netgenium

API

Framework NET Genium

Content

1 Basic information

- API is a web application located on a server, available at a specific address/domain for all devices that have access to the Internet.
- API is used in cases where it is necessary to expose an endpoint that
  - returns data read from a database,
  - receives data from client applications and stores it in the database,
  - returns the contents of file attachments,
  - etc.

1.1 API with full access to the database designed to work with the NETGeniumConnection.dll library

- Each NET Genium has an implemented API that can be used from applications written in C# in conjunction with the "NETGeniumConnection.dll" library.
- This API is intended for clients that have full access to the database. The source code corresponds to this, which is written identically as if it were an application that connects to the database directly (not via web services).
- A typical example of such an application is a console application, but it can also be a desktop application, web application, etc.
- A detailed description of console applications is given in the separate article "Console applications".

1.2 API with limited access to the database

- In each NET Genium, it is possible to activate an API that returns data defined using SQL queries. These queries are configured by the NET Genium administrator in a separate application called "API".
- Each SQL query has its own unique identifier "sqlid", which clients use to display the URL request, for example "http://localhost/netgenium/api/data/{sqlid}".
- The URL can contain multiple parameters, usually used to load a specific record, for example: "http://localhost/netgenium/api/data/{sqlid}/{id}". These parameters are then accessible using the following variables:
  - #path0#: data
  - #path1#: {sqlid}
  - #path2#: {id}
- The SQL query itself can contain server function calls or use variables to limit the range of data retrieved, for example:
  - SELECT * FROM ng_apidata WHERE ng_identifier = FORMATSTRINGSQL(#path1#)
  - SELECT * FROM ng_apidata WHERE id = FORMATINTSQL(#path2#)
  - SELECT * FROM ng_apidata WHERE ng_createdon < FORMATDATESQL(#now#)
- The API is implemented through a separate project in the "Visual Studio 2015" application and higher, or by programming the source codes of this project, followed by compiling the project into a "dll" format file.

2 Importing the application API into NET Genium

- In the first step, it is necessary to import “API.nga” into the NET Genium application, which is located in the “Install” directory of each NET Genium. The “API” application is used to define SQL queries that the resulting API will return to client devices, and to configure tokens that client devices must use to authenticate.
- On the “Tokens” view page, you need to create at least one access token. This token must be passed on securely to client devices that will retrieve data using the API.
- At least one SQL query is required on the “Endpoints” view page, for example:
    - Identifier: susers
    - SQL: SELECT * FROM susers

3 Creating an API project in Visual Studio

- In the second step, you need to create a new web application project in Visual Studio using the following steps:
    - Start Visual Studio
    - From the main menu bar, select “File / New / Project…” (Ctrl+Shift+N)
        - Project type: ASP.NET Web Application (.NET Framework)
        - Project name: api
        - Location: optional project location
        - Solution: Create new solution
        - Place solution and project in the same directory: Yes
        - Create directory for solution: No (Visual Studio 2015)
        - Framework: .NET Framework 4.7.2
        - Click the “Create” button
        - Project Template: Empty
        - Configure for HTTPS: No
        - Click the “Create” button
    - Right-click on “References”
        - select “Manage NuGet Packages…“,
        - select the tab “Installed”,
        - select the item “Microsoft.CodeDom.Providers.DotNetCompilerPlatform”,
        - click the “Uninstall” button.
    - Open the “bin” directory in the explorer via “My Computer”,
        - delete the file “Microsoft.CodeDom.Providers.DotNetCompilerPlatform.dll”,
        - delete the file “Microsoft.CodeDom.Providers.DotNetCompilerPlatform.xml”.
    - Right-click on the project name “api”,

- select “Add” / “New Item…” (Ctrl+Shift+A),
- select the folder “Installed / Visual C# / Web / General”,
- select the file type “Global Application Class”,
- check the contents of the “Name” field to have the value “Global.asax”, and confirm with the “Add” button.

