - Regras

Os campos que são carregados com Lista dinâmica, podem ser customizados utilizando Query:

Algumas regras importantes:

1 - O resultado final da query precisa sempre ser os dados do formulário que você usou como base de dados

2 - Lembre-se de tratar os dados que já foram deletados, usando no where "...deletado = false.."

3 - A query deve retornar duas colunas com os seguintes *alias*: "Name" e "Value". O "Name" deve ser a mesma coluna definida no campo de apresentação (Nome) e o "Value", deve ser a coluna "id" do formulário definido como base de dados (Cliente). Exemplo:

4 - Na query, o campo de apresentação "Name" pode ser um campo concatenado entre outras colunas.

Exemplo simples:

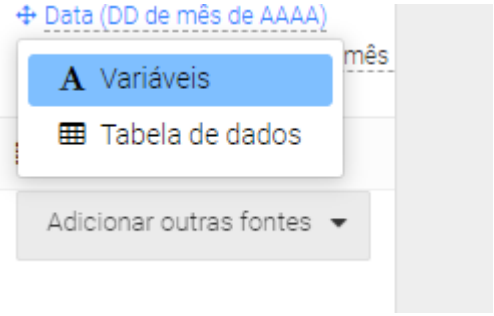
```
SELECT Id as Value, CAMPO_DE_APRESENTACAO as Name FROM [TABELA] where deletado = false
```

Report Print - Relatório de Impressão

Para as labels do relatório, existem algumas regrinhas para a customização via Query Sql

Para acessar a customização, clique no botão "Adicionar Outras Fontes de Dados" e selecione o tipo:

1. **Variáveis:** para retornar uma query com apenas 1 linha e várias colunas, cada coluna se torna um campo separado para ser utilizado no relatório
2. **Tabela de Dados:** para retornar uma tabela com diversas colunas e diversas linhas e usa-las em conjunto em uma tabela



Algumas regras importantes:

1 - Lembre-se de tratar os dados que já foram deletados, usando no where "...deletado = false.."

2 - Utilizar no "where" da query, o parâmetro **@formularioid** para filtrar pelo formulário que está sendo solicitada a impressão, caso contrário trará informações de formulários aleatórios.

3 - A query precisa conter '**limit** ' e o máximo do limit para essa query quando for uma query do tipo:

- **Tabela de Dados** é **500**,
- **Variáveis** é **1**

4 - Essa query pode sim conter regras de 'Order by'

Programação em Html

Para acessar os detalhes desse conteúdo, clique aqui.

Programação em CSS

Para acessar os detalhes desse conteúdo, clique aqui.

Programação em Javascript

Para acessar os detalhes desse conteúdo, clique aqui.

Programação em C# - Na Regra de Negócio

> IMPORTANTE: as variáveis e métodos descritos aqui só funcionarão na programação C# na **Regra de Negócio** do formulário. Para a programação C# nas **APIs** clique aqui.

O agilityflow permite a customização em C#, além de já disponibilizar diversas bibliotecas e funções para facilitar sua programação, incluindo acesso a dados, validações, envio de e-mail, notificação, entre outras.

Abaixo mostramos alguns exemplos, se o material abaixo não for suficiente , entre em contato com nossa equipe.

C# - Propriedades

Variável	**Tipo**	
IsNovoFormulario	bool	retorna true, caso seja um novo formulário, e false caso contrário.
IdFormulario	Guid	retorna o ID do formulário

CurrentStatusFluxo	StatusFluxo? (Enum)	Variável disponível apenas para Workflow
		
		Nessa ENUM será retornado o Status do fluxo.
		As opções podem ser:
		
		• **StatusFluxo.Pendente -** Enquanto o fluxo ainda estiver pendente em alguma das etapas.
		• **StatusFluxo.Aprovado -** Depois do fluxo encerrado e completamente aprovado
		• **StatusFluxo.Reprovado -** Depois do fluxo encerrado e completamente reprovado
		
		Exemplo de utilização:
		
		`if(CurrentStatusFluxo == StatusFluxo.Aprovado){`
		`  //restante do código`
		`}`

(deprecated) ~~FormularioCamposPreenchidos~~	**(deprecated)** ~~dynamic (Json)~~	**(deprecated**: utilizar o método FormContext.GetValue descrito mais abaixo)
		retorna um json com os campos do preenchidos no formulário, exemplo:
		{
		"campo1": "xxxxx",
		"campo2": "yyyyy",
		"campo3": "zzzzz",
		}
		
		Para resgatar a informação do campo1, utilize:
		
		`var campo1 = FormularioCamposPreenchidos["campo1"];`

C# - Buscar e preencher valor de um campo

Método	**Retorno**	

FormContext.SetValue(string idCampo, string value)	void	Esse método preencherá um campo com um novo **Valor** (passado por paramêtro)
		
		Para valores que são **Datas**, utilizar o padrão dd/MM/aaaa HH:mm:ss:
		Exemplo: ***31/12/2010 22:55***
		`var data_e_hora = "31/01/2023 22:55";`
		`FormContext.SetValue("data_e_hora",data_e_hora);`
		
		Para valores **Numéricos**, o número deve estar no formato com ponto no separador decimal e sem utilização de virgulas.
		Exemplo para 2 milhões e 50 centavos**:**
		2000000.50
		`var moeda = "2000000.50";`
		`FormContext.SetValue("moeda",moeda);`
		
