Creatio Academy

Back-end development

Custom web services

Version 8.0

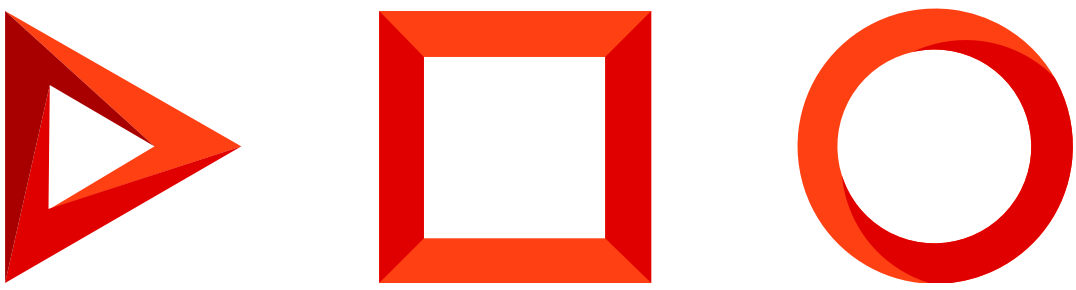


Table of Contents

Custom web services

Medium

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 generates and sends a request to the web service, receives the response, and extracts 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. Learn more in the user documentation guide: 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 and .NET 6 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: Service that runs business processes.

Creatio **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 and .NET 6) 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 or .NET 6 frameworks below.

You can use Postman to test querying a custom web service. Learn more about Postman in the official Postman documentation. Learn more about querying Creatio using Postman in a separate article: Working with requests in Postman. Learn more about calling a web service via Postman in a separate article: Call a custom web service from Postman.

Develop a custom web service that uses cookie-based authentication

1. Create a [*Source code*] schema. Learn more about creating a schema in a separate article: Source code (C#).
2. Create a **service class**.
   a. Add the `Terrasoft.Configuration` namespace or any of its nested namespaces in the Schema Designer. 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 or .NET 6. If you develop the web service using the `System.Web` namespace and have to run it on .NET Core or .NET 6, 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 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

A **custom web service that uses anonymous authentication** does not require the user to pre-authenticate, i. e., you can use the service 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.GetInstanc
   ```

4. Register the **custom web service that uses anonymous authentication**.
   a. Create an *.svc file in the `..\Terrasoft.WebApp\ServiceModel` directory. The file name must match the web service name.
   b. Add the following record to the file.

   Template that registers the custom web service that uses anonymous authentication

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

   Example that registers the custom web service that uses anonymous authentication

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

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

Learn more about the `@ServiceHost` WCF directive in the official Microsoft documentation.

c. Save the file.

5. Register a **non-standard text encoding** (optional).

Since version 8.0.2, Creatio lets you use arbitrary character encodings in .NET Framework web services that use anonymous authentication. For example, you can use such encodings as ISO-8859, ISO-2022, etc. Learn more about encodings in Wikipedia.

To **register an arbitrary character encoding**:

a. Add a `<customBinding>` section to the `..\Terrasoft.WebApp\ServiceModel\http\bindings.config` file.

b. Add the following **attributes** to the `<customBinding>` file section:

- `name` attribute of the `<binding>` element. Fill it with a custom name of the encoding.
- `encoding` attribute of the `<customTextMessageEncoding>` element. Fill it with the code of the encoding, for example, ISO-8859-1.
- `manualAddressing` attribute of the `<httpTransport>` element. Set it to `true`.

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

```
<bindings>
    ...
    <customBinding>
        <binding name="CustomEncodingName">
            <customTextMessageEncoding encoding="ISO-8859-1" />
            <httpTransport manualAddressing="true"/>
        </binding>
        ...
    </customBinding>
</bindings>
```

Register each encoding (i. e., add `<binding>` file section for each) individually.

f. Save the file.

g. Add an identical record to the `..\Terrasoft.WebApp\ServiceModel\https\bindings.config` file.

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

a. Add the following record to the `..\Terrasoft.WebApp\ServiceModel\http\services.config` file.

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="[Binding]"
            bindingConfiguration="[Custom encoding]"
            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.

The `binding` attribute contains the value of the character encoding. Must match the name of the file section where the encoding that the web service uses is registered. Set to "`webHttpBinding`" to use the UTF-8 encoding. Set to "`customBinding`" to use a custom encoding.

The `bindingConfiguration` attribute. Must be present if the `binding` attribute is set to "`customBinding`." The value of the current attribute must match the value of the `<binding>` element's `name` attribute specified on the previous step.

Learn more about the web service configuration elements in the official Microsoft documentation.

b. Save the file.

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

7. Enable all users to **access the custom web service** that uses anonymous authentication.

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

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>
```

b. Add the relative web service path to the `value` attribute of the `<appSettings>` element's `AllowedLocations` key in the `..\Terrasoft.WebApp\Web.config` file.

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

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

c. Save the file.

8. 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 and .NET 6

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.

   To do this, add the web service data to the `AnonymousRoutes` block of the `..\Terrasoft.WebHost\appsettings.json` configuration file.

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

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

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 web service both with and 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 in **several ways**:

- from the browser.
- from the 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 following request string:

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

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

Example URL 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 or .NET 6** 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

To call a **.NET Framework** custom web service that uses anonymous authentication from the browser, use the request string below.

Template URL of a custom web service that uses anonymous authentication

```
[Creatio application URL]/0/ServiceModel/[Custom web service name].svc/[Custom web service endpo
```

Example URL 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 or .NET 6** 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 for executing 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, serviceMethodName, callback, serviceData, scope)` method.
   - Call the `callService(config)` method, where `config` is a configuration object.

     The `config` configuration object 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 handles the web service response.
     - `data` is the object that contains the initialized incoming parameters for the web service method.
     - `scope` is the scope 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 methods.

Migrate an existing custom web service to .NET Core or .NET 6

You can migrate a .NET Framework custom web service that retrieves the scope without inheriting the `Terrasoft.Web.Common.BaseService` base class to .NET Core or .NET 6. 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 or .NET 6. 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.Requ
    public class UsrCustomConfigurationService: BaseService
    {
        /* The web service method. */
        [OperationContract]
        [WebInvoke(Method = "GET", RequestFormat = WebMessageFormat.Json, BodyStyle = WebMessage
        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 **scope retrieval options** for web services developed without inheriting the `Terrasoft.Web.Common.BaseService` class:

- via the `IHttpContextAccessor` **interface** 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: Replacing class factory.

- 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 or .NET 6, 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/.NET 6 (the `Microsoft.AspNetCore.Http` namespace) in the configuration.

Example that adapts the web service to .NET Core or .NET 6

```
namespace Terrasoft.Configuration.UsrCustomNamespace
```

```
{
    /* Use instead of System.Web. */
    using Terrasoft.Web.Http.Abstractions;

    [ServiceContract]
    [AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Requ
    public class UsrCustomConfigurationService
    {
        /* The web service method. */
        [OperationContract]
        [WebInvoke(Method = "GET", RequestFormat = WebMessageFormat.Json, BodyStyle = WebMessage
        ResponseFormat = WebMessageFormat.Json)]
        public void SomeMethod() {
            ...
            var httpContext = HttpContext.Current;
            ...
        }
    }
}
```

Develop a custom web service that uses cookie-based authentication

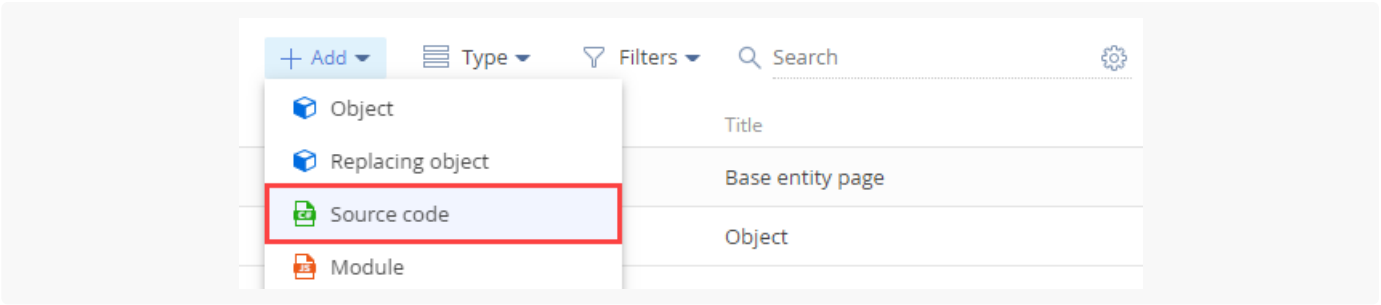
Medium

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.

