Mika Apell

Developing Blazor Web Application Running on Microsoft Azure

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

13 April 2020

Author Title Number of Pages Date	Mika Apell Developing Blazor Web Application Running on Microsoft Azure 30 pages 13 April 2020
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Janne Salonen, Head of School, School of ITC, Metropolia

The purpose of this final year project was to see how easy it is to develop a single-page Blazor Web Application, and could it be done without using JavaScript. The goal of the final year project was to be able to create at least a minimum viable product version of the designed web application and publish it in a cloud service.

To find out how easy it is to develop a Blazor Web Application, a sample application for making notes and calculations needed in brewing beer was created. Inspiration for the project came from personal experience in home-brewing and a need for an easy to use tool.

The goal of the thesis was achieved, and a sample application was created and published. The released version ended up being an MVP as deadline came to soon. This was due the writer’s lack of experience in web development rather than in technology itself.

As a conclusion, Blazor is a great alternative for JavaScript for web development, as there is nothing you could not do with it. Time will tell how popular it might become.

Keywords	Blazor, C#, Web Development

Tekijä Otsikko Sivumäärä Aika	Mika Apell Developing Blazor Web Application Running on Microsoft Azure 30 sivua 13.4.2020
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Ohjelmistotuotanto
Ohjaajat	Osaamisaluepäällikkö, ICT ja tuotantotalous Janne Salonen

Tämän insinöörityön tarkoituksena oli selvittää, kuinka helppoa on luoda web-sovellus Blazor-teknologialla ja voiko sen tehdä käyttämättä JavaScriptiä. Tavoitteena oli kehittää pienin julkaisukelpoinen tuote -versio insinöörityötä varten kehitetystä sovelluksesta ja julkaista se pilvipalvelussa.

Nähdäkseen kuinka helppoa Blazor web-applikaation tekeminen on, suunniteltiin esimerkki sovellus oluen valmistusprosessin muistiinpanojen ja laskujen tekemiseen kotona. Inspiraatio sovellukseen tuli omasta kokemuksesta oluen valmistuksesta kotona.

Insinöörityön tavoite saavutettiin ja sovellus saatiin tehtyä ja julkaistua. Julkaistu versio oli pienin julkaisukelpoinen tuote, koska aikamääreet tulivat vastaan. Tämä johtui ennemminkin kirjoittajan kokemuksen vähyydestä web-sovellusten kehityksen parissa, kuin itse teknologiasta.

Lopputuloksena voidaan todeta Blazor:in olevan hyvä vaihtoehto JavaScriptille web-sovellusten tekemistä varten. Aika näyttää, kuinka suosittu siitä tulee.

Avainsanat	Blazor, C#, Web-kehitys

Contents

List of Abbreviations

.NET Core	An open-source development framework. Allows building cross-platform apps.
.NET Framework	Part of the .NET platform, created by Microsoft.
Angular	Platform for building mobile and desktop web applications.
API	An application programming interface.
ASP.NET	Open source web framework, created by Microsoft.
Azure	Cloud computing service created by Microsoft.
Blazor	Free and open-source web framework.
C#	Object-oriented programming language.
C	Procedural programming language.
C++	Object-oriented programming language.
CSS	Cascading Style Sheets, adds styles to websites,
DOM	The Document Object Model.
HTML	Markup language for creating Web pages.
IDE	An integrated development environment.
JavaScript	Scripting or programming Language, usually used for the Web development.
MVC	Model-View-Controller
MVP	Minimum Viable Product

Razor	ASP.NET programming syntax.
Rust	An open-source systems programming language.
TypeScript	An open-source programming language.
UI	User Interface.
WebAssembly	A portable binary-code format.
VM	Virtual Machine

1 Introduction

Modern web applications consist of two parts: client side and server side. The client side is mostly developed using UI frameworks, such as Angular or React, which use JavaScript or Typescript as their preferred language. The server side of the application is created using frameworks such as .NET and languages such as C#. Switching between two different frameworks and languages can be cumbersome. [1]

For a long time, there has not been any other possibilities, as JavaScript has dominated the front-end side of development. In this absence of options, Blazor was introduced. Blazor provides a full stack .NET development experience by allowing the use of .NET throughout the application. [1]

Purpose of this final year project is to see how easy it is to develop a single-page Blazor web application and can it be done without using JavaScript. As part of the final year project, a small web application will be developed and published in Azure. The goal of this final year project is to be able to create and publish at least the minimum viable product version of the sample web application, without using a line of JavaScript code.

This thesis is structured to three parts. First part of the thesis has a brief overview of technologies behind Blazor web applications and introduction of Azure cloud services. Second part discusses the sample project and in third and last part of the thesis discuss the results.

2 Technologies

In this chapter the technologies behind the Blazor web applications and thesis project are introduced. First the main technologies under the hood of Blazor are presented: WebAssembly and ASP.NET Core. Following is a brief overview of Blazor itself and a brief look to Microsoft Azure.

2.1 WebAssembly

WebAssembly is a low-level assembly-like language. It can run with almost native performance in modern web browsers. It allows the use of languages such as C/C++ and Rust in web development. WebAssembly also runs alongside with JavaScript. It is being developed as a web standard by the W3C WebAssembly Working Group and Community Group and all major browsers vendors participate in the work. [2] WebAssembly Community Group started in April 2015 [3].

WebAssembly allows to run code written in multiple languages on the web at almost native speed. It also allows running client apps, that previously could not have done so, on the web. This can have extensive implications for the web platform. [2]

Web platform can be divided to two parts: virtual machine (VM) that runs the code and the set of APIs that can be called to control web browser functionality. Before, only JavaScript has been able to load to VM. It works fine for smaller scale applications, but can cause performance problems, when trying to use more demanding use cases that need native performance. For example, 3D games or virtual/augmented reality (VR/AR) are very cumbersome. Also, large JavaScript applications can be expensive to download, parse and compile and other resource-constrained platforms can amplify these bottlenecks in performance. [2]

WebAssembly is not intended as a replacement for JavaScript. It is designed to complement and work alongside with it. This way, it allows web developers to take advantage of strong points in both languages. JavaScript has a huge ecosystem that provides powerful frameworks, libraries, and other tools. It is dynamically typed and requires no compiling step. WebAssembly on the other hand has a compact binary format. It provides compilation target for languages such as C++ and Rust, so that they can be run on the web. [2]

WebAssembly has limitations. These include the absence of Garbage Collection (GC), the inability to communicate directly with the Document Object Model (DOM), and the lack of support for older browsers. It is a work-in-progress, so these challenges could be answered in the future. [4]

