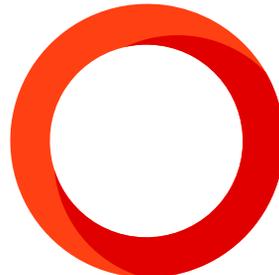
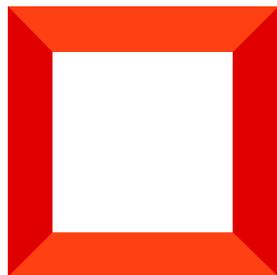
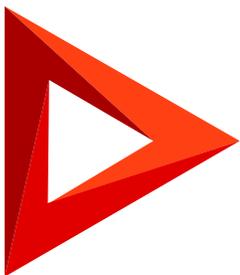


# Back-end development

Custom web services

Version 8.0



This documentation is provided under restrictions on use and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this documentation, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

# Table of Contents

<b>Custom web services</b>	<b>4</b>
Develop a custom web service	5
Call a custom web service	9
Migrate an existing custom web service to .NET Core	10
<b>Develop a custom web service that uses cookie-based authentication</b>	<b>12</b>
1. Create a Source code schema	12
2. Create a service class	13
3. Implement the class method	13
Outcome of the example	15
<b>Develop a custom web service that uses anonymous authentication</b>	<b>16</b>
1. Create a Source code schema	16
2. Create a service class	17
3. Implement the class method	17
4 Register the custom web service that uses anonymous authentication	19
5. Enable both HTTP and HTTPS support for the custom web service that uses anonymous authentication	19
6. Enable all users to access the custom web service that uses anonymous authentication	20
7. Restart Creatio in IIS	21
Outcome of the example	21
<b>Call a custom web service from the front-end</b>	<b>22</b>
1. Create a custom web service	22
2. Create a replacing contact record page	23
3. Add the button to the contact record page	24
Outcome of the example	26
<b>Call a custom web service from Postman</b>	<b>27</b>
1. Create a request collection	27
2. Set up an authentication request	28
3. Execute the authentication request	31
4 Set up the request to the custom web service that uses cookie-based authentication	32
5. Execute the request to the custom web service that uses cookie-based authentication	34
Outcome of the example	34

# Custom web services



A **web service** is software reachable via a unique URL, which enables interaction between applications. The **purpose** of a web service is to integrate Creatio with external applications and systems.

Based on the custom business logic, Creatio will generate and send a request to the web service, receive the response and extract the needed data. Use this data to create or update records in the Creatio database, as well as for custom business logic or automation.

Creatio supports the following web service types:

- **External REST and SOAP services** that you can integrate with low-code tools. Read more in the following user documentation article block: [Web services](#).
- **System web services.**
  - System web services that use cookie-based authentication.
  - System web services that use anonymous authentication.
- **Custom web services.**
  - Custom web services that use cookie-based authentication.
  - Custom web services that use anonymous authentication.

.NET Framework system web services use the [WCF](#) technology and are managed at the IIS level. .NET Core system web services use the [ASP.NET Core Web API](#) technology.

Learn more about the authentication types Creatio provides for web services in a separate article: [Authentication](#). We recommend using authentication based on the OAuth 2.0 open authorization protocol. Learn more about OAuth-based authentication in the user documentation: [Set up OAuth 2.0 authorization for integrated applications](#).

Creatio **system web services that use cookie-based authentication** include:

- `odata` that executes OData 4 external application requests to the Creatio database server. Learn more about using the OData 4 protocol in Creatio in a separate article: [OData](#).
- `EntityDataService.svc` that executes OData 3 external application requests to the Creatio database server. Learn more about using the OData 3 protocol in Creatio in a separate article: [OData](#).
- `ProcessEngineService.svc` that enables external applications to run Creatio business processes. Learn more about the web service in a separate article: [Business process service](#).

Creatio **system web services that use anonymous authentication** include:

- `AuthService.svc` that executes Creatio authentication requests. Learn more about the web service in a separate article: [Authentication](#).

This article covers custom web services. Learn more about system web services in a separate guide: [Integrations & API](#).

## Develop a custom web service

A **custom web service** is a RESTful service that uses the WCF (for .NET Framework) or ASP .NET Core (for .NET Core) technology. Unlike system web services, custom web services let you solve unique integration problems.

The web service development procedure differs for each Creatio deployment framework. View the unique features of the custom web service development for the .NET Framework and .NET Core frameworks below.

### Develop a custom web service that uses cookie-based authentication

1. Create a [ *Source code* ] schema. Learn more about creating schemas in a separate article: [Develop configuration elements](#).
2. Create a service class.
  - a. Add the `Terrasoft.Configuration` namespace or any of its nested namespaces in the Schema Designer. You can name the namespace arbitrarily.
  - b. Add the namespaces the data types of which to utilize in the class using the `using` directive.
  - c. Use the `Terrasoft.Web.Http.Abstractions` namespace if you want the custom web service to support both .NET Framework and .NET Core. If you develop the web service using the `System.Web` namespace and have to run it on .NET Core, [adapt the web service](#).
  - d. Add the class name that matches the schema name (the [ *Code* ] property).
  - e. Specify the `Terrasoft.Nui.ServiceModel.WebService.BaseService` class as a parent class.
  - f. Add the [ `ServiceContract` ] and [ `AspNetCompatibilityRequirement` ] class attributes that contain the needed parameters. Learn more about the [ `ServiceContract` ] attribute in the official [Microsoft documentation](#). Learn more about the [ `AspNetCompatibilityRequirements` ] attribute in the official [Microsoft documentation](#).
3. Implement the class methods that correspond to the web service endpoints.
 

Add the [ `OperationContract` ] and [ `WebInvoke` ] method attributes that contain the needed parameters. Learn more about the [ `OperationContract` ] attribute in the official [Microsoft documentation](#). Learn more about the [ `WebInvoke` ] attribute in the official [Microsoft documentation](#).
4. Implement additional classes whose instances will receive or return the web service methods (optional). Required to pass data of complex types. For example, object instances, collections, arrays, etc.
 

Add the [ `DataContract` ] attribute to the class and the [ `DataMember` ] attribute to the class fields. Learn more about the [ `DataContract` ] attribute in the official [Microsoft documentation](#). Learn more about the [ `DataMember` ] attribute in the official [Microsoft documentation](#).
5. Publish the source code schema.

As a result, you will be able to call the custom web service that uses cookie-based authentication from the source code of configuration schemas, as well as from external applications.

### Develop a custom web service that uses anonymous authentication

**Custom web services that use anonymous authentication** do not require the user to pre-authenticate, i. e., you can use them anonymously.

**Attention.** We do not recommend using anonymous authentication in custom web services. It is insecure and can hurt performance.

Develop a custom web service that uses anonymous authentication for .NET Framework

1. Take steps 1-5 in the [Develop a custom web service that uses cookie-based authentication](#) instruction.
2. Add the `SystemUserConnection` system connection when creating a service class.
3. Specify the user on whose behalf to process the HTTP request when creating a class method. To do this, call the `SessionHelper.SpecifyWebOperationIdentity` method of the `Terrasoft.Web.Common` namespace after retrieving `SystemUserConnection`. This method enables business processes to manage the database entity ( `Entity` ) from the custom web service that uses anonymous authentication.

```
Terrasoft.Web.Common.SessionHelper.SpecifyWebOperationIdentity(HttpContextAccessor.GetInstance
```

4. Register the custom web service that uses anonymous authentication:
  - a. Go to the `..\Terrasoft.WebApp\ServiceModel` directory.
  - b. Create an \*.svc file whose name matches the web service name. Add the following record to the file.

Registration template for a custom web service that uses anonymous authentication

```
<% @ServiceHost
  Service = "Service, ServiceNamespace"
  Factory = "Factory, FactoryNamespace"
  Debug = "Debug"
  Language = "Language"
  CodeBehind = "CodeBehind"
%>
```

Example that registers a custom web service that uses anonymous authentication

```
<% @ServiceHost
  Service = "Terrasoft.Configuration.UsrAnonymousConfigurationServiceNamespace.UsrAnonymous
  Debug = "true"
  Language = "C#"
%>
```

The `Service` attribute must contain the full name of the web service class and specify the namespace.

Learn more about the `@ServiceHost` WCF directive in the official [Microsoft documentation](#).

5. Enable both HTTP and HTTPS support for the custom web service that uses anonymous authentication:
  - a. Open the `..\Terrasoft.WebApp\ServiceModel\http\services.config` file and add the following record to it.

**Example of changes to the `..\Terrasoft.WebApp\ServiceModel\http\services.config` file**

```
<services>
  ...
  <service name="Terrasoft.Configuration.[Custom namespace].[Service name]">
    <endpoint name="[Service name]EndPoint"
      address=""
      binding="webHttpBinding"
      behaviorConfiguration="RestServiceBehavior"
      bindingNamespace="http://Terrasoft.WebApp.ServiceModel"
      contract="Terrasoft.Configuration.[Custom namespace].[Service name]" />
  </service>
</services>
```

The `<services>` element contains the list of Creatio web service configurations (the `<service>` nested elements).

The `name` attribute contains the name of the type (class or interface) that implements the web service contract.

The `<endpoint>` nested element contains the address, binding, and interface that defines the contract of the web service specified in the `name` attribute of the `<service>` element.

Learn more about the web service configuration elements in the official [Microsoft documentation](#).

- b. Add an identical record to the `..\Terrasoft.WebApp\ServiceModel\https\services.config` file.
6. Enable all users to access the custom web service that uses anonymous authentication:
  - a. Open the `..\Terrasoft.WebApp\Web.config` file.
  - b. Add the `<location>` element that defines the relative path and access permissions to the web service.

**Example of changes to the `..\Terrasoft.WebApp\Web.config` file**

```
<configuration>
  ...
  <location path="ServiceModel/[Service name].svc">
    <system.web>
      <authorization>
        <allow users="*" />
      </authorization>
    </system.web>
  </location>
  ...
</configuration>
```

- c. Add the relative web service path to the `value` attribute of the `AllowedLocations` key in the `<appSettings>` element.

**Example of changes to the `..\Terrasoft.WebApp\Web.config` file**

```
<configuration>
  ...
  <appSettings>
    ...
    <add key="AllowedLocations" value="[Previous values];ServiceModel/[Service name].svc" />
    ...
  </appSettings>
  ...
</configuration>
```

7. Restart Creatio in IIS.

As a result, you will be able to call the custom web service that uses anonymous authentication from the source code of configuration schemas, as well as from external applications. You can access the web service both with and without pre-authentication.

Develop a custom web service that uses anonymous authentication for .NET Core

1. Take steps 1-5 in the [Develop a custom web service that uses cookie-based authentication](#) instruction.
2. Enable all users to access the custom web service that uses anonymous authentication:

**Example of changes to the `..\Terrasoft.WebHost\appsettings.json` file**

```
"Terrasoft.Configuration.[Service name]": [
  "/ServiceModel/[Service name].svc"
]
```

- a. Open the `..\Terrasoft.WebHost\appsettings.json` configuration file.
  - b. Add the web service details to the `AnonymousRoutes` file block.
3. Restart Creatio.

As a result, you will be able to call the custom web service that uses anonymous authentication from the source code of configuration schemas, as well as from external applications. You can access the service both with or without pre-authentication.

**Attention.** Reconfigure the web service after updating Creatio. The existing configuration files are overwritten as part of the update.

