PROIECT 3

APLICATIE WEB - CALCULATOR

Grupa 441G

Studenti:

Ilie Robert

Popărascu Cosmin

Popa Sebastian

Profesor:

Anamaria Dumitrescu

Contents

[INTRODUCERE 3](#_Toc153811455)

[1. TEHNOLOGII FOLOSITE 3](#_Toc153811456)

[1.1 C# 3](#_Toc153811457)

[1.2 ASP.NET 3](#_Toc153811458)

[1.3 HTML 3](#_Toc153811459)

[1.4 CSS 3](#_Toc153811460)

[1.5 JavaScript 4](#_Toc153811461)

[1.6 SSMS 4](#_Toc153811462)

[2. FRONT-END 4](#_Toc153811463)

[2.1 HTML & CSS 4](#_Toc153811464)

[2.2 JAVASCRIPT 5](#_Toc153811465)

[3. BACK-END 7](#_Toc153811466)

[3.1 METODE 7](#_Toc153811467)

[4. CONECTAREA LA BAZA DE DATE 9](#_Toc153811468)

[5. MODELUL SI BAZA DE DATE 9](#_Toc153811469)

[5.1 MODELUL 9](#_Toc153811470)

[5.2 DbContext 10](#_Toc153811471)

[6. FUNCTIONAREA 11](#_Toc153811472)

[6.1 MENIUL PRINCIPAL 11](#_Toc153811473)

[Concluzii 13](#_Toc153811474)

[Bibliografie 13](#_Toc153811475)

[Anexe 13](#_Toc153811476)

[Anexa 1 – Front-End 13](#_Toc153811477)

[Anexa 2 – Back-End 18](#_Toc153811478)

[Figure 1 6](#_Toc153811559)

[Figure 2 7](#_Toc153811560)

[Figure 3 7](#_Toc153811561)

[Figure 4 8](#_Toc153811562)

[Figure 5 8](#_Toc153811563)

[Figure 6 10](#_Toc153811564)

[Figure 7 10](#_Toc153811565)

[Figure 8 12](#_Toc153811566)

[Figure 9 13](#_Toc153811567)

[Figure 10 13](#_Toc153811568)

[Figure 11 14](#_Toc153811569)

# INTRODUCERE

Aplicația web Calculator realizează calcule matematice (adunare, scădere, înmulțire, împărțire), după care le salvează într-o bază de date. La final, calculele pot fi descărcate sub formă de fișier Excel.

# TEHNOLOGII FOLOSITE

## 1.1 C#

C# este un limbaj de programare dezvoltat de Microsoft în 2000. Este un limbaj de programare orientat pe obiect proiectat pentru a fi eficient, sigur și ușor de înțeles. C# face parte din familia de limbaje de programare C, moștenind multe dintre caracteristicile acestora, dar introducând o sintaxă mai simplă și modernă. Este un limbaj versatil utilizat pe scară largă pentru dezvoltarea diverselor tipuri de aplicații, inclusiv aplicații desktop, aplicații web, aplicații mobile și servicii cloud.[1]

## 1.2 ASP.NET

ASP.NET este un cadru de dezvoltare web, tot de la Microsoft, construit pe baza limbajului C#. ASP.NET facilitează dezvoltarea și construcția de aplicații web robuste, sigure și scalabile. Furnizează diferite modele de programare, cum ar fi Web Forms, MVC (Model-View-Controller) și, în prezent, în special, ASP.NET Core.[2]

ASP.NET MVC:

ASP.NET MVC (Model-View-Controller) este o arhitectură de programare care împarte aplicația în trei componente principale:

Model: Reprezintă datele și logica de business a aplicației.

View: Furnizează o reprezentare vizuală a datelor și primește intrările utilizatorului.

Controller: Administrează interacțiunea dintre Model și View, procesând solicitările utilizatorului și actualizând datele.

