**INSTYTUT PODSTAW INFORMATYKI**

**POLSKA AKADEMIA NAUK**

**STUDIA PODYPLOMOWE**

**PROGRAMOWANIE NA PLATFORMIE .NET**

Łukasz Dejko

Global Repository – aplikacja webowa do komunikacji

z repozytorium kont, serwerów i systemów.

Praca wykonana pod kierunkiem:

Pana Wiktora Wandachowicza

Warszawa dnia 11/01/2020

**OŚWIADCZENIE**

Oświadczam, że złożoną pracę końcową pod tytułem: Global Repository – aplikacja webowa   
do komunikacji z repozytorium kont, serwerów i systemów napisałem samodzielnie.

Jednocześnie oświadczam, że praca ta:

1. nie narusza praw autorskich osób trzecich w rozumieniu ustawy z dnia 4 lutego 1994 roku   
   o prawie autorskim i prawach pokrewnych (tekst jednolity Dz. U. 2006 nr 90, poz. 631 z późn. zm.);
2. nie zawiera informacji i danych uzyskanych w sposób nielegalny, a w szczególności informacji zastrzeżonych przez inny podmiot albo stanowiących tajemnicę przedsiębiorstwa;
3. nie była wcześniej przedmiotem innych procedur związanych z uzyskaniem dyplomów/świadectw lub tytułów zawodowych wyższych uczelni. Ponadto oświadczam, że treści zawarte w pisemnym egzemplarzu pracy oraz w egzemplarzu tej pracy w formie elektronicznej są jednobrzmiące.

Przyjmuję do wiadomości, że w przypadku stwierdzenia popełnienia przeze mnie czynu polegającego na przypisaniu sobie autorstwa istotnego fragmentu lub innych elementów cudzej pracy lub ustalenia naukowego, właściwy organ stwierdzi nieważność postępowania w sprawie nadania mi tytułu zawodowego (art. 193 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym, tekst jednolity Dz.U. z 2012 r., poz. 572 z późn. zm.).

………………………………

data i podpis

**List of contents**

1. **Vision** 
   1. Introduction.……………………………………………...……………………………4
   2. Positioning…………………………………………………..…………………………4
      1. Problem Statement……………………….…………………………………...……4
   3. Stakeholder Descriptions……………………….…………………………….…..……4
      1. Stakeholder Summary…………………………………………….…………..……4
      2. User Environment…………………………………...…..…………………………5
   4. Product Overview…………………………………...…………………………………5
      1. Needs and Features…………………………………...……………………………5
      2. Other Product Requirements………………………………………...……………..6
      3. Requirements not implemented…………………………………………………....6
      4. Design restrictions…………………………………………………………………7
      5. Requirements relating to the working environment………………………………..7
   5. Database design………………………………………………………………………..7
      1. Main tables of the repository………………………………………………………7
      2. Audit tables………………………………………………………………………...8
      3. Reconciliation tables for web-based communication data…………………………8
2. **Description of the project assumptions**………………………………..……………..8
   1. Source code……………………………………………………………………...……..8
   2. Objective of the project………………………………………………………………...9
   3. Identification of market needs………………………………………………………….9
   4. Technical data and used technologies. ………………………………………...……..10
3. **User documentation**……………………………………………………………………..10
   1. Description of the login proces and create / audit account…………………………..10
      1. Documentation for Web Api users………………………………………………..10
      2. Audit of all changes in the accounts………………………………………………12
      3. Audit of all changes in the accounts………………………………………………13
   2. Description of methods used in Web communication..………………………………15
      1. AdminController ……………………………………………………………....…15
         1. CreateNewUser ……………………………………………...………...…15
      2. AuditController ……………………………………………………………......…15
         1. GetAuditServersCount ………………………………………………...…15
         2. GetAuditSystemsCount………………………………………………..…16
         3. GetAuditAccounts…………………………...………………………...…16
         4. GetAuditServers………………………..……………………………...…17
         5. GetAuditServers…………………………..…………………………...…17
         6. GetAuditSystems………………………………………….…………...…18
      3. GlobalRepositoryDataController ..………………………………………………19
         1. CreateNewAccount ……...……………………………………………….19
         2. GetGlobalRespositoryData……………………………………………….19
         3. GetGlobalRespositoryDatCount….............................................................20
         4. DeleteAccount…………………………………………………………....20
      4. GrTablesController…………………………………………………………….....21
         1. GetGlobalRespositoryDocumentationCount…………………………......21
         2. GetGlobalRespositoryDocumentation……………………………….......21
         3. GetSpecyficInformationDocumentation…................................................22
      5. ReconciliationController........................................................................................23
         1. ReconAccounts...........................................................................................23
         2. ReconServers…..........................................................................................23
         3. ReconSystems ............................................................................................24
4. **Conclusions**………………………………………………..................................................24
5. **Bibliography**…………………………...…………...................................................25
6. **Vision**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 18/05/2019 | 1.0 | Creation of the Vision Document | Łukasz Dejko |
| 01/09/2019 | 1.1 | Change of the database environment. | Łukasz Dejko |
| 02/12/2019 | 1.2 | Adding a database schema and updating the vision document. | Łukasz Dejko |
| 17/12/2019 | 1.3 | Final version of web api | Łukasz Dejko |
| 09/01/2020 | 1.4 | Final version of documentation | Łukasz Dejko |

* 1. **Introduction**

The document concerns a project implemented at postgraduate studies at the Polish Academy of Sciences in the field of: "Programming on the .NET platform". This document is used to present the system of web communication with the repository of accounts that stores information about : systems, servers and accesses. This system will be a bridge between other systems and will only manage this information and will be used for statistical and change reporting purposes. In this project I will use the latest technologies available in Microsoft.