2.2 ASP.NET Core

ASP.NET Core is development framework for the server-side web applications. It allows more efficient way to build web applications. ASP.NET Core is a part of the ASP.NET 5 platform, it runs on top of it.This enables applications to be run on a wide variety of platforms. For example, Linux and Mac OS X are on supported platforms. Microsoft introduced it in early 2002. [5]

ASP.NET has three programming models:

- ASP.NET Web Forms
- ASP.NET Web Pages
- ASP.NET MVC. [5]

ASP.NET Web Forms was the only programming model available to programmers to develop web applications in ASP.NET when it was first introduced. The ASP.NET Web Forms model abstracted the web, so that it can maintain the state even though the web is inherently stateless. It supports the event-driven programming model at the server-side, which has allowed desktop application developers to have a smoother transition into web application development. ASP.NET Web Forms is a file-based framework where users access the web page by means of accessing a file at the server. The server processes the request, convert all server-side components to HTML, and sends it back to the requesting client. Before the arrival of ASP.NET MVC, it was predominant programming model. It is still used to maintain the production applications which were written using this model. [5]

ASP.NET Web Pages is primarily used at small web applications where the data-processing logic is written directly on the web page [5]. It provides a quick way to combine server code with HTML markup for creating dynamic web content [6].

ASP.NET MVC is the implementation of the MVC pattern in ASP.NET. ASP.NET Web Forms disadvantages, such as limited control over the generation of HTML, are resolved in ASP.NET MVC. [5]

Model-View-Controller (MVC) pattern is a software architectural pattern which helps in defining the responsibility for each of the components and how they fit together in achieving the overall goal. It is primarily used in building user interfaces, but it can be used in many areas including developing desktop applications and web applications. [5]

The MVC pattern has three components: Model, view and controller. Model represents your domain data. This component can talk to your database, but the model only represents your domain data. View component is responsible for what is presented to the user. Usually it contains HTML and CSS files and may also include the layout information determining the look of web application. The controller is responsible for interacting with different components. It receives the request, talks to the model, and sends the appropriate view to the user. Separated responsibilities bring flexibility to the web application development, allowing each area to be managed independently. [5]

These three major components in ASP.NET MVC allows separated responsibilities, for example UI developer can work on the View while backend developers can work on the Model to build a data domain for the application. This allows the work to be done in parallel. Separated and compartmentalized components also allow easier unit test case creation. [5]

2.3 Blazor

Blazor is a framework, that allows building web applications using C# programming language. It also allows running any standard .NET library in the browser. Blazor has the same approach to building applications that run in the browser as the ASP.NET MVC. Blazor uses razor files (Blazor is a combination of words Browser and Razor). These razor files execute inside the browser to dynamically build a web page. [7]

Blazor provides many benefits over other client-side frameworks. It allows full stack .NET development experience and removes the need of creating separate model classes for client and server. Other benefits are existing rich .NET APIs and tools and modern and feature-rich language. Blazor is open source framework and has a great community support. [1]

Blazor applications are build on components. These are .NET classes built into .NET assemblies. Component is an element of UI. These can be pages, dialogs or forms. With components it is possible to define UI rendering logic or handle user events. They can also be nested and reused. Distribution is possible as Razor class libraries or NuGet packages. [8]

Components in Blazor are called Razor components. Razor is a syntax, which allows combining of HTML markup with C# code. With it, it is possible to have HTML markup and C# in the same file. Components are used respectively for creating client-side UI logic and composition. [8]

Blazor is divided between three types: server, client and hosted. Server-side Blazor will use server resources, while the client side Blazor runs entirely in the browser. In Client Blazor pages reside on the server but are handled on client-side. This works for a web sites that provide simple services like calculators or web sites which are used only for presentation. In Blazor hosted, logic will run on the browser. This allows saving server resources. It is divided between client Blazor project and an API project, which are connected so that they can be handled in the same project. [9]

2.4 Azure

Microsoft Azure is brand name for Microsoft’s cloud-computing services. It covers a wide range of services that are constantly growing and often form the elementary elements of cloud computing. [10] It was first announced with the code name Red Dog in October 2008 and was officially released in February 2010 with the name Windows Azure. First offered services were web roles and SQL databases. Microsoft rebranded its cloud platform from Windows Azure to Microsoft Azure in March 2014. [11]

The general availability of Azure has added many services to its platform. Microsoft has expanded its data centers to the continents across the globe. Currently the services that Azure supports include, but are not limited to, the following: mobile services, web services, compute services, storage services, messaging, network services, media services, machine learning and internet of things. [11]

Azure has different types of cloud services. The most common ones are the following: Infrastructure as a service (IaaS), Platform as a Service (PaaS) and Software as a service (SaaS). Infrastructure as a service allows running Virtual Machines (VM) on the cloud. In this model the cloud service provider will take care of the underlying infrastructure, such as hardware, network, storage, and virtualization platform. User will be responsible for managing and maintaining VMs. Platform as a service allows running applications on Azure. Underlying infrastructure is managed by the cloud service provider. This allows developer to focus on developing and running applications. Software as a service allows using software on the cloud. The cloud service provider will handle the underlying infrastructure and the application itself. Office 365, for example, provides a variety of solutions ready to be used, such as email services, VOIP services, and so on, without implementing Exchange or Skype. [11]

3 Project Implementation

The project build during final year project was an application for making notes and calculations needed in brewing beer. The idea came from personal experience in home-brewing and a need for simple note taking tool. The project is available here: https://brewingcalculator20200412053241.azurewebsites.net/.

Simplified agile methods were followed during implementation. These include project iteration planning and concepts such as user stories and minimum viable product.

3.1 Agile Software Development

Agile software development is a concept that includes set of practices and frameworks. These are based on the values declared in the Manifesto for Agile Software Development. Agile is a mindset, it provides guidance on how to create and respond to change. It also gives tools for dealing with uncertainty. [12]

3.1.1 Iteration

An iteration is a set duration of which development takes place. This may vary from project to project, thought it is usually ranging from one week to four weeks. Iteration time is usually fixed, for the duration of a given project. A key feature of Agile approaches is that projects consists sequence of iterations. Before the start of development there is a brief planning phase and after a brief closure phase. The fixed length of iterations allows teams to estimate the amount of work remaining during iterations. [13]

3.1.2 User Stories

User stories outline pieces of functionality from the point of view of a user. They can be used to reflect what a particular group of users needs. Example of user stories include:

“As a ‘role’ I can ’objective’ so that ‘reason’”

and

“As a ‘particular user’ I want to ‘be able doing something’ so that ‘I get some value’.[14]

User stories helps to define product with clarity. They allow to articulate the functionality of the product without technicalities and implementation details. This can encourage participation by non-technical members. Therefore, user stories can help to achieve team wide certainty on what to build, why and to whom. [14]

3.1.3 Minimum Viable Product

A minimum viable product (MVP) is a concept. It was introduced in Lean Startup. It stresses the importance of learning, in a development of new projects. The basis of it is producing an actual product, which can be offered to the customers. This is used to observe their behavior with the product. MVP may, for example, just be a landing page for web site. Seeing what people actually do with the product is more useful than asking them about it. [15]

The benefit of MVP is that it allows gaining understanding about interest in the product without developing it fully. More shortly you know the appeal of customers, the less expense and effort will be spent on the product if it won’t succeed in the market. [15]

3.2 Project Background

Before moving to the planning phase, background research needs to be done. To see how why note taking is important, the brewing process must be understood. The brewing process can be divided to six different parts mashing, lautering, boiling, hopping, cooling and fermentation [16]. This is a simplified division, but enough for the purpose of this thesis.

