ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

СОГЛАСОВАНО
Научный руководитель,
доцент департамента программной инженерии факультета компьютерных наук, канд. техн. наук

_____ В.А. Дударев
«_12_» ___мая___ 2022 г.

УТВЕРЖДАЮ
Академический руководитель образовательной программы «Программная инженерия», кандидат технических наук

_____ В.В. Шилов
«_12_» мая 2022 г.

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	RU.17701729.05.15-01 12 01-1

RESTFUL ВЕБ-СЕРВИС ДЛЯ РАБОТЫ С БАЗАМИ ДАННЫХ ПО СВОЙСТВАМ НЕОРГАНИЧЕСКИХ ВЕЩЕСТВ И МАТЕРИАЛОВ

Текст программы

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.05.15-01 12 01-1-ЛУ

Исполнитель:
студент группы БПИ205
_____ / Овчинникова П.А./
«_12_» ___мая___ 2022 г.

2022

УТВЕРЖДЕНО
RU.17701729.05.15-01 12 01-1-ЛУ

RESTFUL ВЕБ-СЕРВИС ДЛЯ РАБОТЫ С БАЗАМИ ДАННЫХ ПО СВОЙСТВАМ НЕОРГАНИЧЕСКИХ ВЕЩЕСТВ И МАТЕРИАЛОВ

Текст программы

RU.17701729.05.15-01 12 01-1

Листов 30

Инв. № подл	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата
RU.17701729.05.15-01 12 01-1				

2022

СОДЕРЖАНИЕ

АННОТАЦИЯ

В настоящем документе «Текст программы» приведен текст «Restful веб-сервис для работы с базами данных по свойствам неорганических веществ и материалов». Программа разработана на языке C# 10.0.

Среда разработки – JetBrains Rider 2021.3.

Функциональным назначением программы является автоматизация процесса синхронизации обновлений из баз данных по свойствам неорганических веществ и материалов в метабазе.

Настоящий документ разработан в соответствии с требованиями:

1) ГОСТ 19.101-77 Виды программ и программных документов;
2) ГОСТ 19.102-77 Стадии разработки;
3) ГОСТ 19.103-77 Обозначения программ и программных документов;
4) ГОСТ 19.104-78 Основные надписи;
5) ГОСТ 19.105-78 Общие требования к программным документам;
6) ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом;
7) ГОСТ 19.401-78 Текст программы. Требования к содержанию и оформлению.

Изменения к данному документу оформляются согласно ГОСТ 19.603-78 [8], ГОСТ 19.604-78.

Перед прочтением данного документа рекомендуется ознакомиться с терминологией, приведенной в Приложении 1.

1. Текст программы

Ссылка на github-репозиторий с исходным текстом программы: https://github.com/poovch/Metabase-update-service.

1.1 Сервис для метабазы (Service for metabase)

1.1.1 Program.cs

```
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.EntityFrameworkCore;
using Service_for_metabase.Model;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationSche
me)
            .AddCookie(options =>
            {
                    options.LoginPath = new PathString("/login");
            });

builder.Services.AddDbContext<MetabaseContext>(options =>
{
            options.UseSqlServer(builder.Configuration.GetConnectionString("Metaba
seConnection"));
});

// Add services to the container.
builder.Services.AddControllersWithViews();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
            app.UseExceptionHandler("/Home/Error");
            // The default HSTS value is 30 days. You may want to change this for
production scenarios, see https://aka.ms/aspnetcore-hsts.
            app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthentication();
app.UseAuthorization();

app.MapControllerRoute(
            name: "default",
            pattern: "{controller=Metabase}/{action=Index}");

app.Run();
```

1.1.2 appsettings.json

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "ConnectionStrings": {
    "MetabaseConnection": "Server=localhost;Persist Security Info=True;initial
catalog=Metabase;Max Pool Size=200;Integrated Security=SSPI;"
  },
  "AllowedHosts": "*"
}
```

1.1.3 Controllers

1.1.3.1 AccountController.cs

```
using EntityLib;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Data.SqlClient;
using Microsoft.EntityFrameworkCore;
using Service_for_database.Model;

namespace Service_for_database.Controllers
{
    public class DatabaseController : Controller
    {
        private readonly DatabaseContext _dbContext;

        public DatabaseController(DatabaseContext dbContext)
        {
            _dbContext = dbContext;
        }

        [HttpGet("/check")]
        public async Task<IActionResult> Check()
        {
            try
            {
                await ExecuteSqlQuery();
            }
            catch (Exception exception)
            {
                return Problem(
                    detail: exception.Message,
                    title: "Data base connection problem.",
                    statusCode: StatusCodes.Status500InternalServerError
                );
            }
```

```
            return Ok();
        }

        [Authorize]
        [HttpGet("properties")]
        public async Task<ICollection<MetabaseProperty>?> GetProperties()
        {
            List<MetabaseProperty> propertiesList;
            try
            {
                await ExecuteSqlQuery();