- Select the “Debug” compilation mode
  - “Debug” mode generates “dll” and “pdb” format files by default
  - Thanks to the “pdb” format file, errors and breaks in the API are easily detected, because the “Stack Trace” of each error also includes the file name and line number where the break occurred
  - “Release” mode is recommended only for the final version of the tuned source codes in the API. By default, “Release” mode generates only a “dll” format file.
- Start the project – from the menu on the main bar, select “Debug” / “Start Debugging” (F5)

3.1 Adding a reference to the “NETGeniumConnection.dll” library

- Right-click on “References”, select “Add Reference…”, and select the path to the “References\NETGeniumConnection.dll” file on your computer’s disk

3.2 Editing the “Web.config” configuration file

- Replace the contents of the “Web.config” file with the following code:

```
<?xml version="1.0"?>
<configuration>

  <appSettings>
    <add key="ConnectionString"
value="driver=firebird;datasource=localhost;user=SYSDBA;password=masterkey;database=C:\Firebir
d\netgenium.fdb;charset=WIN1250;collation=WIN_CZ" />
  </appSettings>

  <system.web>
    <compilation debug="true" targetFramework="4.7.2" />
    <sessionState cookieless="false" />
    <httpRuntime targetFramework="4.7.2" />
  </system.web>

</configuration>
```

3.3 Editing the Global.asax.cs file and the Application_BeginRequest method

- Right-click on the file name “Global.asax”, and select “View Code” (F7)
- Replace the contents of the file “Global.asax.cs” with the following source code:

```
using NETGenium;
using Newtonsoft.Json.Linq;
using System;

namespace api
{
    public class Global : System.Web.HttpApplication
    {
        protected void Application_Start(object sender, EventArgs e)
        {
            NETGenium.Web.Page.Application_Start();
        }

        protected void Session_Start(object sender, EventArgs e)
        {
        }

        protected void Application_BeginRequest(object sender, EventArgs e)
        {
            ApiResponse re = new ApiResponse();

            if ((re.Request.Path[0] == "data" && (re.Request.Path.Length == 2 ||
re.Request.Path.Length == 3)) || re.Request.Path[0] == "import")
            {
                re.Request.LogUrl();
                re.Request.LogBody();

                using (DbConnection conn = DbConnection.UsingSettingsFromWebConfig(true))
                using (DbCommand cmd = new DbCommand(conn))
                    try
                    {
                        string token = null, authorization = Request.Headers["Authorization"];
                        if (authorization != null)
                        {
                            token = authorization.StartsWith("Bearer ") ?
authorization.Substring(7) : authorization;
                        }

                        if (token == null || !new DbRow("SELECT id FROM ng_apitoken WHERE
ng_apitoken = " + conn.Format(token) + " AND ng_expiration > " + conn.Format(DateTime.Today),
conn).Read())
                        {
                            throw new UnauthorizedAccessException();
                        }

                        if (re.Request.Path[0] == "data")
                        {
                            #region data
```

```
                            DbRow dr = new DbRow("SELECT ng_sql FROM ng_apidata WHERE
ng_identifier = " + conn.Format(re.Request.Path[1]), conn);
                            if (!dr.Read())
                            {
                                throw new UnauthorizedAccessException();
                            }

                            string query =
re.Request.ReplaceVariables(dr["ng_sql"].ToString(), conn);
                            if (!DbQuery.IsValidSelect(query))
                            {
                                throw new UnauthorizedAccessException(query);
                            }

                            re.LogVerbose(query);
                            re.DataTable = Data.Get(query, conn);

                            #endregion
                        }
                        else if (re.Request.Path[0] == "import")
                        {
                            #region import

                            var r = JObject.Parse(re.Request.Body);
                            if (r.ContainsKey("value"))
                            {
                                string guid = Guid.NewGuid().ToString();

                                DataSaver ds = new DataSaver("ng_data", 0, cmd);
                                ds.Add("ng_guid", guid);
                                ds.Add("ng_value", (string)r["value"]);
                                ds.Execute();

                                re.HTML = guid;
                            }
                            else
                            {
                                throw new Exception("Invalid request");
                            }

                            #endregion
                        }
                    }
                    catch (UnauthorizedAccessException ex)
                    {
                        re = new ApiResponse(HttpStatusCode.Unauthorized);
                        re.LogWarning("Application_BeginRequest", ex);
                    }
                    catch (ArgumentException ex)
                    {
                        re = new ApiResponse(HttpStatusCode.BadRequest);
                        re.LogWarning("Application_BeginRequest", ex);
                    }
                    catch (Exception ex)
                    {
                        re = new ApiResponse(HttpStatusCode.InternalServerError);
                        re.LogError("Application_BeginRequest", ex);
                    }
```

```
                re.Flush();
            }
        }

        protected void Application_AuthenticateRequest(object sender, EventArgs e)
        {
        }

        protected void Application_Error(object sender, EventArgs e)
        {
        }

        protected void Session_End(object sender, EventArgs e)
        {
        }

        protected void Application_End(object sender, EventArgs e)
        {
        }
    }
}
```