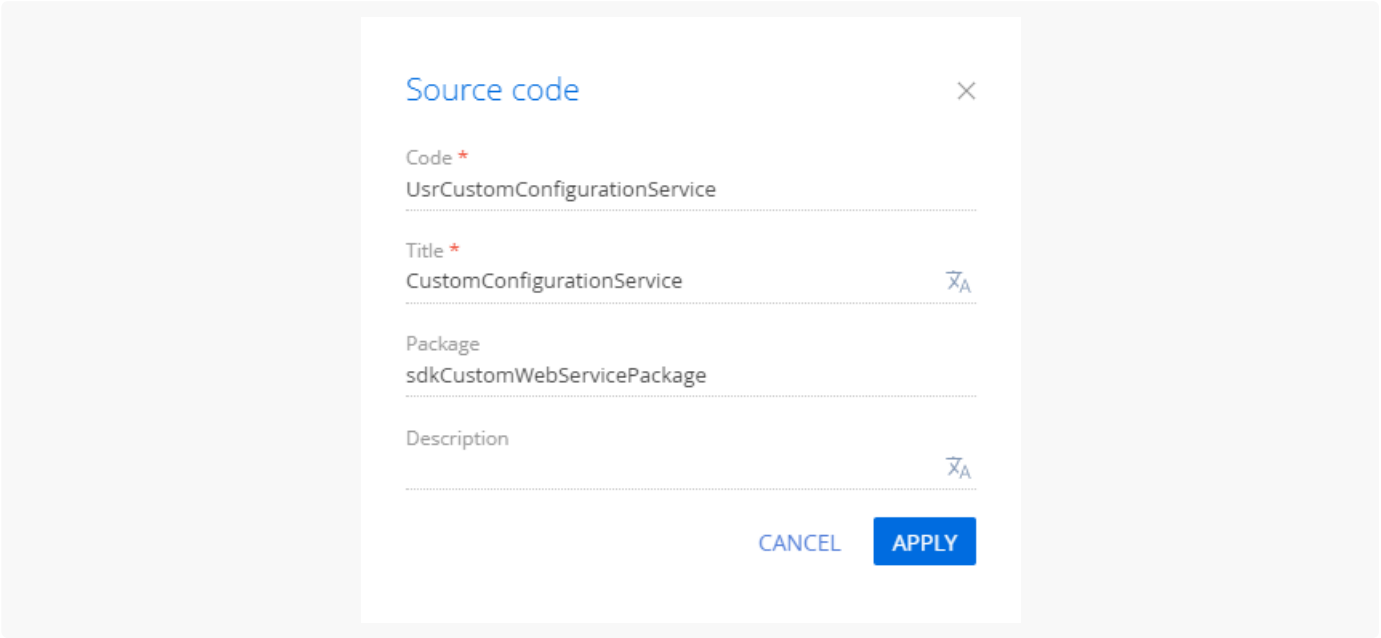
1. Create a [*Source code*] schema

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.Requ
    public class UsrCustomConfigurationService: BaseService
    {

        /* The method that returns the contact ID by the contact name. */
        [OperationContract]
        [WebInvoke(Method = "GET", RequestFormat = WebMessageFormat.Json, BodyStyle = WebMessage
        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", N
            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

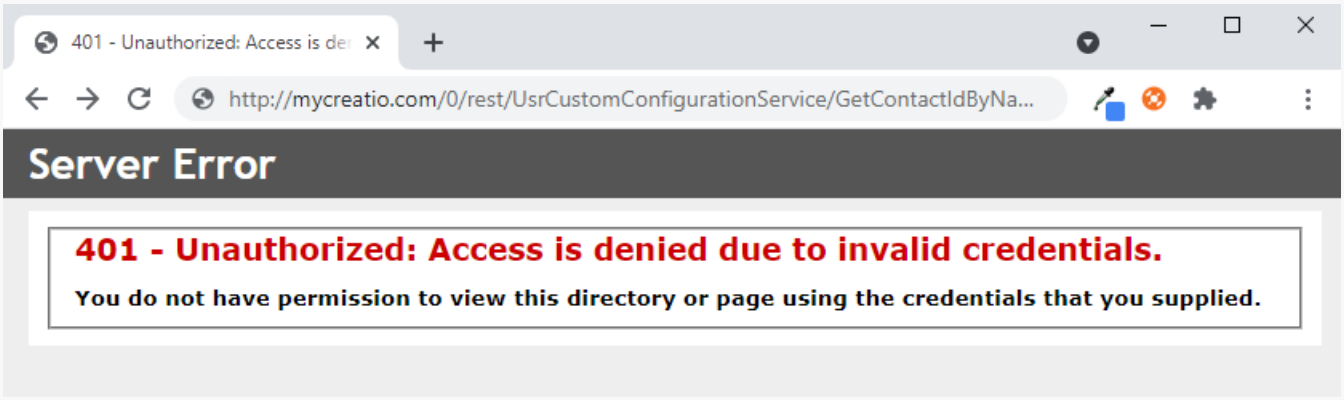
As a result, Creatio will add the custom `UsrCustomConfigurationService` REST web service that has the `GetContactIdByName` endpoint.

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

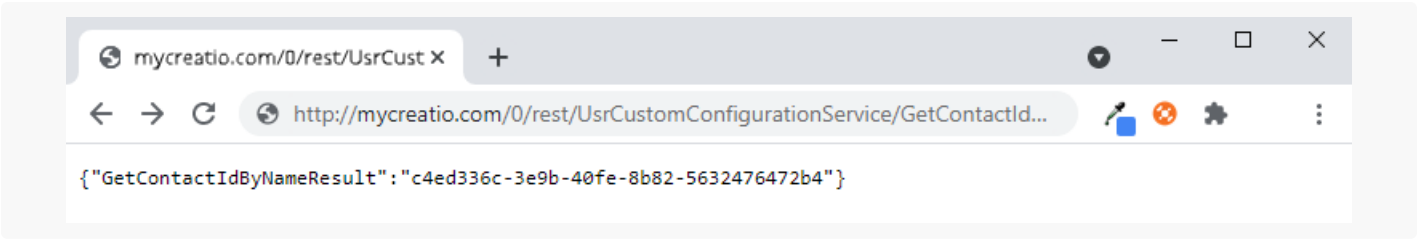
Request string that contains the name of the existing contact

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

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



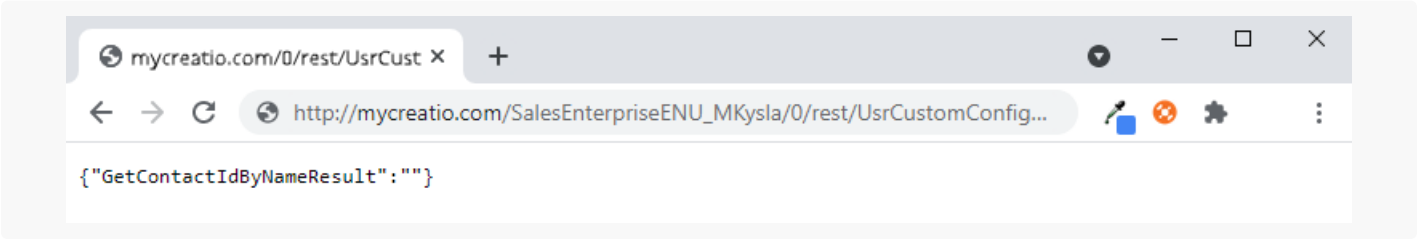
Log in to Creatio and execute the request once more. If Creatio finds the contact from the `Name` parameter in the database, the `GetContactIdByNameResult` property will return the contact ID value.



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

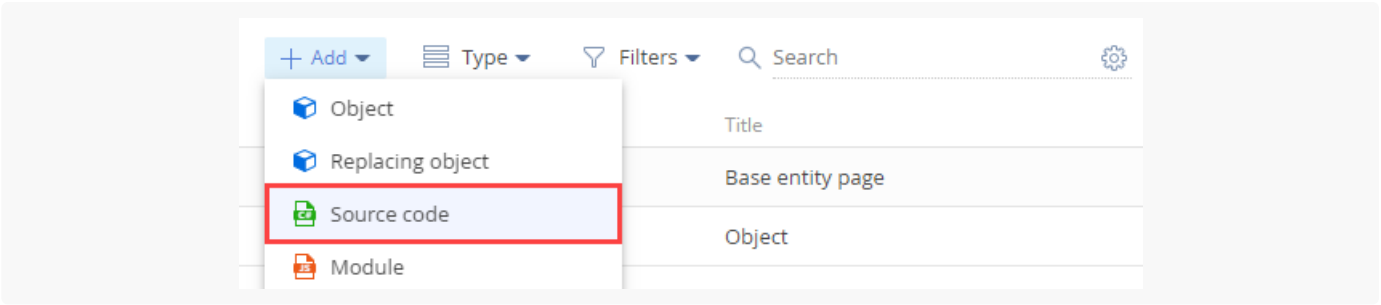
Medium

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.