Mashing is the first step in beer brewing process, in which grist (milled malt) is transferred to the mash tun. This is achieved by combining the grist and water and heating the combination to specific temperature. Mashing makes the natural enzymes in the malt to breakdown and ultimately convert them to sugars. These sugars will eventually become alcohol. [16]

After the mashing, the wort needs to be separated from spent grain. This process is called lautering. [16] The goal of this phase is to get the most sugars form spent grain into the wort [17].

After lautering, the wort is transferred to a boiling kettle for sterilizing and hopping. During the boiling hops are added to the wort. The qualities of aroma, taste and bitterness that hops give to beer depend on which point they are added. Early added hops yield to higher bitterness and hops added late in a boil give more flavor and aroma. [16]

The wort is then chilled to a desired temperature and transferred to a fermentation container for fermentation process. To start the fermentation, yeast needs to be added to the wort. Yeast converses the sugars to alcohol and carbon dioxide. As the converting sugars to alcohol generates heat and higher temperatures may result yeast to yield more esters or other fragrances to beer, fermentation process is monitored closely by brewers. [17]

There is a lot of moving parts in all these processes, which all can affect the taste of beer. Therefore, consistency is needed. The consistent brewer has more great than bad patches of beer. To achieve that, good organization and record keeping is a must. [18] In a light of this, easy to use note keeping tool will benefit a brewer greatly.

Today most homebrewers use recipe formulation software to calculate their original gravity (OG), bitterness and beer color. However, some additional equations that may help homebrewer exists. These include calculations of strike water, water absorption, first wort volume, amount of wort to collect, volume of sparge water required and estimating OG from pre-boil wort concentration and volume. [19]

3.3 Planning and Design

The project iteration duration was chosen to be 8 days, having two day of design and planning phase and one day for retrospective (See Figure 1).

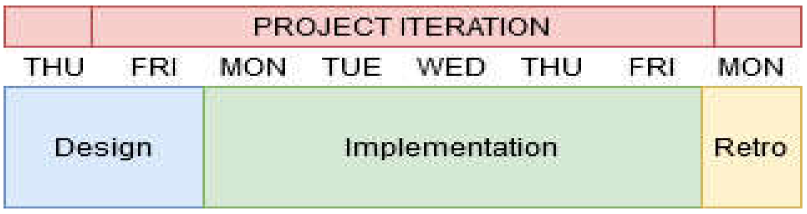


Figure 1. Project timeline.

During the design and planning phase following user stories were created for this product (see Figure 2):

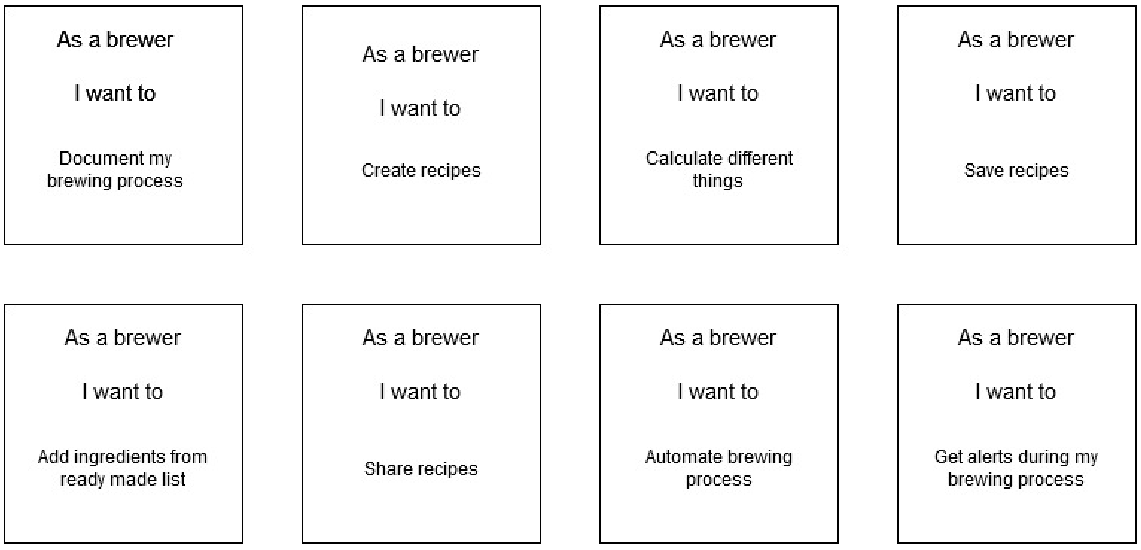


Figure 2. User stories created during design process.

All of the user stories have the value of improving the quality of beer. After going through each user story, five was decided to implement during this iteration: Document my brewing process, calculations, create recipes, save recipes and share recipes. Other stories were considered to be too big for this iteration.

Of five chosen user stories, two stories were chosen to create an MVP for the project. MVP was decided to consist of documenting the brewing process and calculators.

For documenting the brewing process following principles were created:

- Data should be consistent
- The report should be as readable as possible.

To record the brewing process, the ready-made questions were designed. When user records something to the form, it automatically fills the ready-made report. This allows consistency from brew to brew. This auto-generated report is also the base for recipe and there should be option to save and share it.

A sample UX document were created for documenting the brewing process (see Figure 3).