                // Получение данных из View.
                propertiesList = await _dbContext.PropertiesInfo.ToListAsync();
            } catch (Exception)
            {
                return null;
            }

            return propertiesList;
        }

        [Authorize]
        [HttpGet("systems")]
        public async Task<ICollection<MetabaseSystem>?> GetSystems()
        {
            List<MetabaseSystem> systemsList;
            try
            {
                await ExecuteSqlQuery();

                // Получение данных из View.
                systemsList = await _dbContext.SystemInfo.ToListAsync();
            } catch (Exception)
            {
                return null;
            }

            return systemsList;
        }

        private async Task ExecuteSqlQuery()
        {
            // Скрипт создает или обновляет View.
            string sqlScript;
            using (var reader = new StreamReader("DatabaseQuery.sql"))
            {
                sqlScript = await reader.ReadToEndAsync();
            }

            var quires = sqlScript.Split("GO").Where(str => str.Trim() != "");

            foreach (var query in quires)
            {
                await using var sqlConnection = new
SqlConnection(Program.DbConnectionString);
```

```
                await using var command = sqlConnection.CreateCommand();
                command.CommandText = query;
                sqlConnection.Open();
                command.ExecuteNonQuery();
            }
        }
    }
}
```

1.1.3.2 MetabaseController.cs

```
using System.Net;
using System.Net.Http.Headers;
using EntityLib;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Service_for_metabase.Model;

namespace Service_for_metabase.Controllers
{
    [Authorize]
    public class MetabaseController : Controller
    {
        private readonly MetabaseContext _metabaseContext;

        public MetabaseController(MetabaseContext metabaseContext)
        {
            _metabaseContext = metabaseContext;
        }

        public IActionResult Index()
        {
            return View();
        }

        [HttpGet("update")]
        public async Task<IActionResult> Update()
        {
            var updatedProperties = new List<MetabaseProperty>();
            var updatedSystems = new List<MetabaseSystem>();
            var unsuccessfulUpdatesUrls = new List<string>();
            List<MetabaseDb> databases;

            using var client = SpecialHttpClient.GetHttpClient();
            client.DefaultRequestHeaders.Authorization =
                new AuthenticationHeaderValue("Bearer",
TokenGenerator.GetToken(User.Claims));

            try
            {
                databases = await _metabaseContext.DBInfo.ToListAsync();
            }
            catch
            {
                return Problem(detail: "Problem with access to Metabase DBInfo",
```

```
                    statusCode: StatusCodes.Status500InternalServerError);
            }

            var dbServicesUrls = databases
                .Select(info => info.DBServiceHost)
                .Where(host => host is not null);
            foreach (var dbServiceUrl in dbServicesUrls)
            {
                try
                {
                    updatedProperties.AddRange(await UpdateProperties(client,
dbServiceUrl!));
                    updatedSystems.AddRange(await UpdateSystems(client,
dbServiceUrl!));
                }
                catch (Exception)
                {
                    unsuccessfulUpdatesUrls.Add(dbServiceUrl!);
                }
            }

            ViewBag.PropertiesTableModel =
TableDto<MetabaseProperty>.BuildModel(updatedProperties);
            ViewBag.SystemsTableModel =
TableDto<MetabaseSystem>.BuildModel(updatedSystems);
            ViewBag.UnsuccessfulUpdates = databases
                .Where(db => unsuccessfulUpdatesUrls.Contains(db.DBServiceHost!))
                .Select(db => db.Name);

            return View("ShowUpdate");
        }

        [HttpGet("/activeServices")]
        public async Task<IActionResult> GetActivatedServices()
        {
            var activeServices = new List<string>();
            List<MetabaseDb> databases;
            try
            {
                databases = await _metabaseContext.DBInfo.ToListAsync();
            }
            catch
            {
                return Problem(detail: "Problem with access to Metabase DBInfo",
                    statusCode: StatusCodes.Status500InternalServerError);
            }

            const string checkEndpoint = "/check";

            using var client = SpecialHttpClient.GetHttpClient();
            var servicesUrls = databases
                .Select(db => db.DBServiceHost)
                .Where(url => url != null);
            foreach (var serviceUrl in servicesUrls)
            {
                try {
```

```
                    var responseCode = (await client.GetAsync(serviceUrl +
checkEndpoint)).StatusCode;
                    if (responseCode is HttpStatusCode.OK)
                    {
                        activeServices.Add(serviceUrl);
                    }
                }
                catch
                {
                    // ignored
                }
            }

            ViewBag.TableModel = TableDto<MetabaseDbDto>.BuildModel(databases
                .Where(db => activeServices.Contains(db.DBServiceHost))
                .Select(MetabaseDbDto.FromMetabaseDb));
            return View("ShowActiveServices");
        }