* 1. **Positioning**
     1. **Problem Statement**

|  |  |
| --- | --- |
| The problem of | The problem with current interfaces is the performance and ease of implementation of changes. |
| affects | The users of this interface will be business users who want to manage data by transferring and receiving data using json files. |
| a successful solution would be | The assumption of this project will be to design a very flexible interface in which changes can be made quickly and with a small contribution from the development team. |

* 1. **Stakeholder Descriptions**
     1. **Stakeholder Summary**

| **Name** | **Description** | **Responsibilities** |
| --- | --- | --- |
| Programmers | Programmers who have the skills to implement such systems and data processing. | 1. They will read the user documentation before using the system. 2. In order to ensure proper functioning of the system, it is necessary to create a login and password that will be used during the authorization of queries. 3. During the implementation of the system, they will test all new solutions on the development environment. |

* + 1. **User Environment**

The web communication environment is so flexible that it will be possible to use this interface on all platforms, but it will be mainly Microsoft environment. The user will be able to use the application through the mechanisms of web pages. There are no limitations as to the choice of the Internet browser.

The environment in which the web communication interface will be built is .NET Core, and the main programming language to be used on the beckend side will be C# and on the database side TSQL. The user documentation will be available Web Api.

Environment version : ASP.NET Core 2.2

Database environment version : SQL Server 2017

Database model mapping : Entity Framework Core

* 1. **Product Overview**
     1. **Needs and Features**

Priority markings:

1 - the minimum that the created application will have

2 - functionalities that are expected to be implemented

3 - functionalities that will be implemented after the production implementation of the application

|  |  |  |  |
| --- | --- | --- | --- |
| **Need** | **Priority** | **Features** | **Planned Release** |
| Database design (tables, constraints) | 1 |  | December 2019 |
| Create triggers that support database events | 1 |  | December 2019 |
| Implementation of the database model in the interface project | 1 |  | December 2019 |
| Creation of the login and authorization process | 1 |  | December 2019 |
| Creating user documentation accessible from the web api | 1 |  | December 2019 |
| Saving Web Api errors and user activity to a log file. | 1 |  | December 2019 |
| The ability to create accounts from the administrator level. | 1 |  | December 2019 |
| Error and exception handling for Web Api | 1 |  | December 2019 |
| Multithreaded query processing in web Api | 1 |  | December 2019 |

* + 1. **Other Product Requirements**

|  |  |  |
| --- | --- | --- |
| **Requirement** | **Priority** | **Planned Release** |
| User changes in data reconciliation will be saved in audit tables (sql server) | 1 | December 2019 |
| Simultaneous operation of multiple users at the same time in Web Api | 1 | December 2019 |
| Possibility of downloading a report on errors in data reconciliation | 1 | December 2019 |
| Implementation of the Swagger interface for Web Api tests | 1 | December 2019 |
| Creation of a database design with the possibility of easy implementation of these solutions by the built-in interface Visual Studio 2019 | 1 | December 2019 |
| Web page Global Repository | 3 |  |

* + 1. **Requirements not implemented**

|  |
| --- |
| **Requirement** |
| Connection to Active Directory |
| Sending e-mail correspondence confirming the changes. |

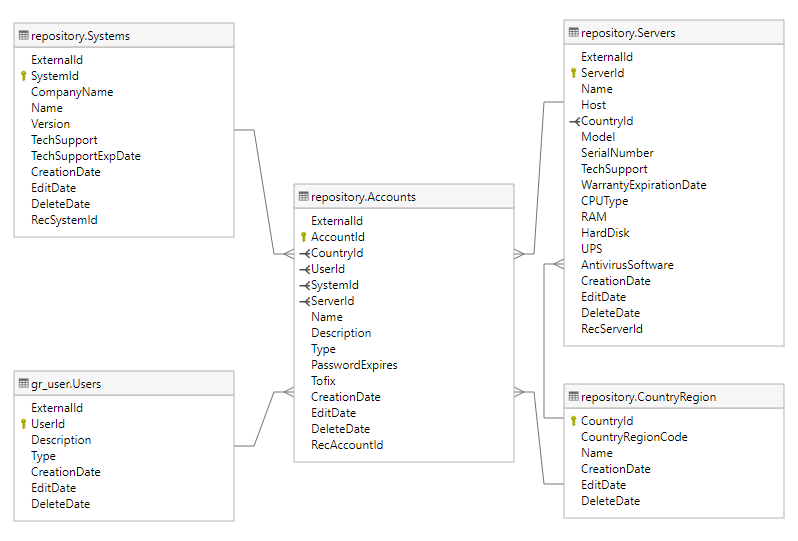
* + 1. **Design restrictions**

The main obstacle is the time in which the project will be implemented. The implementation of functionalities with priority 3 may not be possible within the set time frame. The second limitation is the small experience of the person who designed and implemented the project. This may cause the work to be incomplete. A significant difficulty may also be the fact that the creator of this document runs several other projects at the same time. This fact may lead to unsystematic reporting on the progress of work and a greater number of errors, especially in the testing phase.

