Portfolio Nr. 6 - Überprüfung und Verarbeitung eines String mit ASP.Net

Portfolio

Fakultät für Wirtschaft

Studiengang Wirtschaftsinformatik

Studienjahrgang 2018

Kurs C

DUALE HOCHSCHULE BADEN-WÜRTTEMBERG
VILLINGEN-SCHWENNINGEN

Bearbeiter:

David Bährens

Betreuender Dozent:

Prof. Dr. Kimmig

Dualer Partner:

DATEV eG

Inhaltsverzeichnis

Abkürzungsverzeichnis

Abb.	Abbildung
ASP	Active Server Pages
FCL	Framework Class Library
CLR	Common Language Runtime
UI	User Interface
OS	Operating System
MS	Microsoft
z. B.	zum Beispiel
API	Application-Programming-Interface
ASP	Active Server Pages
REST	Representational State Transfer
MVC	Model-View-Controller
VS	Visual Studio
SPA	Single Page Application
Z.	Zeile
RP	Razor Page
i. d. R.	in der Regel

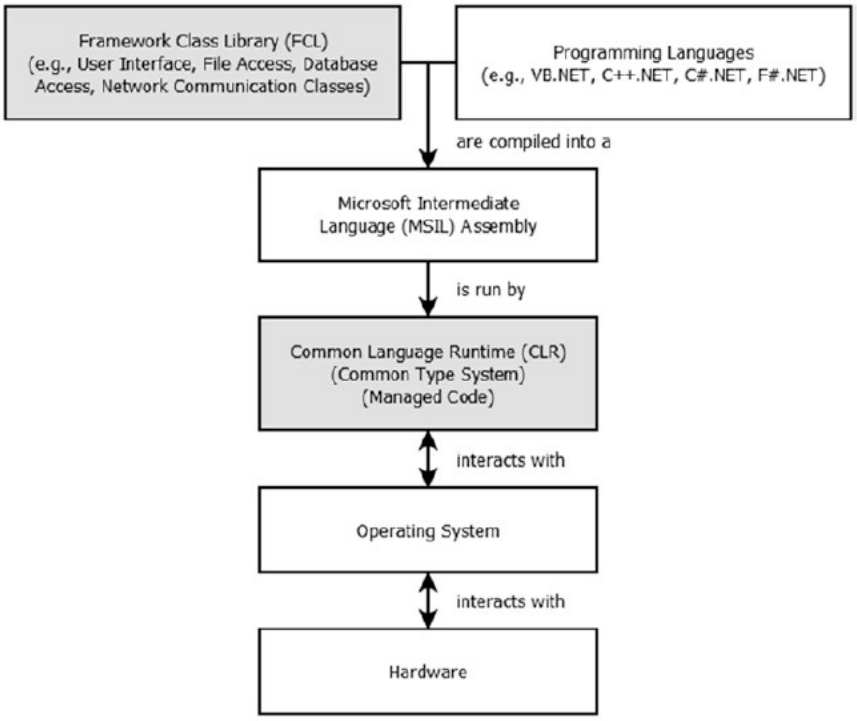
Abbildungsverzeichnis

1 Theoretische Grundlagen

1.1 ASP.Net

Bei ASP.Net handelt es sich um ein modulares und serverseitiges Web-Framework zur Entwicklung von dynamischen Web-Anwendungen. ASP steht für „Acrive Server Pages“. Dieses ist Teil des Microsoft (MS) Softwareentwicklungs und Execution-Framework .Net. .Net dient unter Windows zur Erstellung von Anwendungsprogrammen. Dessen wesentlichen beiden Bestandteile sind zum einen die Framework Class Library (FCL) und die Common Language Runtime (CLR). FCL ist eine umfangreiche Klassenbibliothek in .Net. Sie enthält beispielsweise User Interface (UI)-, File Access- oder Netzwerk-Kommunikationsklassen. Bei CLR handelt es sich um die Laufzeitumgebung in der .Net Anwendungen ausgeführt werden. .Net Programme, beispielsweise eine C# Anwendung, greifen nicht direkt auf das Betriebssystem (OS) zu. Stattdessen wir der Programmcode in die sogenannte MS Intermediate Language Assemby kompiliert und dann in der CLR ausgeführt. Die CLR wiederum greift dann direkt auf das darunterliegende OS zu (siehe Abb. 1).[1]

Abbildung 1: .Net Framework



Quelle: Beasley, Robert E., .Net Basics, 2020, S. 9

Das .Net Framework wird jedoch fortlaufend durch .Net Core abgelöst. Hierbei han-

[1] Vgl. Beasley, Robert E., .Net Basics, 2020, S. 8

delt es sich um eine Open-Source-Plattform von MS und eine Modernisierung des .Net Frameworks. Ein besonderer Vorteil dieses moderneren Frameworks ist seine Plattformunabhängigkeit. Da ASP.Net auf .Net basiert erfolgt die Ablösung hier analog mit ASP.Net Core.[2]

ASP.Net wiederum stellt verschiedene Frameworks für die Entwicklung von Web-Anwendungen zur Verfügung. Die wichtigsten sind Folgende. Zum einen gibt es das inzwischen veraltete, ereignisgesteuerte Framework „Web Forms“. Bei diesem wurden Oberflächen über einen Designer mit einer Drag-and-Drop Mechanik erstellt und die Logik wurde über einen Eventhandler implementiert. In der Vergangenheit kam Web Forms jedoch teilweise mit der Zustandslosigkeit des Webs in Konflikt.
Ein moderneres, aktionsgesteuertes Framework stellt dagegen ASP.Net MVC dar. Dieses folgt den MVC Pattern, wodurch UI, Logik und Daten voneinander getrennt werden. MVC steht dabei für die drei wesentlichen Bestandteile in die eine MVC-Anwendung zerlegt wird, „Model-View-Controller“. Das Model gibt dabei die Datenstruktur, die View die Darstellung bzw. die UI und der Controller beinhaltet die Logik und verbindet das Model mit der View.[3]
Dann sind da noch die Web Pages, die über die neue Razor Syntax verfügen. Razor Pages (RP's) sind eine moderne Alternative zur Entwicklung von dynamischen Websites und sie stellen den Nachfolger von ASP.Net MVC dar. Sie werden in einem gesonderten Kapitel erläutert, da sie für diese Arbeit von größerer Bedeutung sind.
Zuletzt ist noch ASP.Net Web API zu nennen, mit dessen Hilfe Web-Schnittstellen wie z. B. REST entwickelt werden können.[4]