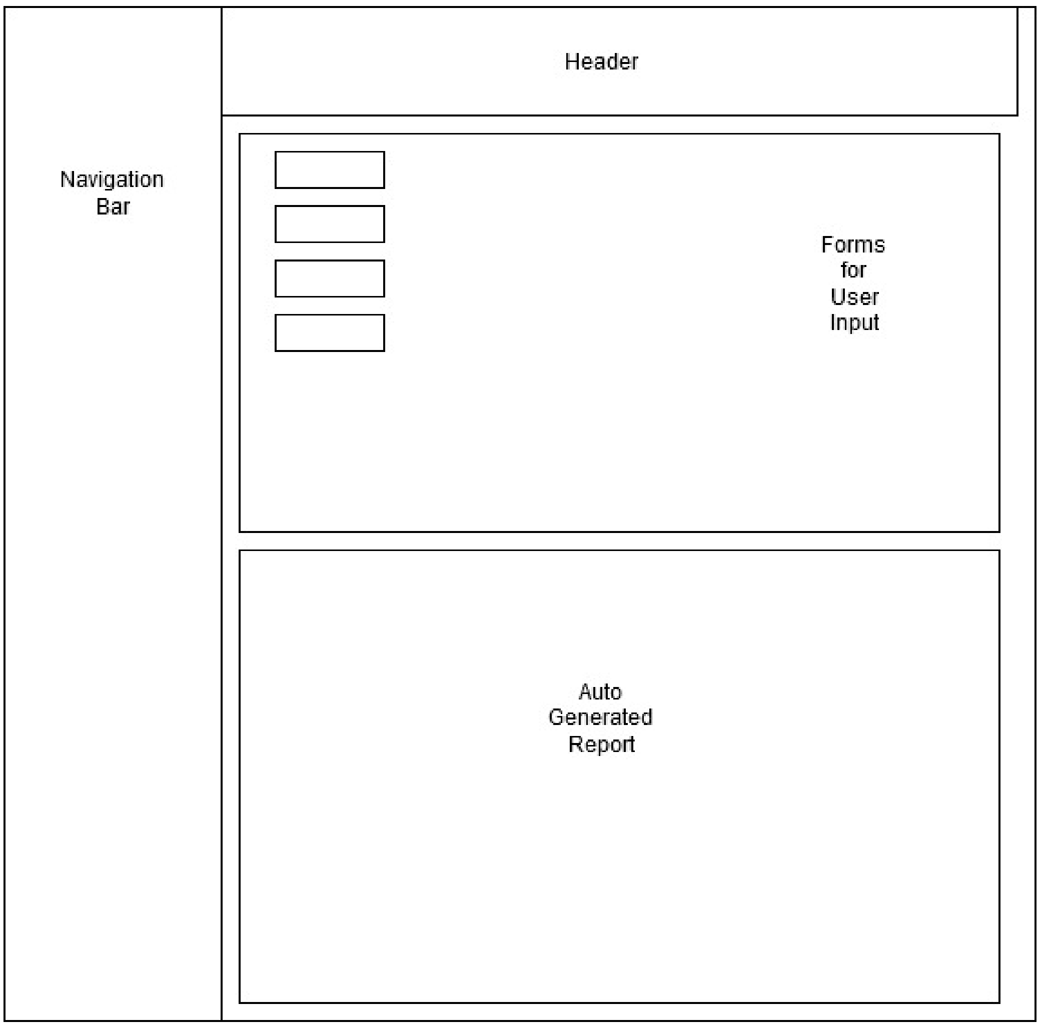


Figure 3. UX design for recording the brewing process.

Following equations were chosen to be the base for calculators:

- Strike Water:

$$V(strike\ water) = W(grain) * \left(MT\frac{volume}{weight}\right)$$

- Water Absorption:

$$AR(volume\ per\ weight) = \frac{V(absorbed)}{W(grain)}$$

- First Wort Volume:

$$V(absorbed) = V(strike\ water) - (V(first\ wort) + V(other))$$

- Amount of Wort to Collect:

$$V(pre\ boil\ wort) = W(grain) * \left(FS\frac{volume}{weight}\right)$$

- Volume of Sparge Water Required:

$$V(sparge\ water) = V(pre\ boil) - (V(first\ wort) + V(other)$$

- Estimating OG from Pre-Boil Wort Concentration and Volume:

$$C(post\ boil) = \frac{C(pre\ boil) * V(pre\ boil)}{V(post\ boil)}$$

Where:

- V(strike water) is the volume of strike water in liters
- W(grain) is the weight of the grain in kilograms
- MT is the mash thickness in liters per kilogram

- AR(volume per weight) is the volume of water absorbed per weight of grain in liters
- V(absorbed) is the volume of water absorbed by the grain
- V(first wort) is the volume of the first wort
- V(other) is the volume of water that end anywhere other than absorbed in grain
- V(pre boil wort) is the volume of wort to collect
- FS is the ratio of wort volume to grain weight
- V(sparge water) is the amount of sparge water needed
- C(post boil) is the post-boil concentration in °Plato
- C(pre boil) is the pre-boil concentration in °Plato. [19]

After choosing the equations for the calculators, following UX document was created (see Figure 4).