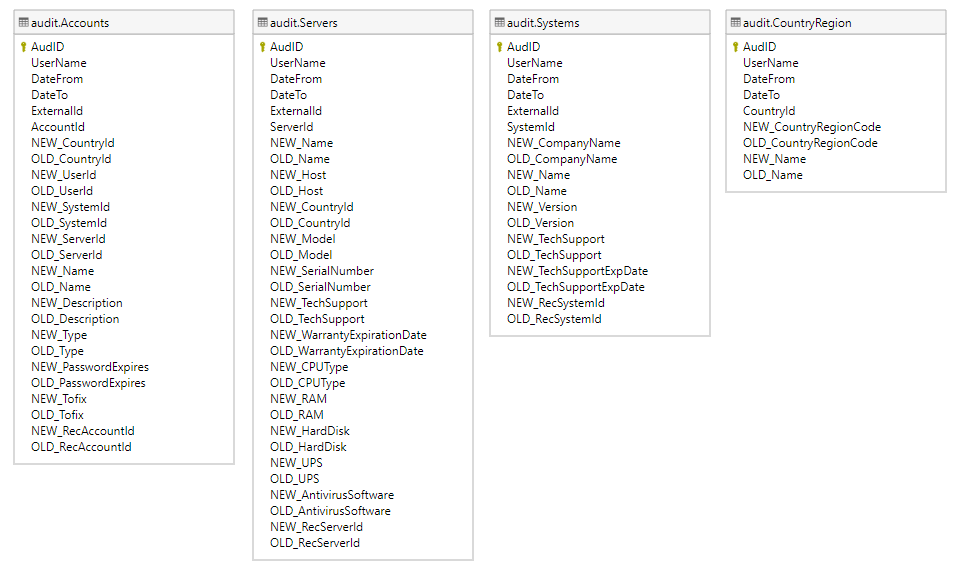
* + 1. **Requirements relating to the working environment**

These requirements include software maintenance costs and time costs associated with the need to monitor system operation, view logs from user traffic and solve emerging errors. It will also be necessary to undertake optimization actions in the event of a sudden increase in the number of users.

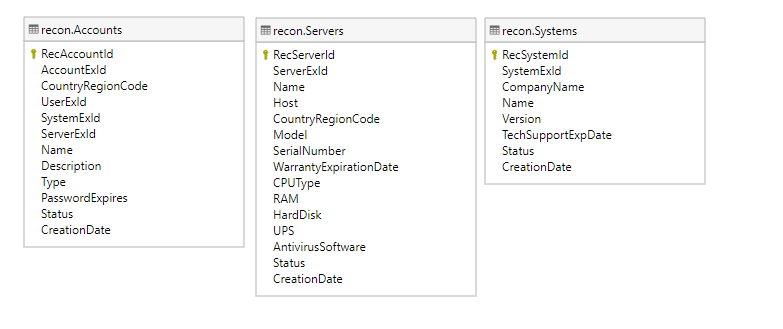
* 1. **Database design** 
     1. **Main tables of the repository**



* + 1. **Audit tables**



* + 1. **Reconciliation tables for web-based communication data**



1. **Description of the project assumptions** 
   1. **Source code - The source code of the project is available at the following links :**
2. Database project

<https://github.com/lukaszFD/IPI-PAN_WEB_API/tree/master/GlobalRepository/DB_GlobalRepository>

1. Database model in Entity Framework

<https://github.com/lukaszFD/IPI-PAN_WEB_API/tree/master/GlobalRepository/DB_ModelEFCore>

1. Web Api project

<https://github.com/lukaszFD/IPI-PAN_WEB_API/tree/master/GlobalRepository/GR_WebApi>

* 1. **Objective of the project**

1. The purpose of the project is to create a system that will be an interface of web communication with the repository of accounts storing information about: systems, servers and access. It will be a bridge between other systems and will manage only this information.
2. The second purpose of this project will be to design a very flexible interface in which changes can be implemented quickly and with little involvement of the development team.
   1. **Identification of market needs**

Global repository of accounts is a product available internally in the largest corporations that want to transparently manage user accounts where information about servers and systems is stored. The application is to be used by other programmers as a flexible tool for implementation in their environments. One of such environments is a portal to order access to specific servers or purchase new operating systems installed on these servers. Flexibility and simplicity in using this repository is the key to success in programming your own interfaces, and the main arguments are :

1. The user documentation is available in the web communication and contains a description of all the columns that are available in the global repository.
2. Programmers who will develop a global repository have a simplified task due to the description of all the methods and classes that are included in the EF database project and the WebApi project.
3. Global repository contains reports: data audit, reconciliation of data coming from external sources, errors in data reconciliation.
4. The repository is very flexible, and the implemented mechanism of paging and limiting returned data is executed in multiple threads.
5. Programmers who will analyze errors have a simplified task because every operation performed in web api is logged.
   1. **Technical data and used technologies.**

During the design and programming works I used the most popular tools provided by Microsoft, Datadeo and Github.

Installed tools :

1. Microsoft SQL Server Express (64-bit) version 14.0.1000.169
2. Microsoft Visual Studio Community 2019 version Version 16.4.1
3. Datadeo version 7.5.4
4. GitHub Desktop version Version 2.2.4

Technologies used:

1. Programming language C# version 7.3
2. Database : Entity Framework Core 2.2
3. Web Api project made in .Net Core 2.2
4. Database project made on : SQL Server 2017
5. **User documentation** 
   1. **Description of the login proces and create / audit account**

For proper operation of Web Api it is necessary to provide the password and login of registered users in the header of the query. The profile of the user who has the ability to create other user accounts is the Global Repository Administrator. Before you start using the repositories, you should apply for such access. Each company has a different interface to process such queries. Data on user types is stored on a global repository database. I prepared WPF applications for the presentation of the login process. The interface of this application presents basic functions of Web Api.