## Call a custom web service

You can call a custom web service from the browser or front-end.

### Call a custom web service from the browser

Call a custom web service that uses cookie-based authentication from the browser

To call a **.NET Framework** custom web service that uses cookie-based authentication from the browser:

1. Retrieve the authentication cookies using the `AuthService.svc` system web service.
2. Call a custom web service using the address bar:

URL template for a custom web service that uses cookie-based authentication

```
[Creatio application URL]/0/rest/[Custom web service name]/[Custom web service endpoint]?[Opt
```

URL example of a custom web service that uses cookie-based authentication

```
http://mycreatio.com/0/rest/UsrCustomConfigurationService/GetContactIdByName?Name=User1
```

The procedure to call a **.NET Core** custom web service that uses cookie-based authentication is identical. That said, the `/0` prefix is not required.

Call a custom web service that uses anonymous authentication from the browser

Call a **.NET Framework** custom web service that uses anonymous authentication using the address bar:

URL template for a custom web service that uses anonymous authentication

```
[Creatio application URL]/0/ServiceModel/[Custom web service name]/[Custom web service endpoint]
```

URL example of a custom web service that uses anonymous authentication

```
http://mycreatio.com/0/ServiceModel/UsrCustomConfigurationService.svc/GetContactIdByName?Name=Us
```

The procedure to call a **.NET Core** custom web service that uses anonymous authentication is identical. That said, the `/0` prefix is not required.

## Call a custom web service from the front-end

1. Add the `ServiceHelper` module as a dependency to the module of the page from which to call the service. This module provides a convenient interface to execute server requests via the `Terrasoft.AjaxProvider` request provider implemented in the client core.
2. Call a custom web service from the `ServiceHelper` module.

You can call a custom web service in **several ways**:

- Call the `callService(serviceName, serviceName, callback, serviceData, scope)` method.
- Call the `callService(config)` method, where `config` is a configuration object that has the following properties:

`serviceName` is the name of the custom web service.

`methodName` is the name of the custom web service method to call.

`callback` is the callback function that processes the web service response.

`data` is the object that contains the initialized incoming parameters for the service method.

`scope` is the context of the request execution.

**Attention.** The `ServiceHelper` module supports only `POST` requests. As such, add the `[WebInvoke]` attribute that contains the `Method = "POST"` parameter to the custom web service method.

## Migrate an existing custom web service to .NET Core

You can migrate a .NET Framework custom web service that retrieves the context without inheriting the `Terrasoft.Web.Common.BaseService` base class to .NET Core. To do this, **adapt the custom web service**.

The `HttpContextAccessor` property of the `Terrasoft.Web.Common.BaseService` provides unified access to context (`HttpContext`) both in .NET Framework and .NET Core. The `UserConnection` and `AppConnection` properties let you retrieve the user connection object and the connection object on the application level. This lets you omit the `HttpContext.Current` property of the `System.Web` library.

**Example that uses the properties of the `Terrasoft.Web.Common.BaseService` parent class.**

```
namespace Terrasoft.Configuration.UsrCustomNamespace
{
    using Terrasoft.Web.Common;

    [ServiceContract]
    [AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.RequirementsMode.None)]
    public class UsrCustomConfigurationService: BaseService
    {
        /* The web service method. */
        [OperationContract]
        [WebInvoke(Method = "GET", RequestFormat = WebMessageFormat.Json, BodyStyle = WebMessageFormat.Json)]
    }
}
```

```

    ResponseFormat = WebMessageFormat.Json)]
    public void SomeMethod() {
        ...
        /* UserConnection is the BaseService property. */
        var currentUser = UserConnection.CurrentUser;
        /* AppConnection is the BaseService property. */
        var sdkHelpUrl = AppConnection.SdkHelpUrl;
        /* HttpContextAccessor is the BaseService property. */
        var httpContext = HttpContextAccessor.GetInstance();
        ...
    }
}
}
}

```

Creatio supports the following **context retrieval options** for web services developed without inheriting the `Terrasoft.Web.Common.BaseService` class:

- via `IHttpContextAccessor` registered in DI (`ClassFactory`)

This option lets you view the explicit class dependencies for thorough automated testing and debugging. Learn more about using the class factory in a separate article: [Replace configuration elements](#).

- via the `HttpContext.Current` static property

Add the `Terrasoft.Web.Http.Abstractions` namespace to the source code using the `using` directive. The `HttpContext.Current` static property implements unified access to `HttpContext`. To adapt the web service code to .NET Core, replace the `System.Web` namespace using `Terrasoft.Web.Http.Abstractions`.

**Attention.** Do not use specific access implementations to request context peculiar to .NET Framework (the `System.Web` library) or .NET Core (the `Microsoft.AspNetCore.Http` library) in the configuration.

### Example that adapts the web service to .NET Core

```

namespace Terrasoft.Configuration.UsrCustomNamespace
{
    /* Use instead of System.Web. */
    using Terrasoft.Web.Http.Abstractions;

    [ServiceContract]
    [AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.RequirementsMode.None)]
    public class UsrCustomConfigurationService
    {
        /* The web service method. */
        [OperationContract]
        [WebInvoke(Method = "GET", RequestFormat = WebMessageFormat.Json, BodyStyle = WebMessageFormat.Json)]
        public void SomeMethod() {

```

```

    ...
    var httpContext = HttpContext.Current;
    ...
  }
}
}

```

# Develop a custom web service that uses cookie-based authentication

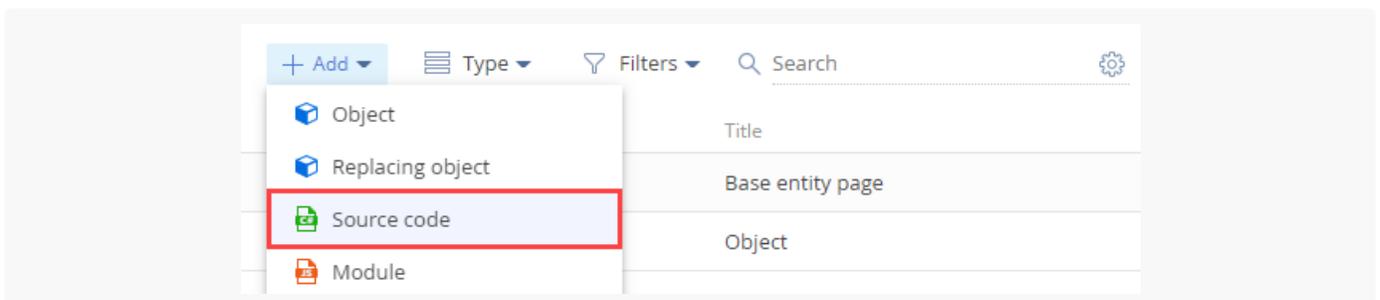
 Medium

**Example.** Create a custom web service that uses cookie-based authentication. The service must execute a Creatio request to return the contact information by the specified name. Creatio must return the following data:

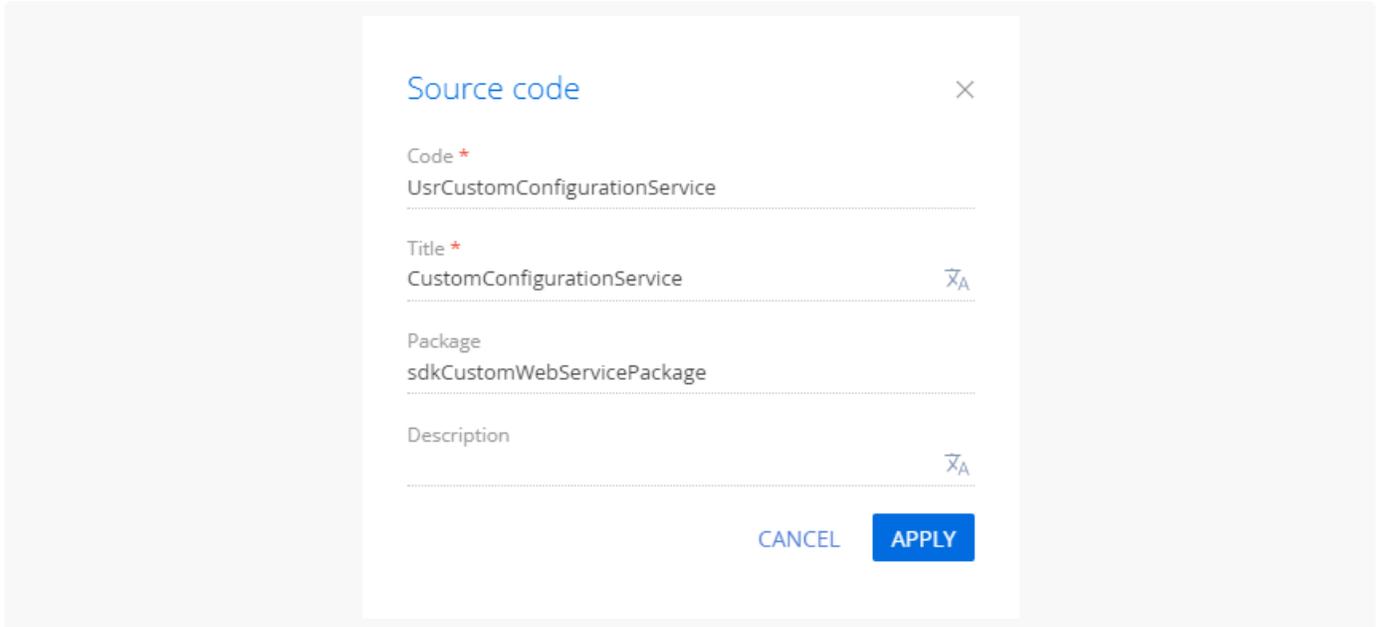
- If the contact is found, return the contact ID.
- If several contacts are found, return the ID of the first contact only.
- If no contacts are found, return an empty string.

## 1. Create a [ *Source code* ] schema

1. [Go to the \[ \*Configuration\* \] section](#) and select a custom [package](#) to add the schema.
2. Click [ *Add* ] → [ *Source code* ] on the section list toolbar.



3. Go to the Schema Designer and fill out the schema properties:
  - Set [ *Code* ] to "UsrCustomConfigurationService."
  - Set [ *Title* ] to "CustomConfigurationService."



Click [ *Apply* ] to apply the properties.

## 2. Create a service class

1. Go to the Schema Designer and add the namespace nested into `Terrasoft.Configuration`. You can use an arbitrary name. For example, `UsrCustomConfigurationServiceNamespace`.
2. Add the namespaces the data types of which to utilize in the class using the `using` directive.
3. Add a class name that matches the schema name (the [ *Code* ] property).
4. Specify the `Terrasoft.Nui.ServiceModel.WebService.BaseService` class as a parent class.
5. Add the `[ServiceContract]` and `[AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Required)]` attributes to the class.

## 3. Implement the class method

Go to the Schema Designer and add the `public string GetContactIdByName(string Name)` class method that implements the endpoint of the custom web service. The method executes database queries using `EntitySchemaQuery`. Depending on the value of the `Name` parameter in the query string, the response body will contain:

- The ID of the contact (string type) if the contact is found.
- The ID of the first found contact (string type) if several contacts are found.
- The empty string if no contacts are found.

View the source code of the `UsrCustomConfigurationService` custom web service below.

```
UsrCustomConfigurationService
```

```

namespace Terrasoft.Configuration.UsrCustomConfigurationServiceNamespace
{
    using System;
    using System.ServiceModel;
    using System.ServiceModel.Web;
    using System.ServiceModel.Activation;
    using Terrasoft.Core;
    using Terrasoft.Web.Common;
    using Terrasoft.Core.Entities;

    [ServiceContract]
    [AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.RequirementsMode)]
    public class UsrCustomConfigurationService: BaseService
    {

        /* The method that returns the contact ID by the contact name. */
        [OperationContract]
        [WebInvoke(Method = "GET", RequestFormat = WebMessageFormat.Json, BodyStyle = WebMessageFormat.Json, ResponseFormat = WebMessageFormat.Json)]
        public string GetContactIdByName(string Name) {
            /* The default result. */
            var result = "";
            /* The EntitySchemaQuery instance that accesses the Contact database table. */
            var esq = new EntitySchemaQuery(UserConnection.EntitySchemaManager, "Contact");
            /* Add columns to the query. */
            var colId = esq.AddColumn("Id");
            var colName = esq.AddColumn("Name");
            /* Filter the query data. */
            var esqFilter = esq.CreateFilterWithParameters(FilterComparisonType.Equal, "Name", Name);
            esq.Filters.Add(esqFilter);
            /* Retrieve the query results. */
            var entities = esq.GetEntityCollection(UserConnection);
            /* If the service receives data. */
            if (entities.Count > 0)
            {
                /* Return the "Id" column value of the first query result record. */
                result = entities[0].GetColumnValue(colId.Name).ToString();
                /* You can also use this option:
                result = entities[0].GetTypedColumnValue<string>(colId.Name); */
            }
            // Return the results.
            return result;
        }
    }
}

```

Click [ *Save* ] then [ *Publish* ] on the Designer's toolbar.

## Outcome of the example

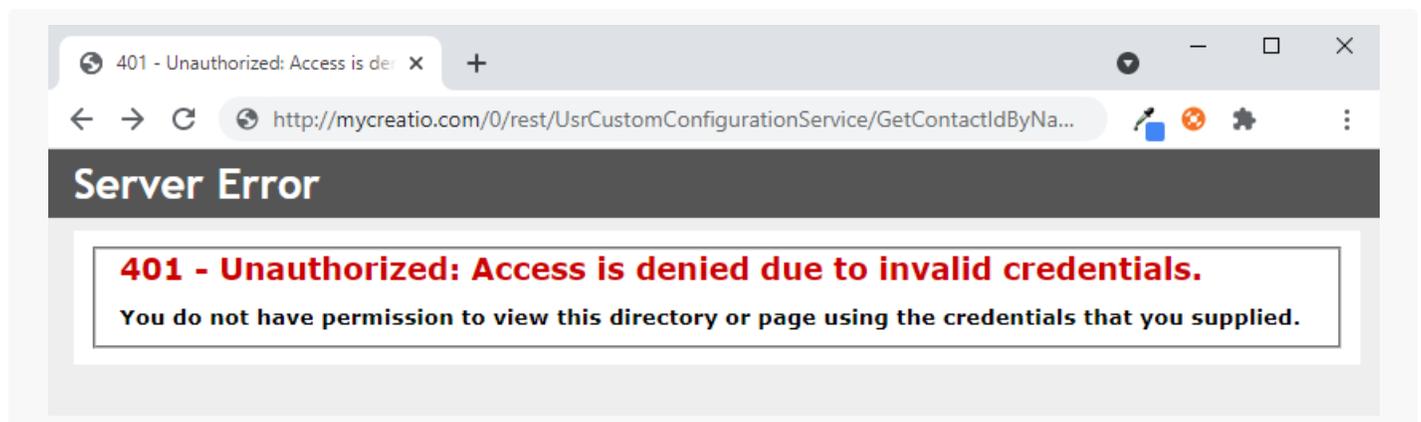
As a result, Creatio will add the custom `UsrCustomConfigurationService` REST web service that has the `GetContactIdByName` endpoint.

Access the `GetContactIdByName` endpoint of the web service from the browser and pass the contact name in the `Name` parameter.

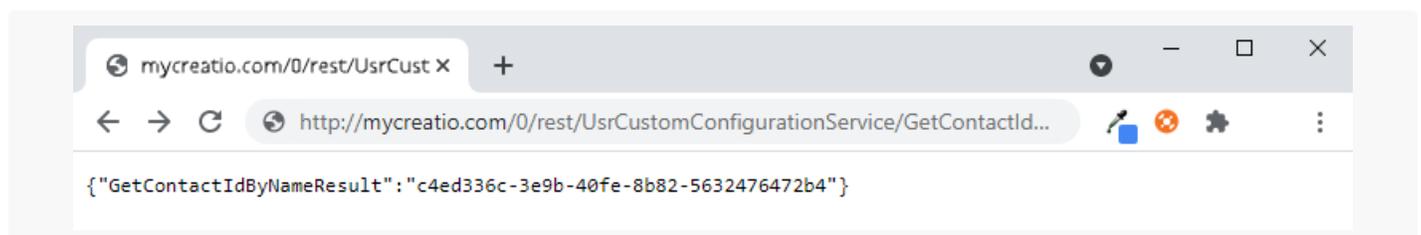
### Request string that contains the name of the existing contact

```
http://mycreatio.com/0/rest/UsrCustomConfigurationService/GetContactIdByName?Name=Andrew%20Baker
```

If you access the web service without preauthorization, an error will occur.



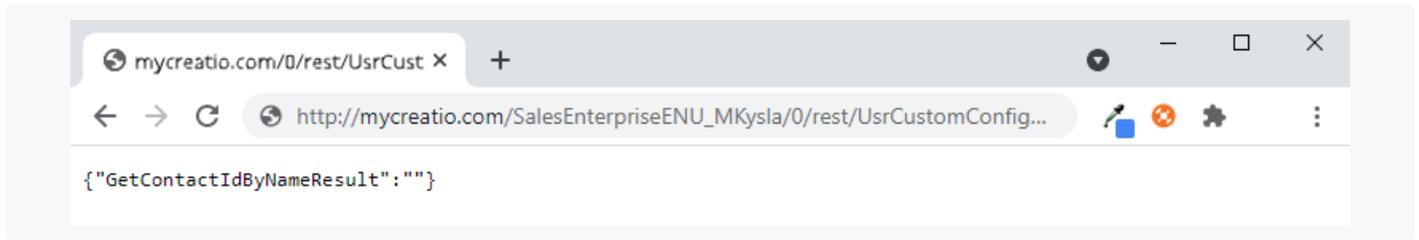
Log in to Creatio and execute the request once more. If Creatio finds the contact from the `Name` parameter in the database, the `GetContactIdByNameResult` property will return the contact ID value.



If Creatio finds no contacts from the `Name` parameter in the database, the `GetContactIdByNameResult` property will return an empty string.

### Request string that contains the name of a non-existing contact

```
http://mycreatio.com/0/rest/UsrCustomConfigurationService/GetContactIdByName?Name=Andrew%20Bake
```



# Develop a custom web service that uses anonymous authentication

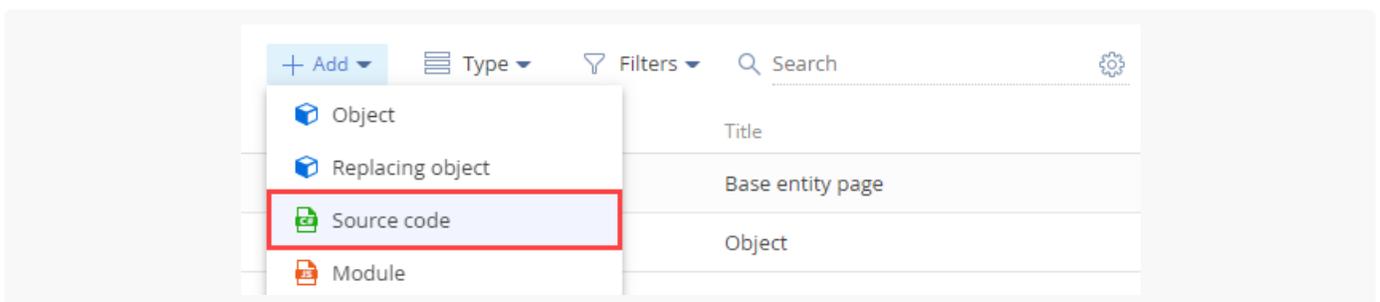
 Medium

**Example.** Create a custom web service that uses anonymous authentication. The service must execute a Creatio request to return the contact information by the specified name. Creatio must return the following data:

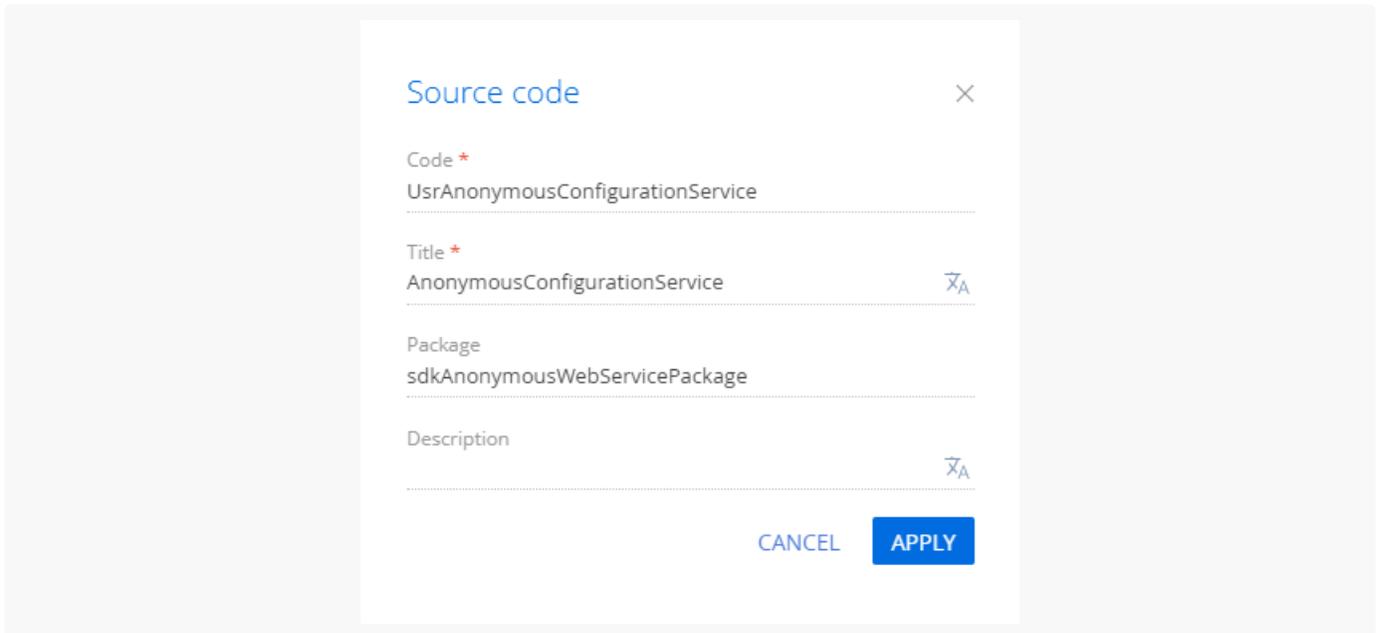
- If the contact is found, return the contact ID.
- If several contacts are found, return the ID of the first contact only.
- If no contacts are found, return an empty string.

## 1. Create a [ *Source code* ] schema

1. [Go to the \[ Configuration \] section](#) and select a custom [package](#) to add the schema.
2. Click [ Add ] → [ *Source code* ] on the section list toolbar.



3. Go to the Schema Designer and fill out the schema properties:
  - Set [ *Code* ] to "UsrAnonymousConfigurationService."
  - Set [ *Title* ] to "AnonymousConfigurationService."



Click [ *Apply* ] to apply the properties.

## 2. Create a service class

1. Go to the Schema Designer and add the namespace nested into `Terrasoft.Configuration`. You can use an arbitrary name. For example, `UsrAnonymousConfigurationServiceNamespace`.
2. Add the namespaces the data types of which to utilize in the class using the `using` directive.
3. Add the class name that matches the schema name (the [ *Code* ] property).
4. Specify the `Terrasoft.Nui.ServiceModel.WebService.BaseService` class as a parent class.
5. Add the `[ServiceContract]` and `[AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Required)]` attributes to the class.
6. Add the `SystemUserConnection` system connection to enable anonymous access to the custom web service.

## 3. Implement the class method

Go to the Schema Designer and add the `public string GetContactIdByName(string Name)` class method that implements the endpoint of the custom web service. The method executes database queries using `EntitySchemaQuery`. Depending on the value of the `Name` parameter in the query string, the response body will contain:

- The ID of the contact (string type) if the contact is found.
- The ID of the first found contact (string type) if Creatio several contacts are found.
- The empty string if Creatio no contacts are found.

Specify the user on whose behalf to process the HTTP request. To do this, call the `SessionHelper.SpecifyWebOperationIdentity` method of the `Terrasoft.Web.Common` namespace after retrieving

`SystemUserConnection`. This method enables business processes to manage the database entity ( `Entity` ) from the custom web service that uses anonymous authentication.

```
Terrasoft.Web.Common.SessionHelper.SpecifyWebOperationIdentity(HttpContextAccessor.GetInstance())
```

View the source code of the `UsrAnonymousConfigurationService` custom web service below.

#### `UsrAnonymousConfigurationService`

```
/* The custom namespace. */
namespace Terrasoft.Configuration.UsrAnonymousConfigurationServiceNamespace
{
    using System;
    using System.ServiceModel;
    using System.ServiceModel.Web;
    using System.ServiceModel.Activation;
    using Terrasoft.Core;
    using Terrasoft.Web.Common;
    using Terrasoft.Core.Entities;

    [ServiceContract]
    [AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.RequirementsMode.None)]
    public class UsrAnonymousConfigurationService: BaseService
    {
        /* The link to the UserConnection instance required to access the database. */
        private SystemUserConnection _systemUserConnection;
        private SystemUserConnection SystemUserConnection {
            get {
                return _systemUserConnection ?? (_systemUserConnection = (SystemUserConnection)A
            }
        }

        /* The method that returns the contact ID by the contact name. */
        [OperationContract]
        [WebInvoke(Method = "GET", RequestFormat = WebMessageFormat.Json, BodyStyle = WebMessageFormat.Json,
            ResponseFormat = WebMessageFormat.Json)]
        public string GetContactIdByName(string Name){
            /* Specify the user on whose behalf to process the HTTP request. */
            SessionHelper.SpecifyWebOperationIdentity(HttpContextAccessor.GetInstance()), SystemUserConnection
            /* The default result. */
            var result = "";
            /* The EntitySchemaQuery instance that accesses the Contact database table. */
            var esq = new EntitySchemaQuery(SystemUserConnection.EntitySchemaManager, "Contact")
            /* Add columns to the query. */
            var colId = esq.AddColumn("Id");
            var colName = esq.AddColumn("Name");
```

```

/* Filter the query data. */
var esqFilter = esq.CreateFilterWithParameters(FilterComparisonType.Equal, "Name", M
esq.Filters.Add(esqFilter);
/* Retrieve the query results. */
var entities = esq.GetEntityCollection(SystemUserConnection);
/* If the service receives data. */
if (entities.Count > 0)
{
    /* Return the "Id" column value of the first query result record. */
    result = entities[0].GetColumnValue(colId.Name).ToString();
    /* You can also use this option:
    result = entities[0].GetTypedColumnValue<string>(colId.Name); */
}
/* Return the results. */
return result;
}
}
}
}

```

Click [ *Save* ] then [ *Publish* ] on the Designer's toolbar.

## 4 Register the custom web service that uses anonymous authentication

1. Go to the `..\Terrasoft.WebApp\ServiceModel` directory.
2. Create a `UsrAnonymousConfigurationService.svc` file and add the following record to it.

```

<% @ServiceHost
    Service = "Terrasoft.Configuration.UsrAnonymousConfigurationServiceNamespace.UsrAnonymous
    Debug = "true"
    Language = "C#"
%>

```

The `service` attribute contains the full name of the web service class and specifies the namespace.

## 5. Enable both HTTP and HTTPS support for the custom web service that uses anonymous authentication

1. Open the `..\Terrasoft.WebApp\ServiceModel\http\services.config` file and add the following record to it.

```

..\Terrasoft.WebApp\ServiceModel\http\services.config file

```

```

<services>

```

```

...
<service name="Terrasoft.Configuration.UsrAnonymousConfigurationServiceNamespace.UsrAnony
  <endpoint name="[Service name]EndPoint"
    address=""
    binding="webHttpBinding"
    behaviorConfiguration="RestServiceBehavior"
    bindingNamespace="http://Terrasoft.WebApp.ServiceModel"
    contract="Terrasoft.Configuration.UsrAnonymousConfigurationServiceNamespace.UsrAn
  </service>
</services>

```

2. Add an identical record to the `..\Terrasoft.WebApp\ServiceModel\https\services.config` file.

## 6. Enable all users to access the custom web service that uses anonymous authentication

1. Open the `..\Terrasoft.WebApp\Web.config` file.
2. Add the `<location>` element that defines the relative path and access permissions to the web service.

`..\Terrasoft.WebApp\Web.config` file

```

<configuration>
...
<location path="ServiceModel/UsrAnonymousConfigurationService.svc">
  <system.web>
    <authorization>
      <allow users="*" />
    </authorization>
  </system.web>
</location>
...
</configuration>

```

3. Add the relative web service path to the `value` attribute of the `AllowedLocations` key in the `<appSettings>` element.

`..\Terrasoft.WebApp\Web.config` file

```

<configuration>
...
<appSettings>
...
  <add key="AllowedLocations" value="[Previous values];ServiceModel/UsrAnonymousConfigu
...
</appSettings>

```

```
...
</configuration>
```

## 7. Restart Creatio in IIS

Restart Creatio in IIS to apply the changes.

## Outcome of the example

As a result, Creatio will add the custom `UsrAnonymousConfigurationService` REST web service that has the `GetContactIdByName` endpoint. You can access the web service from the browser, with or without pre-authentication.

Access the `GetContactIdByName` endpoint of the web service from the browser and pass the contact name in the `Name` parameter.

### Request string that contains the name of the existing contact

```
http://mycreatio.com/0/ServiceModel/UsrAnonymousConfigurationService/GetContactIdByName?Name=Anc
```

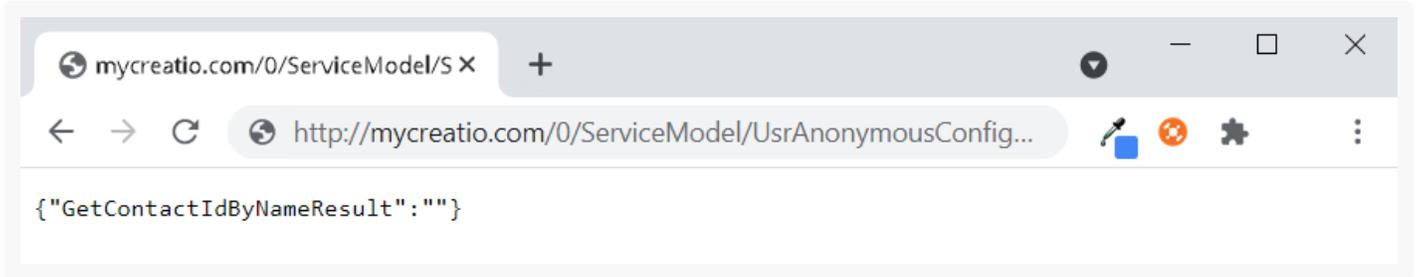
If Creatio finds the contact from the `Name` parameter in the database, the `GetContactIdByNameResult` property will return the contact ID value.



If Creatio finds no contacts from the `Name` parameter in the database, the `GetContactIdByNameResult` property will return an empty string.

### Request string that contains the name of a non-existing contact

```
http://mycreatio.com/0/ServiceModel/UsrAnonymousConfigurationService/GetContactIdByName?Name=Anc
```



# Call a custom web service from the front-end



**Example.** Add a button that calls a custom web service to the contact add page. Display the response returned by the web service in a dialog box.

## 1. Create a custom web service

This example uses the `UsrCustomConfigurationService` custom web service. Learn more about developing the service in a separate article: [Develop a custom web service that uses cookie-based authentication](#).

Change the `Method` parameter of the `WebInvoke` attribute in the `UsrCustomConfigurationService` custom web service to `POST`.

View the source code of the custom web service the example uses below.

### UsrCustomConfigurationService

```
namespace Terrasoft.Configuration.UsrCustomConfigurationServiceNamespace
{
    using System;
    using System.ServiceModel;
    using System.ServiceModel.Web;
    using System.ServiceModel.Activation;
    using Terrasoft.Core;
    using Terrasoft.Web.Common;
    using Terrasoft.Core.Entities;

    [ServiceContract]
    [AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Requ
    public class UsrCustomConfigurationService: BaseService
    {

        /* The method that returns the contact ID by the contact name. */
        [OperationContract]
```

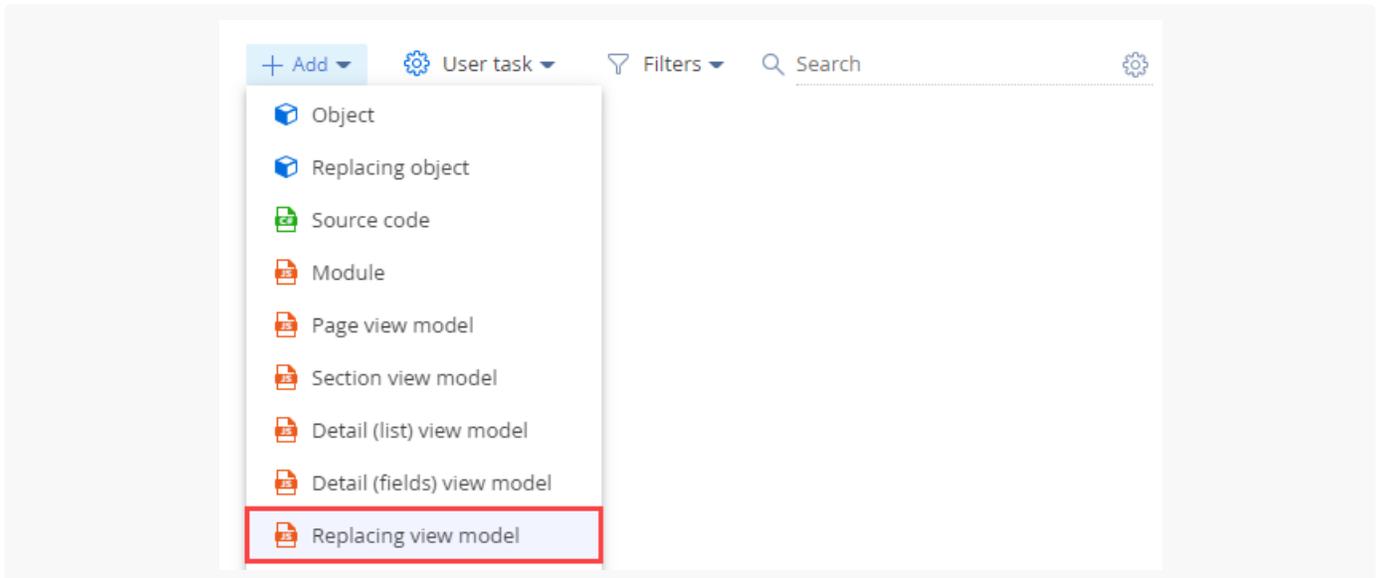
```

[WebInvoke(Method = "POST", RequestFormat = WebMessageFormat.Json, BodyStyle = WebMessageFormat.Json)]
public string GetContactIdByName(string Name) {
    /* The default result. */
    var result = "";
    /* The EntitySchemaQuery instance that accesses the Contact database table. */
    var esq = new EntitySchemaQuery(UserConnection.EntitySchemaManager, "Contact");
    /* Add columns to the query. */
    var colId = esq.AddColumn("Id");
    var colName = esq.AddColumn("Name");
    /* Filter the query data. */
    var esqFilter = esq.CreateFilterWithParameters(FilterComparisonType.Equal, "Name", Name);
    esq.Filters.Add(esqFilter);
    /* Retrieve the query results. */
    var entities = esq.GetEntityCollection(UserConnection);
    /* If the service receives data. */
    if (entities.Count > 0)
    {
        /* Return the "Id" column value of the first query result record. */
        result = entities[0].GetColumnValue(colId.Name).ToString();
        /* You can also use this option:
        result = entities[0].GetTypedColumnValue<string>(colId.Name); */
    }
    /* Return the results. */
    return result;
}
}
}
}

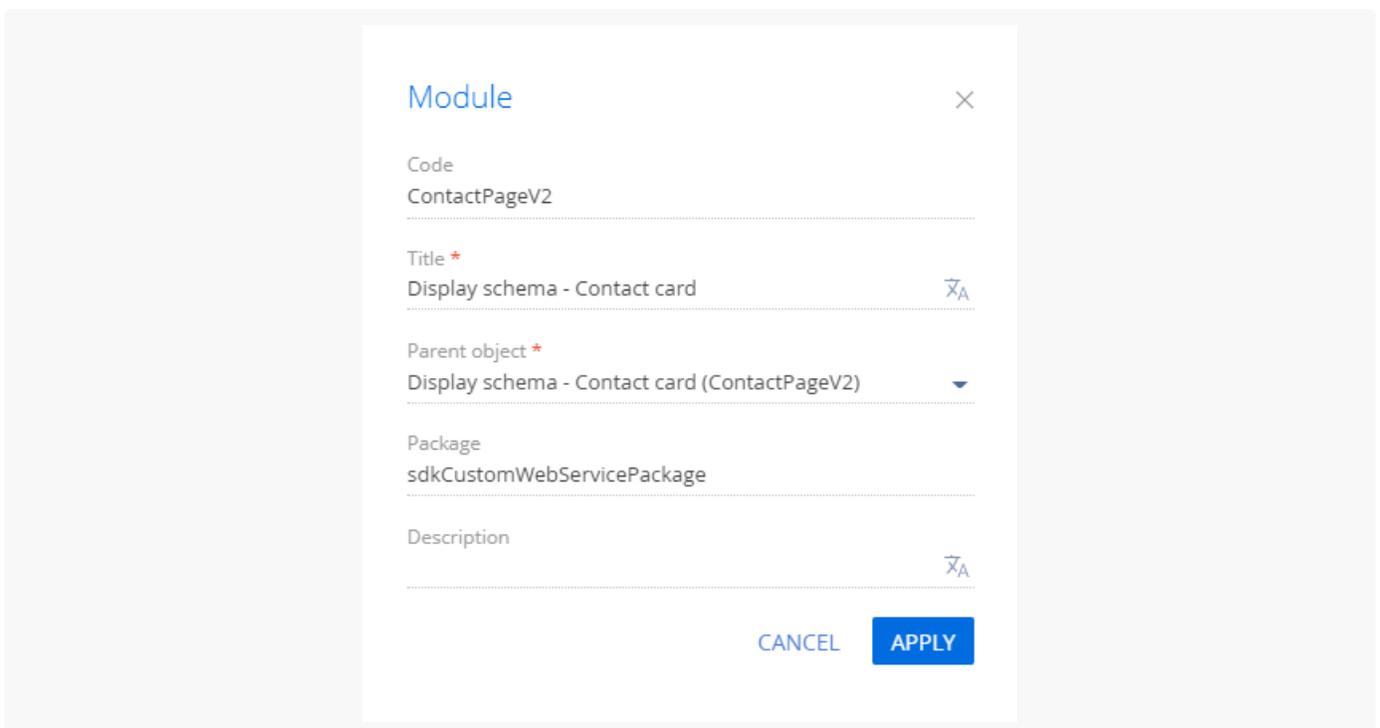
```

## 2. Create a replacing contact record page

1. [Go to the \[ Configuration \] section](#) and select a custom [package](#) to add the schema.
2. Click [ *Add* ] → [ *Replacing view model* ] on the section list toolbar.



3. Select the `ContactPageV2` package's [ *Display schema — Contact card* ] view model schema to replace in the [ *Parent object* ] property. After you confirm the parent object, Creatio will populate the other properties.



4. Enable the `ServiceHelper` module as a dependency in the declaration of the record page module. Learn more about the module dependencies in a separate article: [AMD concept. Module definition](#).

### 3. Add the button to the contact record page

1. Click the **+** button in the [ *Localizable strings* ] block of the properties panel and fill out the **localizable string properties**:
  - Set [ *Code* ] to "GetServiceInfoButtonCaption."

- Set [ *Value* ] to "Call service."

## 2. Add the button handler.

Call the web service using the `callService()` method of the `ServiceHelper` module. Pass the following **parameters** of the `callService()` function:

- `UsrCustomConfigurationService`, the name of the custom web service class
- `GetContactIdByName`, the name of the custom web service method to call
- the callback function in which to process the service output
- `serviceData`, the object that contains the initialized incoming parameters for the custom web service method
- the execution context

View the source code of the `ContactPageV2` replacing view model below.

### ContactPageV2

```
define("ContactPageV2", ["ServiceHelper"],
function(ServiceHelper) {
    return {
        /* The name of the record page object's schema. */
        entitySchemaName: "Contact",
        details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
        /* The methods of the record page's view model. */
        methods: {
            /* Check if the [Full name] page field is filled out. */
            isContactNameSet: function() {
                return this.get("Name") ? true : false;
            },
            /* The button click handler method. */
            onGetServiceInfoClick: function() {
                var name = this.get("Name");
                /* The object that initializes the incoming parameters for the service method
                var serviceData = {
                    /* The name of the property matches the name of the service method's inco
                    Name: name
                };
                /* Call the web service and process the outcome. */
                ServiceHelper.callService("UsrCustomConfigurationService", "GetContactIdByNam
                function(response) {
                    var result = response.GetContactIdByNameResult;
                    this.showInformationDialog(result);
                }, serviceData, this);
            }
        },
        diff: /**SCHEMA_DIFF*/[
            /* The metadata to add the custom button to the page. */
```

```

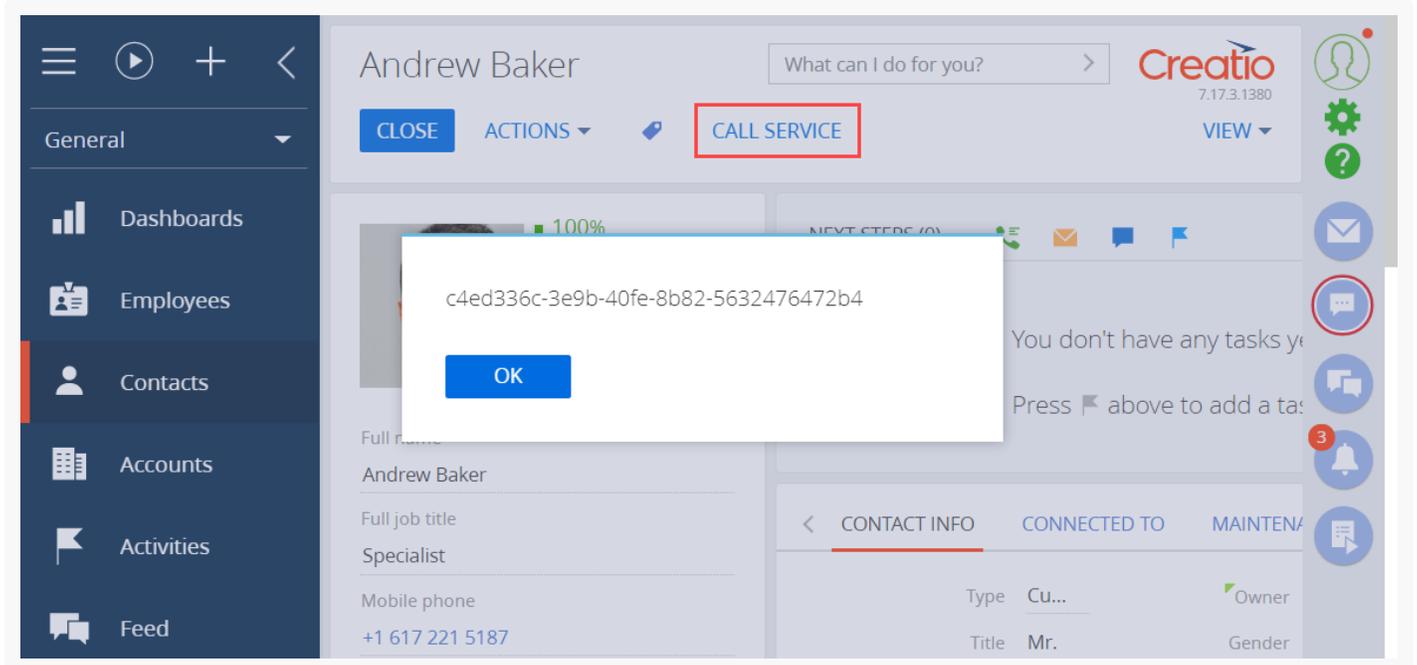
{
    /* Add the element to the page. */
    "operation": "insert",
    /* The name of the parent control to add the button. */
    "parentName": "LeftContainer",
    /* Add the button to the control collection of the parent whose metaname is s
    "propertyName": "items",
    /* The name of the button to add. */
    "name": "GetServiceInfoButton",
    /* The additional field properties. */
    "values": {
        /* Set the type of the added element to button. */
        itemType: Terrasoft.ViewItemType.BUTTON,
        /* Bind the button caption to the localizable schema string. */
        caption: {bindTo: "Resources.Strings.GetServiceInfoButtonCaption"},
        /* Bind the button click handler method. */
        click: {bindTo: "onGetServiceInfoClick"},
        /* Bind the button availability property. */
        enabled: {bindTo: "isContactNameSet"},
        /* Set up the field location. */
        "layout": {"column": 1, "row": 6, "colSpan": 2, "rowSpan": 1}
    }
}
]/**SCHEMA_DIFF*/
};
});

```

3. Click [ Save ] on the Designer's toolbar.

## Outcome of the example

As a result, Creatio will display the [ *Call service* ] button on the contact page after you refresh the Creatio web page. Click the button to call the `GetContactIdByName` method of the `UsrCustomConfigurationService` custom web service. The method returns the ID of the current contact.



# Call a custom web service from Postman

 Medium

Integrate external applications with custom Creatio web services via HTTP requests to the services. Editing and debugging tools, such as [Postman](#) or [Fiddler](#), help to understand the request creation principles.

**Postman** is a request testing toolset. The **purpose** of Postman is to send test requests from the client to the server and receive the server's responses. The example in this article calls a custom web service that uses cookie-based authentication from Postman.

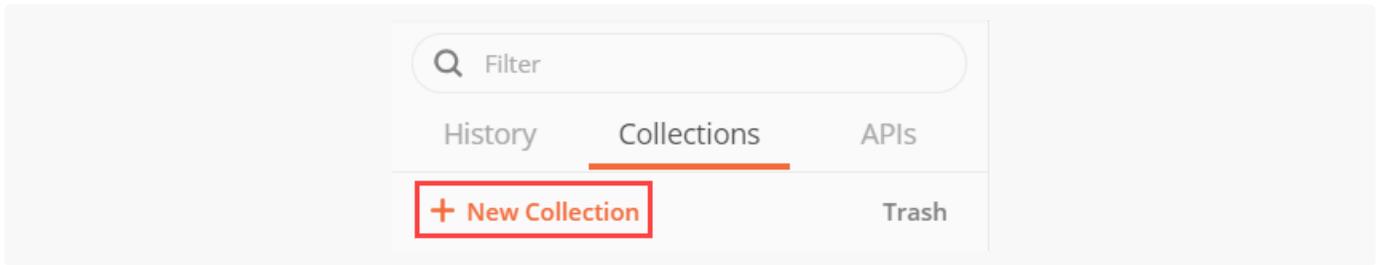
**Example.** Call a custom web service that uses cookie-based authentication from Postman.

This example uses the `UsrCustomConfigurationService` custom web service. Learn more about developing the service in a separate article: [Develop a custom web service that uses cookie-based authentication](#).

Since this custom web service uses cookie-based authentication, authorize in Creatio first. Do this by calling the `AuthService.svc` system web service. Learn more about authentication in a separate article: [Authentication](#).

## 1. Create a request collection

1. Go to the [ *Collections* ] tab on the Postman request toolbar and click [ + *New Collection* ].



2. Fill out the **request collection fields**:

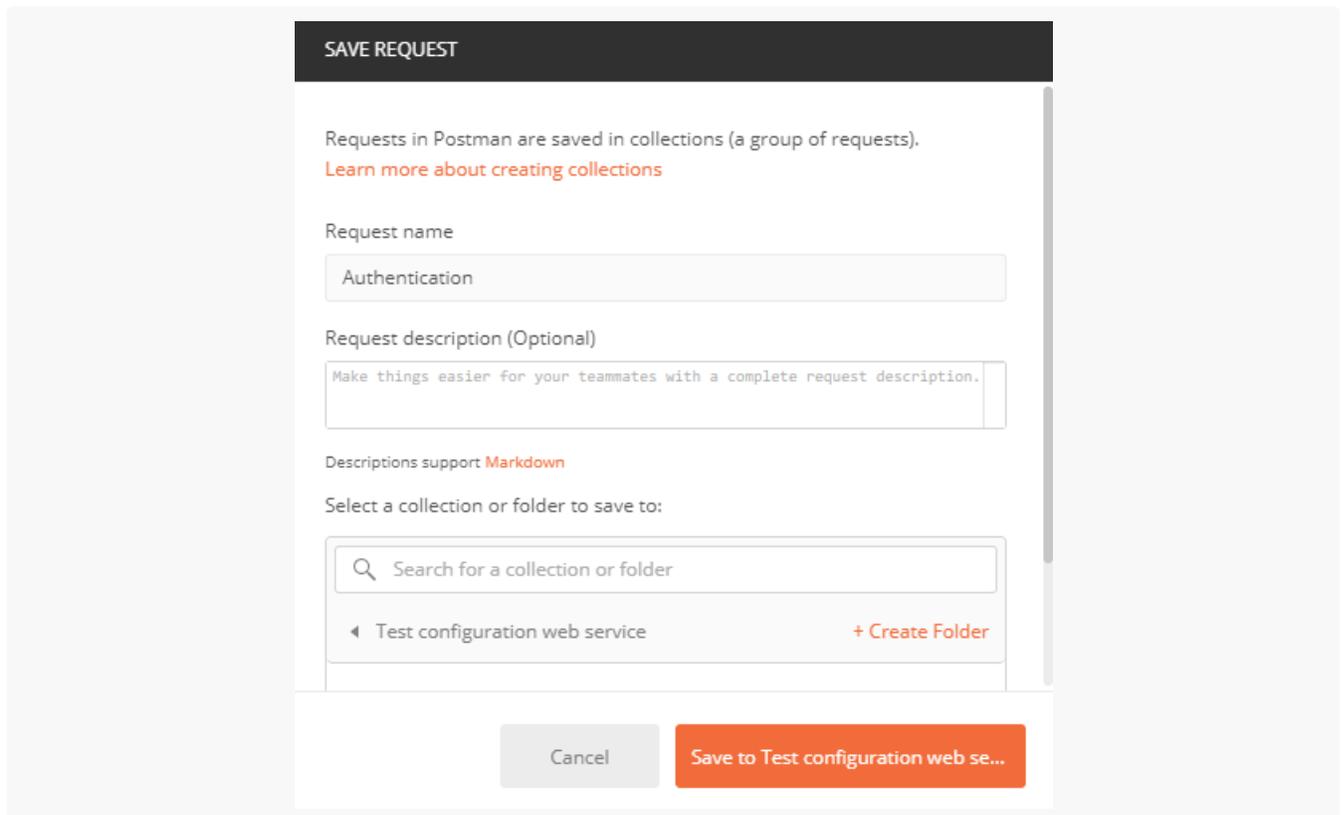
- Set [ *Name* ] to "Test configuration web service."

 A screenshot of the 'CREATE A NEW COLLECTION' dialog box in Postman. The dialog has a dark header with the title 'CREATE A NEW COLLECTION' and a close button. Below the header, there is a 'Name' field containing the text 'Test configuration web service'. Underneath, there are five tabs: 'Description', 'Authorization', 'Pre-request Scripts', 'Tests', and 'Variables'. The 'Description' tab is selected and underlined. Below the tabs, there is a text area for the description with the placeholder text 'Make things easier for your teammates with a complete collection description.' and a note that 'Descriptions support Markdown'. At the bottom right, there are two buttons: 'Cancel' and 'Create'.

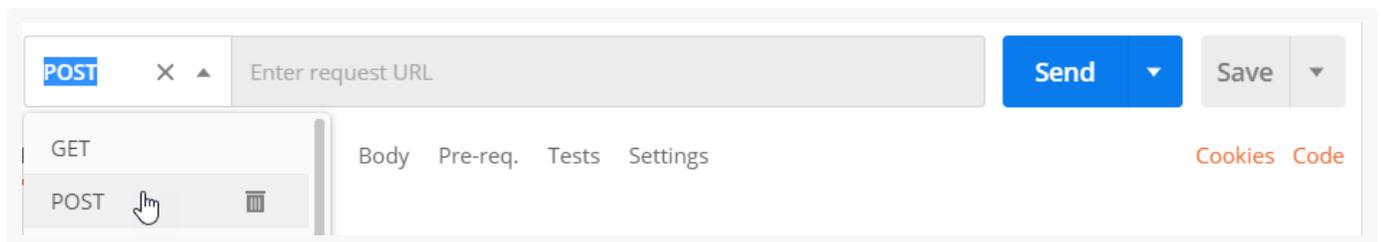
3. Click [ *Create* ] to create a request collection.

## 2. Set up an authentication request

1. Go to the request working area in Postman and right-click the name of the `Test configuration web service` collection → [ *Add request* ].
2. Fill out the **request fields**:
  - Set [ *Request name* ] to "Authentication."



3. Click [ *Save to Test configuration web service* ] to add the request to the collection.
4. Select the `POST` request method in the drop-down list of the Postman workspace toolbar.



5. Enter the string of the authentication service request in the Postman workspace toolbar.

Template of the AuthService.svc service URL

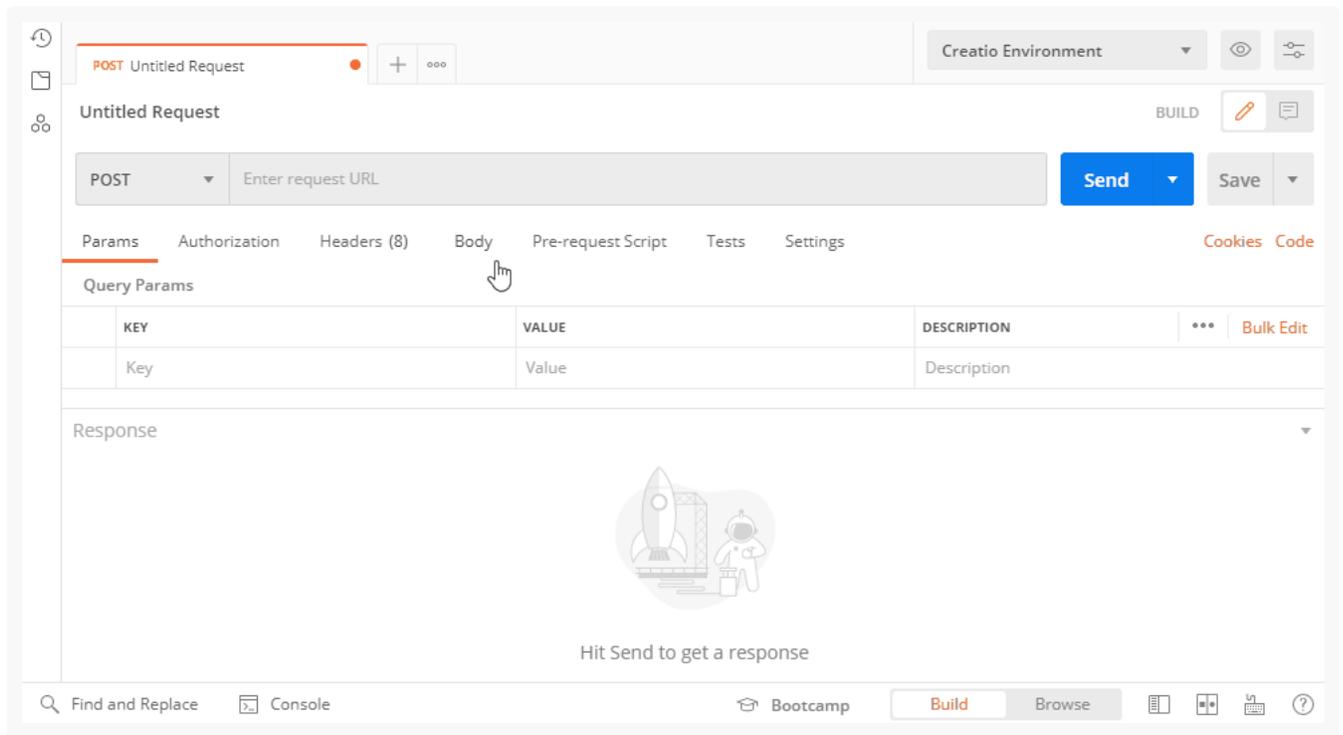
```
[Creatio application URL]/ServiceModel/AuthService.svc/Login
```