Die .Net Entwicklung ist mit einer Reihe kompatibler Programmiersprachen möglich. Hierzu zählen beispielsweise Visual Basic, C# oder F#. Diese Arbeit bezieht sich im folgenden lediglich auf C#. C# ist eine objektorientierte und typsichere höhere Programmiersprache von MS. Ursprünglich war sie primär auf Windows ausgerichtet, inzwischen ist sie jedoch sehr universell und kann für die Entwicklung von Web-Apps eingesetzt werden.[5]

1.2 Razor Pages

RP's basieren auf ASP.Net MVC und zeichnen sich durch die Razor Syntax aus. Mit dieser können statische HTML Webseiten mit C# dynamisch gemacht werden. Dies zeichnet sich dadurch aus, dass eine Webseite durch eine .cshtml Datei er-

[2] Augsten, Stephan, .Net Core, 2020
[3] Rouse, Margaret, MVC, 2016
[4] Gutsch, Jürgen, ASP.Net, 2017
[5] Augsten, Stephan, C#, 2019

stellt wird, also eine Kombination aus C#, mittels der Razor Syntax und HTML. Der dort enthaltene Code wird dabei serverseitig in reines HTML übersetzt. RP's vereinen die Vorteile einer verhältnismäßig einfachen Syntax mit einem leichtgewichtigen und dennoch mächtigen Framework. Anders als ASP.Net MVC nutzen RP's das Model-View-ViewModel-Pattern statt echtem MVC. Hierbei handelt es sich um eine spezielle Form der MVC-Architektur, bei der kein Controller, sondern stattdessen ein ViewModel, das bei RP's PageModel genannt wird, eingesetzt wird. Dieses ist eine spezielle Implementierung eines Controllers, welcher die Logik und Programmcode hinter einer View darstellt. Jede View hat ein eigenes ViewModel, statt einem zentralen Controller, der alle Views steuert. Model und View funktionieren analog zu ihrer Funktionalität bei MVC. Die View erhält ihre benötigten Daten dabei mittels Data Binding. Analog zu MVC ist der Zweck MVVM's die Trennung von Logik, UI und Daten. Diese erfolgt bei MVVM allerdings seitenbasiert.[6] RP's sind automatisch auch bei einem ASP.Net MVC Projekt aktiviert und anders herum kann auch in einer RP mit der MVC-Architektur gearbeitet werden falls dies gewünscht wird. [7]

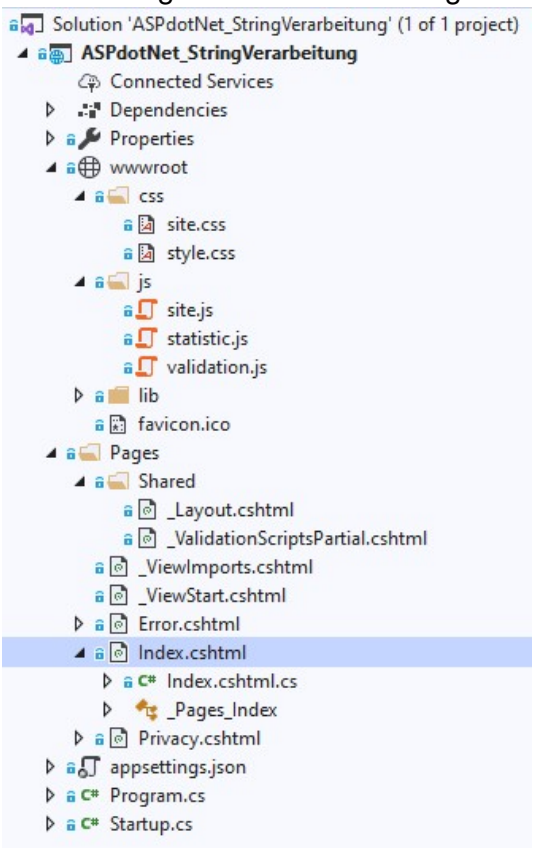
1.2.1 Aufbau eines Razor Page Projektes

Ein Standard Razor Projekt besteht aus verschiedenen unterschiedlichen Dateien (siehe Abb. 2). Die wohl wichtigsten befinden sich in dem „Page“ Ordner. Dieser enthält .cshtml Dateien und .cshtml.cs Dateien. Die .cshtml Dateien sind die Views bzw. die eigentliche RP. Mit Hilfe der Razor Syntax könnte hierin auch Logik implementiert werden und somit jeglicher Quellcode in einer Datei gebündelt werden. Dies widerspricht allerdings dem Prinzip der Datentrennung bei MVVM und ist kein guter Programmierstil. Die .cshtml.cs dagegen sind die PageModels der RP. Diese können sowohl die Datenstruktur, als auch die Logik beinhalten. Dies erkennt man im Übrigen auch aus der Abbildung 2, da hier offensichtlich kein separates Model implementiert wurde. Ein solches würde, falls benötigt, in einem separatem Model-Ordner, direkt unter dem Wurzelverzeichnis implementiert werden. Diese sind, genau wie die PageModels, C# Dateien. PageModels vereinen damit eine breite Menge an Funktionalitäten, wie beispielsweise HTTPContext, den ModelState oder die Behandlung von Requests und Responses, wie z. b. HTTP-Anfragen, die in MVC getrennt würden.[8] Der Startpunkt der Website bildet die Index.cshtml. Eine weitere wichtige Page-Datei ist die _Layout.cshtml, welche ein Design-Template für jede andere Page darstellt. In ihr kann z. B. der Header der Website für alle RP's festgelegt werden. Insgesamt erfüllen Pages mit einem „_“ als Präfix jeweils eine be-

[6] o. V., MVVM, 2017
[7] o. V., RP's, 2019
[8] Jones, Matthew, Razor vs. MVC, 2019

sondere Aufgabe, sind allerdings nicht über einen URL-Aufruf erreichbar. Der Aufruf von „https://Hostname/_Layout“ z. B. würde daher zu einem 404 Http Error führen. Offensichtlich werden RP's, anders als bei ASP.Net MVC, Dateien nach dem Zweck gegliedert. Es gibt nicht nur einen Controller der die gesamte Geschäftslogik in sich vereint, sondern stattdessen gibt es viele C# Dateien, PageModel darstellen und jeweils einer View zugeordnet sind.