* + 1. **Creating a new account in the repository**

Creating a new account in the repository is done by sending a POST request using the Http client.

Required fields :

1. You must enter the correct address to create an account

https://localhost:44396/GlobalRepositoryData/CreateNewAccount

1. All fields for the section: account, server, system must be completed.
2. To execute a POST request, the password and login must be entered in the header.
3. If the data is processed correctly, the new account identifier will be returned.
4. Correctly entered data will be sent in json format.

{

"accountCountryRegionCode": "string",

"accountName": "string",

"accountDescription": "string",

"accountType": "string",

"accountPasswordExpires": "2020-01-09T20:14:33.207Z",

"serverName": "string",

"serverHost": "string",

"serverCountryRegionCode": "string",

"serverModel": "string",

"serverSerialNumber": 0,

"serverWarrantyExpirationDate": "2020-01-09T20:14:33.207Z",

"serverCputype": 0,

"serverRam": 0,

"serverHardDisk": "string",

"serverUps": "string",

"serverAntivirusSoftware": "string",

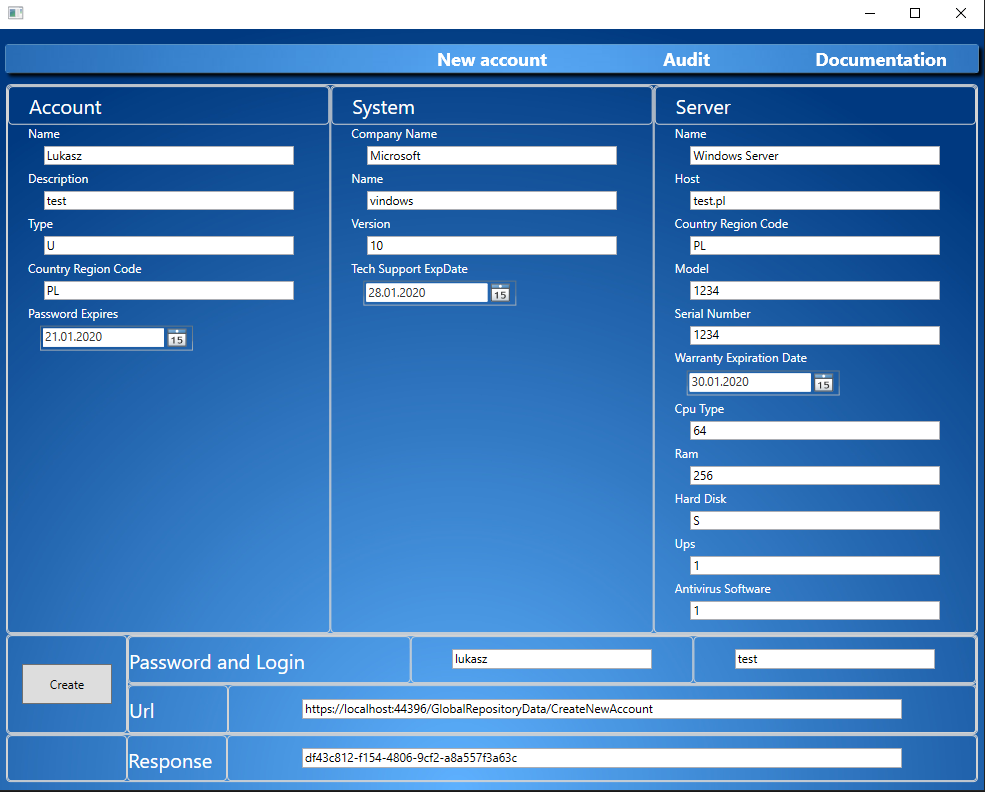
"systemCompanyName": "string",

"systemName": "string",

"systemVersion": "string",

"systemTechSupportExpDate": "2020-01-09T20:14:33.207Z"

}



* + 1. **Documentation for Web Api users**

After logging in, each user has the possibility to display documentation with a description of all fields in the Global Repository.

Required fields :

1. To execute a GET request, the password and login must be entered in the header.
2. If the data is processed correctly, documentation will be returned.
3. All GET queries require the size of pages returned and the amount of data on the page.
4. Other GET queries are used to check the number of pages.

