# Test – Softwareentwicklung 4. JG (1.6.2023)

Name: Johannes Werner

|  |  |  |
| --- | --- | --- |
| **Beispiel** | **Mögliche Punkte** | **Punkte** |
| **Theorie** | 12 |  |
| **Primfaktorenzerlegung** | 10 |  |
| **Klassenbibliothek für DB** | 4 |  |
| Gesamt | **26** |  |

## Primfaktorenzerlegung (10 Punkte)

Erstellen Sie eine Webapplikation mit dem Page-basiertem Ansatz zur Berechnung von Primfaktoren.

Erstellen Sie dazu folgendes:

* Eine Webseite mit dem Namen Primefactors, die per URL-Parameter eine Zahl bekommt und die Primfaktoren dieser Zahl berechnet.
* Eine Webseite mit dem Namen Primeinput, die die Möglichkeit bietet eine Zahl einzugeben. Nach Klick auf einen Button soll die Webseite Primfactors mit der eingegebenen Zahl als URL-Parameter aufgerufen werden.
* Erstellen Sie in Ihrem Menü einen Eintrag, in welchem Sie auf die Seite Primeinput verlinken.

Hilfe zu Primfaktorenzerlegung:

z.B.: 3757208 = 2 \* 2 \* 2 \* 7 \* 13 \* 13 \* 397

Probiere in einer Schleife die eingegebene Zahl durch potenzielle Faktoren (beginnend mit 2) zu teilen. Wenn die Zahl durch den Faktor ohne Rest teilbar ist, ist dies ein Primfaktor. Wenn der Faktor die Zahl nicht ohne Rest teilt, dann erhöhe den Faktor um 1.

Primfactors.cshtml:

@page

@model Test301primeFactorization.Pages.PrimefactorsModel

@{

}

<h1>Calculate Primefactors</h1>

<p></p>

<h2>The primefactor for the number @Model.Number is:</h2>

<h2>@Model.Result</h2>

<br />

<button><**a** class="nav-link text-dark" **asp-area**="" **asp-page**="/Primeinput">Calculate next primefactors</**a**></button>

Primfactors.cshtml.cs:

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.RazorPages;

namespace Test301primeFactorization.Pages

{

public class PrimefactorsModel : PageModel

{

public string Result { get; set; } = "Sry but no Primefactor got calculated";

public int Number { get; set; }

public void OnGet(int number)

{

Number = number;

Result = CalculatePrimefactors(number);

}

private string CalculatePrimefactors(int number)

{

if (number < 2) {

return number.ToString();

}

string solution = "";

while (number % 2 == 0) {

solution += " \* 2";

number /= 2;

}

int factor = 3;

while (factor <= number) {

if(number % factor == 0) {

solution += " \* " +factor.ToString();

number /= factor;

}

else

{

if(factor < number)

{

factor++;

}

else

{

return solution.Substring(2);

}

}

}

return solution.Substring(2);

}

}

}

Primeinput.cshtml:

@page

@model Test301primeFactorization.Pages.PrimeinputModel

@{

}

<**form** method="post" **asp-page-handler**="CalculatePrimefactors">

<div class="row">

<div class="col-6">

Number:

<input type="number" id="number" name="number" value="0" />

</div>

</div>

<br />

<button type="submit">Calculate primefactors</button>

</**form**>

Primeinput.cshtml.cs:

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.RazorPages;

namespace Test301primeFactorization.Pages

{

public class PrimeinputModel : PageModel

{

public void OnGet()

{

}

[HttpPost]

public IActionResult OnPostCalculatePrimefactors(int number)

{

return RedirectToPage("Primefactors", new { number = number});

}

}

}

## Klassenbibliothek für Datenbankzugriff (4 Punkte)

Erstellen Sie eine Klassenbibliothek für den Datenbankzugriff unter Verwendung von Microsoft Entity-Framework Core. Verwenden Sie die Datenbank „chinook.db“ und generieren Sie den DBContext und die Modell-Klassen aller in der Datenbank enthaltenen Entitäten.

**Geben Sie dieses Dokument und Ihre Programme (Zip-Archiv des gesamten Projektordners) in Eduvidual ab.**