Abbildung 2: Aufbau Razor Pages



In wwwroot befinden sich statische Dateien, wie CSS-Style-Sheets, Bilder oder JavaScript(JS)-Dateien. Im lib Ordner wiederum befinden sich Drittanbieter-Pakete. Defaultmäßig sind dies JQuery und Bootstrap, was im Verlauf der Arbeit noch erläutert wird.

Darüber hinaus befinden sich direkt in dem Wurzelverzeichnis eine Konfigurationsdatei im JSON-Format für die gesamte Anwendung namens „appsettings.json“, eine „Program.cs“, die den Startpunkt der Anwendung darstellt und nach eine „Startup.cs“. In letzterer können z. B. benötigte Services hinzugefügt und weitere Konfigurationen vorgenommen werden.[9] Folgendes Beispiel zeigt, wie der RP Service in eine ASP.Net Webanwendung integriert werden können:

```
1  public void ConfigureServices(IServiceCollection services)
2  {
3      services.AddRazorPages();
4  }
```

1.2.2 Aufbau einer Razor Page

Eine RP zeichnet sich, wie bereits erwähnt, vor allem auch durch die Razor Syntax aus, die in den HTML-Code eingefügt wird. Der Server erkennt sie durch ein vorangestelltes „@“. So kann beispielsweise ganzer C#-Code einfach in den HTML-Code eingefügt werden und ihn dadurch mit Funktinalität ausstatten. Dies geschieht innerhalb eines Codeblocks: `@{ C#-Code }`. Darüber hinaus können auch einzelne

[9] o. V., Aufbau RP's, 2020

Funktionen, wie eine if-Funktion, oder auch eine Schleife mit einem vorangestellten @ benutzt werden: `@if ( Condition ) { C#-Code }`. Es ist auch möglich von der View aus auf bestimmte Variablen des PageModels zuzugreifen. Dies geschieht über `@Model.variable`. Um auf diese Variable zugreifen zu können muss sie allerfings als Public deklariert werden. Eine weitere Möglichkeit, Daten vom PageModel an die View zurückzugeben ist ViewData, wobei es sich um ein dictionary verschiedener Objekte handelt. Dieses dictionary wird automatisch an die View übergeben. Somit kann jederzeit auf die darin enthaltenen Objekte, über den jeweiligen Key, zugegriffen werden. ViewData Attribute werden wie folgt im Page Model definiert und anschließend über `@ViewData["Key"]` aufgerufen.[10]

```
public class IndexModel : PageModel
{
    [ViewData]
    public string Key { get; set; }
    // Oder alternativ
    ViewData["Key"] = "Value"
}
```

Außerdem existieren verschiedene Tag-Helpers, also wiederverwendbarer HTML-Code für die Vereinfachung von ASP Funktionalitäten. Ein Beispiel hierfür wäre der Validation Message Tag Helper `asp-validation-for`, der einem span-Element eine Validation Error Message zuordnet, indem er ihm das HTML-Element `data-valmsg-for` zuweist. Bei einem Client seitigen Validationsfehler zeigt jQuery die Fehlermeldung innerhalb des spans. Der Tag kann allerdings auch für eine serverseitige Validation verwendet werden. Diese könnte beispielsweise so aussehen:

```
<input type="text" asp-for="IBAN" />
<span asp-validation-for="IBAN"> </span>
```

```
public class IndexModel : PageModel
{
    [BindProperty]
    [Required]
    public string IBAN { get; set; }
}
```

Wurde nun keine IBAN, bei einem Request an den Server, eingegeben wird unterhalb des Textfeldes eine Fehlermeldung ausgegeben. Ein weiteres Beispiel wäre der `asp-page-handler`, mit dem ein spezieller Page Handler ausgeführt werden kann. Bei Handler Methoden handelt es sich um Funktionen, die automatisch

[10] o. V., ViewData, o. D.

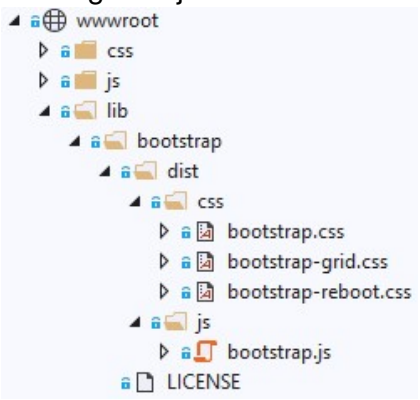
bei einem entsprechenden HTTP-Request ausgeführt werden. Die Handler Methode `OnGet()` wird beispielsweise bei einer Get-Anfrage an den Server ausgeführt. Mit dem `asp-page-handler` können nun verschiedene Page Handler implementiert werden. So würde ein `OnPostProcessing()` Handler nur bei einem bestätigen von `<button type=" submit"asp-page-handler="Processing">Submit</button>` ausgeführt werden.[11] [12]

Jede Page beginnt stehts mit `@page` in der ersten Zeile. Hierdurch weiß ASP.Net, dass es diese datei wie eine RP behandeln muss. Hierdurch kann die Page selber Aktionen ausführen und ist nicht auf einen Controller angewiesen. Des Weiteren muss ein `@model` angegeben werden, mit dem dazugehörigen Model dieser View. Bei RP's ist dies in der Regel (i. d. R.) das PageModel.[13]