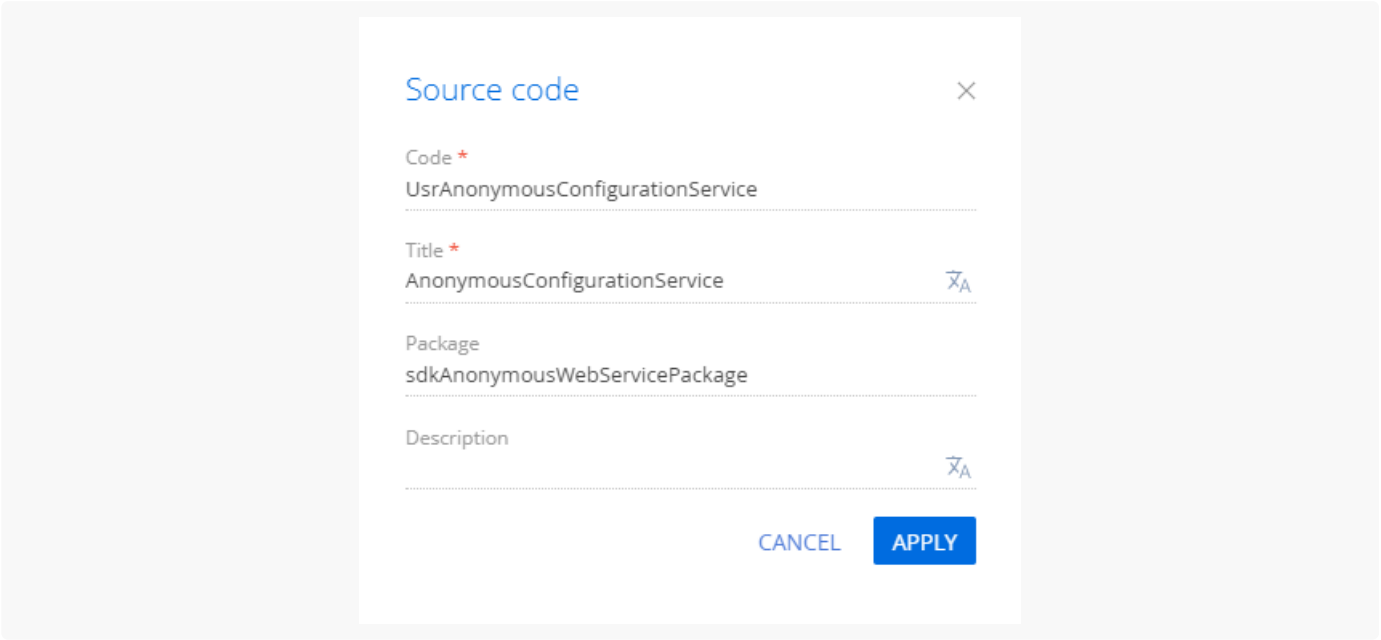
1. Create a [*Source code*] schema

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.Requ
    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 = WebMessage
        ResponseFormat = WebMessageFormat.Json)]
        public string GetContactIdByName(string Name){
            /* Specify the user on whose behalf to process the HTTP request. */
            SessionHelper.SpecifyWebOperationIdentity(HttpContextAccessor.GetInstance(), SystemU
            /* 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", N
            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

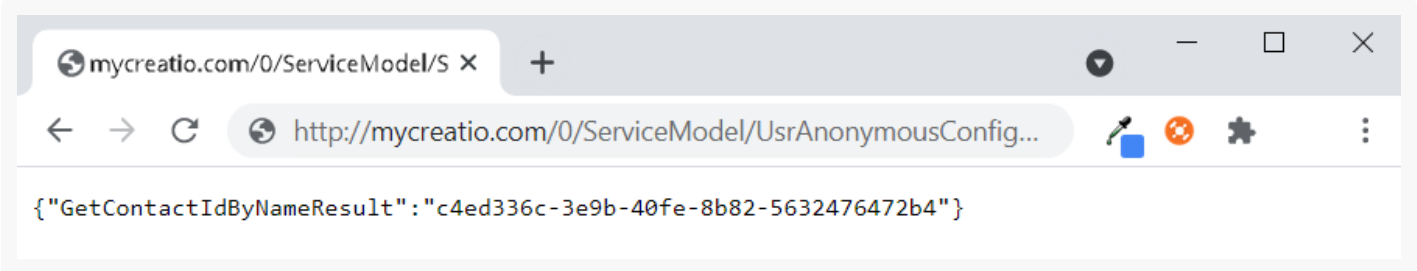
As a result, Creatio will add the custom `UsrAnonymousConfigurationService` REST web service that has the `GetContactIdByName` endpoint. You can access the web service from the browser, with or without pre-authentication.

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.

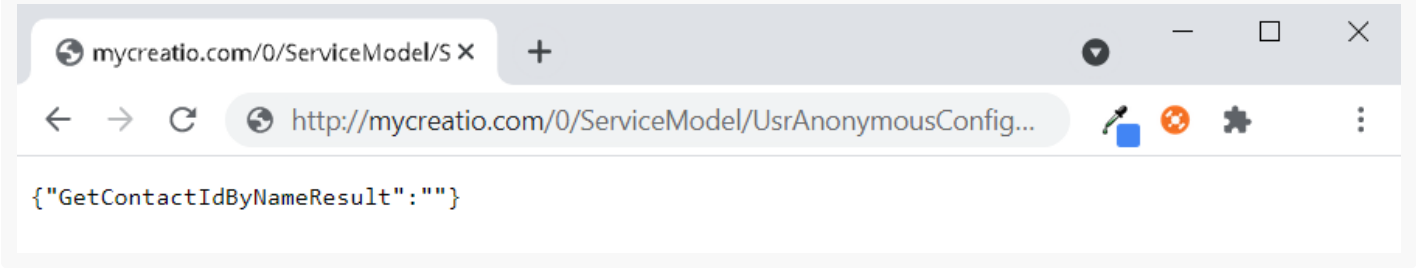


If Creatio finds no contacts from the `Name` parameter in the database, the `GetContactIdByNameResult` property will

return an empty string.

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

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



Develop a custom web service that uses anonymous authentication and non-standard text encoding

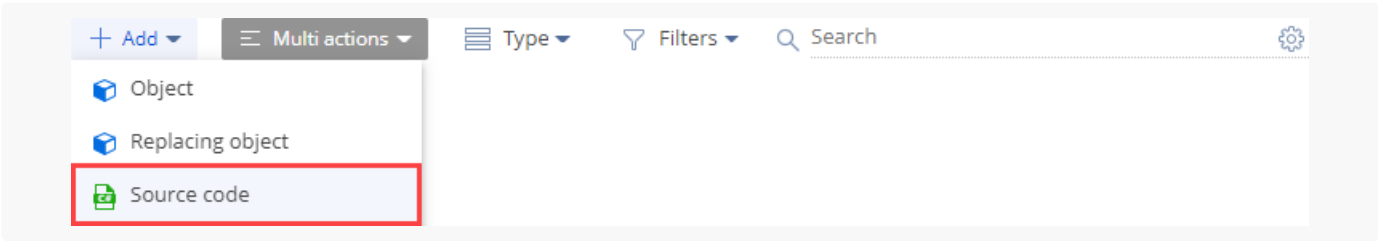
Medium

The example is relevant to Creatio version 8.0.2 and later.

Example. Create a custom web service that uses anonymous authentication and gets an arbitrary text in the ISO-8859-1 encoding. The web service must return identical text that uses the same encoding.

1. Create a [*Source code*] schema

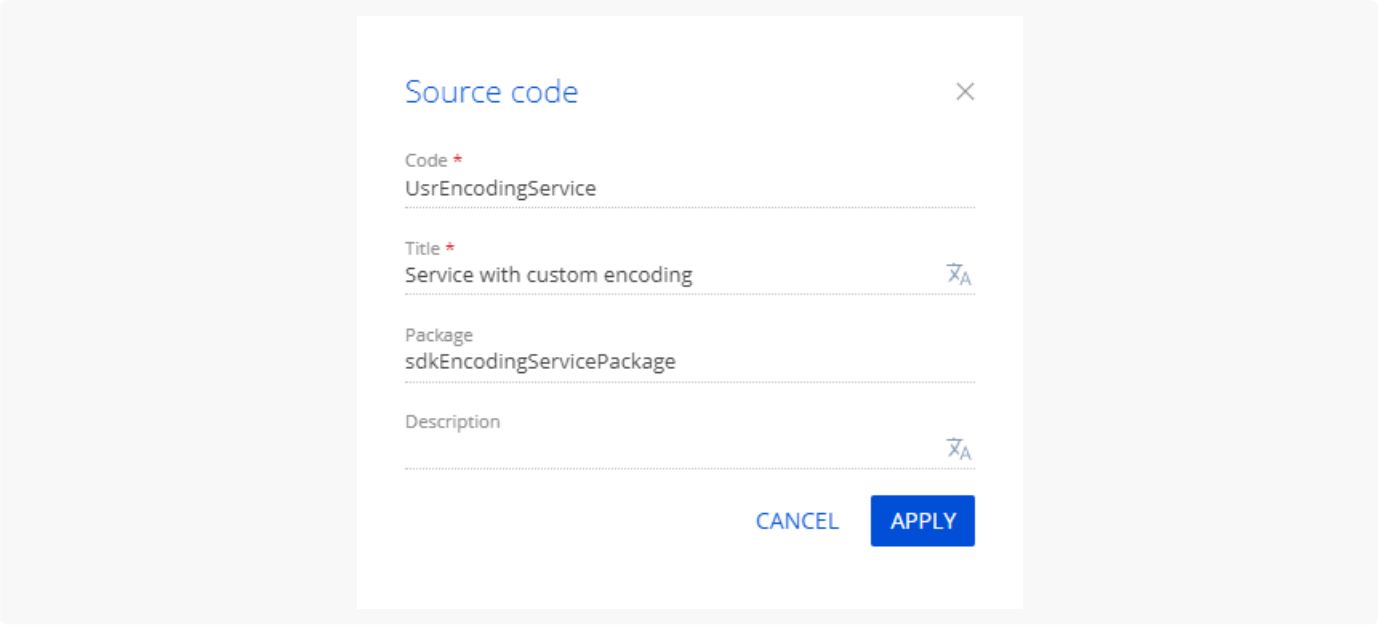
1. Go to the [*Configuration*] section and select a custom package to add the schema.
2. Click [*Add*] → [*Source code*] on the section list toolbar.



3. Fill out the **schema properties** in the Source Code Designer:

   - Set [*Code*] to "`UsrEncodingService`."

- Set [*Title*] to "`Service with custom encoding.`"



Click [*Apply*] to apply the properties.