3.4 Testing the API using a console application

- Create an empty console application with a reference to the library “NETGeniumConnection.dll” see the “Console Applications” manual
- Replace the contents of the file “Program.cs” with the following source code:

```
using NETGenium;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System;
using System.IO;
using System.Text;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            NETGeniumConsole console = new NETGeniumConsole();

            using (DbConnection conn = new DbConnection())
            {
                conn.Open("http://localhost/netgenium", "Administrator",
File.ReadAllText(Config.RootPath + "password.txt"));

                string token = Data.ExecuteScalar("SELECT ng_apitoken FROM ng_apitoken WHERE
ng_expiration > " + conn.Format(DateTime.Today), conn), url, errorresponse = null;

                if (true)
                {
                    #region data

                    url = "http://localhost/netgenium/api/data/susers";

                    try
                    {
                        Uploader.LogToConsole = true;
                        Uploader.Upload(url, null, token, null, Encoding.UTF8, out
errorresponse);
                    }
                    catch (Exception ex)
                    {
                        L.W(url, ex);
                        if (errorresponse != null)
                        {
                            L.V("RESPONSE: " + errorresponse);
                        }
                    }

                    #endregion
                }

                if (false)
                {
                    #region import
```

```
                    url = "http://localhost/netgenium/api/import";

                    StringBuilder sb;

                    using (JsonWriter writer = new JsonTextWriter(new StringWriter(sb = new
StringBuilder())))
                    {
                        writer.WriteStartObject();

                        writer.WritePropertyName("value");
                        writer.WriteValue("test");

                        writer.WriteEndObject();
                    }

                    try
                    {
                        Uploader.LogToConsole = true;
                        Uploader.UploadJSon(url, null, sb.ToString(), token, null,
Encoding.UTF8, out errorresponse);
                    }
                    catch (Exception ex)
                    {
                        L.W(url, ex);
                        if (errorresponse != null)
                        {
                            L.V("RESPONSE: " + errorresponse);
                        }
                    }

                    #endregion
                }
            }

            console.Exit();
        }
    }
}
```

4 Request Serialization and Deserialization

4.1 Changing the Data Parsing Method

- Client-side serialization involves converting data from a specific class (DTO object) to a JSON-formatted text string instead of using the generic "JsonWriter" object.
- Server-side deserialization involves converting a JSON-formatted text string to a specific class (DTO object) instead of using the generic object using "JObject.Parse".