1.3 Bootstrap

Bei Bootstrap handelt es sich um ein Frontend-Framework zur optischen Gestaltung einer Website. Es dient primär der schnellen und einfachen Umsetzung eines Responsive Web-Designs, mit dem Websites für jede Displaygröße optimal gestaltet sein sollen. Damit verfolgt Bootstrap stark die Mobile-first Philosophie. Ursprünglich handelte es sich bei Bootstrap um eine Technologie von Twitter, die dann als Open-Source-Projekt veröffentlicht wurde. Wenn man sich seine Bestandteile anschaut, dann basiert es überwiegend auf CSS, aber auch auf HTML und JS. Seine Bestandteile sind zum einen das Design von Basis-HTML-Elementen und JS Plugins, welche zumeist auf jQuery basieren, bei dem es sich um eine freie JS-Bibliothek handelt. Darüber hinaus gibt es noch verschiedene Komponenten, bei denen es sich um von Bootstrap vordefinierte CSS-Klassen zur Gestaltung von HTML-Elementen handelt. Bootstrap ist zudem leicht anpassbar, sollten die vordefinierten Gestaltungsmöglichkeiten nicht den eigenen Wünschen entsprechen, wie wenn man beispielsweise ein neues Farblayout implementieren möchte.[14]

Abbildung 3: Bootstrap in Razor Page Projekt mit VS



Bootstrap eignet sich besonders gut für die ASP.Net Einbindung, denn bei der

[11] o. V., Page Handler, 2018
[12] Anderson, Rick; Mullen, Taylor; Vicarel, Dan, Razor Syntax, 2020
[13] Jones, Matthew, Razor vs. MVC, 2019
[14] Bhaumik, Snig, Bootstrap, 2015, S. 7-11

Erstellung eines Standard RP Projektes mit Visual Studio (VS) ist Bootstrap bereits vorinstalliert und kann direkt verwendet werden (siehe Abb. 3).

2 Praxisteil: Dokumentation und erklärung des Codes

2.1 Grundfunktionalität

Die Grundfunktinalität beinhaltet eine ASP.Net-Webanwendung, in der der Benutzer eine beliebige Zeichenkette, die mindestens aus zehn Wörtern bestehen muss, eingeben kann. In dieser Zeichenkette sollen dann alle Vokale durch „i“ ersetzt werden. Die Eingabe muss validiert werden, sodass der Nutzer nicht weniger als zehn wörter eingibt. Diese Anwendung wurde als Single Page Application (SPA) realisiert. Das verwendete Framework für diese Aufgabe waren RP's. Diese Wahl liegt vor allem in der Einfachheit des Frameworks begründet. Da es sehr „seitenbasiert“ist und wegen der Razor Syntax ist es relativ einfach hiermit eine SPA umzusetzen. Das ASP.Net MVC Framework wäre hierfür unnötig kompliziert und die erhöhte Datentrennung wäre für ein solches Projekt nicht sinnvoll.
Da es sich um eine SPA handelt, wurde, aus Vereinfachungsgründen, lediglich die Index-Datei als RP verwendet und keine zusätzlichen erstellt.

Zunächst musste die Website für eine SPA vorbereitet werden. Die einfachste Möglichkeit dies umzusetzen ist, indem sich der RP Syntax bedient wird. Hierzu wurde ein boolean namens „started “, der defaultmäßig false ist, im PageModel angelegt und eine if-Bedingung hierzu in der View erstellt. Wenn diese Variable false ist, wird ein Startbildschirm mit einem Startbutton angezeigt. Bei einem Klick auf diesen wird die Variable auf true gesetzt, die if-Bedingung damit wahr und die eigentliche Anwendung wird angezeigt. Dies geschieht über den bereits erwähnten `aps-page-handler`. Dieser Tag helper, wurde dem Startbutton hinzugefügt. Auf Klick ruft er die Handler Methode „OnPotStarting()“ auf, in der die Variable started auf true gesetzt wird und danach die Website erneut zurückgegeben wird. Beim zurückgeben der Page wird die Seite neu geladen, und da der Wert in der if-Bedinung nun true ist, wird die eigentliche Anwendung statt der Startseite angezeigt. Um sie wieder zu beenden ist auf der Anwendungsseite ein exit Buttton der die Variable wieder auf false setzt. Dieser funktioniert analog zum Startbutton

```
1 // Wurde die Anwendung über den Startbutton gestartet, wird die eigentliche
      Anwendung angezeigt
2 @if (@Model.started)
3 {
4     // Deaktiviert die Anwenung über den Handler OnPostExitPage()
5     <button type="submit" asp-page-handler="ExitPage">Exit</button>
6 }
7 // Defaultmäßig wird erst mal die folgende Startseite angezeigt
```

```
8  else
9  {
10 <form method="post">
11     <button type="submit" id="startButton" asp-page-handler="Starting">Anwendung
           starten</button>
12 </form>
13 }
```

```
1  public class IndexModel : PageModel
2  {
3      // Variable, ob die Anwendung gestartet wurde oder nicht
4      public bool started = false;
5  }
6
7  public IActionResult OnPostStarting()
8  {
9      // aktiviert die Anwendung, indem die Startvariable true gesetzt wird und die
           RP returned wird
10     started = true;
11     return Page();
12 }
```

Hiernach wird die Grundfunktionalität umgesetzt, welche in die if-Anweisung ab Zeile (Z.) vier des vorherigen Codes implementiert wird. Der Input erfolgt über eine textarea, die mittels DataBinding an das PageModel übergeben wird. Dort wird die Zeichenkette dann verarbeitet. Es existieren zwei Formen der Validierung. Eine clientseitige und eine serverseitige. Die clientseitige wurde mit HTML und Bootstrap realisiert. Sie validiert, ob überhaupt eine Eingabe erfoglt oder nicht. Hierzu wurden verschiedene Bootstrap-Klassen implementiert und das Attribut `required` zu der textarea hinzugefügt (Zeile 6). Bei einer fehlgeschlagenen Validierung wird Z. vier bis sechs ausgegeben. Damit diese Validierung funktioniert muss allerdigns noch JS hinzugefügt werden (siehe Anhang 1).

```
1  <form method="post" class="needs-validation" id="inputForm" novalidate>
2      <div class="input-group">
3              <textarea asp-for="inputString" placeholder="Zeichenkette eingeben"
                   class="form-control shadow-lg" aria-label="inputString" id="
                   inputString" required></textarea>
4          <div class="invalid-feedback">
5              Bitte geben Sie eine Zeichenkette ein. Sonst funktionert das Ganze
                   nicht...
6          </div>
7      </div>
```

```
8       <button type="submit" asp-page-handler="Processing" form="inputForm">Absenden
            </button>
9   </form>
```