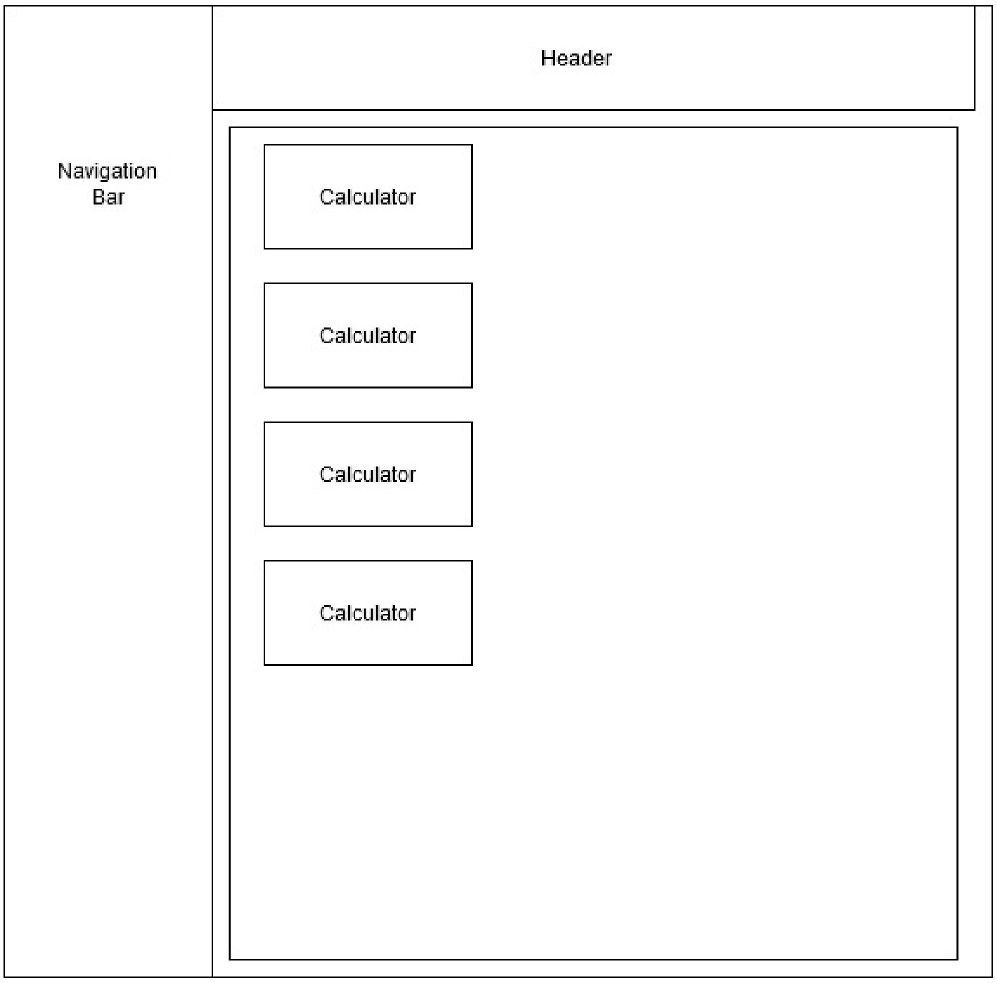


Figure 4. UX design for calculator page.

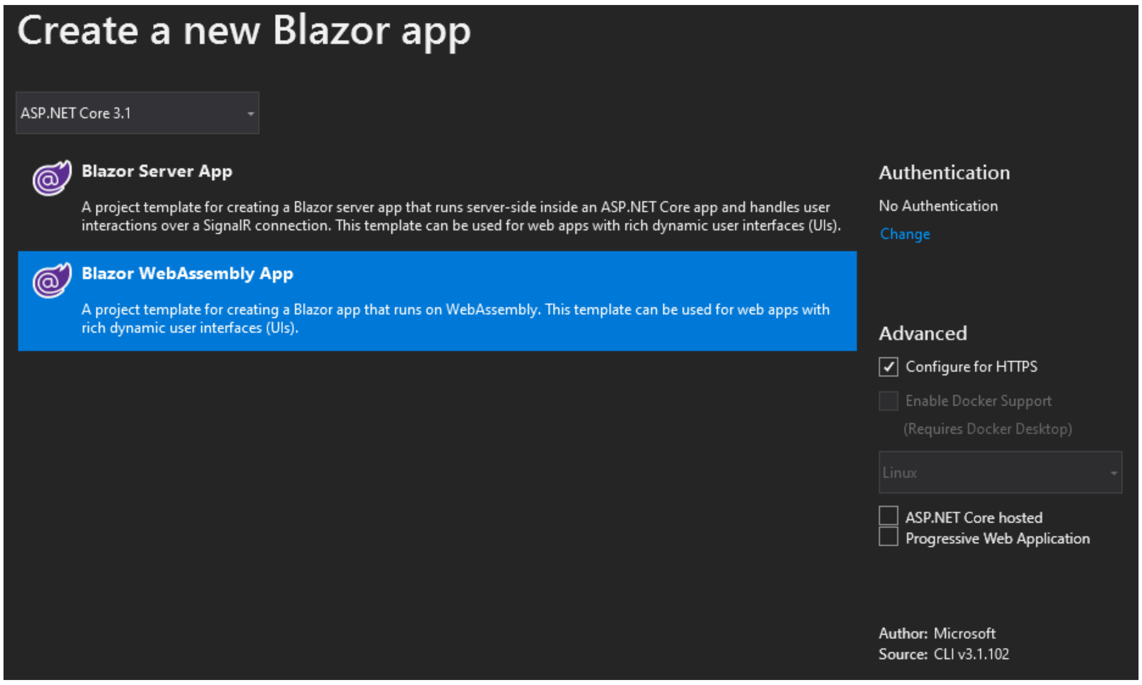
3.4 Building

In this section Blazor application specific things are introduced. As the other build is mainly basic HTML and C#, it will not be discussed in this section.

3.4.1 Installing the Project Environment

An Integrated Development Environment (IDE), enables programmers to bolster the different aspects of coding. It Increases the productivity by combining common activities of writing software into a single application. [20]

There are two alternations for IDEs in Blazor programming: Microsoft Visual Studio or Visual Studio Code. For this project Visual Studio 2019 was chosen, mainly because it has vast tools for releasing the applications. Besides installing an IDE, a workloads for ASP.NET and web development need to be installed. After installation, new project can be created. [8]



Create a new Blazor app

ASP.NET Core 3.1

Blazor Server App

A project template for creating a Blazor server app that runs server-side inside an ASP.NET Core app and handles user interactions over a SignalR connection. This template can be used for web apps with rich dynamic user interfaces (UIs).

Blazor WebAssembly App

A project template for creating a Blazor app that runs on WebAssembly. This template can be used for web apps with rich dynamic user interfaces (UIs).

Authentication

No Authentication

Change

Advanced

- [x] Configure for HTTPS
- [ ] Enable Docker Support (Requires Docker Desktop)

Linux

- [ ] ASP.NET Core hosted
- [ ] Progressive Web Application

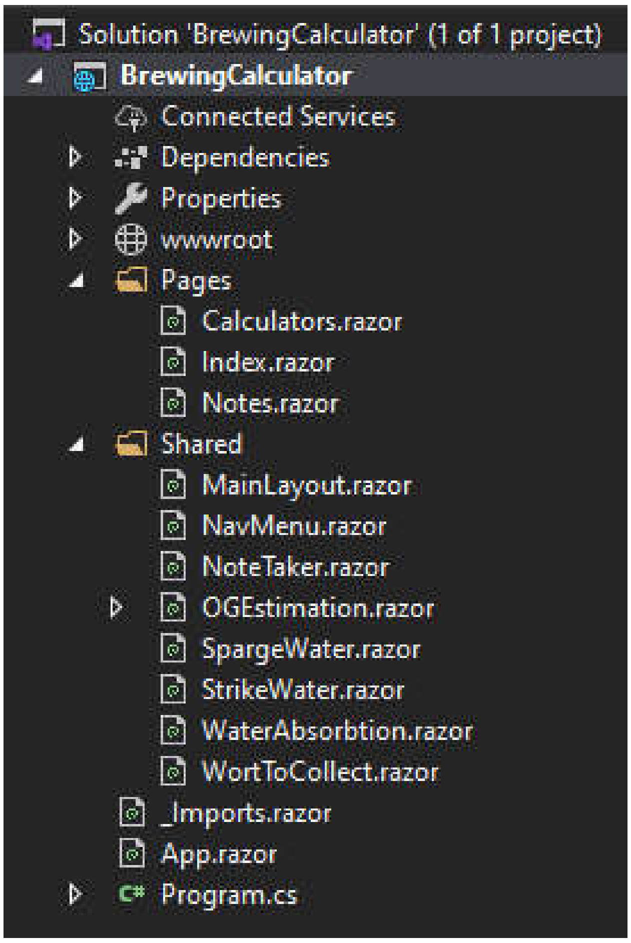
Author: Microsoft
Source: CLI v3.1.102

Figure 5. Creating new project in Visual Studio 2019.