4.2 Console Application Serialization

- Replace the contents of the "import" region in the "Program.cs" file with the following source code:

```
// Original code

StringBuilder sb;

using (JsonWriter writer = new JsonTextWriter(new StringWriter(sb = new StringBuilder())))
{
    writer.WriteStartObject();

    writer.WritePropertyName("value");
    writer.WriteValue("test");

    writer.WriteEndObject();
}

Uploader.UploadJSon(url, null, sb.ToString(), token, null, Encoding.UTF8, out errorresponse);

// New code

namespace ConsoleApp1.DTO
{
    public class Request
    {
        public string data;
    }
}

var request = new DTO.Request();
request.data = "test";

Uploader.UploadJSon(url, null, JsonConvert.SerializeObject(request), token, null,
Encoding.UTF8, out errorresponse);
```

4.3 Deserialization in the server-side API

- Replace the content of the “import” region in the “Global.asax.cs” file with the following source code:

```
// Original code

var r = JObject.Parse(re.Request.Body);
if (r.ContainsKey("value"))
{
    string guid = Guid.NewGuid().ToString();

    DataSaver ds = new DataSaver("ng_data", 0, cmd);
    ds.Add("ng_guid", guid);
    ds.Add("ng_value", (string)r["value"]);
    ds.Execute();

    re.HTML = guid;
}
else
{
    throw new Exception("Invalid request");
}

// New code

var r = JsonConvert.DeserializeObject<DTO.Request>(re.Request.Body);
string guid = Guid.NewGuid().ToString();

DataSaver ds = new DataSaver("ng_data", 0, cmd);
ds.Add("ng_guid", guid);
ds.Add("ng_value", r.data);
ds.Execute();

re.HTML = guid;
```

5 Response Serialization and Deserialization

5.1 ApiResponse

- Serialization on the server side is performed automatically in the “Flush” method of the “ApiResponse” object.
- Deserialization on the client side consists of converting the response to a specific class (DTO object).

5.2 Serialization in the API on the server side

- Automatic serialization of the “ApiResponse” object occurs if none of the following properties of the “ApiResponse” object are set:
  - Dictionary – data type Dictionary<string, string>
    - Automatic conversion to JSon
    - The content of the “Content-Type” header is set to “application/json”
  - DataTable – data type DataTable
    - Automatic conversion to JSon
    - The content of the “Content-Type” header is set to “application/json”
  - DataView – data type DataView
    - Automatic conversion to JSon
    - The content of the “Content-Type” header is set to “application/json”
  - DbRow – data type DbRow
    - Automatic conversion to JSon
    - The content of the “Content-Type” header is set to “application/json”
  - FilePath, or FileName – data type string
    - Automatic conversion to a stream, so the response result is a byte stream with the content of the downloaded file
    - Header content “Content-Type” is set depending on the type of file being downloaded – for example “application/octet-stream”
    - The content of the “Content-Disposition” header is set to “attachment; filename=” plus the file name in the “FileName” variable
    - If the value of the “FileName” variable is null, the file name is automatically evaluated according to the file name in the “FilePath” variable
  - HTML – data type string
    - The response result is this string
    - The content of the “Content-Type” header is set to “text/plain; charset=utf-8”
  - XML – data type string
    - The response result is this string

        - The content of the “Content-Type” header is set to “text/xml; charset=utf-8”
    - JSon – data type string
        - The response result is this string
        - The content of the “Content-Type” header is set to “application/json”

- If none of the above properties are set, the response results in automatic serialization of the “ApiResponse” object into JSON format
    - ApiResponse is
    - The “Content-Type” header is set to “application/json”
    - Typical usage Custom

```
// Original code

ApiResponse re = new ApiResponse();
// ...
re.Flush();

// New code

public class MyResponse : ApiResponse
{
    public string error;
}

// New code – option 1

MyResponse re = new MyResponse();
// ...
re.Flush();

// New code – option 2

ApiResponse re = new ApiResponse();
// ...

if (true)
{
    re = new MyResponse();
}

re.Flush();
```

5.3 Deserialization in a console application

```
// Original code

Uploader.UploadJSon(url, null, JsonConvert.SerializeObject(request), token, null,
Encoding.UTF8, out errorresponse);

// New code

public class MyResponse : ApiResponse
{
    public string error;
}

string response = Uploader.UploadJSon(url, null, JsonConvert.SerializeObject(request), token,
null, Encoding.UTF8, out errorresponse);

var r = JsonConvert.DeserializeObject<MyResponse>(response);
```