Bei der serverseitigen Validierung mit C# mitteln wird überprüft, ob mindestens zehn wörter eingegeben wurden. Dies wird mit Regular Expressions (RegEx) überprüft.

```
1   [BindProperty]
2   [RegularExpression(@"(.*\s){9}..*", ErrorMessage = "Upps. Sie haben zu wenig Wö
        rter eingegeben.")]
3   public string inputString { get; set; }
```

Stimmt die eingegebene Zeichenkette nicht mit der RegEx überein wird eine Fehlermeldung ausgegeben. Diese sollte allerdings abschreckender wirken, als die einfache „required“ Validierung. Daher wird sie an einer anderen Stelle ausgegeben und später mit Bootstrap gestaltet. Die Kennzeichnung, wo die Fehlermeldung stehen soll erfolgt mit `<span asp-validation-for=" inputString"></span>`.

Nun muss der Input verarbeitet werden und alle Vokale durch ein „i“ ersetzt werden. Dies geschieht in der „OnPostProcessing() “ Handler-Methode. Da er nur verarbeitet werden soll, wenn die Validierung positiv war, wird diese in Z. 12 abgefragt. War sie positiv wird über jeden Buchstaben in der Zeichenkette iteriert und mit den Vokalen in dem „vokale-Array“verglichen. Ist ein Buchstabe in dem Vokale Array enthalten, wird er durch ein „i“ ersetzt (Z. 14-24). Der dadurch neu entstandene String wird in Zeile 28 zurückgegeben. Außerdem wird beim eingeben des Wortes „exit“der User auf die Startseite zurück geleitet (Z. 5-10).

```
1   char[] vokale = { 'a', 'e', 'o', 'u' };
2   string[] words = { };
3
4   public IActionResult OnPostProcessing()
5   {
6       inputString = inputString.ToLower();
7       words = inputString.Split(" ");
8
9       if (words.Contains("exit"))
10      {
11          return RedirectToPage("/index");
12      }
13
14      if (ModelState.IsValid)
15      {
```

```
16          foreach (var character in inputString)
17          {
18              if (vokale.Contains(character))
19              {
20                  newChar = 'i';
21                  newString += newChar;
22              }
23              else
24              {
25                  newString += character;
26              }
27          }
28      }
29
30      ViewData["result"] = newString;
31      return Page();
32  }
```

Das Ergebnis wird nun in der View angezeigt. Dabei soll überprüft werden, ob die Validierung positiv oder negativ war. War sie positiv wird, der neue String angezeigt, ansonsten die bereits erwähnte Validierungsfehlermeldung. Es musste hierzu eine weitere Variable „activated“eingeführt werden, die defaultmäßig false ist und beim ersten ausführen von OnPostProcessing() true gesetzt wird. Dies liegt daran, dass ansonsten zu Beginn er Anwendung die textarea bereits validiert würde und eine Fehlermeldung ausgegeben wurde, obwohl der Benutzer noch gar nicht die Möglichkeit hatte eine Eingabe zu tätigen.

```
1  @if (@ModelState.IsValid)
2  {
3      <code>@ViewData["result"]</code>
4  }
5  else if (!@ModelState.IsValid && @Model.activated)
6  {
7      <span asp-validation-for="inputString"></span>
8  }
```

2.2 Erweiterungen

Die primäre Erweiterung stellt eine Historie dar, in der alle eingegeben Zeichenketten, mit der veränderten neuen Zeichenkette ausgegeben werden und die bei Beenden der Anwendung gelöscht wird. Darüber hinaus gibt es eine kleine Statistik, die u. a. mit Google Charts visualiesiert wurde, der man entnehmen kann welche Vokale wie häufig mit dem Buchstaben „i“ersetzt wurden.

2.2.1 Historie

Um die Historie umzusetzen muss eine statische Variable angelegt wird, die auch nach einer POST-Anfrage nicht gelöscht wird (Z. 2) Der `historyEntry`-Liste wird dann einfach am Ende jedes Aufrufes der OnPostProcessing() Handler-Methode die Variable `inputString`, also die eingegebene Zeichenkette und die Variable `newString`, also die überarbeitete Zeichenkette hinzugefügt und diese wiederum der zweidimensionalen Liste `history` hinzugefügt. Danach wird `historyOutput = history` gesetzt, da so einfacher in der View auf die Variable zugegriffen werden kann.

```
1 List<string> historyEntry = new List<string>();
2 static List<List<string>> history = new List<List<string>>();
3 public List<List<string>> historyOutput = new List<List<string>>();
```

In der View wird dann mit einer foreach-Schleife über die Liste iteriert und eine Tabelle mit den einzelnen Werten erstellt.

```
1  @if (@Model.activated)
2  {
3      <table>
4      <!-- ... -->
5      @foreach (var entry in @Model.historyOutput)
6      {
7          <tbody>
8              <tr>
9                  <td>@entry[0]</td>
10                 <td>@entry[1]</td>
11             </tr>
12         </tbody>
13     }
14     </table>
15 }
```

Zuletzt muss die Historie bei jedem Beenden der Anwendung ebenfalls gelöscht werden. Dies geschieht zum Einen in der bereits erwähnten Handler-Methode OnPostExitPage(), indem die statische Variable `history` einfach mit `history = new List<List<string»();` neu initialisiert wird. Zum anderen muss diese Initialisierung auch in die bereits erwähnte if-Bedingung, zur Überprüfung, ob das Wort „exit“ eingegeben wurde, hinzugefügt werden:

```
1 // ...
2 if (words.Contains("exit"))
3 {
4     history = new List<List<string>>();
5     return RedirectToPage("/index");
```

```
}
```