When creating a new Blazor project, there is two options: Blazor Server App and Blazor WebAssembly App (see figure 1). For this project Blazor WebAssembly was selected as the idea was to create simple single page application. After choosing, the Visual Studio will create some base files for the project and the building can be started. [8]

3.4.2 Project Structure

When creating a new Blazor WebAssembly application project, some of the application structure is created for you (see Figure 6). This includes folders for pages and a shared folder which contains shareable components. [8]

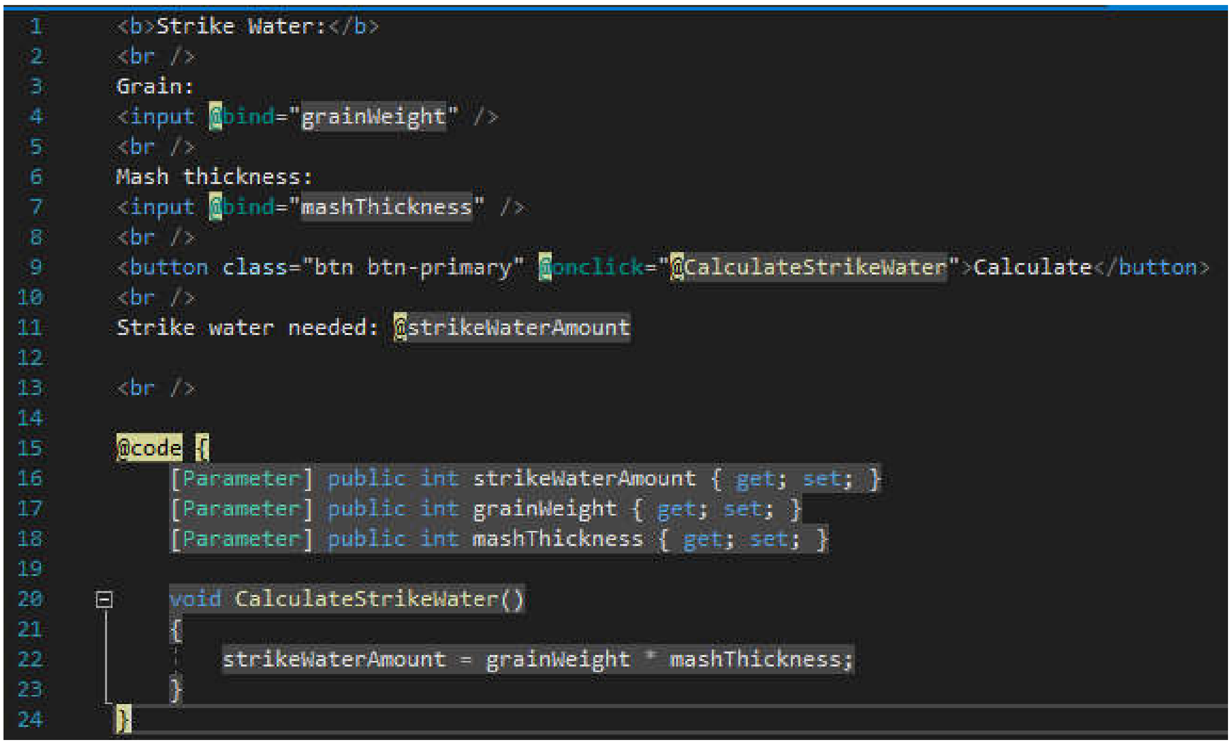


```
Solution 'BrewingCalculator' (1 of 1 project)
  BrewingCalculator
      Connected Services
      Dependencies
      Properties
      wwwroot
      Pages
          Calculators.razor
          Index.razor
          Notes.razor
      Shared
          MainLayout.razor
          NavMenu.razor
          NoteTaker.razor
          OGEstimation.razor
          SpargeWater.razor
          StrikeWater.razor
          WaterAbsorbtion.razor
          WortToCollect.razor
      _Imports.razor
      App.razor
      Program.cs
```

Figure 6. Project structure of ready project.

As we learned before Blazor applications are built by using components. A component is a self-contained part of user interface. In Figure 6, all files which have “.razor” on the end of filename are components.

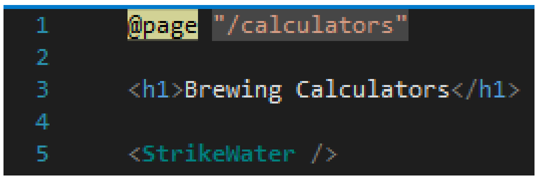
HTML is used to define the UI of component and dynamic rendering logic (loops, for example) are added using an embedded C# syntax called Razor. When the application is compiled these are converted into a component class. Component class members are defined in an “@code” block. There can be more than one “@code” block in a component. [21] Example of component can be seen on Figure 7.



```
<b>Strike Water:</b>
<br />
Grain:
<input @bind="grainWeight" />
<br />
Mash thickness:
<input @bind="mashThickness" />
<br />
<button class="btn btn-primary" @onclick="@CalculateStrikeWater">Calculate</button>
<br />
Strike water needed: @strikeWaterAmount

<br />

@code {
    [Parameter] public int strikeWaterAmount { get; set; }
    [Parameter] public int grainWeight { get; set; }
    [Parameter] public int mashThickness { get; set; }

    void CalculateStrikeWater()
    {
        strikeWaterAmount = grainWeight * mashThickness;
    }
}
```

Figure 7. Component for calculating strike water.

Components can be used by in other components by using HTML element syntax. This looks like an HTML tag where the name is the component type. [21] See Figure 8 for example.



```
@page "/calculators"

<h1>Brewing Calculators</h1>

<StrikeWater />
```

Figure 8. Calculators component which contains “StrikeWater” component.

In Figure 7, “@bind” attribute can be seen. This is what allows data binding in components [21]. In this project, binding is used to get user input to do for example calculations. In Figure 7, user is asked to input grain weight and mash thickness and then component uses them to calculate the needed strike water.