2. Create a web service class

1. Go to the Schema Designer and add the namespace nested into `Terrasoft.Configuration`. For example, `UsrEncodingServiceNamespace`.
2. Add the `using` directive to import the namespaces whose data types are utilized in the class.
3. Add a class name to match 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 a method of the web service class

1. Implement the **endpoint of the custom web service**. To do this, add the `public string Test(string Name)` method to the class in the Source Code Designer. Depending on the `Name` parameter value specified in the ISO-8859-1 encoding and sent in the request string, the response body contains the same parameter value in the same encoding.
2. Specify the **user on whose behalf to process the current 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.GetInstanc
```

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

UsrEncodingService

```
/* Custom namespace. */
namespace Terrasoft.Configuration.UsrEncodingServiceNamespace
{
    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 UsrEncodingService: BaseService
    {
        /* Reference to the UserConnection instance required to call the database. */
        private SystemUserConnection _systemUserConnection;
        private SystemUserConnection SystemUserConnection {
            get {
                return _systemUserConnection ?? (_systemUserConnection = (SystemUserConnection)A
            }
        }

        /* Method that returns the value of the passed parameter in the specified encoding. */
        [OperationContract]
        [WebInvoke(Method = "POST", RequestFormat = WebMessageFormat.Xml, BodyStyle = WebMessage
        public string Test(string Name){
            /* The user on whose behalf to process the HTTP request. */
            SessionHelper.SpecifyWebOperationIdentity(HttpContextAccessor.GetInstance(), SystemU
            /* Return the result. */
            return Name;
        }
    }
}
```

Click [*Publish*] on the Source Code Designer's toolbar to apply the changes on the database level.

4 Register the web service

1. Create a `UsrEncodingService.svc` file in the `..\Terrasoft.WebApp\ServiceModel` directory.
2. Add the following record to the `UsrEncodingService.svc` file.

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

   The `Service` attribute contains the full name of the web service class and specifies the namespace.
3. Save the file.

5. Register a non-standard text encoding

1. Add `<customBinding>` section to the `..\Terrasoft.WebApp\ServiceModel\http\bindings.config` file.
2. Add the following **attributes** to the `<customBinding>` file section:
   - Set the `name` attribute of the `<binding>` element to "ISO88591Encoding."
   - Set the `encoding` attribute of the `<customTextMessageEncoding>` element to "ISO-8859-1."
   - Set the `manualAddressing` attribute of the `<httpTransport>` element to `true`.

`..\Terrasoft.WebApp\ServiceModel\http\bindings.config` file

```
<bindings>
    ...
    <customBinding>
        <binding name="ISO88591Encoding">
            <customTextMessageEncoding encoding="ISO-8859-1" />
            <httpTransport manualAddressing="true"/>
        </binding>
        ...
    </customBinding>
</bindings>
```

3. Save the file.
4. Add an identical record to the `..\Terrasoft.WebApp\ServiceModel\https\bindings.config` file.

6. Enable both HTTP and HTTPS support for the web service

1. Add the following record to the `..\Terrasoft.WebApp\ServiceModel\http\services.config` file.

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

```
<services>
    ...
    <service name="Terrasoft.Configuration.UsrEncodingServiceNamespace.UsrEncodingService">
        <endpoint name="UsrEncodingServiceEndPoint"
            address=""
            binding="customBinding"
            bindingConfiguration="ISO88591Encoding"
            behaviorConfiguration="RestServiceBehavior"
            bindingNamespace="http://Terrasoft.WebApp.ServiceModel"
            contract="Terrasoft.Configuration.UsrEncodingServiceNamespace.UsrEncodingService"
    </service>
</services>
```

The `binding` attribute contains the "`<customBinding>`" value that must match the name of the `<customBinding>` file section that registers the character encoding.

The `bindingConfiguration` attribute contains the name of the registered character encoding. Must match the value of the `<binding>` element's `name` attribute specified on the previous step.

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

7. Enable access to the web service for all users

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

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

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

2. Add the relative web service path to the `value` attribute of the `<appSettings>` element's `AllowedLocations` key in the `..\Terrasoft.WebApp\Web.config` file.

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

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

3. Save the file.

8. Restart Creatio in IIS

Restart Creatio in IIS to apply the changes.

Outcome of the example

Use Postman request testing tool to view the outcome of the example. Learn more about working in Postman in the official Postman documentation. Learn more about using Postman to query Creatio in a separate article: Working with requests in Postman. Learn more about using Postman to call a web service in a separate article: Call a custom web service from Postman.

To **view the outcome of the example**, execute a request to the `UsrEncodingService` web service.

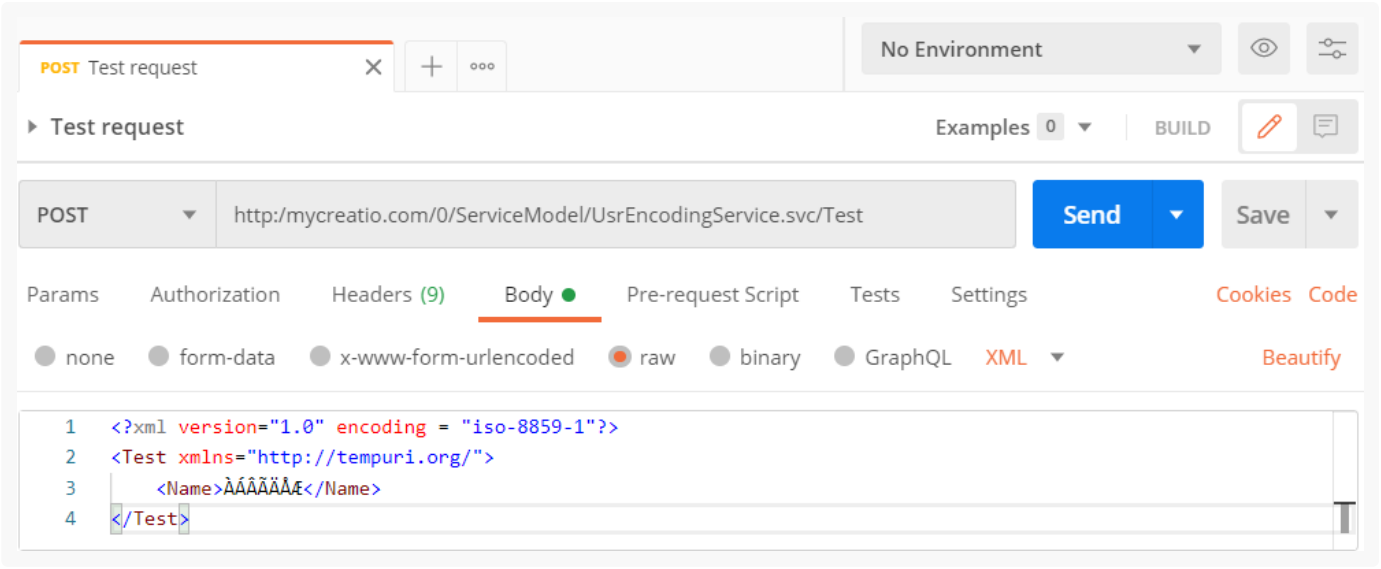
Configure the request in Postman as follows:

- Specify the `POST` request method.
- Specify the `Test` method in the request string to the `UsrEncodingService` custom web service.

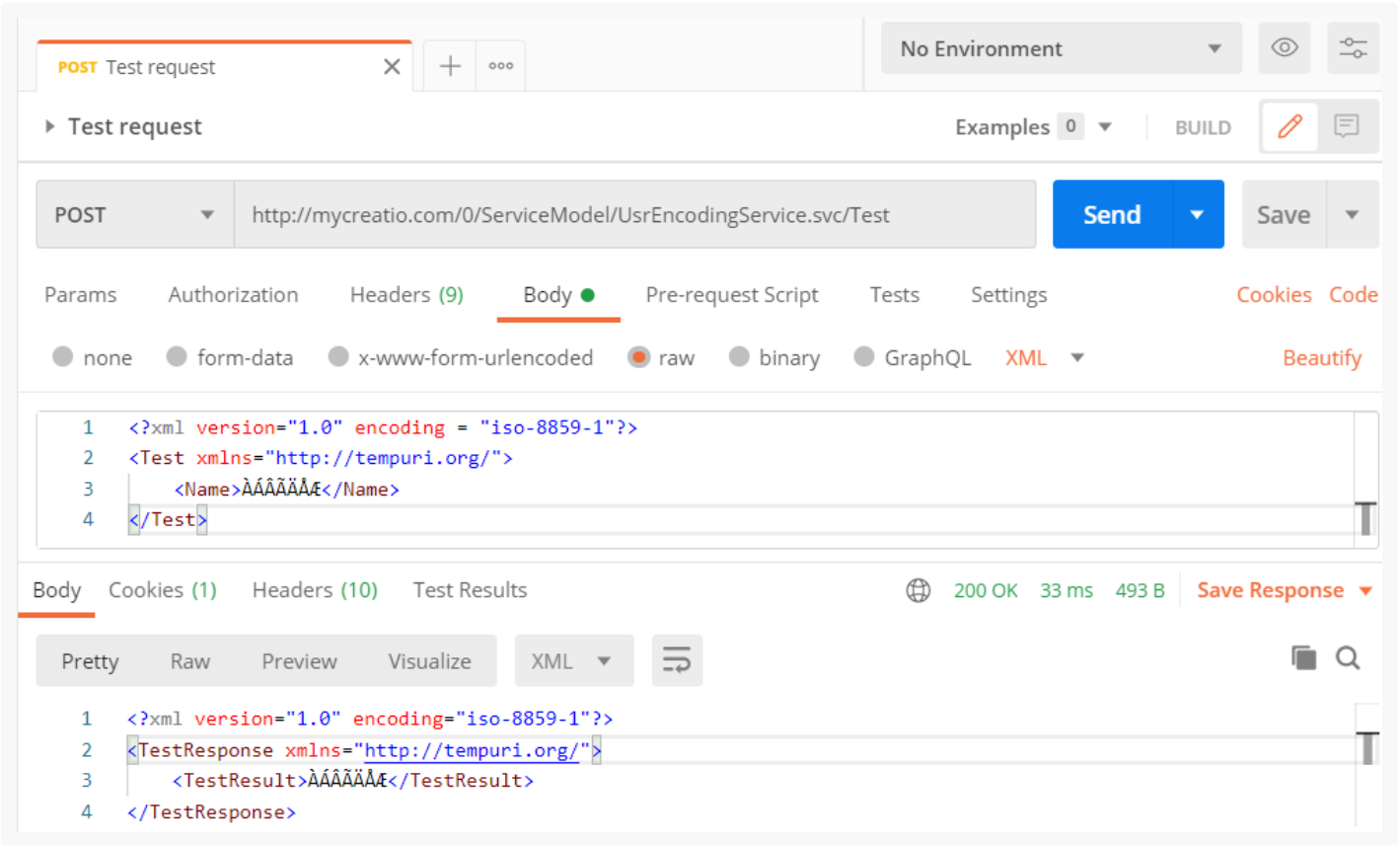
Request string to the `UsrEncodingService` custom web service