        [HttpGet("/properties")]
        public async Task<IActionResult> GetProperties()
        {
            List<MetabaseProperty> properties;
            try
            {
                properties = await _metabaseContext.PropertiesInfo.ToListAsync();
            }
            catch
            {
                return Problem(detail: "Problem with access to Metabase
PropertiesInfo",
                    statusCode: StatusCodes.Status500InternalServerError);
            }

            ViewBag.TableModel =
TableDto<MetabaseProperty>.BuildModel(properties);
            ViewBag.TableName = "Список свойств";
            return View("ShowInfo");
        }

        [HttpGet("/systems")]
        public async Task<IActionResult> GetSystems()
        {
            List<MetabaseSystem> systems;
            try
            {
                systems = await _metabaseContext.SystemInfo.ToListAsync();
            }
            catch
            {
                return Problem(detail: "Problem with access to Metabase
SystemInfo",
                    statusCode: StatusCodes.Status500InternalServerError);
            }

            ViewBag.TableModel = TableDto<MetabaseSystem>.BuildModel(systems);
```

```
            ViewBag.TableName = "Список систем";
            return View("ShowInfo");
        }

        private async Task<List<MetabaseProperty>> UpdateProperties(HttpClient
client, string dbServiceUrl)
        {
            var dbUrl =  dbServiceUrl + "/properties";
            var dbProperties = await (await client.GetAsync(dbUrl))
                .Content
                .ReadFromJsonAsync<List<MetabaseProperty>>();

            var metabaseProperties = await
_metabaseContext.PropertiesInfo.ToListAsync();
            var propsToAdd = dbProperties
                .Where(prop => !metabaseProperties.Contains(prop));
            _metabaseContext.PropertiesInfo.AddRange(propsToAdd);
            await _metabaseContext.SaveChangesAsync();

            var updMetabaseProperties = await
_metabaseContext.PropertiesInfo.ToListAsync();
            var propsToEdit = dbProperties
                .Where(prop => !updMetabaseProperties.Contains(prop, new
FullPropertyEqualityComparer()));
            foreach (var prop in propsToEdit)
            {
                var foundProp = updMetabaseProperties.Find(mProp =>
                    mProp.DBID == prop.DBID && mProp.PropId == prop.PropId);
                var modifyingProperty  =
_metabaseContext.PropertiesInfo.Update(foundProp!).Entity;
                modifyingProperty.Modify(prop);
            }
            await _metabaseContext.SaveChangesAsync();

            var newProperties = new List<MetabaseProperty>();
            newProperties.AddRange(propsToAdd);
            newProperties.AddRange(propsToEdit);
            return newProperties;
        }

        private async Task<List<MetabaseSystem>> UpdateSystems(HttpClient client,
string dbServiceUrl)
        {
            var dbUrl =  dbServiceUrl + "/systems";
            var dbSystems = await (await client.GetAsync(dbUrl))
                .Content
                .ReadFromJsonAsync<List<MetabaseSystem>>();

            var metabaseSystems = await _metabaseContext.SystemInfo.ToListAsync();
            var systemsToAdd = dbSystems
                .Where(sys => !metabaseSystems.Contains(sys));
            _metabaseContext.SystemInfo.AddRange(systemsToAdd);
            await _metabaseContext.SaveChangesAsync();

            var updMetabaseSystems = await
_metabaseContext.SystemInfo.ToListAsync();
```

```
            var systemsToEdit = dbSystems
                .Where(sys => !updMetabaseSystems.Contains(sys, new
FullSystemEqualityComparer()));
            foreach (var system in systemsToEdit)
            {
                var foundSys = updMetabaseSystems.Find(mProp =>
                    mProp.DBID == system.DBID && mProp.SystemId ==
system.SystemId);
                var modifyingSystem  =
_metabaseContext.SystemInfo.Update(foundSys!).Entity;
                modifyingSystem.Modify(system);
            }
            await _metabaseContext.SaveChangesAsync();

            var newSystems = new List<MetabaseSystem>();
            newSystems.AddRange(systemsToAdd);
            newSystems.AddRange(systemsToEdit);
            return newSystems;
        }
    }
}
```

1.1.4 Model

1.1.4.1 JwtAuthenticationOptions.cs

```
using System.Text;
using Microsoft.IdentityModel.Tokens;

namespace Service_for_metabase.Model;

public static class JwtAuthenticationOptions
{
            public const string Issuer = "MetabaseService";
            public const string Audience = "DatabaseService";
            private const string Key = "mysupersecret_secretkey!123";
            public const int Lifetime = 5;
            public static SymmetricSecurityKey GetSymmetricSecurityKey()
            {
                    return new SymmetricSecurityKey(Encoding.ASCII.GetBytes(Key));
            }
}
```

1.1.4.2 MetabaseContext.cs