The following is an example of an count query

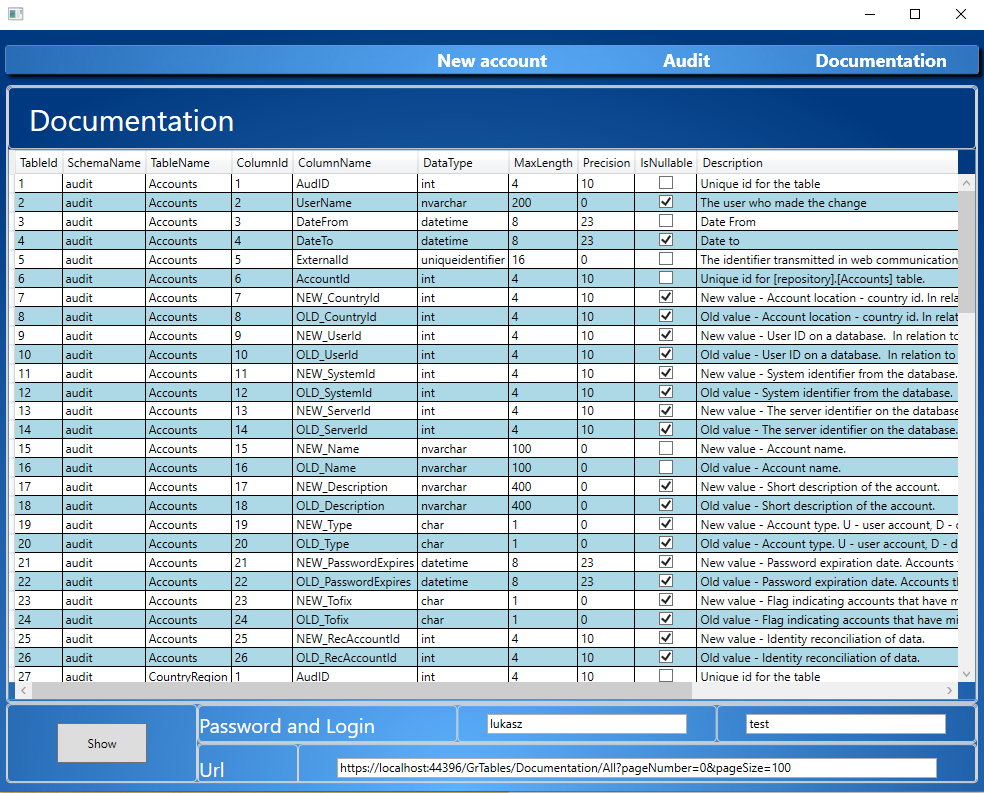
<https://localhost:44396/GrTables/Documentation/All/Count>

In response, the user receives

{

"message": "There are 204 elements in the documentation."

}



* + 1. **Audit of all changes in the accounts**

All accounts created in the global repository are audited.

Required fields :

1. To execute a GET request, the password and login must be entered in the header.
2. If the data is processed correctly, audit data will be returned.
3. All GET queries require the size of pages returned and the amount of data on the page.
4. Other GET queries are used to check the number of pages.

The following is an example of an count query

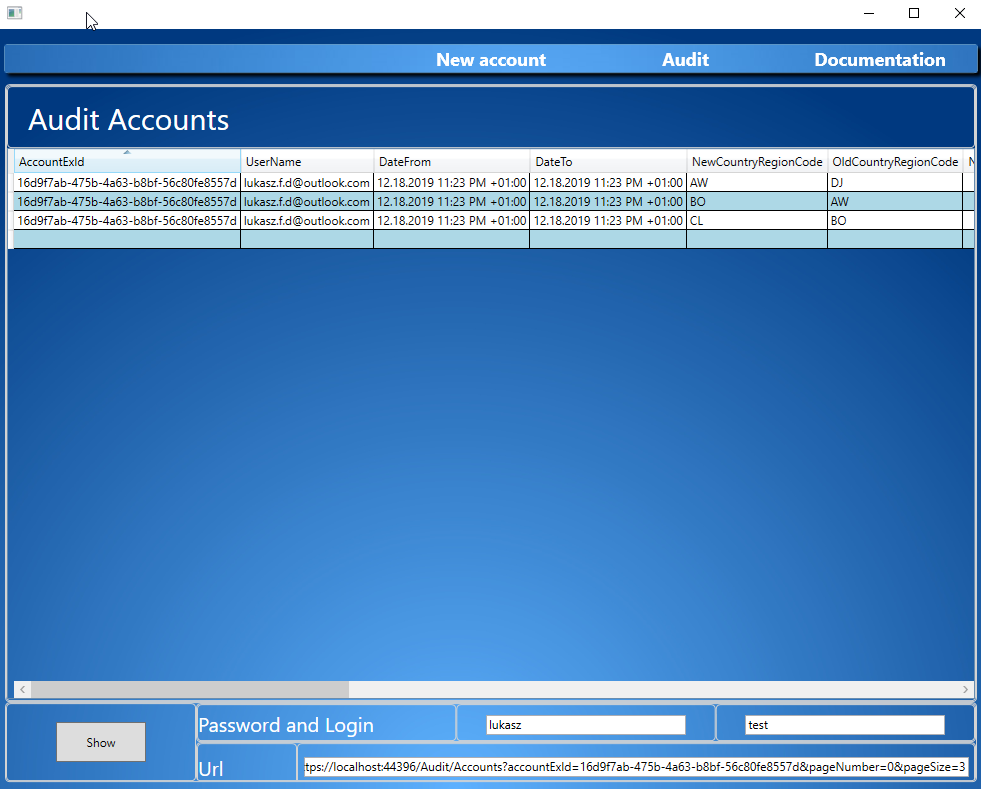
<https://localhost:44396/Audit/Accounts/Count?accountExId=16d9f7ab-475b-4a63-b8bf-56c80fe8557d>

In response, the user receives

{

"message": "For user bb2f882f-f643-4b8a-aa06-3b611d7de44e are assigned 5"

}



* 1. **Description of methods used in Web communication.**
     1. **AdminController**
        1. **CreateNewUser - This method creates a new user of the global repository.**

/// <summary>

/// Create new user

/// </summary>

/// <param name="account"></param>

/// <returns></returns>

[HttpPost("CreateNewUser")]

public async Task<ActionResult<NewUser>> CreateNewUser(NewUser user)

{

string data = null;

foreach (var item in await new GlobalRepositoryData().AdminCheck())

{

if (\_userEx.ToUpper() == item.ToString().ToUpper())

{

try

{

data = Convert.ToString(await new GlobalRepositoryData().CreateNewUser(user));

\_logger.LogInformation($"NewUser : {data}");

if (data == null)

{

\_logger.LogWarning($"No data found for {\_userEx}");

return NotFound(new { message = $"No data found for {\_userEx}" });

}

}

catch (System.Exception ex)

{

\_logger.LogError($"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

return StatusCode(500, $"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

}

}

else

{

return StatusCode(500, $"You don't have the privilege to perform this operation.");

}

}

return Ok(data);

}

* + 1. **AuditController** 
       1. **GetAuditAccountsCount - This method returns the number of pages for a given account.**

/// <summary>

/// Return the number of accounts in the audit.

