netgenium

API

Framework NET Genium

Obsah

1 Základní informace

- API je webová aplikace umístěná na webovém serveru, dostupná na konkrétní adrese/doméně pro všechna zařízení, která mají přístup k internetu.
- API se používá v případech, kdy je potřeba vystavit endpoint, který
  - vrací data načtená z databáze,
  - přijímá data z klientských aplikací a ukládá je do databáze,
  - vrací obsah souborových příloh,
  - apod.

1.1 API s plným přístupem do databáze určené pro práci s knihovnou NETGeniumConnection.dll

- Každé NET Genium má implementované API, které je možné volat z aplikací napsaných v jazyce C# ve spojení s knihovnou „NETGeniumConnection.dll“.
- Toto API je určené pro klienty, kteří mají plný přístup do databáze. Tomu odpovídá i zdrojový kód, který je napsaný identicky, jako by šlo o aplikaci, která se do databáze připojuje napřímo (ne přes webové služby).
- Typickým příkladem takové aplikace je konzolová aplikace, ale může to být i desktopová aplikace, webová aplikace apod.
- Detailní popis psaní konzolových aplikací je uveden v samostatné příručce „Konzolové aplikace“.

1.2 API s omezeným přístupem do databáze

- V každém NET Geniu je možné aktivovat API, které vrací data definovaná pomocí SQL dotazů. Tyto dotazy konfiguruje administrátor NET Genia v samostatné aplikaci s názvem „API“.
- Každý SQL dotaz má svůj unikátní identifikátor „sqlid“, který klienti využívají k sestavení URL požadavku, například „http://localhost/netgenium/api/data/{sqlid}“.
- URL požadavek může obsahovat více parametrů, obvykle sloužících k načtení konkrétního záznamu, například: „http://localhost/netgenium/api/data/{sqlid}/{id}“. Tyto parametry jsou pak přístupné pomocích následujících proměnných:
  - #path0#: data
  - #path1#: {sqlid}
  - #path2#: {id}
- Samotný SQL dotaz může obsahovat volání serverových funkcí nebo používat proměnné, aby bylo možné omezit rozsah načtených dat, například:
  - 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#)
- API se vyvíjí prostřednictvím samostatného projektu v aplikaci „Visual Studio 2015“ a vyšší, resp. programováním zdrojových kódů tohoto projektu, a následnou kompilací projektu do souboru formátu „dll“.

2 Import aplikace API do NET Genia

- V prvním kroku je nutné importovat do NET Genia aplikaci „API.nga“, která je umístěná v adresáři „Install“ každého NET Genia. Aplikace „API“ slouží pro definici SQL dotazů, které bude výsledné API vracet klientským zařízením, a pro konfiguraci tokenů, kterými se musí klientská zařízení autorizovat.
- Na nahlížecí stránce „Tokeny“ je potřeba vytvořit alespoň jeden přístupový token. Tento token je nutné předat zabezpečenou cestou klientským zařízením, která budou data pomocí API načítat.
- Na nahlížecí stránce „Endpointy“ je potřeba definovat alespoň jeden SQL dotaz, například:
    - Identifikátor: susers
    - SQL: SELECT * FROM susers

3 Vytvoření projektu API ve Visual Studiu

- Ve druhém kroku je nutné vytvořit nový projekt webové aplikace ve Visual Studiu pomocí následujících kroků:
    - Spustit Visual Studio
    - Z menu na hlavní liště zvolit „File / New / Project…“ (Ctrl+Shift+N)
        - Project type: ASP.NET Web Application (.NET Framework)
        - Project name: api
        - Location: volitelné umístění projektu
        - Solution: Create new solution
        - Place solution and project in the same directory: Ano
        - Create directory for solution: Ne (Visual Studio 2015)
        - Framework: .NET Framework 4.5.2
        - Kliknout na tlačítko „Create“
        - Project Template: Empty
        - Configure for HTTPS: Ne
        - Kliknout na tlačítko „Create“
    - Kliknout pravým tlačítkem na „References“,
        - zvolit „Manage NuGet Packages…“,
        - vybrat záložku „Installed“,
        - vybrat položku „Microsoft.CodeDom.Providers.DotNetCompilerPlatform“,
        - kliknout na tlačítko „Uninstall“.
    - Přes „Tento počítač“ otevřít v průzkumníkovi adresář „bin“,
        - smazat soubor „Microsoft.CodeDom.Providers.DotNetCompilerPlatform.dll“,
        - smazat soubor „Microsoft.CodeDom.Providers.DotNetCompilerPlatform.xml“.
    - Kliknout pravým tlačítkem na název projektu „api“,
        - zvolit „Add“ / „New Item…“ (Ctrl+Shift+A),
        - vybrat složku „Installed / Visual C# / Web / General“,
        - vybrat typ souboru „Global Application Class“,

- zkontrolovat obsah pole „Name“, aby měl hodnotu „Global.asax“, a potvrdit tlačítkem „Add“.