2.2.2 Statistik

Die Statistik soll beinhalten, welche Vokale wie oft ersetzt wurden und zudem wie viele Wörter eingegeben wurden. Die Anzahl der Vokale kann einfach ermittelt werden, indem die eingegebene Zeichenkette an dem jeweiligen Vokal „gesplittet“ wird, um danach die Anzahl der hieraus entstendenen Elemente, um eins subtrahiert, zu zählen:

```
// aNum bezeichnet dabei die Anzahl aller A's
aNum = inputString.Split('a').Length - 1;
eNum = inputString.Split('e').Length - 1;
oNum = inputString.Split('o').Length - 1;
uNum = inputString.Split('u').Length - 1;
```

Diese Werte werden nun in einem Google Chart visualisiert. Bei einem Google Chart handelt es sich allerdigns um JS Code, weswegen die Variablen hierfür erst einmal zugänglich gemacht werden müssen. Dies kann mit einem versteckten HTML-Input-Feld erreicht werden, dem als Value die C#-Variablen hinzugefügt werden:

```
    <input type="hidden" value="@Model.aNum" id="aNum" />
```

Nun lässt sich in JS einfach auf die Werte dieser Input-Elemente, über den Ausdruck `document.getElementById("aNum").value`, zugreifen. Diese Werte werden dann JS-Variablen zugewiesen und in dem Google Chart verwendet (siehe Anhang 2). Der Chart selber wird dann innerhalb des folgenden div's angezeigt:

```
<div id="piechart" style="width: 100%; height: 500px;"></div>
```

Die eingegebenen Wörter sollen ebenfalls gezählt werden. Hierzu wird die eingegebene Zeichenkette an einem Leerzeichen gesplittet auf das Ergebnis wieder die `Length`-Methode angewendet.

```
words = inputString.Split(" ");
wordNum = words.Length;
```

2.3 Graphische Gestaltung

Für die optische Gestaltung wurde das bereits erläuterte Bootstrap Framework verwendet. Damit wurden verschiedene Bootstrap-Klassen den HTML-Elementen hinzugefügt, um ihr Design zu verändern. Beispielsweise wurden Schatten mit `shadow-lg` oder Buttons mit `bg-success` grün gefärbt. Jegliche HTML-Formulare wurden mit

Bootstrap mitteln designt. So erhielt der „Absenden“-Button z. B. die `btn-outline-sucess` Klasse und die textarea die `form-control` Klasse. Außerdem wurde der Header mit der Klasse `sticky-top` sticky gemacht. Dies bedeutet, dass er beim scrollen der Webseite mit scrollt und immer an der selben Position bleibt. Ein weiterer Aspekt war die Gestaltung der Ausgabe der überarbeiteten Zeichenkette und der Ausgabe eines serverseitigen Validierungsfehlers. Beides wurde mit einem `Jumbotron`-Element realsiert, also einer Art Textbox, die bei einer erfolgreichen Anfrage grün und bei einem Validierungsfehler rot gefärbt wird. Die clientseitige Valdierung wird über die Klasse `invalid-feedback` realisiert. Die Tabelle in der die Historie aufgelistet ist wurde ebenfalls über Bootstrap mit verschiedenen table-Klassen realisiert.
Die Webseite sieht nach dieser Gestaltung folgendermaßen aus:

Abbildung 4: RP - Startseite

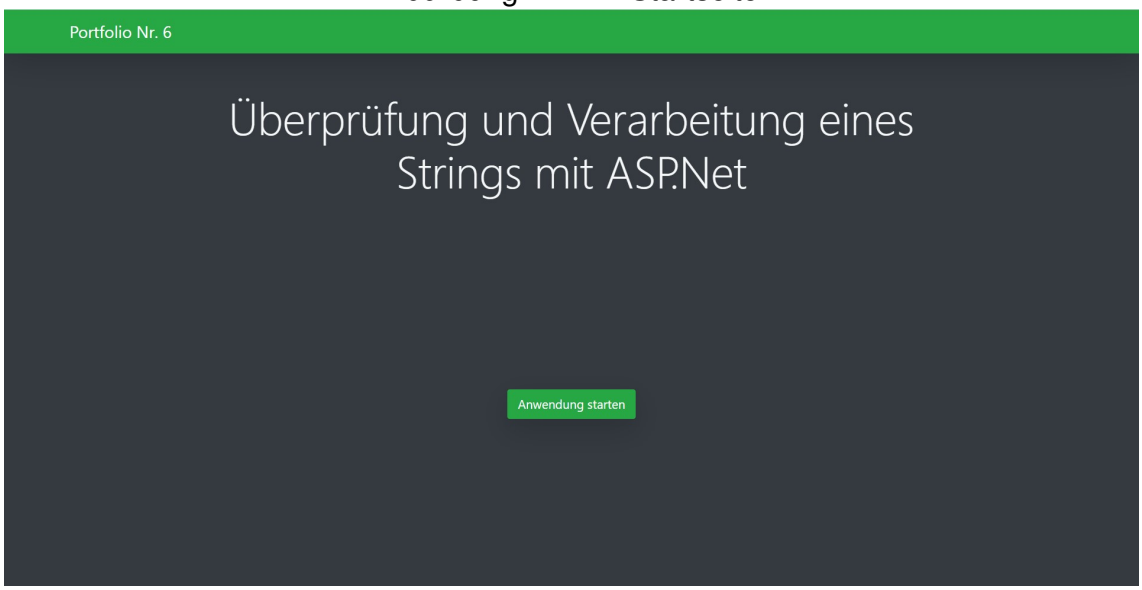


Abbildung 5: RP - Eingabe

Abbildung 6: RP - Ausgabe mit Historie

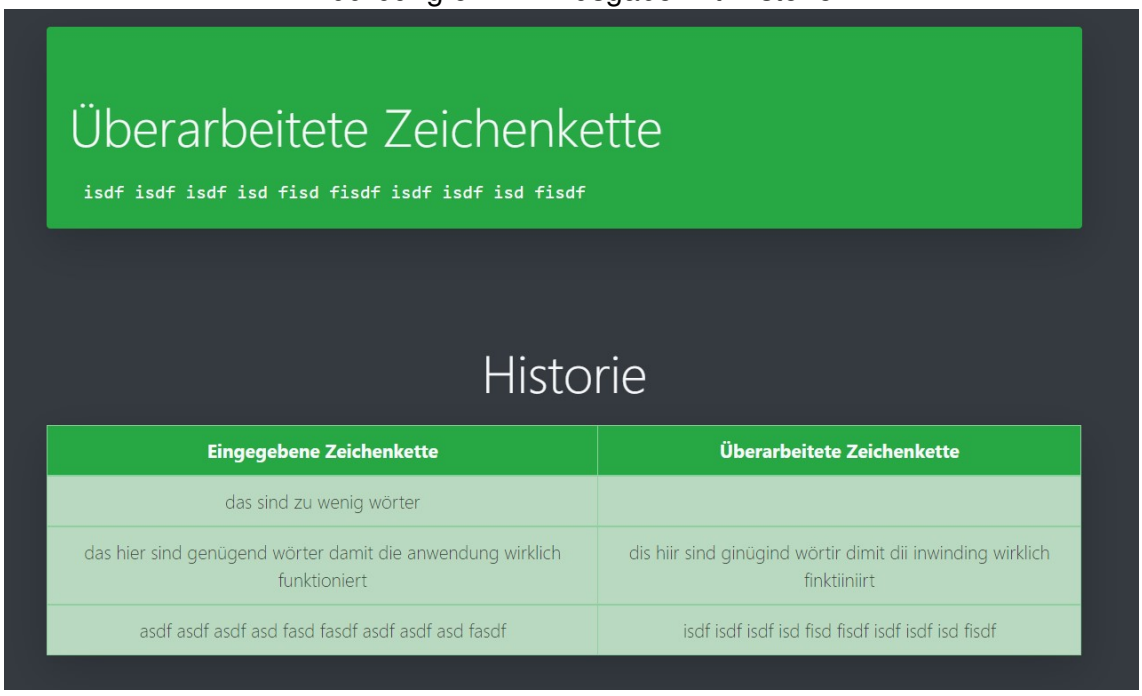


Abbildung 7: RP - Validierungsfehler

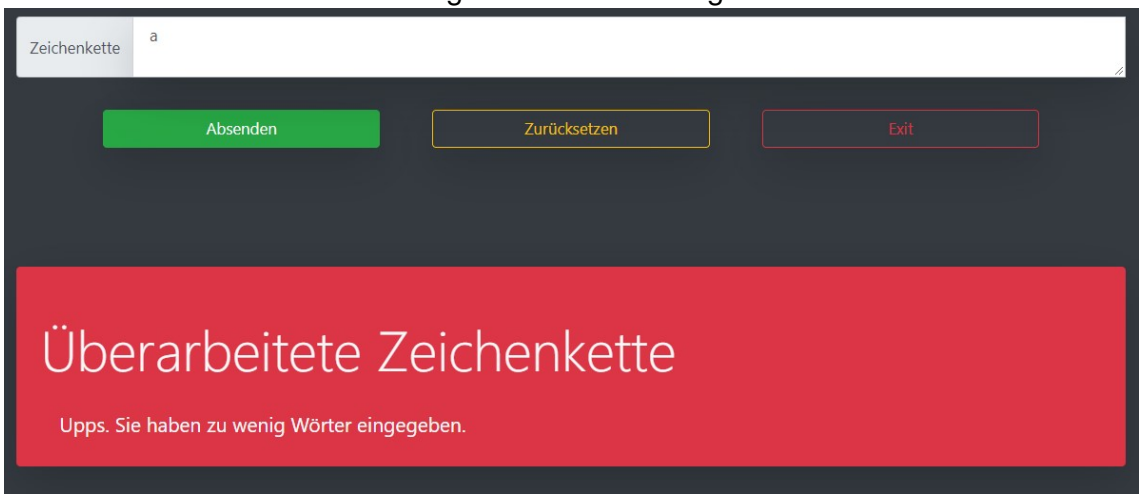
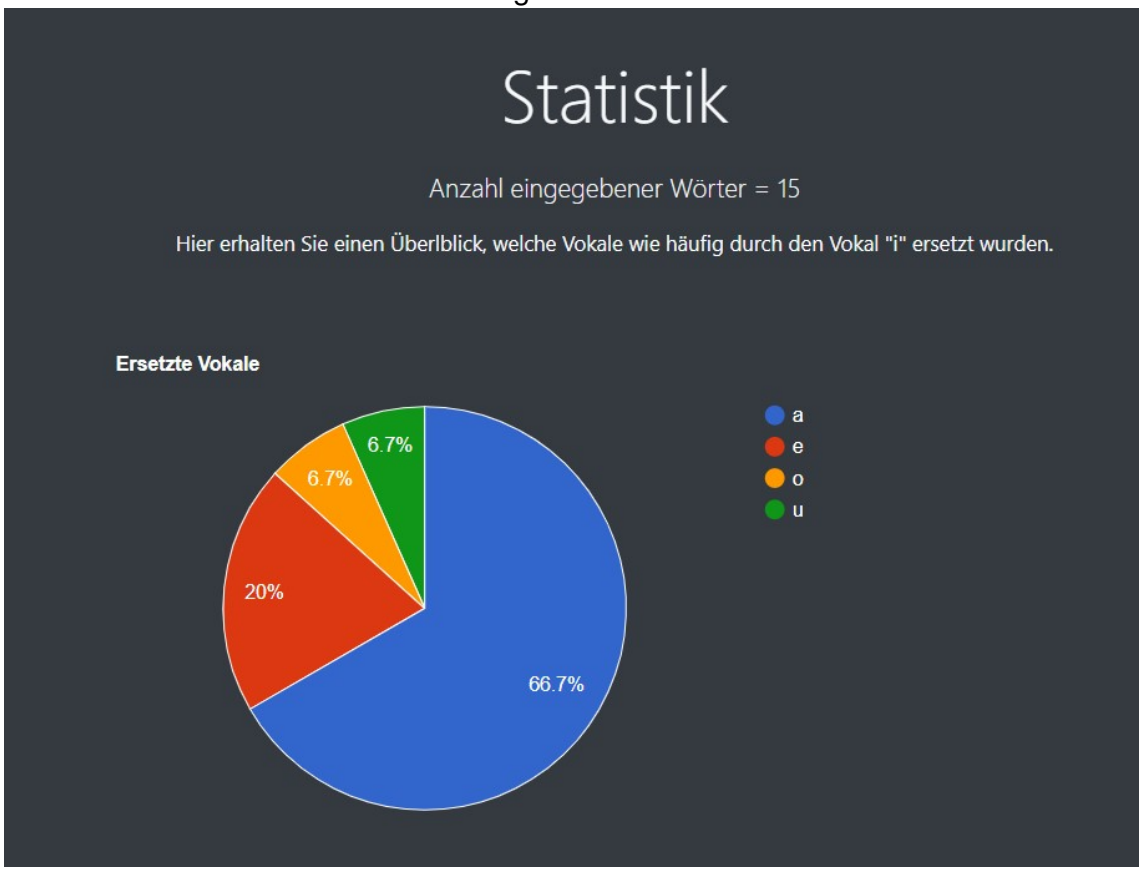


Abbildung 8: RP - Statistik



Anhang 1: Clientseitige Validierung

```
<script>
// Example starter JavaScript for disabling form submissions if there are invalid fields
(function() {
    'use strict';
    window.addEventListener('load', function() {
        // Fetch all the forms we want to apply custom Bootstrap validation styles to
        var forms = document.getElementsByClassName('needs-validation');
        // Loop over them and prevent submission
        var validation = Array.prototype.filter.call(forms, function(form) {
            form.addEventListener('submit', function(event) {
                if (form.checkValidity() === false) {
                    event.preventDefault();
                    event.stopPropagation();
                }
                form.classList.add('was-validated');
            }, false);
        });
    }, false);
})();
</script>
```

Quelle: o. V., Bootstrap Validation, o. D.

Anhang 2: Google Chart

```
a = Number(document.getElementById("aNum").value);
e = Number(document.getElementById("eNum").value);
o = Number(document.getElementById("oNum").value);
u = Number(document.getElementById("uNum").value);

google.charts.load('current', { 'packages': ['corechart'] });
google.charts.setOnLoadCallback(drawChart);

function drawChart() {

    var data = google.visualization.arrayToDataTable([
    ['Vokal', 'Änderungen'],
    ['a', a],
    ['e', e],
    ['o', o],
    ['u', u]
    ]);

    var options = {
        title: 'Ersetzte Vokale',
        backgroundColor: { fill: 'transparent' },
        titleTextStyle: {
            color: 'white'
        },
        hAxis: {
            textStyle: {
                color: 'white'
            },
            titleTextStyle: {
                color: 'white'
            }
        },
        vAxis: {
            textStyle: {
                color: 'white'
            },
            titleTextStyle: {
                color: 'white'
            }
        },
        legend: {
            textStyle: {
```

```
                color: 'white'
            }
        }
    };

    var chart = new google.visualization.PieChart(document.getElementById('
        piechart'));

    chart.draw(data, options);
}
```

Literatur

Beasley, Robert E., [.Net Basics] Essential ASP.NET Web Forms Development, Berkeley, CA 2020

Augsten, Stephan, [.Net Core] Definition „Microsoft .NET Core Platform“, Was ist .NET Core?, 09.04.2020, https://www.dev-insider.de/was-ist-net-core-a-914978/ (24.05.2020)

Rouse, Margaret, [MVC] Model View Controller (MVC), 10.2016, https://www.computerweekly.com/de/definition/Model-View-Controller-MVC (26.05.2020)

Gutsch, Jürgen, [ASP.Net] Microsofts Web-Frameworks im Vergleich, Die Qual der Wahl, 15.06.2017, https://www.dotnetpro.de/frontend/qual-wahl-1226135.html (26.05.2020)

Augsten, Stephan, [C#] Definition „C-Sharp“, Was ist C#?, 09.08.2019, https://www.dev-insider.de/was-ist-c-a-846162/ (26.05.2020)

o. V., [MVVM] The Model-View-ViewModel Pattern, 07.08.2017, https://docs.microsoft.com/de-de/xamarin/xamarin-forms/enterprise-application-patterns/mvvm (26.05.2020)

o. V., [Razor Pages] Welcome To Learn Razor Pages, 08.05.2019, https://www.learnrazorpages.com/ (26.05.2020)

o. V. [Aufbau Razor Pages] First look at Razor Pages, 08.01.2020 https://www.learnrazorpages.com/first-look (26.05.2020)

Jones, Matthew, [Razor vs. MVC] How Does Razor Pages Differ From MVC In ASP.NET Core?, 04.03.2019, https://exceptionnotfound.net/razor-pages-how-does-it-differ-from-mvc-in-asp-net-core/ (26.05.2020)

Anderson, Rick; Mullen, Taylor; Vicarel, Dan, [Razor Syntax] Razor syntax reference for ASP.NET Core, 12.02.2020, https://docs.microsoft.com/de-de/aspnet/core/mvc/views/razor?view=aspnetcore-3.1 (26.05.2020)

o. V., [ViewData] Working With ViewData in Razor Pages, o. D., https://www.learnrazorpages.com/razor-pages/viewdata (26.05.2020)

o. V., [Page Handler] Handler Methods in Razor Pages, 24.10.2018. https://www.learnrazorpages.com/razor-pages/handler-methods (26.05.2020)

o. V., [Bootstrap Validation] Forms, o. D., https://getbootstrap.com/docs/4.3/components/forms/ (26.05.2020)

Selbstständigkeitserklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit mit dem Thema: „Portfolio Nr. 6 - Überprüfung und Verarbeitung eines String mit ASP.Net“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort, Datum **Unterschrift**

Nürnberg, den 30.05.2020