```
using EntityLib;
using Microsoft.EntityFrameworkCore;

namespace Service_for_metabase.Model;

    public class MetabaseContext : DbContext
    {
        public DbSet<MetabaseProperty> PropertiesInfo { get; set; } = null!;
        public DbSet<MetabaseSystem> SystemInfo { get; set; } = null!;

        public DbSet<MetabaseDb> DBInfo { get; set; } = null!;
```

```
        public MetabaseContext(DbContextOptions<MetabaseContext> options)
            : base(options)
        {
            Database.EnsureCreated();
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<MetabaseProperty>()
                        .HasKey(nameof(MetabaseProperty.DBID),
                                nameof(MetabaseProperty.PropId));
            modelBuilder.Entity<MetabaseSystem>()
                .HasKey(nameof(MetabaseSystem.DBID),
                    nameof(MetabaseSystem.SystemId));
        }
    }
```

1.1.4.3 MetabaseDb.cs

```
using System.ComponentModel.DataAnnotations;

namespace Service_for_metabase.Model;

public class MetabaseDb
{
            [Key]
            public int DBID { get; set; }

            public Boolean Enabled { get; set; }

            public DateTime LastUpdate { get; set; }

            public string Name { get; set; }

            public string Login { get; set; }

            public string Password { get; set; }

            public string DBURL { get; set; }

            public string DBGateRedirect { get; set; }

            public string DBGateURL { get; set; }

            public string EmailManager { get; set; }

            public string WWWTemplatePage { get; set; }

            public string Language { get; set; }

            public string Description { get; set; }

            public string? DBServiceHost { get; set; }
}
```

1.1.4.4 MetabaseDbDto.cs

```
namespace Service_for_metabase.Model;

public class MetabaseDbDto
{
            public int DBID { get; set; }

            public string Name { get; set; }

            public string EmailManager { get; set; }

            public string Language { get; set; }

            public string Description { get; set; }

            public string? ServiceHost { get; set; }

            public static MetabaseDbDto FromMetabaseDb(MetabaseDb metabaseDb)
            {
                    return new MetabaseDbDto()
                    {
                            DBID = metabaseDb.DBID,
                            Name = metabaseDb.Name,
                            EmailManager = metabaseDb.EmailManager,
                            Language = metabaseDb.Language,
                            Description = metabaseDb.Description,
                            ServiceHost = metabaseDb.DBServiceHost
                    };
            }
}
```

1.1.4.5 ServiceUser.cs

```
namespace Service_for_metabase.Model;

[Serializable]
public class ServiceUser
{
            public string Username { get; set; }

            public string Password { get; set; }

            public string Role { get; set; }
}
```

1.1.4.6 ServiceUserDto.cs

```
using System.ComponentModel.DataAnnotations;

namespace Service_for_metabase.Model;

public class ServiceUserDto
{
            [Required(ErrorMessage = "Не указано имя пользователя")]
```

```
        public string Username { get; set; }

        [Required(ErrorMessage = "Не указан пароль")]
        [DataType(DataType.Password)]
        public string Password { get; set; }
}
```

1.1.4.7 SpecialHttpClient.cs

```
namespace Service_for_metabase.Model;

public static class SpecialHttpClient
{
        public static HttpClient GetHttpClient()
        {
                var clientHandler = new HttpClientHandler()
                {
                        ServerCertificateCustomValidationCallback = (sender,
cert, chain, sslPolicyErrors) => true
                };

                return new HttpClient(clientHandler);
        }
}
```

1.1.4.8 TableDto.cs

```
using System.Reflection;

namespace Service_for_metabase.Model;

public class TableDto<T>
{
        public IEnumerable<PropertyInfo> Columns { get; set; }

        public IEnumerable<T> Items { get; set; }

        public static TableDto<T> BuildModel(IEnumerable<T> properties)
        {
                return new TableDto<T>
                {
                        Columns = typeof(T).GetProperties()
                                .Where(col => col.Name != "UpdateStatus"),
                        Items = properties
                };
        }
}
```

1.1.4.9 TokenGenerator.cs

```
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using Microsoft.IdentityModel.Tokens;

namespace Service_for_metabase.Model;
```

```
public static class TokenGenerator
{
            public static string GetToken(IEnumerable<Claim> claims)
            {
                    var now = DateTime.UtcNow;

                    var jwt = new JwtSecurityToken(
                            JwtAuthenticationOptions.Issuer,
                            JwtAuthenticationOptions.Audience,
                            notBefore: now,
                            claims: claims,
                            expires:
now.Add(TimeSpan.FromMinutes(JwtAuthenticationOptions.Lifetime)),
                            signingCredentials: new
SigningCredentials(JwtAuthenticationOptions.GetSymmetricSecurityKey(),
                                    SecurityAlgorithms.HmacSha256));
                    var encodedJwt = new
JwtSecurityTokenHandler().WriteToken(jwt);

                    return encodedJwt;
            }
}
```

1.1.5 Resources

UserInfo.txt:

```
[{"Username":"Admin","Password":"12345","Role":"admin"},{"Username":"User","Passwo
rd":"11111","Role":"user"}]
```

1.1.6 Views

1.1.6.1 Account

Login.cshtml:

```
@model Service_for_metabase.Model.ServiceUserDto

@{
    ViewData["Title"] = "Login";
}
<head>
    <link rel="stylesheet" href="css/login.css">
</head>
<h1>Metabase service</h1>
<h3>Вход</h3>

@using (Html.BeginForm())
{
    @Html.ValidationSummary(true)
    <div>
        <p>@Html.ValidationMessageFor(m => m.Username)</p>
         <label >Имя пользователя:</label><br/>
```

```
        @Html.EditorFor(model => model.Username)
    </div>
    <div>
        <p>@Html.ValidationMessageFor(m => m.Password)</p>
        <label >Пароль:</label><br/>
        @Html.PasswordFor(model => model.Password)
    </div>
    <input type="submit" value="Войти"/>
}
```

1.1.6.2 Metabase

Index.cshtml:

```
@model dynamic

<head>
    <link rel="stylesheet" href="css/index.css">
</head>

@{
    ViewBag.Title = "title";
    Layout = "_Layout";
}

<div>
    <h2 class="menu-header">Доступные функции</h2>
    <ul id="menu">
        <li class="menuitem"><a class="menu-reference"
href="/activeServices">Подключенные сервисы</a></li>
        <li class="menuitem"><a class="menu-reference" href="/properties">Список
свойств</a></li>
        <li class="menuitem"><a class="menu-reference" href="/systems">Список
систем</a></li>
        <li class="menuitem"><a class="menu-reference" href="/update">Обновить
метабазу</a></li>
    </ul>
</div>
```

ShowActiveServices.cshtml:

```
@model dynamic

@{
    ViewBag.Title = "title";
    Layout = "_Layout";
}

<h2>Подключенные сервисы</h2>

@if (Enumerable.Count(ViewBag.TableModel.Items) > 0)
{
    @await Html.PartialAsync("_Table", ViewBag.TableModel, null)
}
else
```

```
{
    <p> Нет активных баз данных :(</p>
}
```

ShowInfo.cshtml:

```
@model dynamic

@{
    Layout = "_Layout";
}

<h2>@ViewBag.TableName</h2>

@await Html.PartialAsync("_Table", ViewBag.TableModel, null);
```

ShowUpdate.cshtml:

```
@model dynamic

@{
    ViewBag.Title = "title";
    Layout = "_Layout";
}

<h2>Результаты обновления метабазы</h2>

@if (Enumerable.Count(ViewBag.UnsuccessfulUpdates) > 0)
{
    <p>Не получилось взять обновления из следующих баз данных:</p>
    <ul>
        @foreach (var mistake in ViewBag.UnsuccessfulUpdates)
        {
            <li style="color: red">@mistake</li>
        }
    </ul>
}

@if (Enumerable.Count(ViewBag.PropertiesTableModel.Items) == 0 &&
     Enumerable.Count(ViewBag.SystemsTableModel.Items) == 0)
{
    <h3>Обновлений нет</h3>
}

@if (Enumerable.Count(ViewBag.PropertiesTableModel.Items) > 0)
{
    <h3>Добавленные свойства</h3>
    @await Html.PartialAsync("_Table", ViewBag.PropertiesTableModel, null)
}

@if (Enumerable.Count(ViewBag.SystemsTableModel.Items) > 0)
{
    <h3>Добавленные системы</h3>
    @await Html.PartialAsync("_Table", ViewBag.SystemsTableModel, null)
}
```

1.1.6.3 Shared

_Layout.cshtml:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8"/>
    <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
    <title>@ViewData["Title"] - WebApplication1</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css"/>
    <link rel="stylesheet" href="~/css/site.css" asp-append-version="true"/>

</head>
<body>
<header>
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm border-bottom
box-shadow mb-3
        header-bg">
            <a class="service-name" href="/">Metabase service</a>
            <ul class="navbar-nav flex-grow-1">
                <li class="nav-item">
                    <a class="nav-link text-dark" href="/">Главная</a>
                </li>
                <li class="nav-item logout">
                    <a class="nav-link text-dark" href="/logout">Выйти</a>
                </li>
            </ul>
        </nav>
        </header>
        <div class="container">
            <main role="main" class="pb-3">
                @RenderBody()
            </main>
        </div>
</body>
</html>
```

_Table.cshtml:

```
@using Microsoft.AspNetCore.Html
@model dynamic

<head>
    <link rel="stylesheet" href="css/table-style.css">
</head>

<table class="items-table">
    <thead>
    <tr>
        @foreach (var column in @Model.Columns)
        {
            <th class="th-style">@column.Name</th>
        }
    </tr>
    </thead>
    <tbody>
```

```
    @foreach (var item in @Model.Items)
    {
        <tr>
            @foreach (var column in @Model.Columns)
            {
                @if (column.Name == "SystemInfo" && column.GetValue(item) != null)
                {
                    <td>
                        @{
                            var html = new
HtmlString(@column.GetValue(item).ToString());
                        }
                        @html
                    </td>
                }
                else
                {
                    <td>@column.GetValue(item)</td>
                }
            }
        </tr>
    }
    </tbody>
</table>
```