## 1.3 HTML

HTML (HyperText Markup Language):

HTML este limbajul standard de marcă folosit pentru a crea și proiecta structura paginilor web. Dezvoltat de World Wide Web Consortium (W3C), HTML furnizează un set de etichete și elemente care definesc diferitele părți ale unei pagini web, cum ar fi titluri, paragrafe, legături, imagini și altele. Este fundamentul conținutului web și servește ca bază pentru construirea unor site-uri web vizual atractive și interactive.[3]

## 1.4 CSS

CSS (Cascading Style Sheets):

CSS este un limbaj de foi de stil care completează HTML prin controlul prezentării și a structurii paginilor web. Cu ajutorul CSS, dezvoltatorii pot defini stiluri, cum ar fi culori, fonturi, spațiere și poziționare, pentru a îmbunătăți aspectul vizual al elementelor HTML. CSS urmează o natură "cascadă", permițând stilurilor să fie moștenite, anulate sau combinate, oferind o apariție consecventă și elegantă pe întregul site.[4]

## 1.5 JavaScript

JavaScript este un limbaj de programare versatil și dinamic care adaugă interactivitate și comportament paginilor web. Inițial dezvoltat pentru a permite scripting-ul la nivelul clientului, JavaScript s-a evoluat pentru a deveni un limbaj puternic utilizat atât pentru dezvoltarea la nivelul frontend-ului, cât și a backend-ului. Permite dezvoltatorilor să manipuleze Modelul Obiectului Document (DOM), să gestioneze evenimentele utilizatorului și să comunice cu serverele, facilitând crearea de aplicații web dinamice și responsive.[5]

Integrarea HTML, CSS și JavaScript:

HTML, CSS și JavaScript lucrează împreună în mod armonios pentru a crea experiențe web moderne și interactive. HTML structurează conținutul, CSS stilizează prezentarea, iar JavaScript adaugă funcționalitate și interactivitate. Această triadă formează baza dezvoltării web, oferind dezvoltatorilor posibilitatea de a construi site-uri web și aplicații web atrăgătoare și ușor de utilizat.

## 1.6 SSMS

SSMS (SQL Server Management Studio):

SQL Server Management Studio (SSMS) este o unealtă grafică cu interfață de utilizator (GUI) dezvoltată de Microsoft pentru gestionarea bazelor de date SQL Server. Oferă administratorilor de baze de date și dezvoltatorilor un mediu cuprinzător pentru proiectarea bazelor de date, scrierea și executarea interogărilor SQL, gestionarea securității și monitorizarea performanțelor. SSMS simplifică administrarea bazelor de date SQL Server, oferind o serie de instrumente pentru eficientizarea sarcinilor legate de dezvoltarea și întreținerea bazelor de date.[6]

# FRONT-END

## 2.1 HTML & CSS

HTML și Stilizare:

Structura HTML definește o simplă calculatoare cu o zonă de afișare și butoane pentru cifre, operatori și alte controale.

Stilizarea este aplicată folosind CSS pentru a crea o interfață de utilizator vizual atrăgătoare și receptivă. Calculatorul este centrat pe pagină, având un fundal verde deschis.

## 2.2 JAVASCRIPT

Funcții JavaScript:

appendToDisplay(value):

Figure 1


Figure 1

Adaugă valoarea specificată la display-ul calculatorului.

Verifică ultimul caracter din display pentru a asigura o intrare corectă, evitând operatori consecutivi sau combinații invalide.

clearDisplay():

A computer screen with white text

Description automatically generated

Figure 2

Șterge conținutul display-ului calculatorului și resetează valoarea ascunsă a inputului folosită pentru trimiterea formularului.

calculateAndDisplayResult():

A computer screen with text

Description automatically generated

Figure 3

Analizează expresia matematică din display folosind eval() și calculează rezultatul.

Gestionează erorile potențiale și afișează "Calcul invalid" pentru expresiile invalide.

Rotunjește rezultatul la cinci zecimale pentru precizie.

updateResultDisplay(value):

A screen shot of a computer code

Description automatically generated

Figure 4

Actualizează display-ul cu valoarea furnizată.