```
http://mycreatio.com/0/ServiceModel/UsrEncodingService.svc/Test
```

- Configure the **request data format** on the [*Body*] tab.
  - Set the "`raw`" option.
  - Select the "`XML`" type.
  - Fill out the body of the `POST` request. Pass the characters in the ISO-8859-1 character encoding in the request body. Learn more about the characters the ISO-8859-1 character encoding uses in Wikipedia.



As a result, you will receive a response to the `POST` request. The response format is `XML`, the code is ● **200 OK**. Postman will display the response body on the [*Body*] tab. The body will contain the value of the `Name` parameter in the ISO-8859-1 character encoding.



Call a custom web service from the front-end

Medium

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.

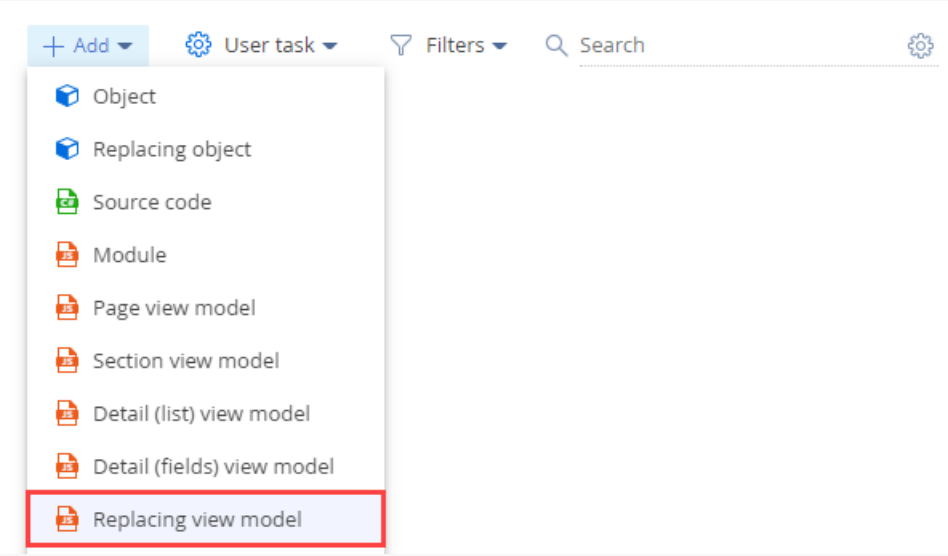
UsrCustomConfigurationService

```
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 = WebMessag
        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", N
            esq.Filters.Add(esqFilter);
```

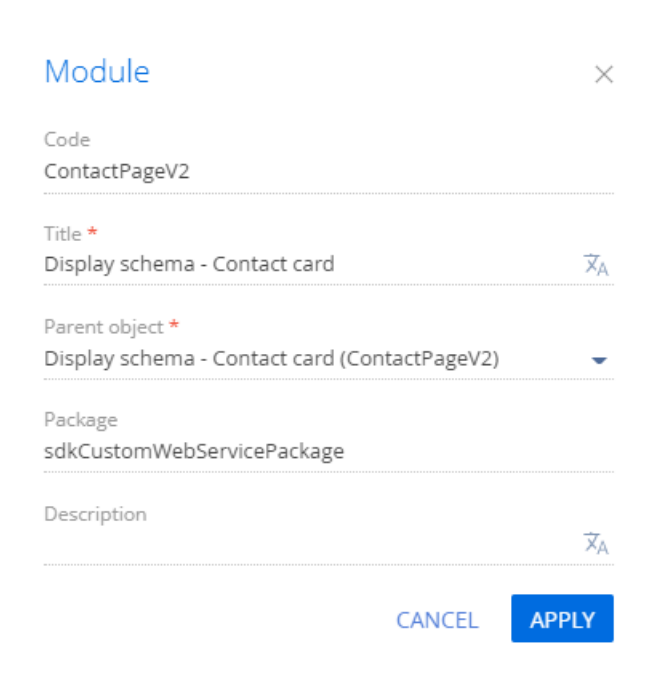
```
            /* Retrieve the query results. */
            var entities = esq.GetEntityCollection(UserConnection);
            /* If the service receives data. */
            if (entities.Count > 0)
            {
                /* Return the "Id" column value of the first query result record. */
                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`