/// </summary>

/// <param name="accountExId"></param>

/// <returns></returns>

[HttpGet("Accounts/Count")]

public async Task<ActionResult<int>> GetAuditAccountsCount(string accountExId)

{

int data;

try

{

\_logger.LogInformation($"Count for GetAuditAccountsCount by user : {\_userEx}");

data = await new AuditData().AuditAccountsCount(accountExId);

if (data == 0)

{

return NotFound(new { message = $"No account is assigned to the user : {\_userEx}" });

}

}

catch (System.Exception ex)

{

\_logger.LogError($"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

return StatusCode(500, $"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

}

return Ok(new { message = $"For user {\_userEx} are assigned {data}" });

}

* + - 1. **GetAuditServersCount - This method returns the number of pages for a given server.**

/// <summary>

/// Return the number of servers in the audit.

/// </summary>

/// <param name="serverExId"></param>

/// <returns></returns>

[HttpGet("Servers/Count")]

public async Task<ActionResult<int>> GetAuditServersCount(string serverExId)

{

int data;

try

{

\_logger.LogInformation($"Count for GetAuditServersCount by user : {\_userEx}");

data = await new AuditData().AuditServersCount(serverExId);

if (data == 0)

{

return NotFound(new { message = $"No servers is assigned to the user : {\_userEx}" });

}

}

catch (System.Exception ex)

{

\_logger.LogError($"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

return StatusCode(500, $"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

}

return Ok(new { message = $"For user {\_userEx} are assigned {data}" });

}

* + - 1. **GetAuditSystemsCount - This method returns the number of pages for a given system.**

/// <summary>

/// Return the number of systems in the audit.

/// </summary>

/// <param name="systemExId"></param>

/// <returns></returns>

[HttpGet("Systems/Count")]

public async Task<ActionResult<int>> GetAuditSystemsCount(string systemExId)

{

int data;

try

{

\_logger.LogInformation($"Count for GetAuditSystemsCount by user : {\_userEx}");

data = await new AuditData().AuditSystemsCount(systemExId);

if (data == 0)

{

return NotFound(new { message = $"No systems is assigned to the user : {\_userEx}" });

}

}

catch (System.Exception ex)

{

\_logger.LogError($"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

return StatusCode(500, $"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

}

return Ok(new { message = $"For user {\_userEx} are assigned {data}" });

}

* + - 1. **GetAuditAccounts - This method returns the audit for the account given in the query. The page number and page size are required.**

/// <summary>

/// Return accounts saved in the audit.

/// </summary>

/// <param name="accountExId"></param>

/// <param name="pageNumber"></param>

/// <param name="pageSize"></param>

/// <returns></returns>

[HttpGet("Accounts")]

public async Task <ActionResult<IEnumerable<AuditAccounts>>> GetAuditAccounts(string accountExId, [FromQuery] int pageNumber, [FromQuery] int pageSize)

{

IEnumerable<AuditAccounts> data;

try

{

\_logger.LogInformation($"Data downloading for account : {accountExId} by user : {\_userEx} (GetAuditServers)");

data = await new AuditData().AuditAccounts(accountExId, pageSize, pageNumber);

if (data == null)

{

\_logger.LogWarning($"No data found for user : {\_userEx} and account : {accountExId}");

return NotFound(new { message = $"No data found for user : {\_userEx} and account : {accountExId}" });

}

}

catch (System.Exception ex)

{

\_logger.LogError($"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

return StatusCode(500, $"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

}

return Ok(data);

}

* + - 1. **GetAuditServers - This method returns the audit for the servers given in the query. The page number and page size are required.**

/// <summary>

/// Return servers saved in the audit.

/// </summary>

/// <param name="serverExId"></param>

/// <param name="pageNumber"></param>

/// <param name="pageSize"></param>

/// <returns></returns>

[HttpGet("Servers")]

public async Task<ActionResult<IEnumerable<AuditServers>>> GetAuditServers(string serverExId, [FromQuery] int pageNumber, [FromQuery] int pageSize)

{

IEnumerable<AuditServers> data;

try

{

\_logger.LogInformation($"Data downloading for server : {serverExId} by user : {\_userEx} (GetAuditServers)");

data = await new AuditData().AuditServers(serverExId, pageSize, pageNumber);

if (data == null)

{

\_logger.LogWarning($"No data found for user : {\_userEx} and server : {serverExId}");

return NotFound(new { message = $"No data found for user : {\_userEx} and server : {serverExId}" });

}

}

catch (System.Exception ex)

{

\_logger.LogError($"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

return StatusCode(500, $"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

}

return Ok(data);

}

* + - 1. **GetAuditSystems This method returns the audit for the systems given in the query. The page number and page size are required.**

/// <summary>

/// Return systems saved in the audit.