Formular HTML:

A computer screen with text

Description automatically generated

Figure 5

Un formular este inclus cu un câmp de input ascuns (displayValue) pentru a captura valoarea display-ului calculatorului în vederea trimiterii.

Butonul egal (=) declanșează trimiterea formularului, apelând funcția calculateAndDisplayResult().

Link de Descărcare:

Un link de descărcare este furnizat în partea de jos a interfeței calculatorului (<a asp-action="Download" class="download">). Acest link duce către o acțiune de descărcare în cadrul aplicației ASP.NET asociate.

În ansamblu, codul integrează HTML pentru structură, CSS pentru stilizare și JavaScript pentru comportament interactiv, creând un calculator funcțional și vizual plăcut. Funcțiile JavaScript gestionează intrarea utilizatorului, efectuează calcule și actualizează dinamic afișajul. Formularul HTML permite trimiterea rezultatului calculat, iar linkul de descărcare sugerează funcționalități suplimentare în cadrul aplicației ASP.NET asociate.

# BACK-END

Controller și Constructor:

Controllerul (CalculatorsController) gestionează cererile primite de la interfața web asociată.

Constructorul primește un obiect CalculatorDbContext, care este utilizat pentru interacțiunea cu baza de date.

Acțiuni de Rute:

Index: Metoda care returnează o vedere pentru pagina principală.

Create: Metoda care primește datele postate de la formularul calculatorului.

Download: Metoda pentru descărcarea rezultatelor calculatorului din baza de date sub formă de fișier Excel.

## METODE

Metoda Create:

* Metoda Create este responsabilă pentru gestionarea cererilor de tip POST care vin de la interfața web atunci când utilizatorul realizează o operație de calcul.
* În primul rând, se primește o valoare de afișare (displayValue) care conține expresia matematică introdusă de utilizator în calculator.
* Se creează o instanță a clasei Calculator care reprezintă un model pentru datele de calcul.
* Se verifică validitatea expresiei matematice folosind metoda IsValidExpression.
* Dacă expresia este validă, se efectuează calculul utilizând metoda CalculateExpression și se salvează rezultatul și expresia în baza de date.
* Dacă expresia nu este validă, nu se întâmplă nimic (de exemplu, int y = 2; - o linie care pare să fie acolo doar pentru a satisface compilatorul, dar nu are altă funcționalitate).
* Se returnează NoContent pentru a indica că operația s-a încheiat fără a returna vreo valoare.

IsValidExpression:

* Această metodă primește o expresie matematică sub formă de șir de caractere (expression) și un parametru de ieșire (result). Scopul acestei metode este de a verifica dacă expresia este validă și, în caz afirmativ, de a calcula rezultatul expresiei.
* Metoda încearcă să evalueze expresia folosind metoda CalculateExpression și returnează true dacă evaluarea nu generează excepții și rezultatul nu este NaN (Not a Number).
* Dacă expresia nu este validă sau evaluarea eșuează, se generează o excepție, iar metoda returnează false. Rezultatul este setat pe 0 în acest caz.

CalculateExpression:

* Această metodă primește o expresie matematică sub formă de șir de caractere (expression) și returnează rezultatul evaluării expresiei sub formă de double.
* Metoda utilizează un obiect DataTable pentru a evalua expresia folosind un rând (DataRow) și o coloană (DataColumn) definite în DataTable.
* Se adaugă expresia dată în coloana "expression" a obiectului DataTable, iar apoi se încearcă să se evalueze această expresie. Rezultatul este returnat sub formă de double.

Metoda Download:

* Metoda Download este responsabilă pentru generarea unui fișier Excel care conține informațiile despre calculele realizate și stocate în baza de date.
* Se utilizează biblioteca EPPlus pentru manipularea fișierelor Excel.
* Se definește conținutul fișierului Excel, inclusiv un antet frumos cu culori și stiluri diferite.
* Se obțin calculele din baza de date și se adaugă în fișierul Excel.
* Fișierul Excel este generat și transmis către client pentru descărcare.