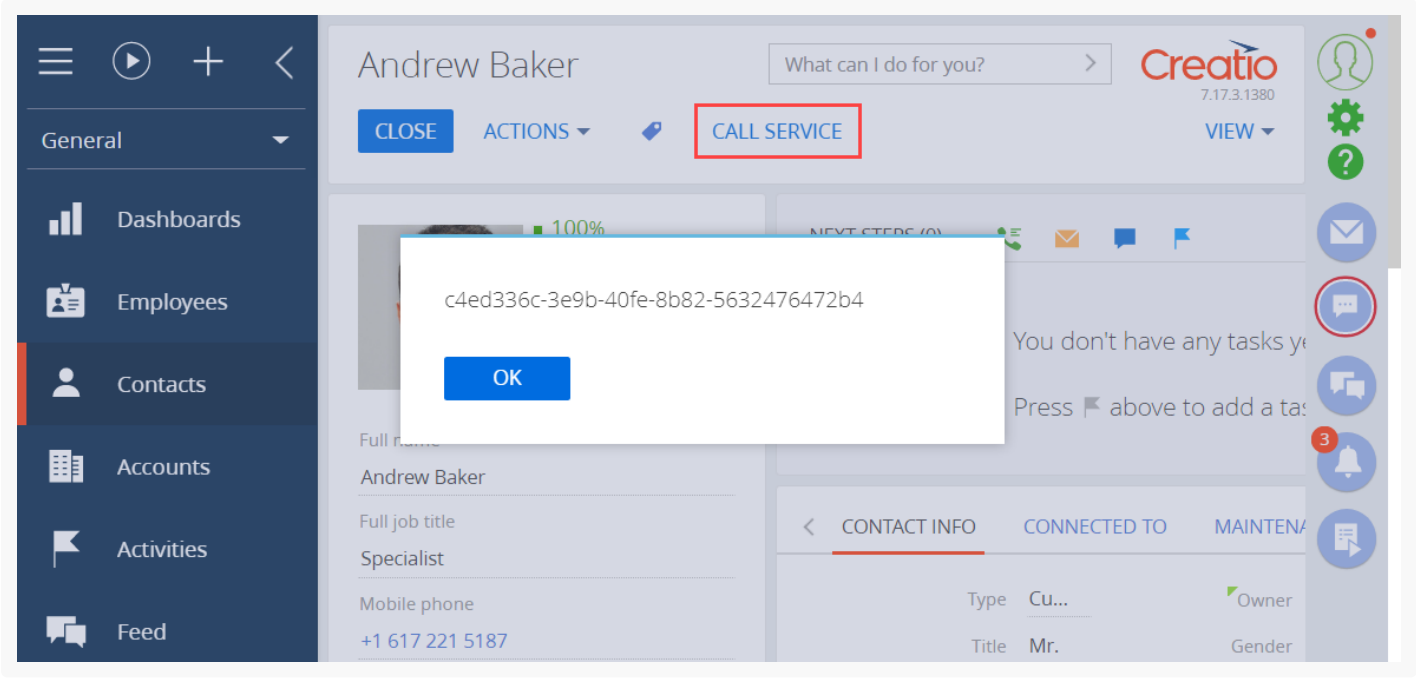
```
define("ContactPageV2", ["ServiceHelper"],
function(ServiceHelper) {
    return {
        /* The name of the record page object's schema. */
        entitySchemaName: "Contact",
        details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
        /* The methods of the record page's view model. */
        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.

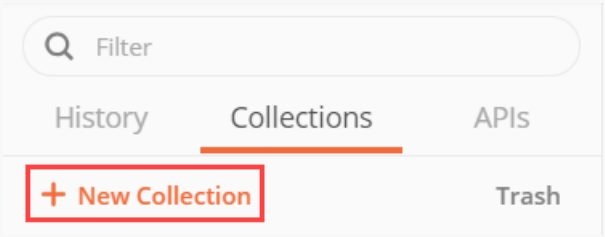
Example. Call a custom web service that uses cookie-based authentication from Postman.

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.

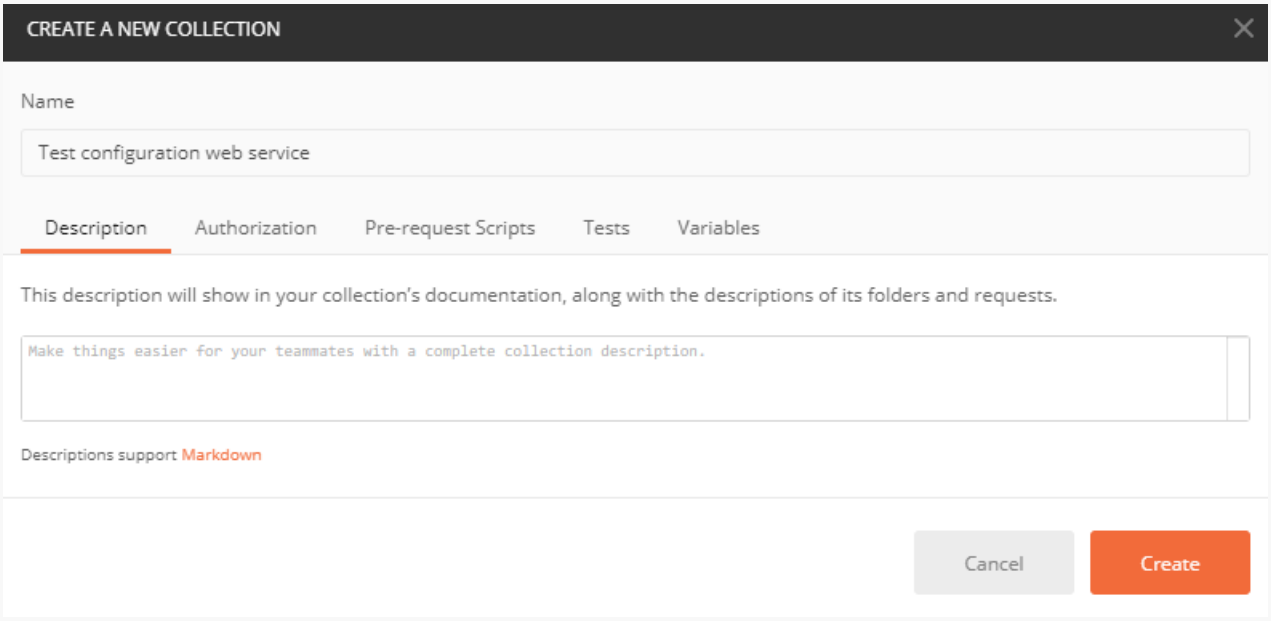
1. Create a request collection

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`."



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

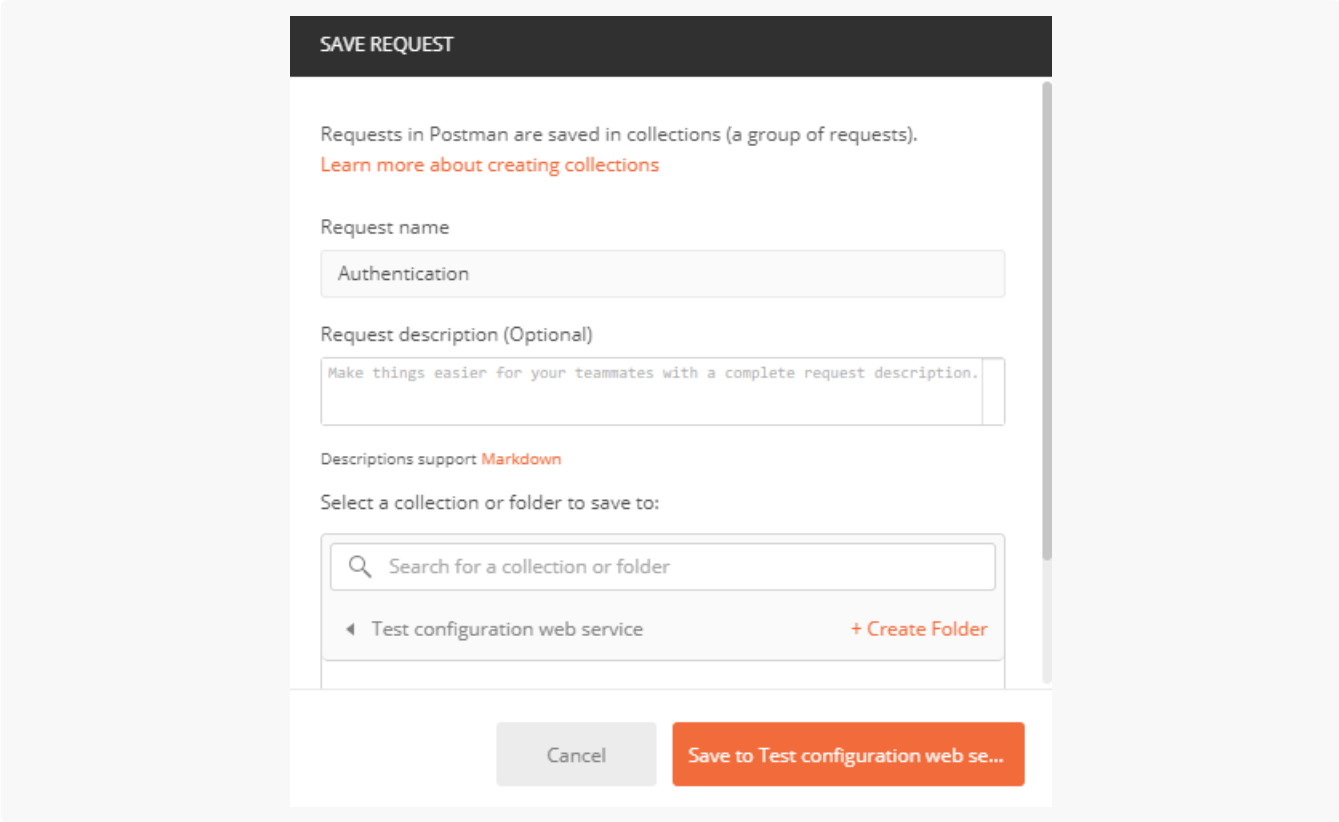
2. Set up an authentication request

1. Go to the request working area in Postman and right-click the name of the `Test configuration web service`

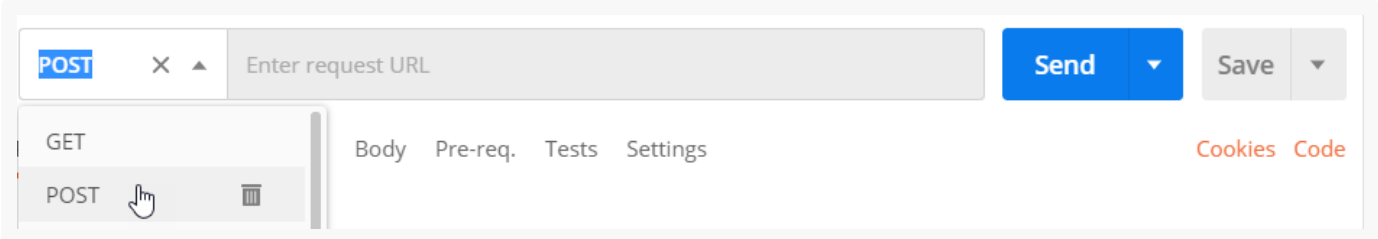
collection → [*Add request*].

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

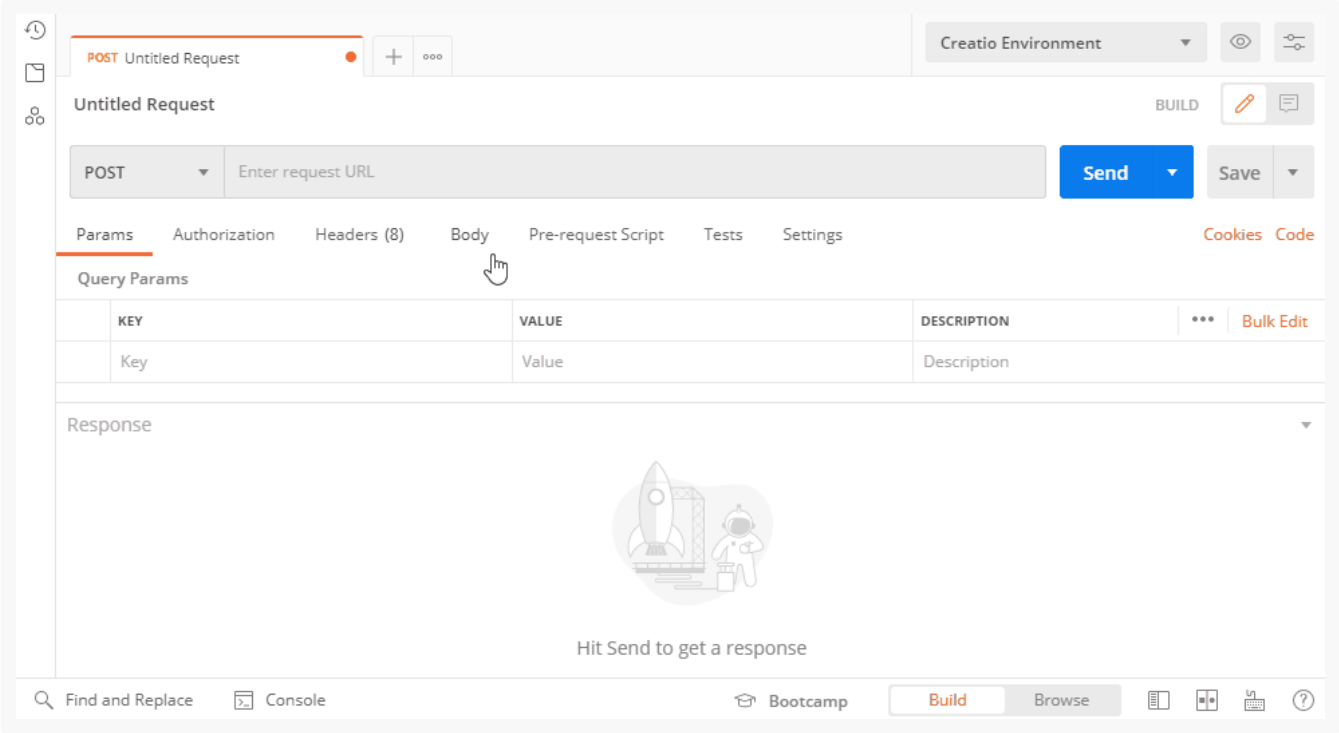
```