Last thing to look in Figure 7, is the “[Parameter]” declaration. Components can have component parameters. These are defined using public properties on the component class using [Parameter] attribute. [21] With parameters it is possible to set the value of one control with the properties on another control [22].

All of the project components were created on similar way to create the application. Testing was conducted all the time during the development process.

3.4.3 Publishing

Publishing the application in Azure can be achieved in simple steps. For publishing the application, free Azure account is required. See Figures 9, 10, 11 for publishing process.

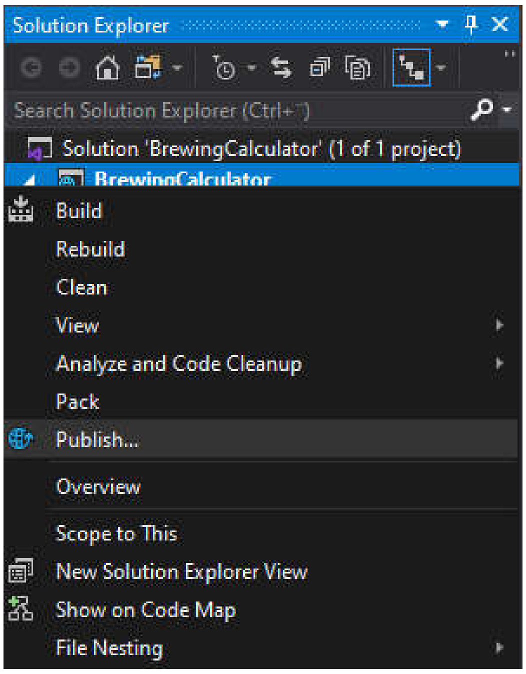
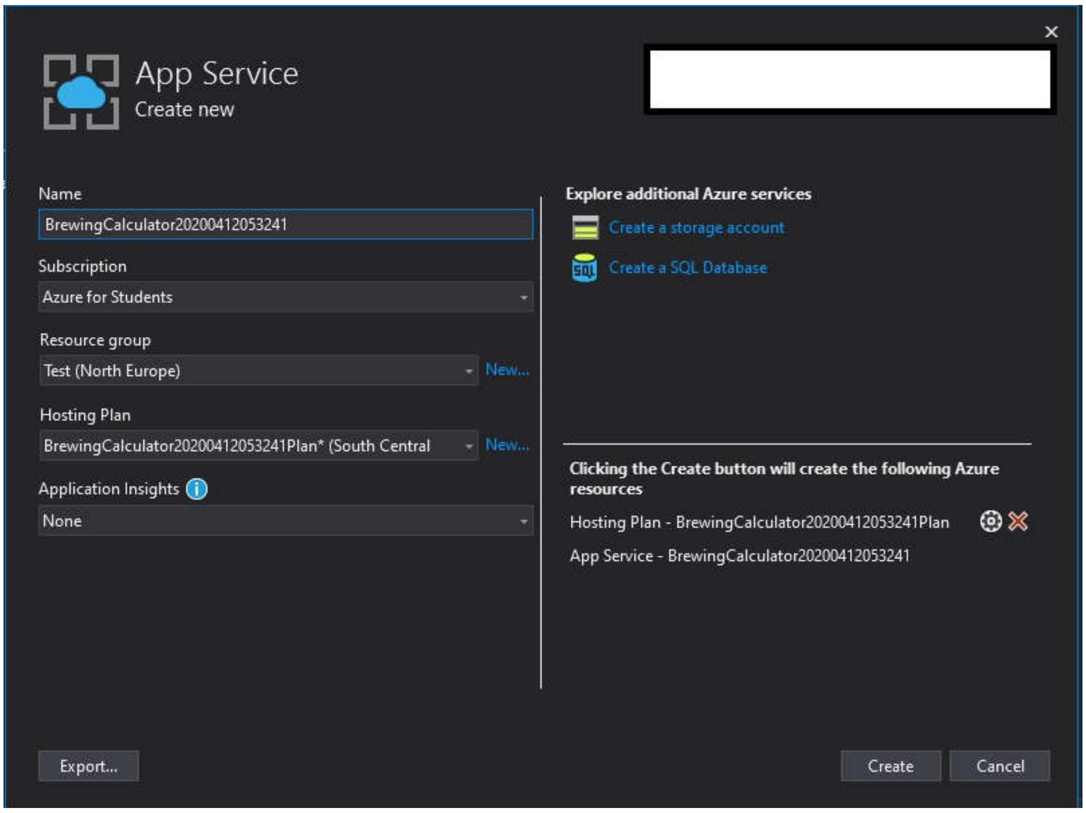


Figure 9. Publishing can be done straight from Visual Studio.

While in visual studio, the publishing process can be started by right clicking the solution’s name in Solution Explorer and choosing “Publish” (Figure 9) [23].

After that the App Service window will open (Figure 10).



App Service
Create new

Name
BrewingCalculator20200412053241

Subscription
Azure for Students

Resource group
Test (North Europe) New...

Hosting Plan
BrewingCalculator20200412053241Plan* (South Central New...