# CONECTAREA LA BAZA DE DATE

In 'Program.cs' se adaugă contextul bazei de date (CalculatorDbContext) utilizând Entity Framework Core, specificând conexiunea la bază de date prin citirea configurației din appsettings.json.

A black background with text on it

Description automatically generated

Figure 6

In appsettings.json se definește fișierul de configurare care conține setările aplicației, inclusiv nivelul de jurnalizare și informații de conectare la bază de date.

În secțiunea "ConnectionStrings", se specifică o conexiune la bază de date SQL Server sub numele "CalculatorConnectionString". Aceasta include detalii precum serverul, numele bazei de date și opțiuni de securitate.

A black screen with white text

Description automatically generated

Figure 7

# MODELUL SI BAZA DE DATE

## 5.1 MODELUL

Clasa Calculator din spațiul de nume Calculator\_P3.Models reprezintă o clasă de entitate care modelează structura datelor pentru calculele efectuate în aplicația noastră.

Proprietatea ID:

Este de tip Guid (Globally Unique Identifier), care reprezintă un identificator unic global. În contextul bazei de date, acesta este adesea folosit ca și cheie primară pentru a identifica în mod unic fiecare înregistrare în tabel.

Proprietatea Calcul:

Este de tip string și stochează expresia matematică asociată cu calculul efectuat. Acesta reprezintă intrarea de la utilizator sau expresia matematică care a fost evaluată.

Proprietatea Rezultat:

Este de tip string și stochează rezultatul calculului. Deși rezultatul ar putea fi de tip numeric, se pare că este stocat sub formă de șir de caractere (string). Acest lucru poate fi util, de exemplu, pentru a păstra formatul specific sau pentru a gestiona cazurile în care rezultatul nu este numeric (cum ar fi "Calcul invalid").

Proprietatea Data:

Este de tip DateTime și stochează data și ora când a fost efectuat calculul. Această proprietate permite urmărirea temporală a calculelor.

Această clasă este o reprezentare a unei entități care va fi mapată pe o tabelă într-o bază de date. În contextul unei aplicații ASP.NET Core care utilizează Entity Framework Core, această clasă este asociată cu un context de bază de date. Prin intermediul migrărilor, se creează și se aplică schema corespunzătoare în baza de date, conform definiției clasei Calculator.

## 5.2 DbContext

A computer screen shot of a black screen

Description automatically generated

Figure 8

Acest fișier conține clasa CalculatorDbContext, care este o subclasă a clasei DbContext din Entity Framework Core. Clasa CalculatorDbContext este responsabilă pentru gestionarea interacțiunilor cu baza de date și maparea entităților (cum ar fi Calculator) la tabelele corespunzătoare în baza de date.

Câteva aspecte notabile:

Constructorul primește un obiect de tip DbContextOptions și îl transmite constructorului clasei de bază (base(options)).

Proprietatea Calculators este de tip DbSet<Calculator>, reprezentând tabela asociată entității Calculator în baza de date.

Această clasă acționează ca un mediu centralizat pentru configurarea și gestionarea interacțiunilor cu baza de date, inclusiv definirea schemelor și gestionarea migrărilor.

În general, clasa CalculatorDbContext servește ca punte între aplicația ASP.NET Core și baza de date, facilitând operațiunile de citire și scriere în cadrul aplicației.

# FUNCTIONAREA

## 6.1 MENIUL PRINCIPAL

A screenshot of a computer

Description automatically generated

Figure 9

Calculatorul este încadrat într-un container (div cu clasa “calculator”) cu o înălțime de 75% și o lățime de 50% a paginii.

Ecranul calculatorului este reprezentat de un câmp de text de tip input (<input type="text">) cu id-ul "display", configurat să fie readonly și stilizat pentru a afișa expresiile matematice și rezultatele.

Butoanele numerice și de operații sunt reprezentate de input-uri de tip buton (<input type="button">), fiecare având o clasă specifică pentru stilizare și acțiuni associate (Figure 9).