		Veja exemplo

FormContext.**GetValue**(string idCampo)	string	Esse método retornará o **Valor** de um determinado campo
FormContext.GetText(string idCampo**)**	string	Esse método retornará o **Texto** de um determinado campo, utilizado apenas se o campo for um campo de **lista dinâmica**, exemplo: Combo, Auto complete, Campo de Múltipla escolha, radio etc. Veja exemplo
FormContext.GetInt(string idColuna)	int?	Retorna o valor da coluna no tipo INT (inteiro), caso esse campo no formulário seja um número, se o valor for 10.000,99 Será retornado: 10000 Veja exemplo

FormContext.GetDecimal(string idColuna)	decimal?	Retorna o valor da coluna no tipo Decimal, caso esse campo no formulário seja um número, se o valor for 10.000,99 Será retornado: 10000.99 Veja exemplo
FormContext.GetDateTime(string idColuna)	DateTime?	Retorna o valor da coluna no tipo DateTime para valores que estão no formato de data dd/MM/yyyy ou dd/MM/yyyy hh:mm Veja exemplo

C# - Buscar Rascunho por Id

Método	Retorno	

FormContext.**GetDrafJsontById**(string draftid)
ou
FormContext.**GetDrafJsontById**(Guid?draftid)
	dynamic	var str_preco_de_venda = "";
		var str_quantidade = "";
		var temRascunho = false;
		var rascunho_json = PageContext.GetDraftJsontById(str_draftid);
		if(rascunho_json != null)
		{
		temRascunho = true;
		str_preco_de_venda = rascunho_json.preco_de_venda;
		str_quantidade = rascunho_json.quantidade;
		}

C# - Executar uma consulta SQL (Query SQL)

Método	Retorno	
await FormContext.**GetDataTableAsync**(string sql, params DbParameter[] parameters)	DataTable	Permite retornar informações dos bancos de dados através de uma query. Veja exemplo

C# - Inserir (salvar/criar), alterar, deletar e aprovar outros formulários

Método	**Retorno**	
await FormContext.SaveEntityAsync (Guid estruturaformularioid_ASerCriadoOuAtualizado, DataDictionary campos_e_valores)	Guid	Salva (Insere e Atualiza) as informações de um formulário no bancos de dados. Para atualizar um registro, no parâmetro **campos_e_valores**, apenas adicione um parâmetro chamado "**id**" com o id do elemento que você deseja atualizar. Obs: Para cadastrar valores numéricos, o números devem estar no formato com ponto no separador decimal: 999999.99 sem utilização de virgulas. Veja exemplo
await FormContext.ApproveEntityAsync (Guid estruturaformularioid_ASerAprovado, DataDictionary campos_e_valores)	Guid	Avança uma etapa em um formulário com Workflow. As regras de usuário aprovador, permissão de aprovação e responsabilidade por uma etapa devem ser validadas antes desse método ser executado. Obs: Para cadastrar valores numéricos, o números devem estar no formato com ponto no separador decimal: 999999.99 sem utilização de virgulas. Veja exemplo
await FormContext.DeleteEntityAsync(Guid formularioid)	void	Deletar um um registro

C# - Apresentar mensagem para o usuário

Método	**Retorno**	
FormContext.WarningMessage (string message)	void	Apresenta um aviso para o usuário veja detalhes aqui

C# - Buscar informações do usuário logado

Método	Retorno	
FormContext. GetUsuarioLogadoNome();	string	Nome Completo do usuário logado
FormContext. GetUsuarioLogadoPrimeiroNome();	string	Primeiro nome do usuário logado
FormContext. GetUsuarioLogadoPrimeiroESegundoNome();	string	Primeiro e Segundo nome do usuário logado
FormContext. GetUsuarioLogadoEmail();	string	E-mail do usuário logado
FormContext. GetUsuarioLogadoId();	Guid	Id do usuário logado

C# - Buscar variáveis de ambiente

Método	Retorno	
FormContext.GetEnvironmentVariable(string nomeVariavel) ou **FormContext.GetEnvironmentVariable_Text**(string nomeVariavel) **	string	Recupera o Valor de uma variável de ambiente. ** Recupera a Descrição (texto), no caso de variáveis do Tipo "Query com Id + Descrição":

C# - Formatação Numérica

> ***Se você está utilizando o método GetValue(..) para buscar o valor de um campo numérico, esse método já retornará o número formatado e não será necessário nenhuma formatação adicional.***
>
> ***Se você está utilizando o método GetDecimal(..) ele não virá formatado pois ele retorna o tipo DECIMAL do C#, sendo assim, se você precisa do dado formatado você pode utilizar o GetText(..)***

string **FormContext.FormatDecimalToString**(object numDecimal, string mascaraTipo = "dec2")	formata object 999999999.55 para o formato da mascara de acordo com o idioma. - se for ingles, será 999,999,999.55 - se for portugues, será 999.999.999,55
string **FormContext.FormatNumberToString_EnglishFormat**(object num, [optional]int? qtdCasasDecimais)	converte decimal para string, porém sempre a string vem no formato Inglês. coloca a a mesma qtd de casas decimais, caso o parâmetro qtdCasasDecimais estivar em branco

C# - Gerar log

Método	Retorno	
FormContext.Log(string log, string logTipo)	void	Cria um log na tabela de log geral, o log pode ser consultado entrando na Área de Customização e clicando na opção "Log Geral >> consultar Log"

C# - Métodos Úteis

Método	Retorno	
FormContext.GetUrlBase()		Retorna a URL base da sua aplicação exemplo: https://suaempresa.agilityflow.io/ *A url sempre virá com uma barra no final **/**

Método principal para utilização da Programação em C# na opção "Regra de Negócio"

A única regra é que o método se chame Execute, não tenha parâmetros de entrada e o retorno seja void.