/// </summary>

/// <param name="systemrExId"></param>

/// <param name="pageNumber"></param>

/// <param name="pageSize"></param>

/// <returns></returns>

[HttpGet("Systems")]

public async Task<ActionResult<IEnumerable<AuditSystems>>> GetAuditSystems(string systemrExId, [FromQuery] int pageNumber, [FromQuery] int pageSize)

{

IEnumerable<AuditSystems> data;

try

{

\_logger.LogInformation($"Data downloading for system : {systemrExId} by user : {\_userEx} (GetAuditSystems)");

data = await new AuditData().AuditSystems(systemrExId, pageSize, pageNumber);

if (data == null)

{

\_logger.LogWarning($"No data found for user : {\_userEx} and system : {systemrExId}");

return NotFound(new { message = $"No data found for user : {\_userEx} and system : {systemrExId}" });

}

}

catch (System.Exception ex)

{

\_logger.LogError($"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

return StatusCode(500, $"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

}

return Ok(data);

}

* + 1. **GlobalRepositoryDataController**
       1. **CreateNewAccount - This method creates a new account.**

/// <summary>

/// Create a new account with information about servers and systems.

/// </summary>

/// <param name="account"></param>

/// <returns></returns>

[HttpPost("CreateNewAccount")]

public async Task<ActionResult<NewAccount>> CreateNewAccount(NewAccount account)

{

string data;

try

{

data = Convert.ToString(await new GlobalRepositoryData().CreateNewAccount(account, \_userEx));

\_logger.LogInformation($"CreateNewAccount {data} for {\_userEx}");

if (data == null)

{

\_logger.LogWarning($"No data found for {\_userEx}");

return NotFound(new { message = $"No data found for user : {\_userEx} and account : {account}" });

}

}

catch (System.Exception ex)

{

\_logger.LogError($"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

return StatusCode(500, $"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

}

R

* + - 1. **GetGlobalRespositoryData - This method displays data for the account specified in the query. The page number and page size are required.**

/// <summary>

/// Collect all data about existing accounts.

/// </summary>

/// <param name="pageNumber"></param>

/// <param name="pageSize"></param>

/// <returns></returns>

[HttpGet("/AllAccounts")]

public async Task<ActionResult<IEnumerable<GrAccount>>> GetGlobalRespositoryData([FromQuery] int pageNumber, [FromQuery] int pageSize)

{

IEnumerable<GrAccount> data;

try

{

\_logger.LogInformation($"Data downloading for GrAccount by user : {\_userEx}");

data = await new GlobalRepositoryData().GrData(\_userEx, pageSize, pageNumber);

if (data == null)

{

\_logger.LogWarning($"No data found for {\_userEx}");

return NotFound(new { message = $"No data found for user {\_userEx}" });

}

}

catch (System.Exception ex)

{

\_logger.LogError($"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

return StatusCode(500, $"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

}

return Ok(data);

}

* + - 1. **GetGlobalRespositoryDatCount - This method displays how many pages there are for the account.**

/// <summary>

/// Checking how many accounts are assigned to a user.

/// </summary>

/// <returns></returns>

[HttpGet("/AllAccounts/Count")]

public async Task<ActionResult<int>> GetGlobalRespositoryDatCount()

{

int data;

try

{

\_logger.LogInformation($"Count for GrAccount by user : {\_userEx}");

data = await new GlobalRepositoryData().GrDataCount(\_userEx);

if (data == 0)

{

return NotFound(new { message = $"No account is assigned to the user : {\_userEx}" });

}

}

catch (System.Exception ex)

{

\_logger.LogError($"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

return StatusCode(500, $"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

}

return Ok(new { message = $"For user {\_userEx} are assigned {data}." });

}

* + - 1. **DeleteAccount - This method deletes the account specified in the query.**

/// <summary>

/// DEelte account

/// </summary>

/// <param name="accountExId"></param>

/// <returns></returns>

[HttpDelete("DeleteAccount")]

public async Task<ActionResult<bool>> DeleteAccount(string accountExId)

{

try

{

await new GlobalRepositoryData().DeleteAccount(accountExId);

}

catch (System.Exception ex)

{

\_logger.LogError($"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

return StatusCode(500, $"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

}

return Ok(true);

}

* + 1. **GrTablesController**
       1. **GetGlobalRespositoryDocumentationCount - This method displays how many pages there are for the documentation.**

/// <summary>

/// Checking how many objects are available for download from the documentation.

/// </summary>

/// <returns></returns>

[HttpGet("Documentation/All/Count")]

public async Task<ActionResult<int>> GetGlobalRespositoryDocumentationCount()

{

int data;

try

{

\_logger.LogInformation($"GetGlobalRespositoryDocumentationCount for user : {\_userEx}");

data = await new Documentation().DBDocumentationCount();

if (data == 0)

{

return NotFound(new { message = $"Not found documentation." });

}

}

catch (System.Exception ex)

{

\_logger.LogError($"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

return StatusCode(500, $"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

}

return Ok(new { message = $"There are {data} elements in the documentation." });

}

* + - 1. **GetGlobalRespositoryDocumentation - This method displays documentation. The page number and page size are required.**

/// <summary>

/// Getting all the objects from the documentation.

/// </summary>

/// <param name="pageNumber"></param>

/// <param name="pageSize"></param>

/// <returns></returns>

[HttpGet("Documentation/All")]

public async Task<ActionResult<IEnumerable<GrTables>>> GetGlobalRespositoryDocumentation([FromQuery] int pageNumber, [FromQuery] int pageSize)

{

IEnumerable<GrTables> data;

try

{

\_logger.LogInformation($"GetGlobalRespositoryDocumentation for user : {\_userEx}");

data = await new Documentation().DBDocumentation(pageNumber, pageSize);

if (data == null)

{

\_logger.LogWarning("No data found for GetGlobalRespositoryDocumentation/Documentation/All");

return NotFound(new { message = $"No data found in documentation." });

}

}

catch (System.Exception ex)

{

\_logger.LogError($"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

return StatusCode(500, $"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

}

return Ok(data);

}

* + - 1. **GetSpecyficInformationDocumentation - This method displays documentation for selected schemes and tables. The page number and page size are required.**

/// <summary>

/// To retrieve objects from the documentation with table and schema names.