A screenshot of a calculator

Description automatically generated

Figure 10

Funcția calculateAndDisplayResult() este apelată la apăsarea butonului "=" și utilizează funcția eval() pentru a evalua expresia matematică și a afișa rezultatul pe ecran (Figure 10).

A screenshot of a calculator

Description automatically generated

Figure 11

Butonul de descărcare (<a asp-action="Download" class="download">Download</a>) care, când este apăsat, declanșează o acțiune pentru descărcarea calculelor în format Excel (Figure 11).

# Concluzii

În concluzie, proiectul oferă o soluție funcțională și bine structurată pentru realizarea de calcule matematice online, cu posibilitatea de a păstra un istoric al acestora și de a descărca rezultatele într-un format ușor de accesat.

# Bibliografie

[1]Wikipedia: <https://ro.wikipedia.org/wiki/C_sharp>

[2]Wikipedia: <https://ro.wikipedia.org/wiki/ASP.NET>

[3]Wikipedia: <https://ro.wikipedia.org/wiki/HyperText_Markup_Language>

[4]Wikipedia: <https://ro.wikipedia.org/wiki/Cascading_Style_Sheets>

[5]Wikipedia: <https://ro.wikipedia.org/wiki/JavaScript>

[6]Wikipedia: <https://en.wikipedia.org/wiki/SQL_Server_Management_Studio>

# Anexe

## Anexa 1 – Front-End

@model Calculator\_P3.Models.Calculator

@{

ViewData["Title"] = "Calculator";

Layout = "";

}

<style>

body{

background-color: lightseagreen;

}

.container{

height:100%;

width:100%;

align-items: center;

display: flex;

justify-content: center;

}

.calculator{

border: 2px black solid;

width: 50%;

height: 75%;

position: absolute;

top: 8%;

}

table{

height: 80%;

width: 100%;

background-color: black;

}

td{

width:25%;

}

.display{

height: 100%;

width: 75%;

}

#display{

height: 100%;

width: 100%;

position: relative;

font-size: 2.7rem;

padding: 2%;

background-color: black;

color: white;

border: none;

user-select: none;

cursor: default;

}

display:focus{

outline:none;

}

.clear{

width: 25%;

height: 100%;

font-size: 1rem;

border: none;

}

#clear{

width:100%;

height:100%;

background-color: #dbdbdb;

font-size: 2rem;

}

#clear:hover{

background-color: #f0efed;

}

.button{

height: 100%;

width: 100%;

font-size: 1rem;

border: none;

}

.nr:hover{

background-color: #e6a14e;

}

.op:hover{

background-color:#999896;

}

.nr{

background-color: #e88813;

border: none;

}

.op{

background-color: #73716e;

border: none;

}

.top-container{

position: relative;

height: 20%;

width: 100%;

align-items: center;

display: flex;

}

.download{

background-color:white;

color:black;

position: absolute;

bottom: 3%;

height: 10%;

width: 20%;

border: solid 5px green;

border-radius: 20px;

font-size: 200%;

align-items: center;

display: flex;

justify-content: center;

text-decoration: none;

}

.download:hover {

background-color: #c6f7cc;

}

</style>

<div class="container">

<div class="calculator">

<div class="top-container">

<div class="display">

<input type="text" id="display" readonly>

</div>

<div class="clear">

<button id="clear" value="c" onclick="clearDisplay()">C</button>

</div>

</div>

<table>

<tr>

<td>

<input type="button" value="7" class="button nr" onclick="appendToDisplay('7')">

</td>

<td>

<input type="button" value="8" class="button nr" onclick="appendToDisplay('8')">

</td>

<td>

<input type="button" value="9" class="button nr" onclick="appendToDisplay('9')">

</td>

<td>

<input type="button" value="\*" class="button op" onclick="appendToDisplay('\*')">