1.2 Сервис для базы данных (Service for database)

1.2.1 Program.cs

```
using Microsoft.EntityFrameworkCore;
using Service_for_database.Model;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.IdentityModel.Tokens;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
            .AddJwtBearer(options =>
            {
                    options.RequireHttpsMetadata = true;
                    options.TokenValidationParameters = new
TokenValidationParameters()
                    {
                            ValidateIssuer = true,
                            ValidIssuer = AuthenticationOptions.Issuer,

                            ValidateAudience = true,
                            ValidAudience = AuthenticationOptions.Audience,

                            ValidateLifetime = true,

                            ValidateIssuerSigningKey = true,
                            IssuerSigningKey =
AuthenticationOptions.GetSymmetricSecurityKey()
                    };
            });
```

```
DbConnectionString =
builder.Configuration.GetConnectionString("DatabaseConnection");
builder.Services.AddDbContext<DatabaseContext>(options =>
{
            options.UseSqlServer(DbConnectionString);
});

// Add services to the container.
builder.Services.AddControllersWithViews();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
            app.UseExceptionHandler("/Home/Error");
            // The default HSTS value is 30 days. You may want to change this for
production scenarios, see https://aka.ms/aspnetcore-hsts.
            app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthentication();
app.UseAuthorization();

app.MapControllerRoute(
            name: "default",
            pattern: "{controller=Database}/{action=Index}/");

app.Run();

internal partial class Program
{
            public static string DbConnectionString = null!;
}
```

1.2.2 appsettings.json

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "ConnectionStrings": {
    "DatabaseConnection": "Server=localhost;initial catalog=BandGap;Integrated
Security=SSPI;"
  },
  "AllowedHosts": "*"
```

```
}
```

1.2.3 DatabaseController.cs

```
using EntityLib;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Data.SqlClient;
using Microsoft.EntityFrameworkCore;
using Service_for_database.Model;

namespace Service_for_database.Controllers
{
    public class DatabaseController : Controller
    {
        private readonly DatabaseContext _dbContext;

        public DatabaseController(DatabaseContext dbContext)
        {
            _dbContext = dbContext;
        }

        [HttpGet("/check")]
        public async Task<IActionResult> Check()
        {
            try
            {
                await ExecuteSqlQuery();
            }
            catch (Exception exception)
            {
                return Problem(
                    detail: exception.Message,
                    title: "Data base connection problem.",
                    statusCode: StatusCodes.Status500InternalServerError
                );
            }

            return Ok();
        }

        [Authorize]
        [HttpGet("properties")]
        public async Task<ICollection<MetabaseProperty>?> GetProperties()
        {
            List<MetabaseProperty> propertiesList;
            try
            {
                await ExecuteSqlQuery();

                // Получение данных из View.
                propertiesList = await _dbContext.PropertiesInfo.ToListAsync();
            } catch (Exception)
            {
                return null;
            }
```

```
            return propertiesList;
        }

        [Authorize]
        [HttpGet("systems")]
        public async Task<ICollection<MetabaseSystem>?> GetSystems()
        {
            List<MetabaseSystem> systemsList;
            try
            {
                await ExecuteSqlQuery();

                // Получение данных из View.
                systemsList = await _dbContext.SystemInfo.ToListAsync();
            } catch (Exception)
            {
                return null;
            }

            return systemsList;
        }

        private async Task ExecuteSqlQuery()
        {
            // Скрипт создает или обновляет View.
            string sqlScript;
            using (var reader = new StreamReader("DatabaseQuery.sql"))
            {
                sqlScript = await reader.ReadToEndAsync();
            }

            var quires = sqlScript.Split("GO").Where(str => str.Trim() != "");

            foreach (var query in quires)
            {
                await using var sqlConnection = new
SqlConnection(Program.DbConnectionString);
                await using var command = sqlConnection.CreateCommand();
                command.CommandText = query;
                sqlConnection.Open();
                command.ExecuteNonQuery();
            }
        }
    }
}
```

1.2.4 Model

1.2.4.1 AuthenticationOptions.cs

```
using System.Text;
using Microsoft.IdentityModel.Tokens;
```

```
namespace Service_for_database.Model;

public static class AuthenticationOptions
{
            public const string Issuer = "MetabaseService";
            public const string Audience = "DatabaseService";
            private const string Key = "mysupersecret_secretkey!123";
            public static SymmetricSecurityKey GetSymmetricSecurityKey()
            {
                    return new SymmetricSecurityKey(Encoding.ASCII.GetBytes(Key));
            }
}
```

1.2.4.2 DatabaseContext.cs