Example of the AuthService.svc service URL

```
http://mycreatio.com/creatio/ServiceModel/AuthService.svc/Login
```

6. Set the **request data format**:

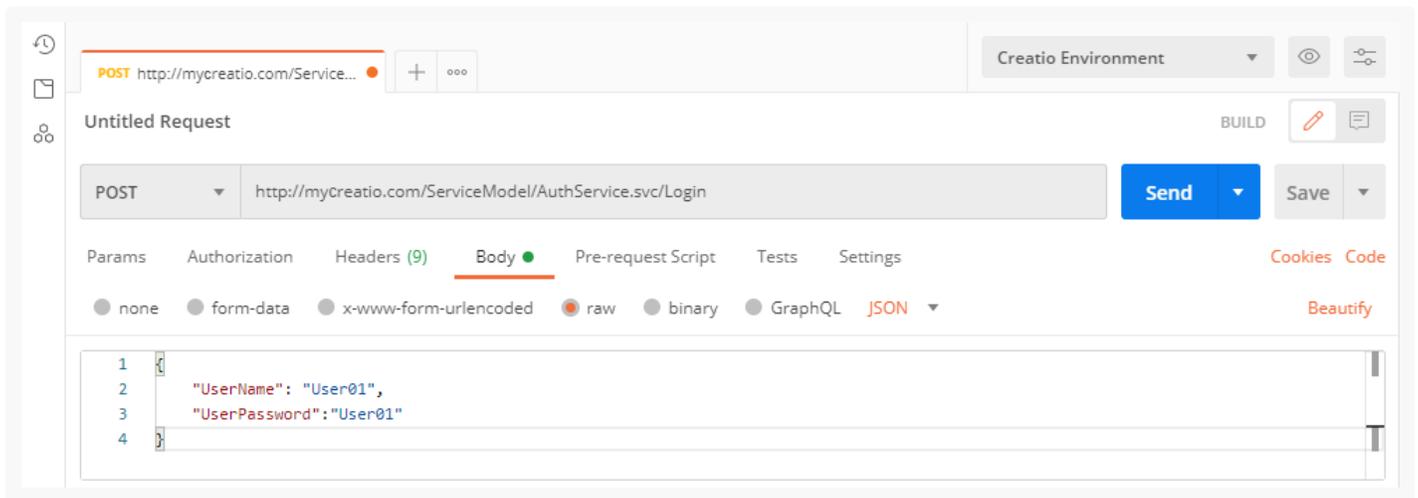
- a. Go to the [ *Body* ] tab.
- b. Set the "raw" option.
- c. Select the "JSON" type.



7. Go to the [ *Body* ] tab in the Postman workspace and fill out the body of the `POST` request. The body is a JSON object that contains the login credentials.

### Body of the `POST` request

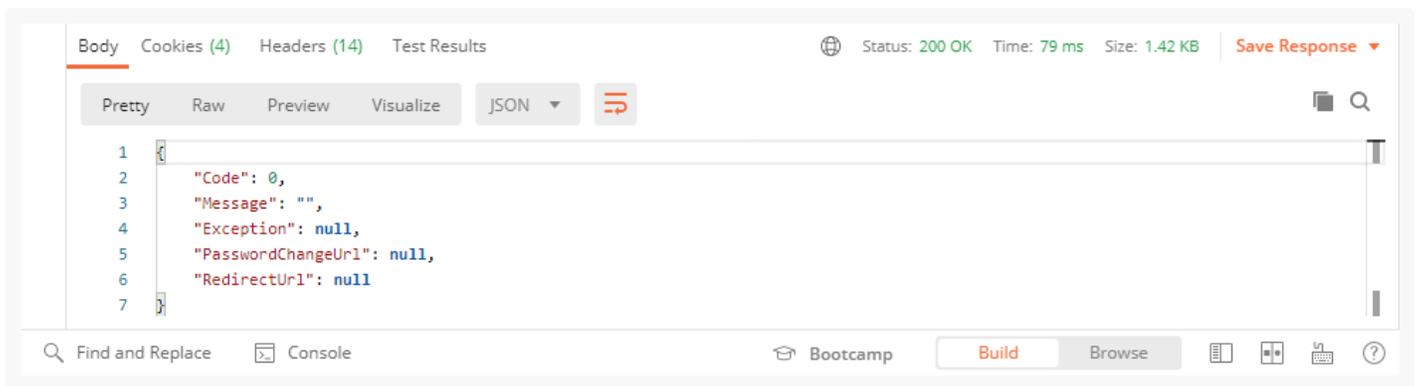
```
{
  "UserName": "User01",
  "UserPassword": "User01"
}
```



### 3. Execute the authentication request

Click [ `Send` ] in the Postman workspace toolbar to **execute the request from Postman**.

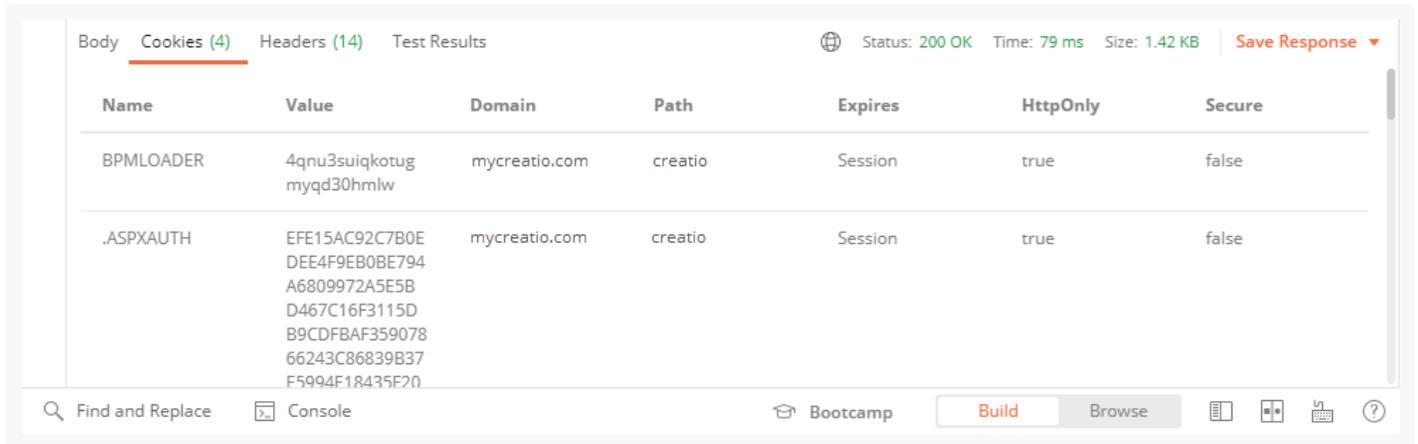
As a result, Postman will receive a response that contains a JSON object. View the response body on the Postman `Body` tab.



The indicators of a **successfully executed request** are as follows:

- The server returns the `200 OK` status code.
- The `code` parameter of the response body contains `"0"`.

The response also contains `BPMLOADER`, `.ASPXAUTH`, `BPMCSRF`, and `UserName` cookies. Postman displays them on the `Cookies` and `Headers` tab.



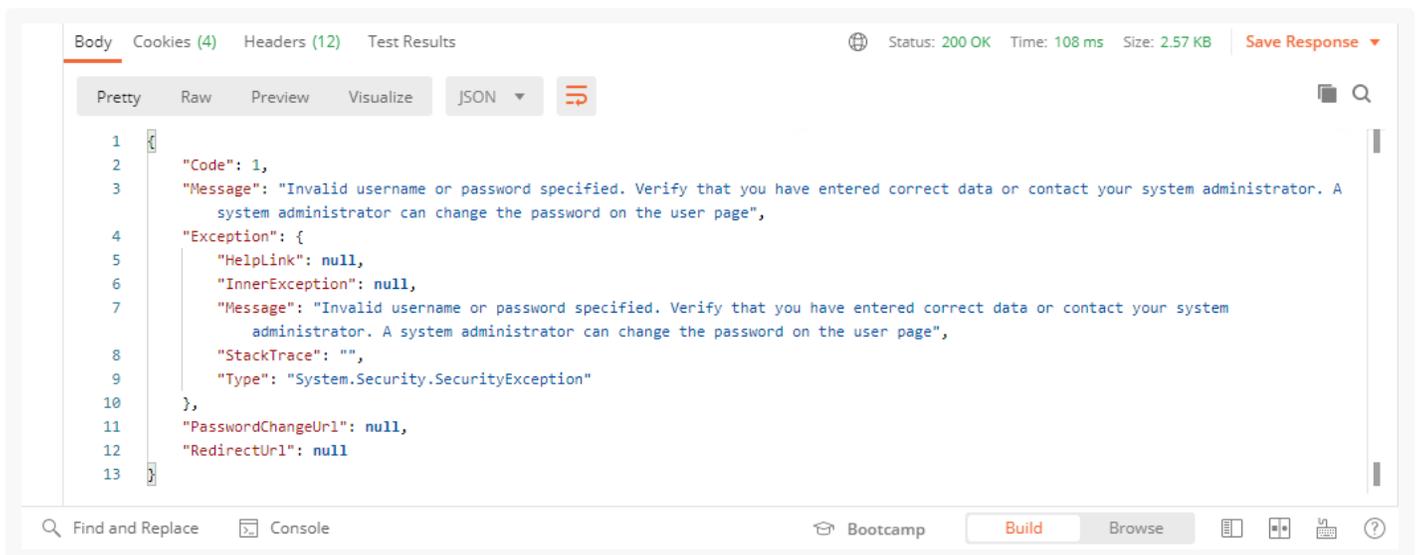
Use these cookies in further requests to Creatio services that use cookie-based authentication.

If you enabled the [CSRF attack protection](#), always use the `BPMCSRF` cookie for request methods ( `POST` , `PUT` , `DELETE` ) that modify (add, change, or delete) the entity. If you do not use the `BPMCSRF` cookie, the server returns the **403 Forbidden** status code. Creatio does not check for the `BPMCSRF` cookie for `GET` requests. You do not have to use the `BPMCSRF` cookie with [Creatio demo sites](#) since they have CSRF attack protection disabled by default.

The request fails if it contains errors in the string or the body.

The indicators of an **unsuccessfully executed request** are as follows:

- The `Code` parameter of the response body contains "1."
- The `Message` parameter of the response body contains the reason for the authentication failure.