/// </summary>

/// <param name="schema\_name"></param>

/// <param name="table\_name"></param>

/// <param name="pageNumber"></param>

/// <param name="pageSize"></param>

/// <returns></returns>

[HttpGet("Documentation/{schema\_name}/{table\_name}")]

public async Task<ActionResult<IEnumerable<GrTables>>> GetSpecyfikInformationDocumentation(string schema\_name, string table\_name, [FromQuery] int pageNumber, [FromQuery] int pageSize)

{

IEnumerable<GrTables> data;

try

{

\_logger.LogInformation($"GetSpecyfikInformationDocumentation for user : {\_userEx}");

data = await new Documentation().DBDocumentation(schema\_name, table\_name, pageNumber, pageSize);

if (data == null)

{

\_logger.LogWarning($"No data found for GetGlobalRespositoryDocumentation/Documentation/{schema\_name}/{table\_name}");

return NotFound(new { message = $"No data found for GetGlobalRespositoryDocumentation/Documentation/{schema\_name}/{table\_name}" });

}

}

catch (System.Exception ex)

{

\_logger.LogError($"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

return StatusCode(500, $"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

}

return Ok(data);

}

* + 1. **ReconciliationController**
       1. **ReconAccounts - This method is used to reconcile data for accounts.**

/// <summary>

/// Accounts reconciliations.

/// </summary>

/// <param name="recon"></param>

/// <returns></returns>

[HttpPost("Accounts")]

public async Task<ActionResult<ReconAccounts>> ReconAccounts(ReconAccounts recon)

{

try

{

await new ReconciliationData().ReconciliationAccounts(recon);

\_logger.LogInformation($"The data have been accepted for user: {\_userEx} (ReconAccounts).");

}

catch (System.Exception ex)

{

\_logger.LogError($"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

return StatusCode(500, $"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

}

return Ok(new { message = "The data have been accepted" });

}

* + - 1. **ReconServers - This method is used to reconcile data for servers.**

/// <summary>

/// Servers reconciliations.

/// </summary>

/// <param name="recon"></param>

/// <returns></returns>

[HttpPost("Servers")]

public async Task<ActionResult<ReconServers>> ReconServers(ReconServers recon)

{

try

{

await new ReconciliationData().ReconciliationServers(recon);

\_logger.LogInformation($"The data have been accepted for user: {\_userEx} (ReconServers).");

}

catch (System.Exception ex)

{

\_logger.LogError($"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

return StatusCode(500, $"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

}

return Ok(new { message = "The data have been accepted" });

}

* + - 1. **ReconSystems - This method is used to reconcile data for systems.**

/// <summary>

/// System reconciliations.

/// </summary>

/// <param name="recon"></param>

/// <returns></returns>

[HttpPost("Systems")]

public async Task<ActionResult<ReconSystems>> ReconSystems(ReconSystems recon)

{

try

{

await new ReconciliationData().ReconciliationSystems(recon);

\_logger.LogInformation($"The data have been accepted for user: {\_userEx} (ReconSystems).");

}

catch (System.Exception ex)

{

\_logger.LogError($"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

return StatusCode(500, $"An error has occurred during data processing. For : {\_userEx}. Error : {ex.Message}");

}

return Ok(new { message = "The data have been accepted" });

}

1. **Conclusions**

Training material which I received during the postgraduate studies "Programming on the .Net platform" and workshops with specialists in object-oriented programming allowed me to create a web application "Global Repositories" of accounts, servers and systems. One of the biggest advantages of such projects is to learn the basic principles of object-oriented programming on the .net platform, project management and source code management. Currently, the level of my advancement in the design and development of web applications has definitely increased.

Web application which I created is a simple but comprehensive solution for every company that in its portfolio manages data and information about the infrastructure. With simplicity in access to data and the ability to easily check the documentation on how to use Web Api "Global Repository" is a very easy tool to implement in any system.

While working on the application I encountered many problems, especially related to the lack of experience in using the capabilities offered by the .Net Core platform. I encountered many optimization and functional difficulties. Therefore, I used Microsoft documentation, online courses and actively used the "Stackoverflow" platform. However, not all the plans and assumptions contained in the Vision document were implemented.

Despite minor complications, I managed to create a very good web application. I can certainly say that the design of the database, the entity framework database model and the application itself is at a high level of code advancement.

1. **Bibliography**
2. Microsft's documentation .Net Core 2.2

<https://docs.microsoft.com/pl-pl/aspnet/core/?view=aspnetcore-2.2>

1. Stackoverflow

<https://stackoverflow.com/questions/59321310/easy-basic-page-numbering-and-limiting-the-size-of-returned-data-in-the-net-cor>

1. Course C# and .net Core WebApi on YouTube

<https://www.youtube.com/channel/UC-ptWR16ITQyYOglXyQmpzw>

1. Microsoft’s database project documentation

<https://docs.microsoft.com/en-us/visualstudio/data-tools/create-a-sql-database-by-using-a-designer?view=vs-2019>

1. Język C# 6.0 i platforma .NET 4.6 Troelsen Andrew, Japikse Phiplip