</td>

</tr>

<tr>

<td>

<input type="button" value="4" class="button nr" onclick="appendToDisplay('4')">

</td>

<td>

<input type="button" value="5" class="button nr" onclick="appendToDisplay('5')">

</td>

<td>

<input type="button" value="6" class="button nr" onclick="appendToDisplay('6')">

<td>

<input type="button" value="/" class="button op" onclick="appendToDisplay('/')">

</td>

</tr>

<tr>

<td>

<input type="button" value="1" class="button nr" onclick="appendToDisplay('1')">

</td>

<td>

<input type="button" value="2" class="button nr" onclick="appendToDisplay('2')">

</td>

<td>

<input type="button" value="3" class="button nr" onclick="appendToDisplay('3')">

</td>

<td>

<input type="button" value="-" class="button op" onclick="appendToDisplay('-')">

</td>

</tr>

<tr>

<td>

<input type="button" value="0" class="button nr" onclick="appendToDisplay('0')">

</td>

<td>

<input type="button" value="." class="button nr" onclick="appendToDisplay('.')">

</td>

<**form** **asp-action**="Create">

<td>

<input type="hidden" name="displayValue" id="displayValue">

<**input** **asp-for**="Calcul" **type**="submit" **value**="=" class="button op" onclick="calculateAndDisplayResult()">

</td>

</**form**>

<td>

<input type="button" value="+" class="button op" onclick="appendToDisplay('+')">

</td>

</tr>

</table>

</div>

<**a** **asp-action**="Download" class="download"><p>Download</p></**a**>

</div>

<script>

function appendToDisplay(value) {

const display = document.getElementById('display');

const displayValueField = document.getElementById('displayValue');

const lastChar = display.value.charAt(display.value.length - 1);

if (display.value == '' || lastChar == '\*' || lastChar == '-'

|| lastChar == '+' || lastChar == '/' || lastChar == '.') {

if (value == '\*' || value == '+' || value == '/' || value == '.') {

display.value = display.value;

displayValueField.value = display.value;

}

else {

display.value += value;

displayValueField.value = display.value;

}

}

else

{

display.value += value;

displayValueField.value = display.value;

}

}

function clearDisplay() {

const display = document.getElementById('display');

const displayValueField = document.getElementById('displayValue');

display.value = '';

displayValueField.value = '';

}

function calculateAndDisplayResult() {

const displayValueField = document.getElementById('displayValue');

const display = document.getElementById('display');

try {

const result = eval(displayValueField.value);

if (isNaN(result)) {

updateResultDisplay('Calcul invalid');

} else {

const roundedResult = parseFloat(result.toFixed(5));

updateResultDisplay(roundedResult);

}

} catch (error) {

updateResultDisplay('Calcul invalid');

}

}

function updateResultDisplay(value) {

const display = document.getElementById('display');

display.textContent = value;

}

function updateResultDisplay(result) {

const display = document.getElementById('display');

display.value = result;

}

</script>

## Anexa 2 – Back-End

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using Calculator\_P3.Data;

using Calculator\_P3.Models;

using System.Data;

using Microsoft.IdentityModel.Tokens;

using System.Diagnostics.Contracts;

using OfficeOpenXml;

using OfficeOpenXml.Style;

namespace Calculator\_P3.Controllers

{

public class CalculatorsController : Controller

{

private readonly CalculatorDbContext \_context;

public CalculatorsController(CalculatorDbContext context)

{

\_context = context;

}

// GET: Calculators

public async Task<IActionResult> Index()

{

return View();

}

// GET: Calculators/Create

public IActionResult Create()

{

return View();

}

// POST: Calculators/Create

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(string displayValue)

{

Calculator calculator = new Calculator();

double result;

if (IsValidExpression(displayValue, out result))

{

calculator.Rezultat = result.ToString();

calculator.Calcul = displayValue;

calculator.ID = Guid.NewGuid();

calculator.Data = DateTime.Now;

\_context.Add(calculator);

await \_context.SaveChangesAsync();

}

else

{

int y = 2;

}

return NoContent();

}

private bool IsValidExpression(string expression, out double result)

{

try

{

result = CalculateExpression(expression);

return !double.IsNaN(result);

}

catch (Exception)