http://mycreatio.com/creatio/ServiceModel/AuthService.svc/Login
```

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

   a. Go to the [*Body*] tab.

   b. Set the "raw" option.

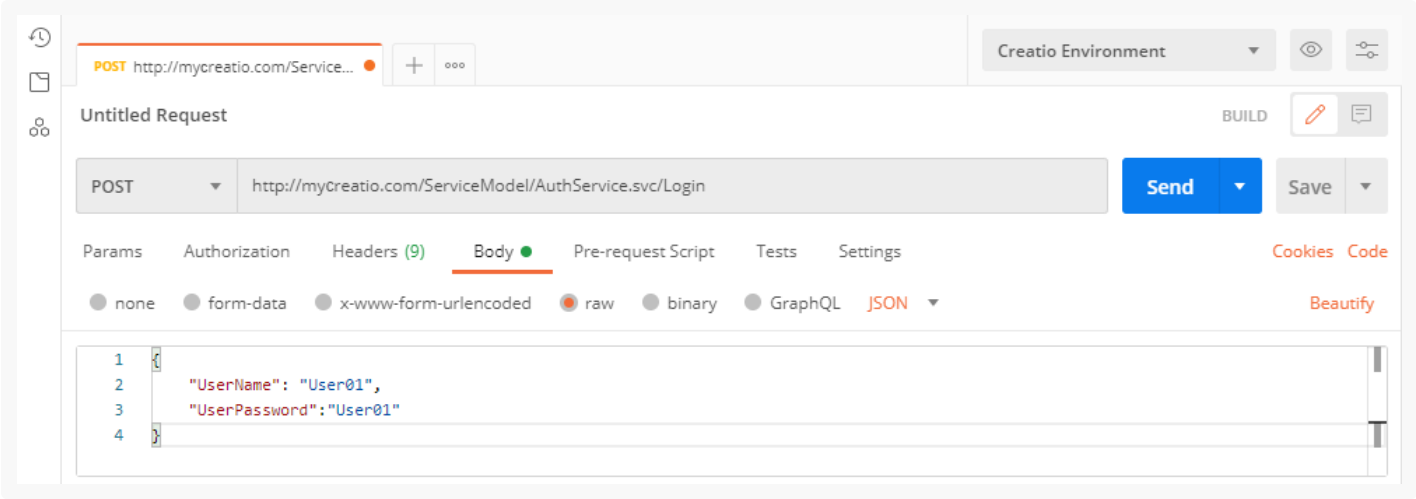
   c. Select the "JSON" type.



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

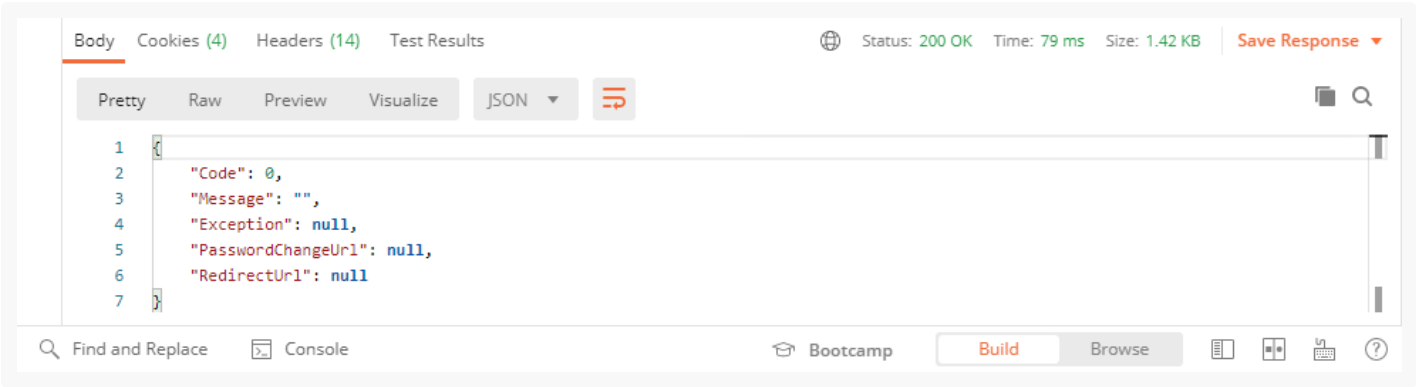
```
{
    "UserName": "User01",
    "UserPassword":"User01"
}
```



3. Execute the authentication request

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

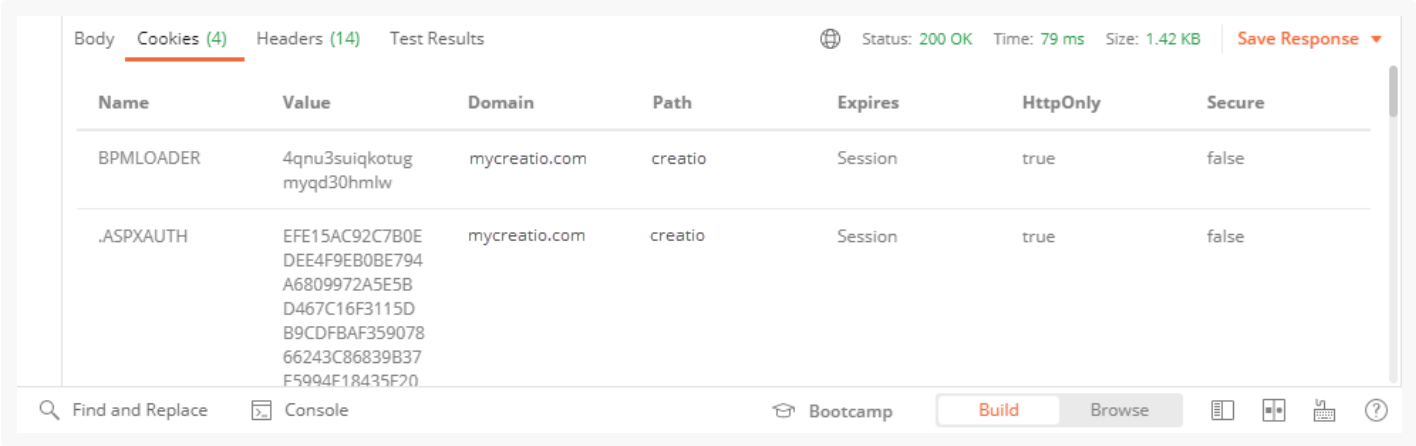
As a result, Postman will receive a response that contains a JSON object. View the response body on the Postman `Body` tab.



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.



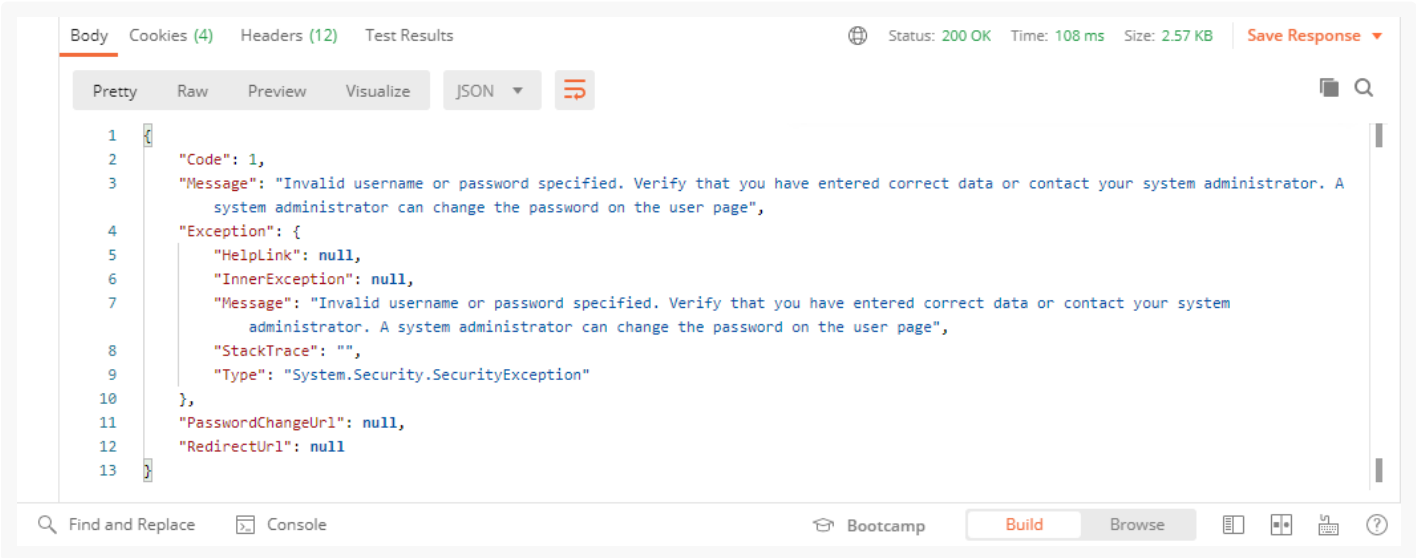
Use these cookies in further requests to Creatio services that use cookie-based authentication.

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.

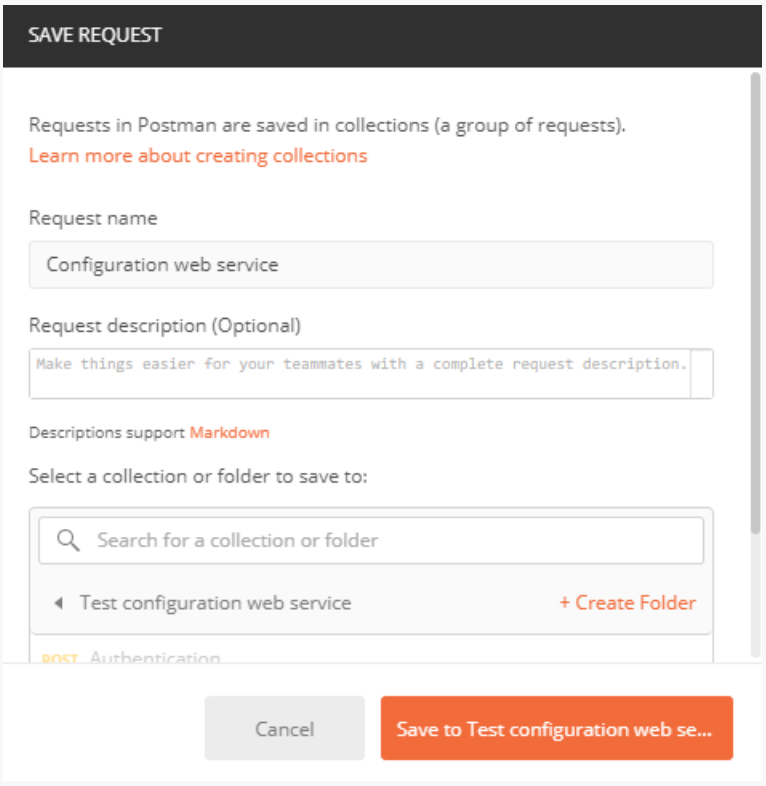


4 Set up the request to the custom web service that uses cookie-based authentication

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.

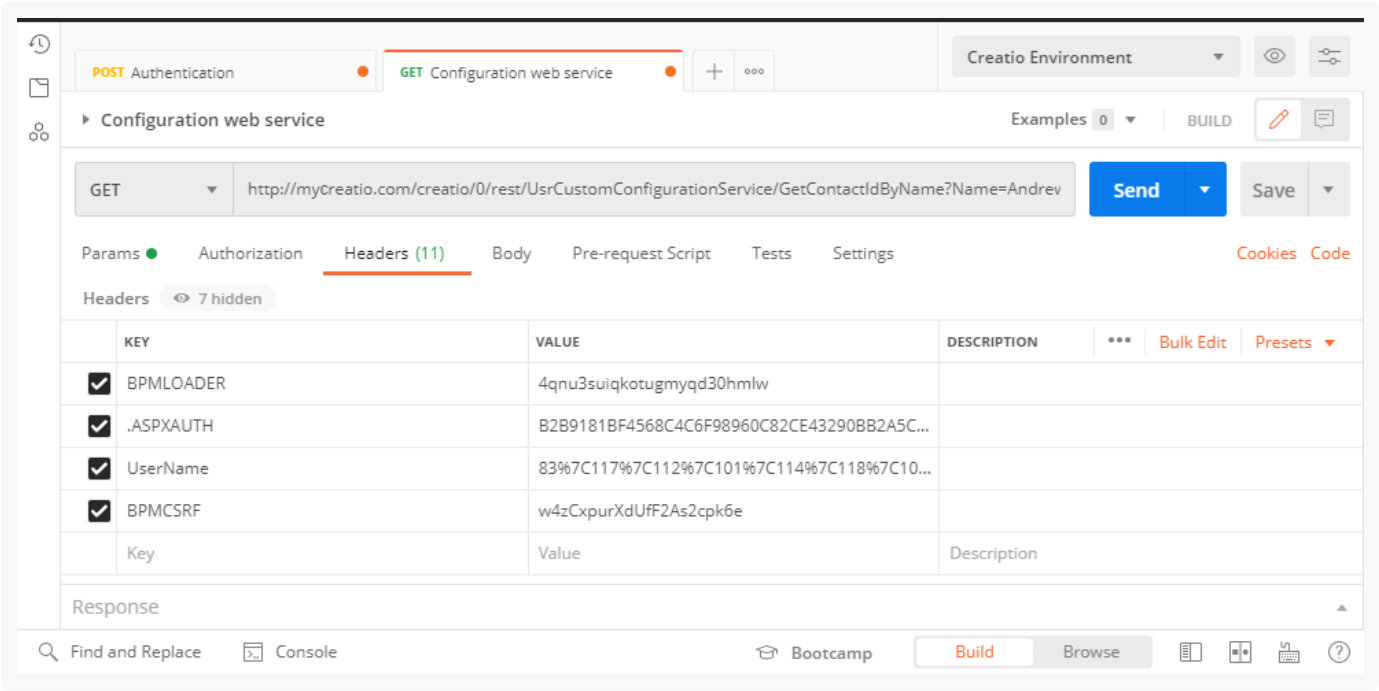
Template of the custom web service's URL