```
using EntityLib;
using Microsoft.EntityFrameworkCore;

namespace Service_for_database.Model;

    public class DatabaseContext : DbContext
    {
        public DbSet<MetabaseProperty> PropertiesInfo { get; set; } = null!;
        public DbSet<MetabaseSystem> SystemInfo { get; set; } = null!;

        public DatabaseContext(DbContextOptions<DatabaseContext> options)
            : base(options)
        {
            Database.EnsureCreated();
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<MetabaseProperty>()
                        .HasKey(nameof(MetabaseProperty.DBID),
                                nameof(MetabaseProperty.PropId));
            modelBuilder.Entity<MetabaseSystem>()
                .HasKey(nameof(MetabaseSystem.DBID),
                    nameof(MetabaseSystem.SystemId));
        }
    }
```

1.3 Библиотека сущностей (EntityLib)

1.3.1 FullPropertyEqualityComparer.cs

```
namespace EntityLib;

public class FullPropertyEqualityComparer : IEqualityComparer<MetabaseProperty>
{
            public bool Equals(MetabaseProperty? x, MetabaseProperty? y)
            {
                    if (ReferenceEquals(x, y)) return true;
```

```
                    if (ReferenceEquals(x, null)) return false;
                    if (ReferenceEquals(y, null)) return false;
                    if (x.GetType() != y.GetType()) return false;
                    return x.DBID == y.DBID && x.PropId == y.PropId && x.Name ==
y.Name &&
                           x.Description == y.Description && x.WWWTemplatePage ==
y.WWWTemplatePage &&
                           x.UpdateStatus == y.UpdateStatus;
            }

            public int GetHashCode(MetabaseProperty obj)
            {
                    return HashCode.Combine(obj.DBID, obj.PropId, obj.Name,
obj.Description,
                            obj.WWWTemplatePage, obj.UpdateStatus);
            }
}
```

1.3.2 FullSystemEqualityComparer.cs

```
namespace EntityLib;

public class FullSystemEqualityComparer: IEqualityComparer<MetabaseSystem>
{
            public bool Equals(MetabaseSystem x, MetabaseSystem y)
            {
                    if (ReferenceEquals(x, y)) return true;
                    if (ReferenceEquals(x, null)) return false;
                    if (ReferenceEquals(y, null)) return false;
                    if (x.GetType() != y.GetType()) return false;
                    return x.DBID == y.DBID && x.SystemId == y.SystemId &&
x.ElemNumber == y.ElemNumber &&
                           x.UpdateStatus == y.UpdateStatus && x.Elements ==
y.Elements &&
                           x.SystemInfo == y.SystemInfo && x.Description ==
y.Description;
            }

            public int GetHashCode(MetabaseSystem obj)
            {
                    return HashCode.Combine(obj.DBID, obj.SystemId,
obj.ElemNumber,
                            obj.UpdateStatus, obj._date, obj.Elements,
obj.SystemInfo,
                            obj.Description);
            }
}
```

1.3.3 MetabaseProperty.cs

```
namespace EntityLib
{
    public class MetabaseProperty
```

```
    {
        public int DBID { get; set; }

        public int PropId { get; set; }

        public string Name { get; set; }

        public string Description { get; set; }

        public string WWWTemplatePage { get; set; }

        public int UpdateStatus { get; set; }

        public override bool Equals(Object other)
        {
            // TODO пофиксить наллы
            var otherProperty = (MetabaseProperty) other;
            return DBID == otherProperty.DBID && PropId == otherProperty.PropId;
        }

        public override int GetHashCode()
        {
            return HashCode.Combine(DBID, PropId);
        }

        public void Modify(MetabaseProperty property)
        {
            Name = property.Name;
            Description = property.Description;
            WWWTemplatePage = property.WWWTemplatePage;
            UpdateStatus = property.UpdateStatus;
        }
    }
}
```

1.3.4 MetabaseSystem.cs

```
namespace EntityLib;

public class MetabaseSystem
{
            public int DBID { get; set; }

            public int SystemId { get; set; }

            public int ElemNumber { get; set; }

            public int UpdateStatus { get; set; }

            public DateTime _date { get; set; }

            public string Elements { get; set; }

            public string SystemInfo { get; set; }
```

```
            public string? Description { get; set; }

            public override bool Equals(Object other)
            {
                    var otherSystem = (MetabaseSystem) other;
                    return DBID == otherSystem.DBID && SystemId ==
otherSystem.SystemId;
            }

            public override int GetHashCode()
            {
                    return HashCode.Combine(DBID, SystemId);
            }

            public void Modify(MetabaseSystem system)
            {
                    ElemNumber = system.ElemNumber;
                    UpdateStatus = system.UpdateStatus;
                    _date = system._date;
                    Elements = system.Elements;
                    SystemInfo = system.SystemInfo;
                    Description = system.Description;
            }
}
```

1.4 Данные для подключения к БД

1.4.1 ConnectionStrings.txt