- Zvolit režim kompilace „Debug“
  - Režim „Debug“ ve výchozím nastavení generuje soubory formátu „dll“ a „pdb“
  - Díky souboru formátu „pdb“ se snadno odhalují chyby a přerušení v API, protože součástí „Stack Trace“ každé chyby je i název souboru a číslo řádky, na které došlo k přerušení
  - Režim „Release“ se doporučuje až pro finální verzi vyladěných zdrojových kódů v API. Ve výchozím nastavení režim „Release“ generuje pouze soubor formátu „dll“.
- Spustit projekt – z menu na hlavní liště zvolit „Debug“ / „Start Debugging“ (F5)

3.1 Přidání reference na knihovnu „NETGeniumConnection.dll“

- Kliknout pravým tlačítkem na „References“, zvolit „Add Reference…“, a vybrat cestu k souboru „References\NETGeniumConnection.dll“ na disku počítače

3.2 Úprava konfiguračního souboru „Web.config“

- Obsah souboru „Web.config“ vyměnit za následující kód:

```
<?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="4.5.2" />
    <sessionState cookieless="false" />
    <httpRuntime targetFramework="4.5.2" />
  </system.web>

</configuration>
```

3.3 Úprava souboru Global.asax.cs a metody Application_BeginRequest

- Kliknout pravým tlačítkem na název souboru „Global.asax“, a zvolit „View Code“ (F7)
- Obsah souboru „Global.asax.cs“ vyměnit za následující zdrojový kód:

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

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

        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 Testování API pomocí konzolové aplikace

- Vytvořit prázdnou konzolovou aplikaci s referencí na knihovnu „NETGeniumConnection.dll“ viz příručka „Konzolové aplikace“
- Obsah souboru „Program.cs“ vyměnit za následující zdrojový kód:

```
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 Serializace a deserializace requestu

4.1 Změna způsobu parsování dat

- Serializace na straně klienta spočívá v převodu dat z konkrétní třídy (DTO objektu) do textového řetězce ve formátu JSON namísto použití obecného objektu „JsonWriter“.
- Deserializace na straně serveru spočívá v převodu textového řetězce ve formátu JSON do konkrétní třídy (DTO objektu) namísto použití obecného objektu pomocí „JObject.Parse“.

4.2 Serializace v konzolové aplikaci

- Obsah regionu „import“ v souboru „Program.cs“ vyměnit za následující zdrojový kód:

```
// Původní kód

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);

// Nový kód

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 Deserializace v API na straně serveru

- Obsah regionu „import“ v souboru „Global.asax.cs“ vyměnit za následující zdrojový kód:

```
// Původní kód

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");
}

// Nový kód

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 Serializace a deserializace response

5.1 ApiResponse

- Serializace na straně serveru je prováděna automaticky v metodě „Flush“ objektu „ApiResponse“.
- Deserializace na straně klienta spočívá v převodu response do konkrétní třídy (DTO objektu).

5.2 Serializace v API na straně serveru

- K automatické serializaci objektu „ApiResponse“ dochází v případě, že není nastavena ani jedna z následujících properties objektu „ApiResponse“:
  - Dictionary – datový typ Dictionary<string, string>
    - Automatická konverze do JSon
    - Obsah hlavičky „Content-Type“ je nastaven na „application/json“
  - DataTable – datový typ DataTable
    - Automatická konverze do JSon
    - Obsah hlavičky „Content-Type“ je nastaven na „application/json“
  - DataView – datový typ DataView
    - Automatická konverze do JSon
    - Obsah hlavičky „Content-Type“ je nastaven na „application/json“
  - DbRow – datový typ DbRow
    - Automatická konverze do JSon
    - Obsah hlavičky „Content-Type“ je nastaven na „application/json“
  - FilePath, případně FileName – datový typ string
    - Automatická konverze do streamu, takže výsledkem response je datový proud bajtů s obsahem stahovaného souboru
    - Obsah hlavičky „Content-Type“ je nastaven v závislosti na typu stahovaného souboru – například „application/octet-stream“
    - Obsah hlavičky „Content-Disposition“ je nastaven na „attachment; filename=“ plus název souboru v proměnné „FileName“
    - Pokud je hodnota proměnné „FileName“ null, je název souboru vyhodnocen automaticky podle názvu souboru v proměnné „FilePath“
  - HTML – datový typ string
    - Výsledkem response je právě tento string
    - Obsah hlavičky „Content-Type“ je nastaven na „text/plain; charset=utf-8“
  - XML – datový typ string
    - Výsledkem response je právě tento string
    - Obsah hlavičky „Content-Type“ je nastaven na „text/xml; charset=utf-8“

- JSon – datový typ string
  - Výsledkem response je právě tento string
  - Obsah hlavičky „Content-Type“ je nastaven na „application/json“

- Pokud není nastavena hodnota ani jedné z výše uvedených properties, je výsledkem response automatická serializace objektu „ApiResponse“ do formátu JSON
  - ApiResponse je
  - Obsah hlavičky „Content-Type“ je nastaven na „application/json“
  - Typické použití Vlastní

```
// Původní kód

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

// Nový kód

public class MyResponse : ApiResponse
{
    public string error;
}

// Nový kód – varianta 1

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

// Nový kód – varianta 2

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

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

re.Flush();
```

5.3 Deserializace v konzolové aplikaci

```
// Původní kód

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

// Nový kód

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);
```