```
[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.



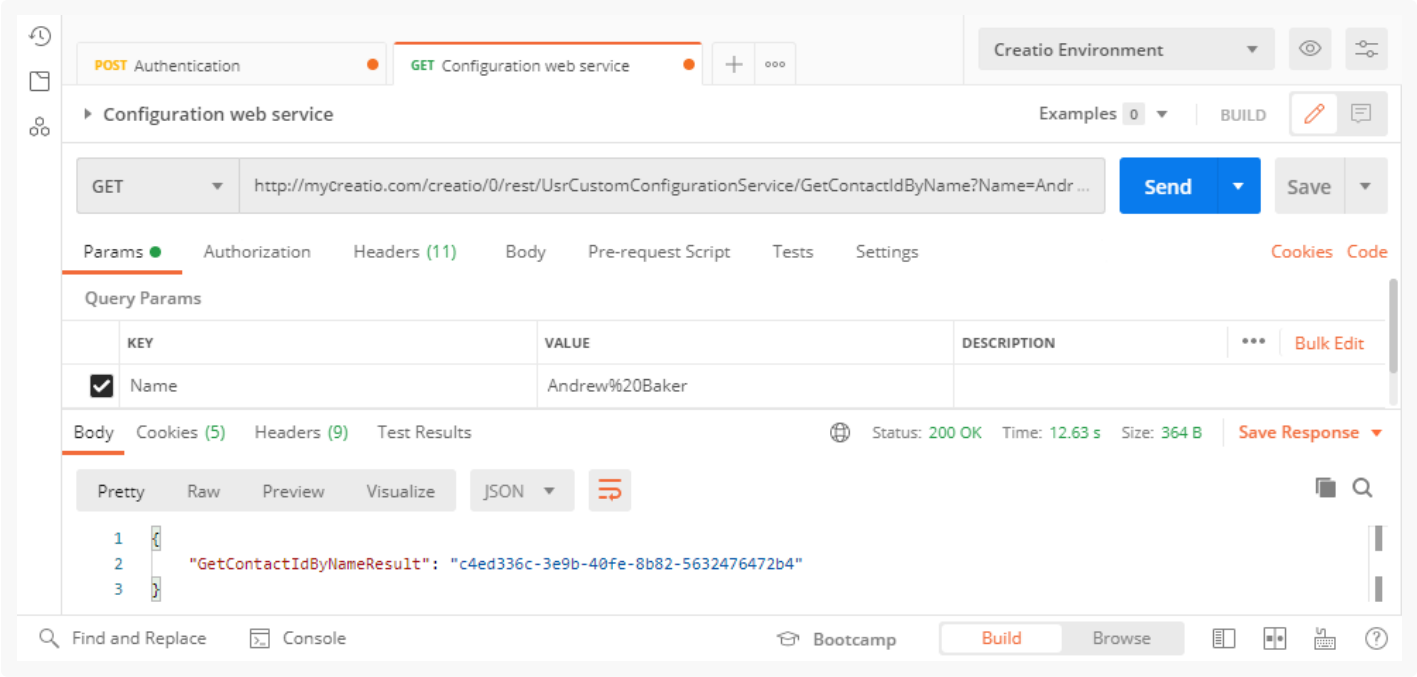
5. Execute the request to the custom web service that uses cookie-based authentication

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.



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