Exemplo:

```
public void Execute(){



  //aqui você pode programar



}
```

```
  //aqui você pode programar outros métodos
```

Exemplos

Recuperando dados do banco de dados através de uma query SQL

```
//parametros da query
var paramsQuery = new List<NpgsqlParameter>();
paramsQuery.Add(new NpgsqlParameter("@param1", "xxx"));
paramsQuery.Add(new NpgsqlParameter("@param2", "yyy"));
paramsQuery.Add(new NpgsqlParameter("@param3", "zzz"));

//query usando (nolock) nas tabelas para evitar problema com a transação que está ativa
var sql = "select column1,column2,column3 from table (nolock) where column1 = @param1 or  column2 =
@param2 or  column3 = @param3 ";

//executar no banco de dados
var dt = await FormContext.GetDataTableAsync(sql, paramsQuery.ToArray());

//percorrendo as linhas de retorno da tabela
foreach (DataRow dr in dt.Rows)
{
  if (dr["column1"] != DBNull.Value)
  FormContext.WarningMessage(dr["column1"].ToString());
}
```

Modificando os campos do próprio formulário que está sendo salvo

```
//concatenando o texto "abcdef" ao campo de texto chamado "texto_simples"
var texto_simples = FormContext.GetValueField("texto_simples");
texto_simples += " abcdef ";
FormContext.SetValue("texto_simples",texto_simples);

//colocando uma data fixa em um campo de data chamado "data_e_hora"
var data_e_hora = "31/01/2023 22:55";
FormContext.SetValue("data_e_hora",data_e_hora);



//colocando um valor decimal fixo em um campo de data chamado "moeda"
var moeda = "2000000.50";
```

```
FormContext.SetValue("moeda",moeda);



//somando 200 dias a um campo de data chamado "data_e_hora_2"
var dt_data_e_hora = FormContext.GetDateTime("data_e_hora_2");
if(dt_data_e_hora != null){
  var nova_data = dt_data_e_hora.Value.AddDays(200);
  FormContext.SetValue("data_e_hora_2", nova_data);
}

//somando 900.12 a um campo de moeda chamado "moeda2"
var num_moeda = FormContext.GetDecimal("moeda2");
if(num_moeda != null){
  var novo_valor = num_moeda.Value + 9000000.12m;

  FormContext.SetValue("moeda2",novo_valor);
}
```

Salvando/Criar um outro formulário através do método SaveEntityAsync

* Para cadastrar valores numéricos, o números devem estar no formato com ponto no separador decimal: 999999.99 sem utilização de virgulas.

```
//guarda em uma variável o ID da  E S T R U T U R A  do formulário de pessoa
var idEstruturaFormulario_PESSOA = Guid.Parse("0b56b66a-4f6f-4ded-ad04-016d7c0724e1");

var values = new DataDictionary();
values.Add("nome", FormContext.GetValue("nome"));
values.Add("email", FormContext.GetValue("email"));



//salva a informação no banco de dados
await FormContext.SaveEntityAsync(idEstruturaFormulario_PESSOA, values);
```

Atualizar um outro formulário através do método SaveEntityAsync

* Para cadastrar valores numéricos, o números devem estar no formato com ponto no separador decimal: 999999.99 sem utilização de virgulas.

```
//guarda em uma variável o ID da  E S T R U T U R A  do formulário de pessoa
var idEstruturaFormulario_PESSOA = Guid.Parse("0b56b66a-4f6f-4ded-ad04-016d7c0724e1");

var values = new DataDictionary();
values.Add("id", "33f6bdf8-26d9-42b9-94ae-ad44c62725b5"); //insere aqui o ID do registro que você deseja atualizar
values.Add("nome", FormContext.GetValue("nome"));
values.Add("email", FormContext.GetValue("email"));



//salva a informação no banco de dados
await FormContext.SaveEntityAsync(idEstruturaFormulario_PESSOA, values);
```

> Caso você queira atualizar informações do próprio formulário que está sendo salvo, utilize a função SetValue, descrita mais acima nos métodos da Classe FormContext

Aprovar um outro formulário através do método ApproveEntityAsync

* Para cadastrar valores numéricos, o números devem estar no formato com ponto no separador decimal: 999999.99 sem utilização de virgulas.

```
//guarda em uma variável o ID da  E S T R U T U R A  do formulário de pessoa
var idEstruturaFormulario_PESSOA = Guid.Parse("0b56b66a-4f6f-4ded-ad04-016d7c0724e1");

var values = new DataDictionary();
values.Add("nome", FormContext.GetValue("nome"));
values.Add("email", FormContext.GetValue("email"));



//salva a informação no banco de dados
await FormContext.ApproveEntityAsync(idEstruturaFormulario_PESSOA, values);
```

Salvando um outro formulário com Tabela Associativa através do método SaveEntityAsync

* Para cadastrar valores numéricos, o números devem estar no formato com ponto no separador decimal: 999999.99 sem utilização de virgulas.

```
//guarda em uma variável o ID da  E S T R U T U R A  do formulário de pessoa
var idEstruturaFormulario_PESSOA = Guid.Parse("0b56b66a-4f6f-4ded-ad04-016d7c0724e1");

var values = new DataDictionary();
values.Add("nome", json["nome"].ToString());
values.Add("email", json["email"].ToString());



    //no caso do campo do formulário ser uma tabela associativa (tabela relacional N:N)
    var idCampoTabelaAssociativa = "70b6f362-2587-4fd2-a2fb-148bd0caf437";

    //itens da tabela associativa que serão adicionados
    var idItem1 = "51cf02f1-3787-4dca-8a2c-e219a5ce1298";
    var idItem2 = "f999f103-c775-4245-92d3-034cb3ded5e4";
    var idItem3 = "XXXXf103-c775-4245-92d3-034cb3ded5e4";
    var idItem4 = "YYYYf103-c775-4245-92d3-034cb3ded5e4";
    var idsJuntos_ConcatenadosComVirgula = idItem1 + "," + idItem2 + "," + idItem3 + "," + idItem4 + ",";

    //adiciona
    values.Add(idCampoTabelaAssociativa, idItem1); //adiciona idItem1
    values.Add(idCampoTabelaAssociativa, idItem2); //adiciona idItem2
    values.Add(idCampoTabelaAssociativa, idItem3); //adiciona idItem3
    values.Add(idCampoTabelaAssociativa, idItem4); //adiciona idItem4
    values.Add("tabela_associativa_added-"+idCampoTabelaAssociativa,idsJuntos_ConcatenadosComVirgula);//
itens concatenados com virgula
    values.Add("tabela_associativa_removed-"+idCampoTabelaAssociativa,"");//em branco
    values.Add("tabela_associativa_colunas_adicionais-"+idCampoTabelaAssociativa,"");//em branco
    //----------------------------------------



//salva a informação no banco de dados
await FormContext.SaveEntityAsync(idEstruturaFormulario_PESSOA, values);
```

Para recuperar o Texto de um campo que seja uma Lista Dinâmica (Combo, Auto completar, Múltipla seleção, Radio):

Suponhamos que você tenha um formulário de Pedido e nesse formulário você tenha um campo chamado 'Produto'.
Esse campo é do tipo 'Pesquisa com Auto Completar' e listará os 'Produtos' por 'Nome'.

Quando o usuário salvar o formulário, no Json de envio não retornará o Nome do Produto, apenas o Id do produto selecionado (Esse id estará no formato de um GUID).
Para recuperar o Nome do produto, você precisará executar o método **GetText** da seguinte forma:

```
var nomeProduto = FormContext.GetText("produto");
```

Recuperar os valores das "Variáveis de Ambiente"

Para saber mais sobre variáveis de ambiente entre em Variáveis de ambiente

```
var valorDaVariavel = FormContext.GetEnvironmentVariable("var_nomeDaVariavel");
```

Recuperar a Descrição (texto), no caso de variáveis do Tipo "Query com Id + Descrição":

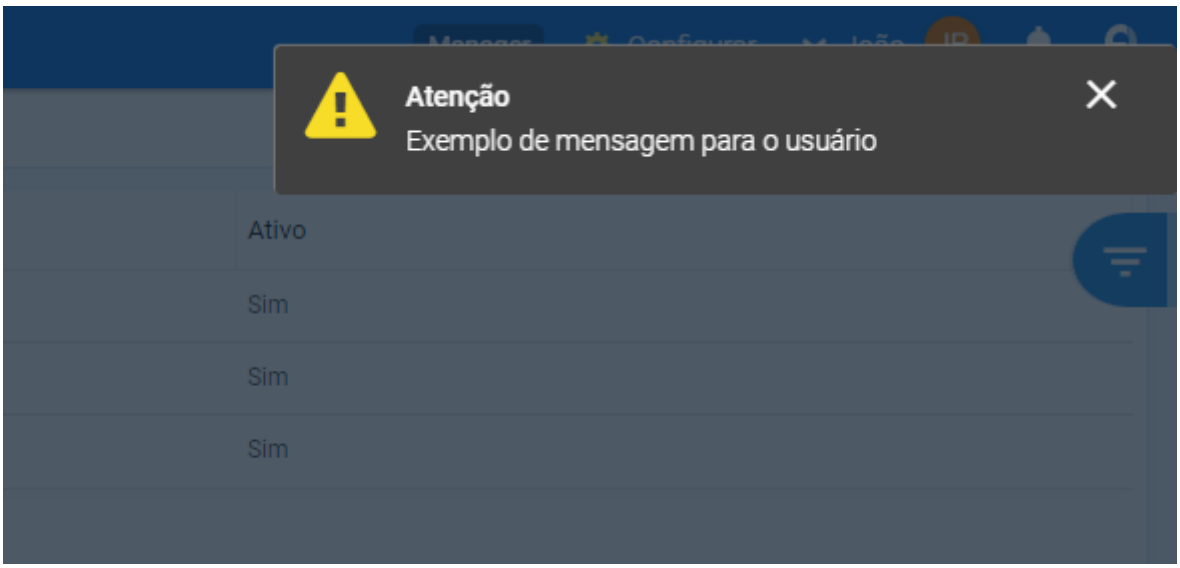
```
var textDaVariavel = FormContext.GetEnvironmentVariable_Text("var_nomeDaVariavel");
```

Enviar Mensagem para o usuário: A execução do código abaixo apresentará uma mensagem para o usuário

```
FormContext.WarningMessage("Exemplo de mensagem para o usuário");
```

> O código C# se encerrará no exato momento em que for chamado o método para apresentação de mensagem.

Retorno da mensagem



Envio de e-mail

Responsável pelo envio de e-mail

Método	**Retorno**	
EmailSender.SendToGrupoUsuario(Guid grupoUsuarioId, string subject, string htmlContent)	void	Enviar um e-mail para um determinado Grupo de Usuario
EmailSender.SendEmail(string toEmail, string subject, string htmlContent)	void	Enviar um e-mail
EmailSender.SendEmail(string toEmail, string subject, string htmlContent, string ccEmail**)**	void	Enviar um e-mail, com copia (CC)

Enviar e-mail para um Grupo de Usuário

Antes de iniciar, crie um "Grupo de Usuário" no menu "Segurança e Acesso". Depois de criado, associe os usuários que receberão e-mail ao novo grupo de usuário que você acabou de criar.

Pegue o ID desse novo Grupo de usuário. Você vai precisar dele para enviar o e-mail.

```
var grupoUsuarioId = Guid.Parse("af7c9275-0e23-4b64-a433-f238bb457005"); //substitua o ID "af7c9275-0e23-4b64-a433-f238bb457005" pelo Id do grupo de usuario que você deseja enviar o e-mail
var assunto = "Novo E-mail para um Grupo de usuario ";
var html = "Olá Grupo, <BR><BR> Teste para envio de e-mail para um Grupo de Usuário no AgilityFlow!";

EmailSender.SendToGrupoUsuario(grupoUsuarioId, assunto, html);
```

Enviar e-mail

```
var htmlContent = "<strong>Olá,</strong> teste envio de e-mail.";
var emailTo = "john@email.com";
var subject = "Subject do E-mail";

EmailSender.SendEmail(emailTo, subject, htmlContent);
```

Programação em C# - Na API de integração (POST)

As informações descritas abaixo referem-se as APIs de **POST**

> IMPORTANTE: as variáveis e métodos descritos aqui só funcionarão na programação C# na **API de integração**. Para a programação C# nas Regras de Negócio de um formulário clique aqui

O agilityflow permite a customização da API em C#, além de já disponibilizar diversas bibliotecas e funções para facilitar sua programação, incluindo acesso a dados, validações, envio de e-mail, notificação, entre outras.

> Para cada requisição na API é criada uma transação de banco de dados, essa transação é única em toda execução da API e é gerenciada automaticamente pelo Agilityflow, caso você prefira, você pode fazer esse gerenciamento seguindo esses passos, veja mais detalhes aqui

Abaixo mostramos alguns exemplos, se o material abaixo não for suficiente , entre em contato com nossa equipe.

Variáveis disponíveis

Variável	**Tipo**	
content	string	Conteúdo enviado no Body da API No caso se ser enviado uma string com o JSON, por exemplo no formato: `{ nome: 'José', email: 'jose@xxxx.com'}` Você pode converter essa string para JSON e utilizá-la como object `var json = Newtonsoft.Json.JsonConvert.DeserializeObject<dynamic>(content);`

Classe JsonHelper e outras para auxiliar o tratamento de JSON

Para utilizar execute `JsonHelper.NomeDoMetodo(....)`

Método	**Retorno**	

HasProperty(dynamic json, string nomePropriedade)	bool
	(true or false)
		Testa se o JSON tem uma determinada propriedade.
		Por exemplo, a sua API espera receber o JSON
		no formato:
		`{ nome: 'José', email: 'jose@xxxx.com'}`
		
		Porém recebe o JSON formato:
		`{ nomecompleto: 'José da Silva', idade : '21'}`
		
		Como a propriedade NOME e EMAIL não existem no JSON recebido, o seu programa pode dar erro. Para evitar o erro, utilize o método HasProperty(...) Como no exemplo abaixo:
		
		`if (json != null)`
		`{`
		
		`if (JsonHelper.HasProperty(json, "nome") && JsonHelper.HasProperty(json, "email") ){`
		
		`var nome = json["nome"].ToString();`
		`var email= json["email"].ToString();`
		
		`}`
		
		`}`

`Newtonsoft.Json.JsonConvert.DeserializeObject<dynamic>(content);`	dynamic	converte o conteúdo string para json
		
		`var content = "{ nome: 'José', email: 'jose@xxxx.com'}";`
		`var json = Newtonsoft.Json.JsonConvert.DeserializeObject<dynamic>(content);`
		
		`var nome = json["nome"].ToString();`

Métodos Disponíveis no contexto da API

Método	**Retorno**	
await ApiContext.GetDataTableAsync (string sql, params DbParameter[] parameters)	DataTable	Recuperar informações dos bancos de dados através de uma query sql. Veja exemplo

await ApiContext.SaveEntityAsync(Guid estruturaformularioid_ASerCriadoOuAtualizado, DataDictionary campos_e_valores)	Guid	Salva as informações de um formulário no bancos de dados. Obs: Para cadastrar valores numéricos, o números devem estar no formato com ponto no separador decimal: 999999.99 sem utilização de virgulas. Caso seja necessário, utilizar o .ToString() para formatar um campo decimal, não use. Utilize a conversão usando o string.Format, como no exemplo abaixo: `string.Format(new System.Globalization.CultureInfo("en-gb"), "{0:F2}", json["valor"]);` Veja exemplo
await ApiContext.ApproveEntityAsync (Guid estruturaformularioid_ASerAprovado, DataDictionary campos_e_valores)	Guid	Avança uma etapa em um formulário com Workflow. As regras de usuário aprovador, permissão de aprovação e responsabilidade por uma etapa devem ser validadas antes desse método ser executado. Obs: Para cadastrar valores numéricos, o números devem estar no formato com ponto no separador decimal: 999999.99 sem utilização de virgulas. Caso seja necessário, utilizar o .ToString() para formatar um campo decimal, não use. Utilize a conversão usando o string.Format, como no exemplo abaixo: `string.Format(new System.Globalization.CultureInfo("en-gb"), "{0:F2}", json["valor"]);` Veja exemplo
await ApiContext.DeleteEntityAsync(Guid formularioId)	void	deletar um formulario
ApiContext.GetEnvironmentVariable(string nomeVariavel) ou **ApiContext.GetEnvironmentVariable_Text**(string nomeVariavel) **	string	Recupera o Valor de uma variável de ambiente. ** Recupera a Descrição (texto), no caso de variáveis do Tipo "Que

await ApiContext.LogAsync(string log, string logTipo)	void	Cria um log na tabela de log geral, o log pode ser consultado entrando na Área de Customização e clicando na opção "Log Geral >> consultar Log" ** Além do Log Padrão da API que o sistema gera, você pode gerar esses Logs customizados para o seu próprio controle.
await ApiContext.DbTransaction_BeginTransactionAsync() **await ApiContext.DbTransaction_CommitAsync()** **await ApiContext.DbTransaction_RollbackAsync()**	void	Para gerenciar a transação de banco de dados manualmente. Leia os detalhes aqui mais abaixo
ApiContext.GetUrlBase()	void	Retorna a URL base da sua aplicação exemplo: https://suaempresa.agilityflow.io/ *A url sempre virá com uma barra no final /

Recuperando dados do banco de dados através de uma query SQL

```
//parametros da query
var paramsQuery = new List<SqlParameter>();
paramsQuery.Add(new SqlParameter("@param1", "xxx"));
paramsQuery.Add(new SqlParameter("@param2", "yyy"));
paramsQuery.Add(new SqlParameter("@param3", "zzz"));

//query usando (nolock) nas tabelas para evitar problema com a transação que está ativa
var sql = "select column1,column2,column3 from table (nolock) where column1 = @param1 or  column2 = @param2 or  column3 = @param3 ";

//executar no banco de dados
var dt = await ApiContext.GetDataTableAsync(sql, paramsQuery.ToArray());
```

```
//percorrendo as linhas de retorno da tabela
foreach (DataRow dr in dt.Rows)
{
  if (dr["column1"] != DBNull.Value){
  var xx = dr["column1"].ToString();
  }
}
```

Salvando os dados recebidos através do método SaveEntityAsync

* Para cadastrar valores numéricos, o números devem estar no formato com ponto no separador decimal: 999999.99 sem utilização de virgulas.

```
public void Execute(){

  //opcional para registrar no log de controle
  await ApiContext.LogAsync("Entrou da API", "log_api_xpto");

        //converte o conteúdo enviado no body da API para JSON
        var json = Newtonsoft.Json.JsonConvert.DeserializeObject<dynamic>(content);

        //guarda em uma variável o ID da  E S T R U T U R A  do formulário de pessoa
        var idEstruturaFormulario_PESSOA = Guid.Parse("0b56b66a-4f6f-4ded-ad04-016d7c0724e1");



        if (json != null)
        {

            //verifica se o JSON recebido contém a propriedade "nome", "email" e "cel" que são necessários p
cadastro
            if (JsonHelper.HasProperty(json, "nome") && JsonHelper.HasProperty(json, "email") &&
JsonHelper.HasProperty(json, "perfil")){

                var values = new DataDictionary();
                values.Add("nome", json["nome"].ToString());
                values.Add("email", json["email"].ToString());

                //no caso do campo do formulário ser uma tabela associativa (tabela relacional N:N)
                var idCampoTabelaAssociativa = "70b6f362-2587-4fd2-a2fb-148bd0caf437";
```

```
                if (json["perfil"].ToString() == "todos_os_perfis" ) {

                  var idPerfilAdmin = "51cf02f1-3787-4dca-8a2c-e219a5ce1298";
                  var idPerfilColaborador = "f999f103-c775-4245-92d3-034cb3ded5e4";
                  var idPerfisConcatenadosComVirgula = idPerfilAdmin + "," + idPerfilColaborador + ",";

                  //adiciona os perfis
                  values.Add(idCampoTabelaAssociativa, idPerfilAdmin); //adiciona perfil de admin
                  values.Add(idCampoTabelaAssociativa, idPerfilColaborador); //adiciona perfil de colaborador
                  values.Add("tabela_associativa_added-
"+idCampoTabelaAssociativa,idPerfisConcatenadosComVirgula);// perfis concatenados com virgula
                  values.Add("tabela_associativa_removed-"+idCampoTabelaAssociativa,"");//em branco, se fosse
pra remover, era só colocar os perfis concatenados com virgula
                  //----------------------------------------

                }
                else if (json["perfil"].ToString() == "apenas_colaborador" ) {


                  var idPerfilColaborador = "f999f103-c775-4245-92d3-034cb3ded5e4";
                  var idPerfilConcatenadoComVirgula = idPerfilColaborador + ",";

                  //adiciona o perfil
                  values.Add(idCampoTabelaAssociativa, idPerfilColaborador); //adiciona perfil de colaborador
                  values.Add("tabela_associativa_added-
"+idCampoTabelaAssociativa,idPerfilConcatenadoComVirgula);// perfis concatenados com virgula
                   values.Add("tabela_associativa_removed-"+idCampoTabelaAssociativa,"");//em branco, se
fosse pra remover, era só colocar os perfis concatenados com virgula
                  //----------------------------------------


                }

                //salva a informação no banco de dados
                await ApiContext.SaveEntityAsync(idEstruturaFormulario_PESSOA, values);

              }

          }
//opcional para registrar no log de controle
```

```
  await ApiContext.LogAsync("Saiu da API", "log_api_xpto");

   }
```

Aprovando uma etapa através do método ApproveEntityAsync

```
public void Execute(){

  //opcional para registrar no log de controle
  await ApiContext.LogAsync("Entrou da API", "log_api_xpto");

        //converte o conteúdo enviado no body da API para JSON
        var json = Newtonsoft.Json.JsonConvert.DeserializeObject<dynamic>(content);

        //guarda em uma variável o ID da  E S T R U T U R A  do formulário de pessoa
        var idEstruturaFormulario_PESSOA = Guid.Parse("0b56b66a-4f6f-4ded-ad04-016d7c0724e1");



        if (json != null)
        {

            //verifica se o JSON recebido contém a propriedade "nome", "email" e "cel" que são necessários p
cadastro
            if (JsonHelper.HasProperty(json, "nome") && JsonHelper.HasProperty(json, "email") &&
JsonHelper.HasProperty(json, "perfil")){

                var values = new DataDictionary();
                values.Add("nome", json["nome"].ToString());
                values.Add("email", json["email"].ToString());
                values.Add("id", json["id"].ToString());

                //no caso do campo do formulário ser uma tabela associativa (tabela relacional N:N)
                var idCampoTabelaAssociativa = "70b6f362-2587-4fd2-a2fb-148bd0caf437";
                if (json["perfil"].ToString() == "todos_os_perfis" ) {

                  var idPerfilAdmin = "51cf02f1-3787-4dca-8a2c-e219a5ce1298";
                  var idPerfilColaborador = "f999f103-c775-4245-92d3-034cb3ded5e4";
                  var idPerfisConcatenadosComVirgula = idPerfilAdmin + "," + idPerfilColaborador + ",";

                  //adiciona os perfis
```

```
                    values.Add(idCampoTabelaAssociativa, idPerfilAdmin); //adiciona perfil de admin
                    values.Add(idCampoTabelaAssociativa, idPerfilColaborador); //adiciona perfil de colaborador
                    values.Add("tabela_associativa_added-
"+idCampoTabelaAssociativa,idPerfisConcatenadosComVirgula);// perfis concatenados com virgula
                    values.Add("tabela_associativa_removed-"+idCampoTabelaAssociativa,"");//em branco, se fosse
pra remover, era só colocar os perfis concatenados com virgula
                    //----------------------------------------


                }
                else if (json["perfil"].ToString() == "apenas_colaborador" ) {



                    var idPerfilColaborador = "f999f103-c775-4245-92d3-034cb3ded5e4";
                    var idPerfilConcatenadoComVirgula = idPerfilColaborador + ",";

                    //adiciona o perfil
                    values.Add(idCampoTabelaAssociativa, idPerfilColaborador); //adiciona perfil de colaborador
                    values.Add("tabela_associativa_added-
"+idCampoTabelaAssociativa,idPerfilConcatenadoComVirgula);// perfis concatenados com virgula
                    values.Add("tabela_associativa_removed-"+idCampoTabelaAssociativa,"");//em branco, se fosse
pra remover, era só colocar os perfis concatenados com virgula
                    //----------------------------------------



                }

                //salva a informação no banco de dados
                await ApiContext.ApproveEntityAsync(idEstruturaFormulario_PESSOA, values);


            }


        }
//opcional para registrar no log de controle
    await ApiContext.LogAsync("saiu da API", "log_api_xpto");


    }
```

Recuperar os valores das "Variáveis de Ambiente"