{

result = 0;

return false;

}

}

private double CalculateExpression(string expression)

{

DataTable table = new DataTable();

table.Columns.Add("expression", typeof(string), expression);

DataRow row = table.NewRow();

table.Rows.Add(row);

return double.Parse((string)row["expression"]);

}

public ActionResult Download()

{

ExcelPackage.LicenseContext = LicenseContext.NonCommercial;

var package = new ExcelPackage();

var worksheet = package.Workbook.Worksheets.Add("Calculations");

var calcule = \_context.Calculators.OrderBy(c => c.Data).ToList();

worksheet.Cells["E1:H1"].Merge = true;

worksheet.Cells["E2:E3"].Merge = true;

worksheet.Cells["F2:F3"].Merge = true;

worksheet.Cells["G2:G3"].Merge = true;

worksheet.Cells["H2:H3"].Merge = true;

var mergedCell = worksheet.Cells["E1:H1"];

mergedCell.Value = "Calculations";

mergedCell.Style.HorizontalAlignment = OfficeOpenXml.Style.ExcelHorizontalAlignment.Center;

mergedCell.Style.VerticalAlignment = OfficeOpenXml.Style.ExcelVerticalAlignment.Center;

var fill = mergedCell.Style.Fill;

fill.PatternType = OfficeOpenXml.Style.ExcelFillStyle.Solid;

fill.BackgroundColor.SetColor(System.Drawing.Color.FromArgb(242, 152, 48)); // Orange

mergedCell.Style.Font.Size = 24;

int j = 3;

for (int i = 0; i < calcule.Count + 1; i++)

{

if (i == 0)

{

worksheet.Column(5).Width = 40;

worksheet.Column(6).Width = 20;

worksheet.Column(7).Width = 20;

worksheet.Column(8).Width = 20;

worksheet.Cells["E2:E3"].Value = "ID";

worksheet.Cells["F2:F3"].Value = "Calculation";

worksheet.Cells["G2:G3"].Value = "Result";

worksheet.Cells["H2:H3"].Value = "Date";

for (int colIndex = 5; colIndex <= 8; colIndex++)

{

var comb = worksheet.Cells[2, colIndex, 3, colIndex];

comb.Merge = true;

comb.Style.Font.Bold = true;

comb.Style.HorizontalAlignment = OfficeOpenXml.Style.ExcelHorizontalAlignment.Center;

comb.Style.VerticalAlignment = OfficeOpenXml.Style.ExcelVerticalAlignment.Center;

comb.Style.Font.Size = 14;

var fill1 = comb.Style.Fill;

fill1.PatternType = OfficeOpenXml.Style.ExcelFillStyle.Solid;

fill1.BackgroundColor.SetColor(System.Drawing.Color.FromArgb(255, 235, 156)); // Gold

}

}

else

{

worksheet.Cells["E" + (j + 1)].Value = calcule[i - 1].ID;

worksheet.Cells["F" + (j + 1)].Value = calcule[i - 1].Calcul;

worksheet.Cells["G" + (j + 1)].Value = calcule[i - 1].Rezultat;

worksheet.Cells["H" + (j + 1)].Value = calcule[i - 1].Data;

worksheet.Cells["H" + (j + 1)].Style.Numberformat.Format = "yyyy-mm-dd hh:mm:ss";

j++;

for (int colIndex = 5; colIndex <= 8; colIndex++)

{

var valori = worksheet.Cells[j, colIndex];

valori.Style.HorizontalAlignment = OfficeOpenXml.Style.ExcelHorizontalAlignment.Center;

var fill2 = valori.Style.Fill;

fill2.PatternType = OfficeOpenXml.Style.ExcelFillStyle.Solid;

fill2.BackgroundColor.SetColor(System.Drawing.Color.FromArgb(75, 222, 78)); // Green

}

}

}

var stream = new MemoryStream(package.GetAsByteArray());

return new FileStreamResult(stream, "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet")

{

FileDownloadName = "Calculations.xlsx"

};

}

}

}