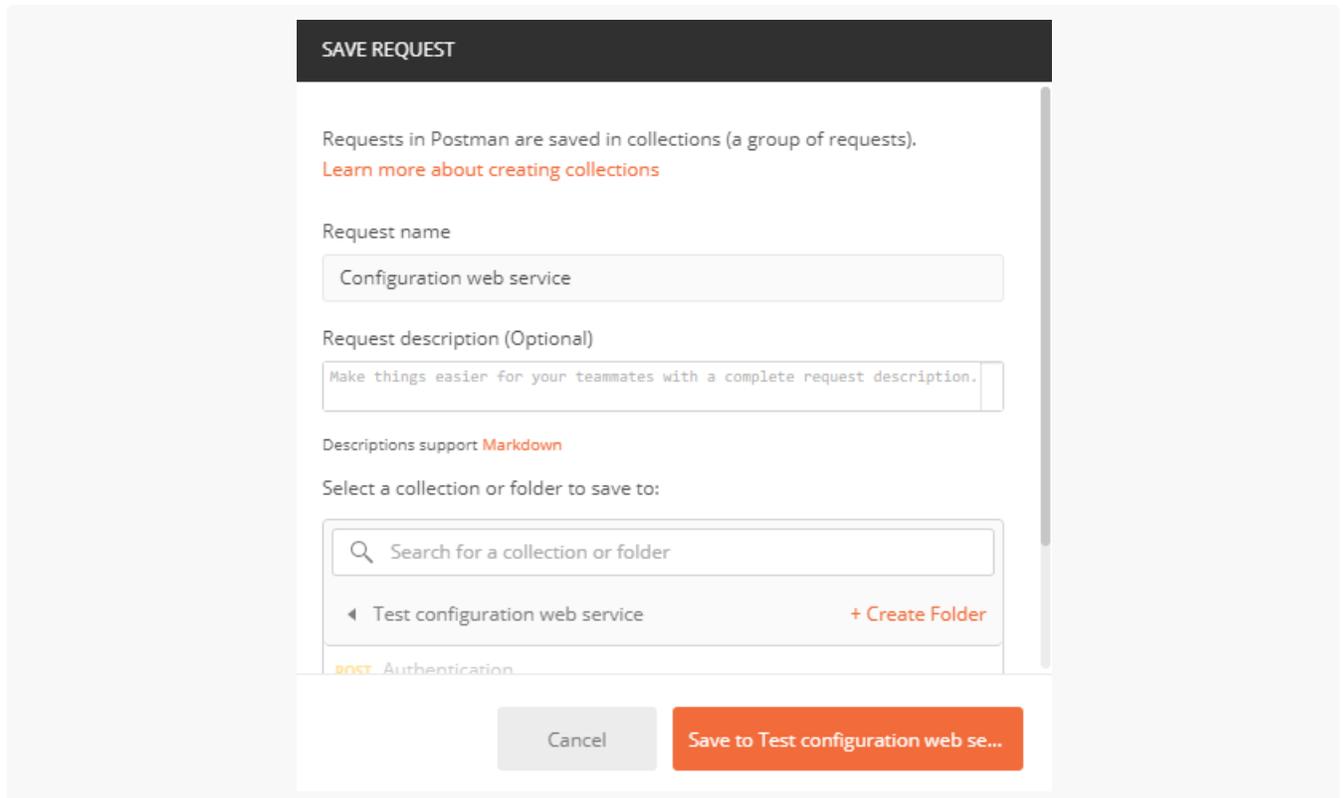


## 4 Set up the request to the custom web service that uses cookie-based authentication

The `UsrCustomConfigurationService` custom web service accepts `GET` requests only.

To **set up the request to the custom web service that uses cookie-based authentication**:

1. Go to the request working area in Postman and right-click the name of the `Test configuration web service` collection → [ *Add request* ].
2. Fill out the **request fields**:
  - Set [ *Request name* ] to "Configuration web service."



3. Click [ *Save to Test configuration web service* ] to add the request to the collection.
4. Postman selects the `GET` method by default. Enter the string of the `UsrCustomConfigurationService` custom web service request in the request field of the Postman workspace toolbar.

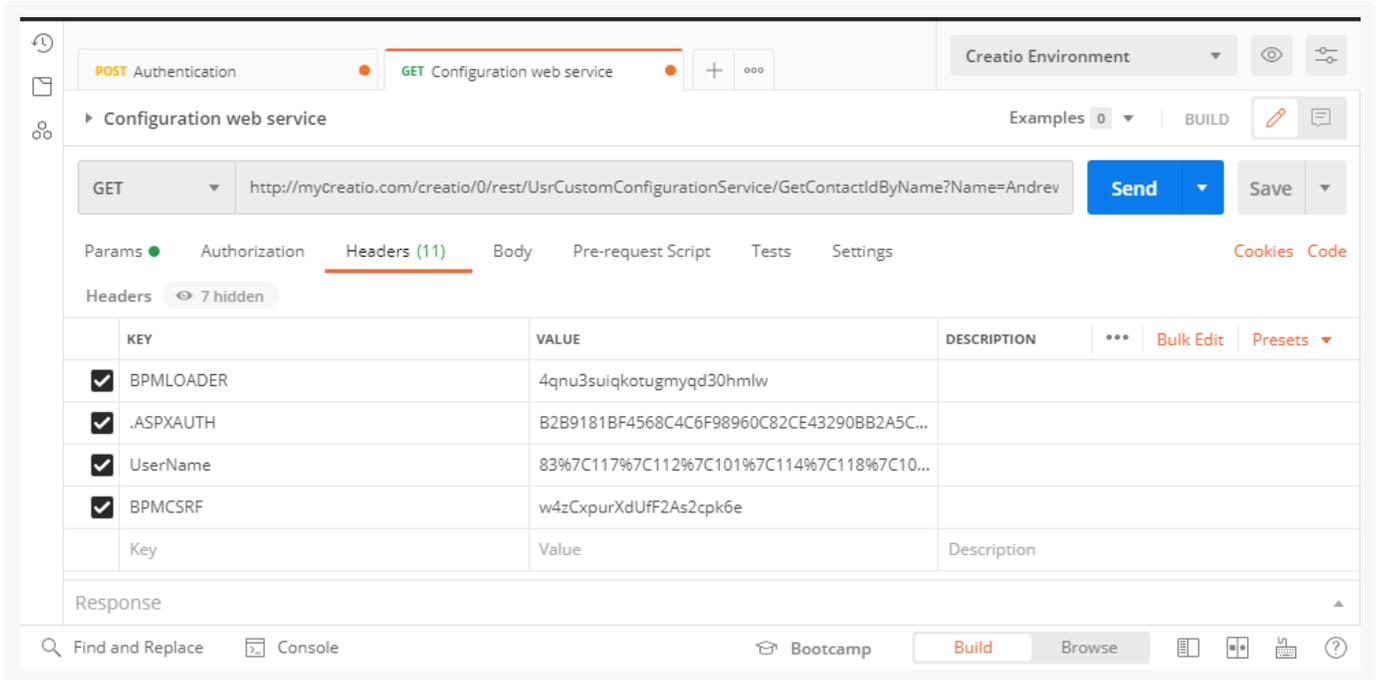
Template of the custom web service's URL

[Creatio application URL]/0/rest/UsrCustomConfigurationService/GetContactIdByName?Name=[Conta

Example of the custom web service's URL

http://mycreatio.com/creatio/0/rest/UsrCustomConfigurationService/GetContactIdByName?Name=And

5. Go to the [ *Headers* ] tab in the Postman workspace and add the cookies received as a result of the authorization request to the headers of the custom web service request. Add the cookie name to the [ *Key* ] field and copy the corresponding cookie value to the [ *Value* ] field.



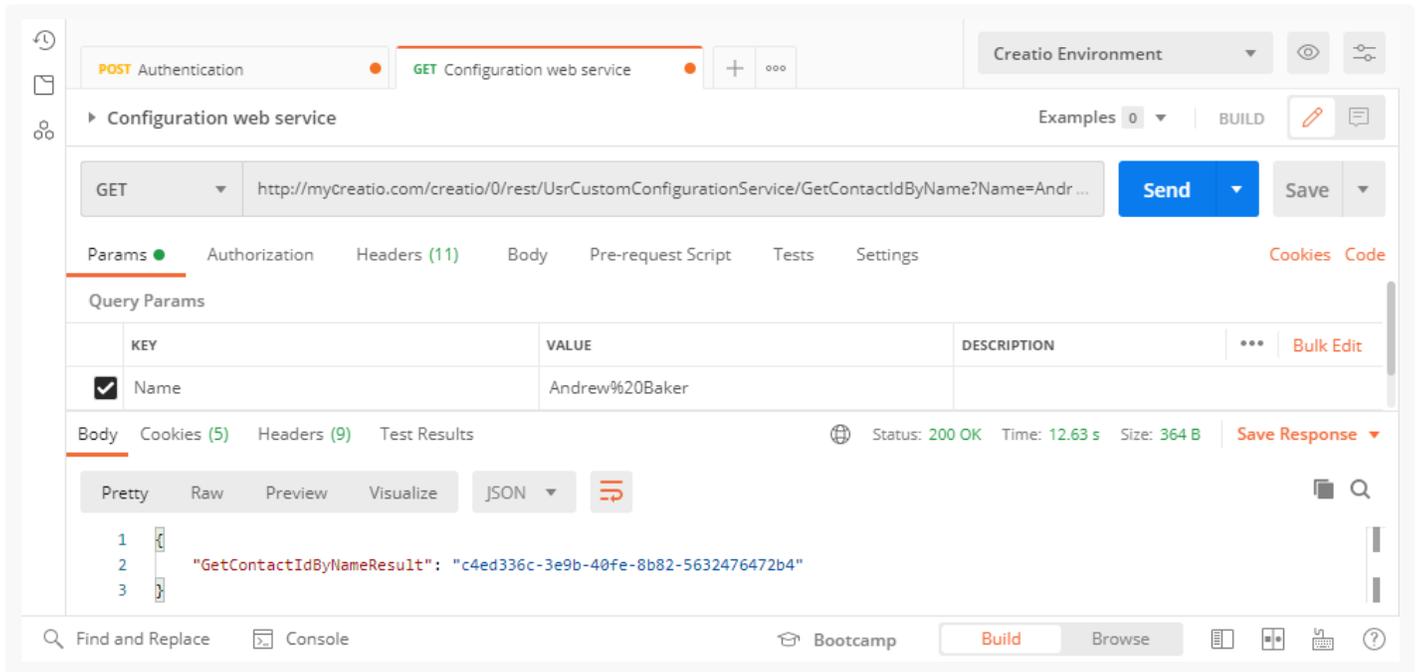
## 5. Execute the request to the custom web service that uses cookie-based authentication

Click [ *Send* ] on the workspace toolbar to execute a request from Postman.

### Outcome of the example

As a result, Postman will receive a response that contains a JSON object. View the response body on the Postman `Body` tab.

If Creatio finds the contact from the `Name` parameter in the database, the `GetContactIdByNameResult` property will return the contact ID value.



The screenshot shows the Postman interface for a GET request to the endpoint `http://mycreatio.com/creatio/0/rest/UsrCustomConfigurationService/GetContactIdByName?Name=Andr...`. The request parameters are set to `Name=Andrew%20Baker`. The response status is `200 OK` with a time of `12.63 s` and a size of `364 B`. The response body is displayed in JSON format, showing a single property: `"GetContactIdByNameResult": "c4ed336c-3e9b-40fe-8b82-5632476472b4"`.

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> Name	Andrew%20Baker	

```
1 {
2   "GetContactIdByNameResult": "c4ed336c-3e9b-40fe-8b82-5632476472b4"
3 }
```

If Creatio finds no contacts from the `Name` parameter in the database, the `GetContactIdByNameResult` property will return an empty string.