```
"BandGapConnection": "Server=localhost;initial catalog=BandGap;Integrated
Security=SSPI;",
"CrystalConnection": "Server=localhost;initial catalog=Crystal;Integrated
Security=SSPI;"
com.example.application.data.entity;
```

1.4.2 BandGapQuery.sql

```
USE [BandGap]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO


create OR ALTER view [dbo].[SystemInfo] as
select  3 as [DBID],
                    Substances.[SubstanceId] as SystemId,
                    count(SubstanceId) as ElemNumber,
                    1 as UpdateStatus,
                    GETDATE() as _date,
                    dbo.GetElements(NumElements, El1, El2, El3, El4, [Elements],
Compound) as [Elements],
                    Compound as SystemInfo,
```

```
                    null as [Description]
            from Substances
cross apply string_split(trim('-' from dbo.GetElements(NumElements, El1, El2, El3,
El4, [Elements], Compound)), '-')
group by SubstanceId, NumElements, El1, El2, El3, El4, [Elements], Compound
GO

create OR ALTER view [dbo].[PropertiesInfo] as
select  3 as [DBID],
                    [NOMPROP] as PropId,
                    [NAZVPROP] as [Name],
                    '' as [Description],
                    [HTML] as [WWWTemplatePage],
                [UpdateStatus]
              from _PropertiesConv
GO
```

1.4.3 CrystalQuery.sql

```
USE [Crystal]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

create OR ALTER view [dbo].[SystemInfo] as
select  1 as [DBID],
                    [HeadClue] as SystemId,
                    count(HeadClue) as ElemNumber,
                    1 as UpdateStatus,
                    GETDATE() as _date,
                    [Help] as [Elements],
                    [System] as SystemInfo,
        null as [Description]
            from dbo.HeadTabl
cross apply string_split(trim('-' from Help), '-')
group by HeadClue, Help, System
GO

create OR ALTER view [dbo].[PropertiesInfo] as
select  1 as [DBID],
                    [NOMPROP] as PropId,
                    [NAZVPROP] as [Name],
                    '' as [Description],
                    [HTML] as [WWWTemplatePage],
                    1 as [UpdateStatus]
            from dbo.Properties
GO
```

ПРИЛОЖЕНИЕ 1

ТЕРМИНОЛОГИЯ

Таблица 1

Термин	Определение
Веб-сервис	Идентифицируемая уникальным веб-адресом программная система со стандартизированными интерфейсами.
API	Описание способов, которыми одна компьютерная программа может взаимодействовать с другой программой. API может представлять собой набор классов, процедур, функций, структур или констант. Расшифровка: Application Programming Interface.
HTTP	Прикладной протокол для передачи гипертекстовых документов, таких как HTML. Расшифровка: Hypertext Transfer Protocol.
REST	Стиль архитектуры программного обеспечения для распределенных систем, как правило, используется для построения веб-служб. Расшифровка: Representational state transfer.
Микросервисная архитектруа	Подход, при котором единое приложение строится как набор небольших сервисов, каждый из которых работает независимо от других и коммуницирует с остальными используя протоколы, такие как HTTP.
Метаданные (метаинформация)	Информация о другой информации, или данные, относящиеся к дополнительной информации о содержимом или объекте.
База данных	Упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе.
Метабаза	База данных, хранящая информацию о метаданных.
Аутентификация	Процесс проверки подлинности учетных данных пользователя для его идентификации.
Авторизация	Процесс проверки прав пользователя и определения возможности доступа к различным ресурсам.
Cookie (куки)	Небольшой фрагмент данных, отправленный веб-сервером и хранимый на компьютере пользователя.
Cookie-based аутентифкация	Вид аутентификации, при котором для идентификации пользователя используются файлы cookie.
Токен аутентификации	Нечто, свидетельствующее об аутентичности и использующееся для идентификации его владельца.
JSON	Текстовый формат обмена данными, основанный на JavaScript. Расшифрока: JavaScript Object Notation.

Продолжение таблицы 1

JWT-токен	Открытый стандарт для создания токенов доступа, основанный на формате JSON. Расшифровка: JSON Web Token.
JWT аутентификация	Вид аутентификации, при котором для идентификации пользователя используются JWT-токены.
ADO.NET Entity Framework	Объектно-ориентированная технология доступа к данным.
Microsoft SQL Server	Система управления реляционными базами данных, разработанная корпорацией Microsoft.
Нереляционная база данных	База данных, в которой не используется табличная схема строк и столбцов.
NoSql	Обозначение широкого класса разнородных систем управления базами данных.

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

Лист регистрации изменений									
Номера листов (страниц)					Всего листов (страниц в докум.)	№ документа	Входящий № сопроводительного докум. и дата	Подп.	Дата
Изм.	Измененных	Замененных	Новых	Аннулированных					

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.15-01 12				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата