



DOCUMENTAÇÃO TÉCNICA
DE INTEGRAÇÕES

Sumário

Introdução	3
Integrando com o Zuri	3
Integrações de orquestração/saída	4
WebService SOAP	4
WebService REST	5
Procedure	6
Integrações de entrada	8
API REST	8
Autenticação	8
Lista de Processos	9
Lista de itens pendentes	9
Detalhe de item (Etapa Item)	9
Iniciar Processo	9
Exemplo de utilização	10
API Simplificada SOAP	13
Método IniciaProcesso	13
Método IniciaProcessoDetalhado	13
Método CriaItem	13
Método ExecutaPontoEntrada	13
Método ExecutaDecisao	14
Método ExecutarDecisao	14
Método BuscaIdItemProcesso	14
Método BuscaIdPontodeEntrada	14

Introdução

Este documento descreve os principais meios de integração disponíveis no Zuri, como interagir com outros sistemas a partir de processos assim como para permitir que outros sistemas interajam com itens e processos do Zuri.

Este documento não é uma descrição completa de todas as formas de integração e extensibilidade possíveis no Zuri, contendo apenas os modos de integração nativos mais utilizados. O Zuri contém outros meios de integração e diversos pontos de extensibilidade para atender situações mais específicas ou que não sejam contempladas de forma nativa pelo Zuri, mas esses pontos de extensibilidade não são parte do escopo deste documento e serão mencionadas apenas para dar uma dimensão do que é possível estender.

Integrando com o Zuri

Os tipos de integração com o Zuri mais comuns, e portanto o que será abordado neste documento, são:

- **Integrações de orquestração/saída:** Integração de processos do Zuri com outros sistemas, seja para obter ou gerar informações/ações. Essas integrações são realizadas pelo uso de módulos de integração adicionados ao desenho do processo. Esses módulos serão descritos em mais detalhes mais adiante neste documento.
- **Integrações de entrada:** Integração de outros sistemas com processos e itens de processo do Zuri, com o intuito de iniciar novos itens de processo ou dar andamento/tomar decisões em itens de processo existentes. Essas integrações são realizadas por meio de APIs SOAP ou REST disponibilizadas pelo Zuri.

Em casos específicos também é possível interagir de forma mais direta, via código, fazendo uso das DLLs e classes de controle do Zuri. Mas como este tipo de integração é necessário apenas em situações muito específicas e requer conhecimentos de programação e do funcionamento interno do Zuri, portanto não será abordado neste documento.

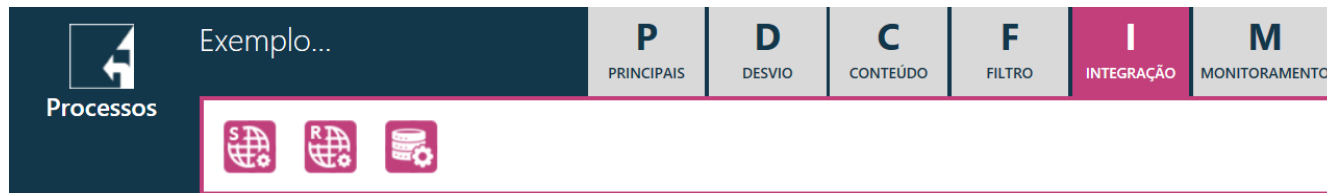
Em caso de dúvidas sobre o melhor tipo de integração para atender às necessidades do negócio, basta entrar em contato com a equipe de suporte da WF.

Integrações de orquestração/saída

Integrações de saída são realizadas pela adição de módulos de integração no desenho do processo. Abaixo temos uma descrição dos módulos de integração disponíveis e suas capacidades.

Todos os módulos a seguir podem ser usados tanto para gerar ações em outros sistemas como para obter dados para serem usados no próprio item de processo que está sendo desenhado.

Os módulos de integração do Zuri se encontram na aba “Integração” da tela de desenho de processo, conforme mostra a figura a seguir:



WebService SOAP



Este módulo deve ser utilizado para acessar web services expostos por protocolo SOAP (em .NET normalmente são urls com extensão .asmx ou web services criados usando WCF, mas outras tecnologias que exportem serviços SOAP também são suportadas).

Na tela de configuração do módulo de WebService SOAP podem ser definidos:

- nome do módulo
- apelido do anexo (necessário apenas quando o web service a ser chamado retorna dados para uso no processo)
- url do web service

Após informar a url do web service, deve-se clicar no botão “testar url” para que a tela de configurações exiba a lista de métodos disponíveis.

Após selecionar um método, será apresentada uma tabela contendo todos os argumentos necessários para sua execução. Esta tabela contém 2 colunas, uma chamada “Execução” e outra chamada “Teste”. Na coluna “execução” deve ser preenchido ou escolhido o valor que será passado para o web service durante a execução do item de processo ao chegar neste módulo. A coluna “teste” deve ser preenchida com valores de validação para que este módulo possa testar o web service.

Parâmetro	Tipo	Execução	Teste
userId	System.Int32	<input type="text" value="NULL"/>	<input type="text" value="NULL"/>
processId	System.Int32	<input type="text" value="NULL"/>	<input type="text" value="NULL"/>

Após configurar os argumentos do método com valores de “Execução” e de “Teste”, deve-se clicar no botão “testar webservice”. **Não é possível concluir a configuração deste módulo sem testar o método.**

Se o método retorna dados que serão usados mais adiante no processo, é necessário marcar o checkbox “Adicionar anexo”. Se este checkbox não for marcado, quaisquer dados retornados pelo método serão ignorados.

Este módulo suporta diretamente apenas métodos com argumentos de tipo primitivo (texto, números, datas). Este módulo não suporta chamar métodos de web services que esperam objetos como argumentos de entrada.

Este módulo não suporta diretamente a execução de web services soap que necessitem de algum tipo de autenticação ativa. Web services com autenticação passiva (Windows Authentication, ip de origem...) não são ativamente implementadas no módulo mas pode ser que funcionem, dependendo de configurações no ambiente.

Em caso de ser necessário chamar web services SOAP com algum tipo de autenticação ou com objetos como argumentos, sugerimos entrar em contato com o suporte da WF para identificarmos a melhor abordagem.

WebService REST



Este módulo deve ser usado para executar web services que tenham sido implementados seguindo padrões REST.

Este módulo suporta transporte de dados nos formatos JSON ou XML, tanto na requisição como na resposta do web service.

Este módulo suporta realizar requisições usando os 4 métodos HTTP padrão: GET, POST, PUT e DELETE.

Este módulo permite definir headers personalizados para as requisições, tornando possível chamar web services que necessitem de header específicos para autenticação, por exemplo.

A tela de configuração a seguir mostra um exemplo de configuração de web service REST usando POST, enviando dados em formato JSON, com valores provenientes de um formulário preenchido anteriormente. Neste exemplo a chamada não retorna dados para o Zuri e não estão sendo adicionados headers personalizados no request.

O ideal é que a configuração deste módulo seja feita por um profissional com conhecimento técnico de REST.

Após a configuração de todos os parâmetros necessários, é necessário clicar no botão “testar serviço”. **Não é possível concluir a configuração deste módulo sem testar o serviço.**

WebService REST

Informações Iniciais
Nome*:

Url*:

Método:

Tipo de conteúdo do Request :

Parâmetros
Novo parâmetro:

adicionar

Nome	Valor	Valor de teste
------	-------	----------------

Conteúdo

```
{  
  name:"%[criarpoc]CRIARPOC/Item/@NOME%",  
  package:"%[criarpoc]CRIARPOC/Item/@PACOTE%",  
  contentLevel:"%[criarpoc]CRIARPOC/Item/@CONTEUDO%"  
}
```

Headers
Header:

adicionar

Nome	Valor	Valor de teste
------	-------	----------------

Resultado
Tipo de resultado :

testar serviço

cancelar

Procedure



Este módulo pode ser utilizado para executar stored procedures existentes em um banco de dados acessível ao Zuri (inclusive no próprio banco de dados do Zuri).

Os pontos abaixo devem ser levados em consideração antes de configurar a stored procedure desejada:

- A string de conexão com o banco de dados onde a procedure se encontra deve estar configurada na sessão “connectionStrings” do web.config do Zuri
- Argumentos de saída em procedures no SQL Server serão ignorados.
- O Zuri suporta a execução de stored procedures em SQL Server e Oracle, mas para conectar com Oracle é necessário primeiro entrar em contato com a equipe de suporte da WF para que configuremos os drivers Zuri/Oracle necessários.
 - Stored procedures Oracle retornam dados de forma diferente do SQL Server. Se houver necessidade de integrar com sistemas em Oracle via stored procedure, favor entrar em contato com a equipe de suporte da WF.

Configurando:

- O campo Nome é o nome do módulo no desenho do processo
- O campo Apelido Anexo só é necessário se a procedure for retornar dados que serão usados mais adiante no processo. Se for este o caso, o checkbox “Adicionar anexo” também deverá estar marcado.
- O campo Conexão permite que seja escolhida uma das strings de conexão existentes no web.config.
- O campo Nome da procedure deve ser preenchido com o nome exato da stored procedure que se deseja executar. Após preencher este campo, deve clicar no botão “carregar a procedure” para que o Zuri verifique se a procedure existe e obtenha sua lista de parâmetros.
 - A configuração dos parâmetros da procedure é semelhante à configuração mencionada acima no web service SOAP, onde a coluna “Execução” contém os valores que serão passados para a procedure durante a execução do item no Zuri e a coluna “Teste” deve ser preenchida com algum valor de teste para que a tela de configuração possa validar a procedure e seu retorno.
- Após preencher os valores dos parâmetros, deve-se clicar no botão “testar a procedure”. **Não é possível concluir a configuração deste módulo sem testar a procedure.**
- O checkbox “Permitir retorno vazio” deve ser preenchido em caso de procedures que retornam dados que não sejam obrigatórios para a execução do processo.

Procedure

Nome

Procedure

Apelido Anexo

Conexão

ConnectionString

Nome da procedure

procedure_exemplo

carregar a procedure

Parâmetro	Execução	Teste
@a	NULL	NULL

testar a procedure

☐ Adicionar anexo

☐ Permitir retorno vazio

Acesso

☐ Usuários ☒ Grupos ☐ Perfis

Selecione um grupo

Nenhum acesso definido

Remover todos

ok

cancelar

Integrações de entrada

Todas as urls de apis mostradas a seguir são urls relativas à url de instalação do Zuri.

Nas definições das urls a seguir, trechos entre colchetes [] representam informações opcionais e valores entre chaves {} representam um dado que deverá ser informado.

API REST

O Zuri tem um conjunto de APIs REST nativas para interagir com processos e seus itens. Estas APIs são utilizadas pela tela de desenho de processo do Zuri e também pelo aplicativo para iOS e versão mobile.

Devido ao fato de ser uma API mais robusta e complexa, esta documentação irá apenas listar os métodos mais relevantes para integrações entre sistemas e explicar alguns conceitos básico do funcionamento. Para de fato usar esta API, recomendamos que seja lido o documento Zuri.MobileAPI.pdf existente no pacote do Zuri. Se não tiver acesso a este documento, basta solicitar à equipe de suporte da WF.

Autenticação

GET /api/auth/

Antes de realizar qualquer operação pela API REST é necessário obter o token de autenticação. O token então deverá ser passado no header “Authorization” como tipo “Bearer” em todos os requests das operações a seguir.

Não é necessário obter um novo token para cada operação executada pelo mesmo usuário, a não ser que o token tenha expirado. A duração padrão do token de autenticação é de 60 minutos e pode ser configurada no web.config do Zuri, na tag “WF.Api.TokenExpirationMinutes” na sessão appSettings.

Para obter o token de autenticação, basta realizar um request GET na url “/api/auth”, com um header “Authorization” em formato basic com as credenciais do usuário desejado. (https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication#Basic_authentication_scheme)

Exemplo em C#, usando WebClient, para um usuário cujo login seja “username” e a senha seja “password”:

```
var c = new WebClient();  
var authRawString = String.Concat("username", ":", "password");  
var authUtf8Bytes = Encoding.UTF8.GetBytes(authRawString);  
var authBase64Encoded = Convert.ToBase64String(authUtf8Bytes);  
c.Headers(HttpRequestHeader.Authorization) = String.Concat("Basic ", authBase64Encoded);  
var token = c.DownloadString("http://url-do-zuri/api/auth");
```

Com este exemplo de código acima, o header gerado ficaria conforme mostrado abaixo:

```
Authorization: Basic dXN1cm5hbWU6cGFzc3dvcmQ=
```

Este request devolve uma string que é o token de acesso para ser usado nas outras operações. Para usar o token, o código será semelhante ao exibido acima, mudando o header Authorization de “Basic” para “Bearer” e passando o token no lugar onde foram passadas as credenciais codificadas no exemplo acima.

Por exemplo, se o token retornado for o texto a seguir, sem as aspas: “!@7!w/KEspvvlr3m3gSA+BZnvnEv4IsMXPPokjRTY7WglHi14HNFhpEll8CSS5WXmov=”, o header das operações deverá ficar assim:

`Authorization: Bearer !@7!w/KEspvvlr3m3gSA+BZnvnEv4IsMXPPokjRTY7WglHi14HNFhpEll8CSS5WXmov=`

Todas as operações a seguir requerem este header de autenticação para funcionar.

Lista de Processos

GET /api/zuri/procs/

Array contendo a lista dos processos que o usuários pode iniciar ou processos onde haja itens na caixa de entrada do usuário.

Lista de itens pendentes

GET /api/zuri/item/inbox[?procid={id-do-processo}]

Array dos itens na caixa de entrada do usuário. Se for informado um id de processo, serão listados apenas os itens deste processo que estejam na caixa de entrada do usuário.

Detalhe de item (Etapa Item)

Os detalhes de um item são obtidos realizando um GET na url presente na propriedade “detailsUrl” do objeto retornado na lista de itens pendentes. O detalhe do item contém informações como id, nome, url de execução, tipo de execução e status e dados de execução.

O tipo de execução é usado para indicar se o item está parado no preenchimento de um formulário ou em uma tomada de decisão.

Se for uma tomada de decisão, o objeto na propriedade execData conterá a mensagem da decisão e a lista de decisões possíveis. Para tomar uma decisão, será necessário realizar um POST para a url em execUrl, informando qual decisão está sendo tomada. Para mais detalhes, consulte o documento Zuri.MobileAPI.pdf.

Se for um formulário, o objeto em execData conterá a definição do formulário (campos e valores atuais). Para preencher um formulário, será necessário realizar um POST para a url em execURL, contendo um objeto com os valores preenchidos de cada campo do formulário. Para mais detalhes, consulte o documento Zuri.MobileAPI.pdf.

Iniciar Processo

POST /api/zuri/procs/{id}/items/

Inicia um novo item do processo identificado pelo {id} na url. Essa url de inicio de processo também pode ser obtida vendo a propriedade “startUrl” do processo desejado retornado pelo método Lista de Processos.

Exemplo de utilização

Segue abaixo um exemplo de uma sequência de requests e respostas usando a API REST e tomando uma decisão.

1 - A aplicação chama a API de autenticação em modo basic para o usuário “usuario”, cuja senha é “usuario”.

Request
GET http://zuri/api/auth/ HTTP/1.1 Authorization: Basic dXN1YXJpbzplc3Vhcmlv
Resposta
HTTP/1.1 200 OK Content-Type: text/plain; charset=utf-8 !@7!IHliBunXzuIuWHjKAiggsefpZxfnC+6l97gbSZPyJ7pZt1SzbFbJcgejrAwRZ8sL

O token recebido é guardado para ser usado no header dos próximos requests.

2 - A aplicação obtém a lista de processos

Request
GET http://zuri/api/zuri/procs/ HTTP/1.1 Authorization: Bearer !@7!IHliBunXzuIuWHjKAiggsefpZxfnC+6l97gbSZPyJ7pZt1SzbFbJcgejrAwRZ8sL
Resposta
HTTP/1.1 200 OK Content-Type: application/json; charset=utf-8 [{ "name": "teste", "inboxCount": 0, "startUrl": "/api/zuri/procs/2/items/", "inboxUrl": "/api/zuri/items/inbox/?procid=2" }, { "name": "teste 2", "inboxCount": 0, "startUrl": "/api/zuri/procs/3/items/", "inboxUrl": "/api/zuri/items/inbox/?procid=3" }]

3 - A aplicação obtém a lista de itens no processo “teste 2”

Request
GET http://zuri/api/zuri/items/inbox/?procid=3 HTTP/1.1 Authorization: Bearer !@7!IHliBunXzuIuWHjKAiggsefpZxfnC+6l97gbSZPyJ7pZt1SzbFbJcgejrAwRZ8sL
Resposta
HTTP/1.1 200 OK Content-Type: application/json; charset=utf-8 <pre>[{ "detailsUrl":"/api/zuri/items/1/2/", "name":"teste 2_1", "id":1, "status":"Inicializado", "step":"Etapa 1", "info":"" }, { "detailsUrl":"/api/zuri/items/2/6/", "name":"teste 2_2", "id":2, "status":"Inicializado", "step":"Etapa 1", "info":"" }, { "detailsUrl":"/api/zuri/items/6/12/", "name":"teste 2_6", "id":6, "status":"Inicializado", "step":"Etapa 1", "info":"" }]</pre>

4 - A aplicação obtém os detalhes do item “teste 2_1” usando a url da propriedade “detailsUrl”.

Request
GET http://zuri/api/zuri/items/1/2/ HTTP/1.1 Authorization: Bearer !@7!IHliBunXzuIuWHjKAiggsefpZxfnC+6l97gbSZPyJ7pZt1SzbFbJcgejrAwRZ8sL
Resposta
HTTP/1.1 200 OK Content-Type: application/json; charset=utf-8 <pre>{ "id":1, "activityId":0, "name":"teste 2_1", "status":"Inicializado", "execUrl":"/api/zuri/items/2/6/exec/", "execMessage":null, "execType":"b4aa1704-c8b5-46ae-8e70-ce00253ee023", "execStatus":0, "execData": { "message":"Aprovar o ítem de teste?", "decisions":[{"id":4,"name":"Aprovar"}, {"id":5,"name":"Reprovar"}] } }</pre>

5 - A aplicação toma a decisão “Reprovar”

Request

```
POST http://zuri/api/zuri/items/1/2/exec/ HTTP/1.1
Authorization: Bearer !@7!IHliBunXzuIuWHjKAiggsefpZxfnC+6l97gbSZPyJ7pZt1SzbFbJcgejrAwRZ8sL
Content-Type: application/json; charset=utf-8
```

```
{
  "id":1,
  "activityId":0,
  "name":"teste 2_1",
  "status":"Inicializado",
  "execUrl":"/api/zuri/items/2/6/exec/",
  "execMessage":null,
  "execType":"b4aa1704-c8b5-46ae-8e70ce00253ee023",
  "execStatus":0,
  "execData":{"id":5,
    "name":"Reprovar"
  }
}
```

Resposta

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
```

```
{
  "id":1,
  "activityId":0,
  "name":"teste 2_1",
  "status":"Reprovado",
  "execUrl":"/api/zuri/items/2/6/exec/",
  "execMessage":null,
  "execType":"","
  "execStatus":1,
  "execData":null
}
```

Se durante o processamento algum dos requests retornar status HTTP 401, significa que o token de autenticação venceu e é necessário gerar um novo token.

API Simplificada SOAP

Além da API REST, o Zuri oferece também uma API SOAP para realizar as operações mais comuns de integração. Para usar a API SOAP do Zuri, a aplicação deverá fazer uma referência à url `"/wfcontent/webservices/zuricontrols.asmx"`. Se for necessário obter apenas o WSDL do serviço para fazer a referência, basta acrescentar `"?WSDL"` a esta url, resultando em `"/wfcontent/webservices/zuricontrols.asmx?WSDL"`.

Para iniciar itens de processo no Zuri será necessário ter no mínimo o id do item a ser iniciado e o id do usuário que irá constar como criador do item. Em cenários de integração pode ser interessante cadastrar um usuário no Zuri para representar cada sistema externo e usar esses usuários para criar os itens e executar outras operações, tornando assim mais fácil identificar os itens criados e operações tomadas de forma automática por sistemas externos.

Segue abaixo uma descrição dos métodos

Método IniciaProcesso

Cria e inicia um item de processo em nome do usuário informado. Retorna um xml com dados da operação e indicando se houve sucesso ou falha.

O id de usuário informado aparecerá como criador do item e executor do início.

Será retornado um xml semelhante ao exibido abaixo com os dados da operação:

```
<ZURICREATE>
  <RESULT
    ExecResult="OK"
    ExecMessage="ITEM CREATED"
    ItemId="40"
    ItemName="Exemplo_40" />
</ZURICREATE>
```

Método IniciaProcessoDetalhado

Funciona da mesma forma do método IniciaProcesso, mas permite definir o nome do item e os parâmetros opcionais 1, 2 e 3.

Retorna um xml no mesmo formato demonstrado anteriormente na descrição do método IniciaProcesso.

Método CriaItem

Apenas cria um novo item de processo em nome do usuário informado, sem inicia-lo. Retorna o id do item criado. Este método deve ser usado quando se deseja iniciar um item a partir de um ponto de entrada ao invés do início.

Método ExecutaPontoEntrada

Executa o item informado a partir de um ponto de entrada. Recebe como parâmetros o id do usuário que constará como executor, id do processo, id do item workflow e id do ponto de entrada desejado.

Retorna um xml no mesmo formato demonstrado anteriormente na descrição do método IniciaProcesso.

Método ExecutaDecisao

Toma uma decisão em uma etapa do item informado. Recebe como parâmetros o id do usuário que constará como executor, id da etapa, id do item workflow e id da decisão que será tomada.

Retorna um xml no mesmo formato demonstrado anteriormente na descrição do método IniciaProcesso.

Método ExecutarDecisao

Semelhante ao método ExecutaDecisao, mas ao invés de receber o id da etapa + id do item workflow, este método recebe o id da “etapa item”.

Toda vez que um item chega em uma etapa, é criado um novo “etapa item”, portanto para usar este método essa informação terá que ser armazenada ou obtida de alguma forma após o item ter chegado na etapa. No método anterior, como está sendo usado o id da etapa, só seria necessário passar um novo id se o desenho do processo for alterado e a etapa em questão for removida. O id do item workflow é definido quando o item é criado e não muda ao longo da sua existência.

Retorna um xml no mesmo formato demonstrado anteriormente na descrição do método IniciaProcesso.

Método BuscaIdItemProcesso

Obtém o id de item processo a partir de um id item workflow e id do processo. Item workflow é uma construção lógica do Zuri, usada para agrupar diversos itens processos relacionados. Um item workflow pode conter vários itens processo de um ou vários processos distintos, mas um item processo faz parte de apenas um item workflow.

Método BuscaIdPontodeEntrada

Obtém o id de um ponto de entrada a partir do id do processo e do nome do ponto de entrada desejado.



Em caso de dúvidas,
entre em contato:

(11) 3474-1790
suporte@solucoes.wf