Para saber mais sobre variáveis de ambiente entre em Variáveis de ambiente

```
var valorDaVariavel = ApiContext.GetEnvironmentVariable("var_nomeDaVariavel");
```

Recuperar a Descrição (texto), no caso de variáveis do Tipo "Query com Id + Descrição":

```
var textDaVariavel = ApiContext.GetEnvironmentVariable_Text("var_nomeDaVariavel");
```

Envio de e-mail

Responsável pelo envio de e-mail

Método	**Retorno**	
EmailSender.SendToGrupoUsuario(Guid grupoUsuarioId, string subject, string htmlContent)	void	Enviar um e-mail para um determinado Grupo de Usuario
EmailSender.SendEmail(string toEmail, string subject, string htmlContent)	void	Enviar um e-mail
EmailSender.SendEmail(string toEmail, string subject, string htmlContent, string ccEmail)	void	Enviar um e-mail, com copia (CC)

Enviar e-mail para um Grupo de Usuário

Antes de iniciar, crie um "Grupo de Usuário" no menu "Segurança e Acesso". Depois de criado, associe os usuários que receberão e-mail ao novo grupo de usuário que você acabou de criar.

Pegue o ID desse novo Grupo de usuário. Você vai precisar dele para enviar o e-mail.

```
var grupoUsuarioId = Guid.Parse("af7c9275-0e23-4b64-a433-f238bb457005"); //substitua o ID "af7c9275-0e23-4b64-a433-f238bb457005" pelo Id do grupo de usuario que você deseja enviar o e-mail
var assunto = "Novo E-mail para um Grupo de usuario ";

var html = "Olá Grupo, <BR><BR> Teste para envio de e-mail para um Grupo de Usuário no AgilityFlow!";

EmailSender.SendToGrupoUsuario(grupoUsuarioId, assunto, html);
```

Enviar e-mail

```
var htmlContent = "<strong>Olá,</strong> teste envio de e-mail.";
var emailTo = "john@email.com";
var subject = "Subject do E-mail";

EmailSender.SendEmail(emailTo, subject, htmlContent);
```

Método principal para utilização da Programação em C# na api de post

A única regra é que o método se chame Execute, não tenha parâmetros de entrada e o retorno seja void.

Exemplo:

```
public void Execute(){



  //aqui você pode programar



}
```

Gerenciar as transações de Banco de dados manualmente

Caso você escolha por gerenciar as transactions de banco de dados manualmente, você precisará obrigatoriamente utilizar as 3 funções abaixo:

> Atenção: Configuração recomendada apenas para especialista.

1. **ApiContext.DbTransaction_BeginTransaction()** Para abrir uma transação, antes de iniciar as alterações no banco de dados
2. **ApiContext.DbTransaction_Commit()** Para finalizar e confirmar com sucesso as alterações no banco de dados
3. **ApiContext.DbTransaction_Rollback()** Para finalizar e desfazer as alterações no banco de dados

Exemplo de uso:

> Atenção: Utilize as transactions sempre dentro de TRY e CATCH

> Podem ser abertas uma ou mais transações na mesma API. Você ficará responsável por todas as aberturas e encerramentos das transações criadas.

```
public async Task RunAsync(){

        //converte o conteúdo enviado no body da API para JSON
        var json = Newtonsoft.Json.JsonConvert.DeserializeObject<dynamic>(content);

        //guarda em uma variável o ID da  E S T R U T U R A  do formulário de pessoa
        var idEstruturaFormulario_PESSOA = Guid.Parse("0b56b66a-4f6f-4ded-ad04-016d7c0724e1");

        try
        {

          await ApiContext.DbTransaction_BeginTransactionAsync(); //ABRE A TRANSAÇÃO, DENTRO DO TRY CATCH

          var values = new DataDictionary();
          values.Add("nome", "exemplo1_PRIMEIRA_transacao");

          await ApiContext.SaveEntityAsync(idEstruturaFormulario_PESSOA, values); //salva a informação no banco de dados

          await ApiContext.DbTransaction_CommitAsync(); //CONFIRMA A TRANSAÇÃO, DENTRO DO TRY CATCH

        }
        catch
        {
          await ApiContext.DbTransaction_RollbackAsync(); //DESFAZ A TRANSAÇÃO, DENTRO DO CATCH em caso de erro
          throw;
        }


  □□□//mais código
```

```
   //...
   //mais código
   //...
    //mais código
   //...
   //mais código
   //...




   try
          {

           await ApiContext.DbTransaction_BeginTransactionAsync(); //ABRE A TRANSAÇÃO, DENTRO DO TRY
CATCH

           var values2 = new DataDictionary();
           values2.Add("nome", "exemplo2_SEGUNDA_transacao");

           await ApiContext.SaveEntityAsync(idEstruturaFormulario_PESSOA, values2); //salva a informação no
banco de dados

           await ApiContext.DbTransaction_CommitAsync(); //CONFIRMA A TRANSAÇÃO, DENTRO DO TRY CATCH

          }
          catch
          {
           await ApiContext.DbTransaction_RollbackAsync(); //DESFAZ A TRANSAÇÃO, DENTRO DO CATCH em
caso de erro
           throw;
          }
   


    }
```

Tabela de dados customizada através de programação: Query SQL, C#, HTML e CSS

Para acessar os detalhes desse conteúdo, clique aqui.