Application Insights
None

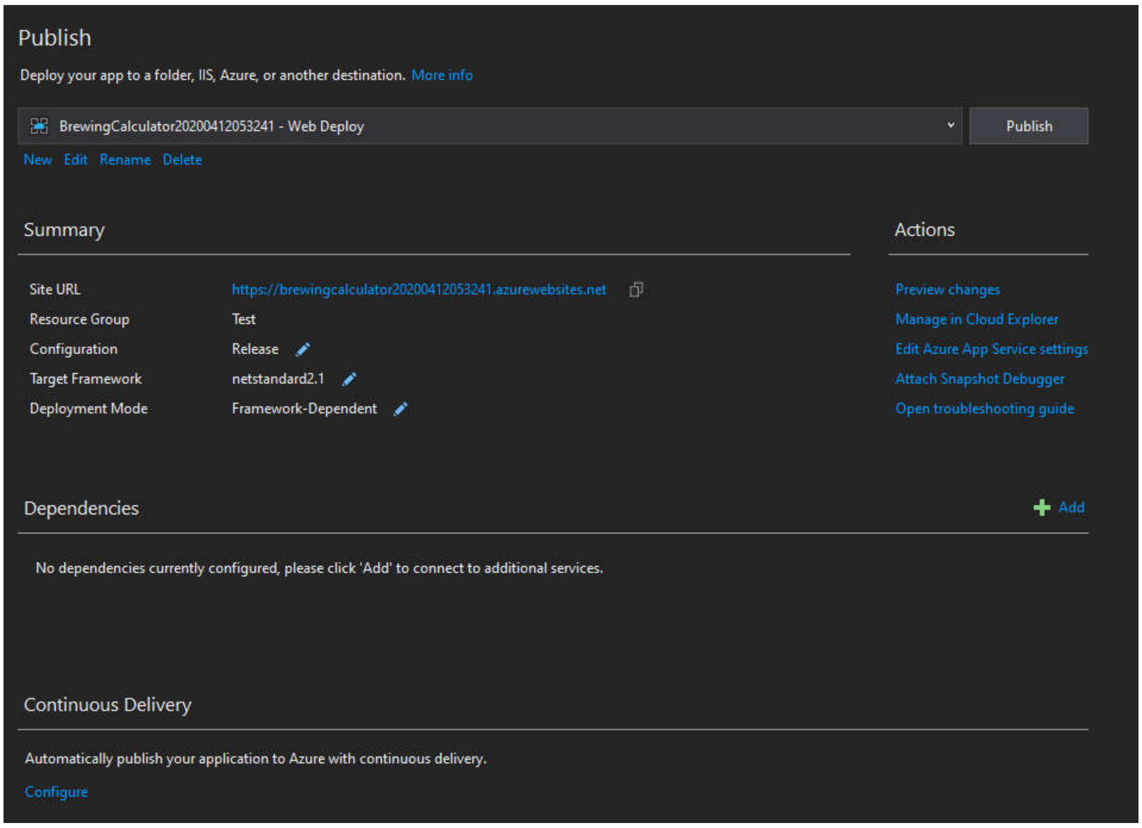
Explore additional Azure services
Create a storage account
Create a SQL Database

Clicking the Create button will create the following Azure resources
Hosting Plan - BrewingCalculator20200412053241Plan
App Service - BrewingCalculator20200412053241

Figure 10. Creating new App Service.

From this window a name can be given for the app service. Here the choices for the wanted Azure subscription, resource group and hosting plan can be made. [24] For this project, Azure for Students were used, but it is possible to anyone to create free subscription to Azure. After clicking create Visual Studio deploys the app to your Azure App Service and it opens to a web browser [24].

After pressing create on App Service window, the “Publish” window will open (Figure 11). In this window you can choose the



Publish

Deploy your app to a folder, IIS, Azure, or another destination. More info

BrewingCalculator20200412053241 - Web Deploy

Summary

Site URL	https://brewingcalculator20200412053241.azurewebsites.net
Resource Group	Test
Configuration	Release
Target Framework	netstandard2.1
Deployment Mode	Framework-Dependent

Dependencies

No dependencies currently configured, please click 'Add' to connect to additional services.

Continuous Delivery

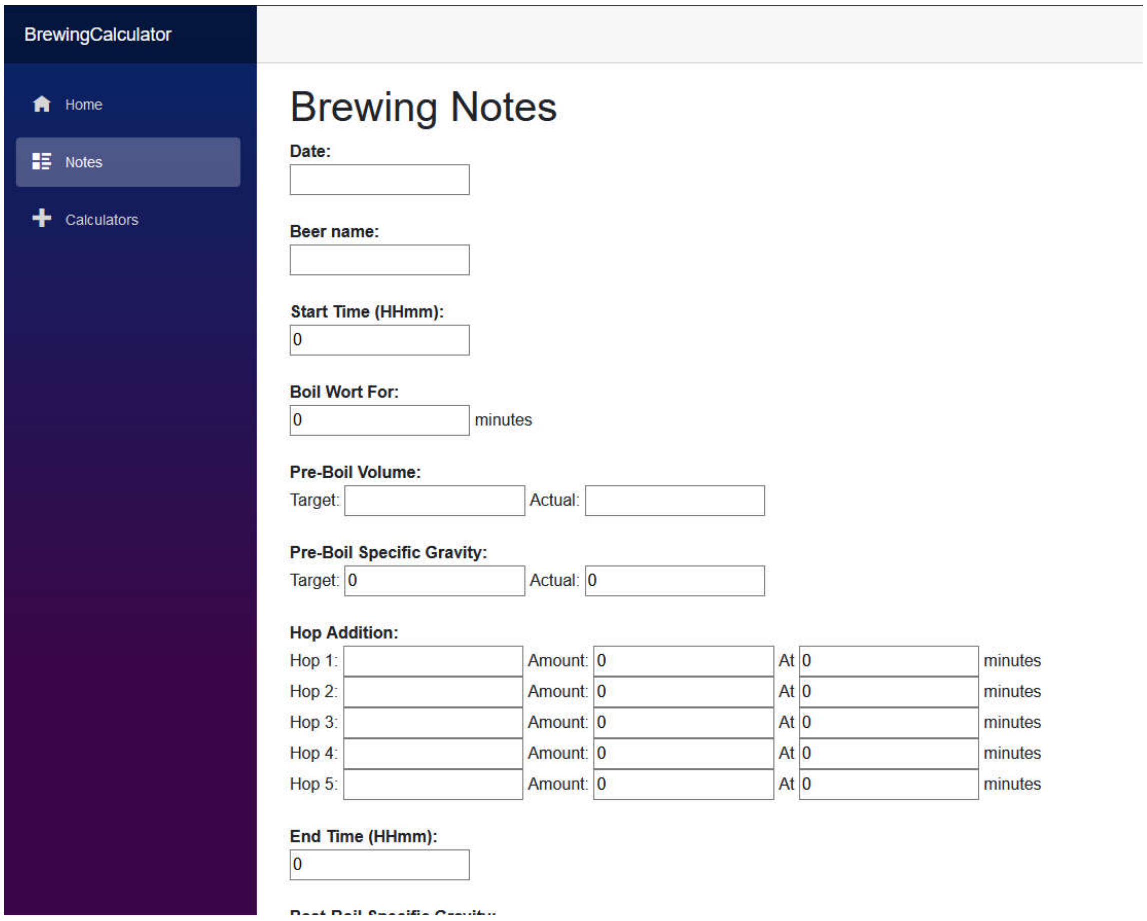
Automatically publish your application to Azure with continuous delivery.

Figure 11. Making last adjustments before publishing.

3.5 Project Retrospective

The project iteration goal was to create tools for note taking, calculations and creating recipes. As seen from Figures 12, 13 and 14, only note taking and calculations were made. The given time for the project was not enough, as there were some challenges along the way. These are discussed in the next section. For next iteration, better planning would be advised.

However, during the iteration, the MVP was created. This would be enough to present the project for customers, if this was a customer project.



Brewing Notes

Date:

Beer name:

Start Time (HHmm): 0

Boil Wort For: 0 minutes

Pre-Boil Volume: Target: Actual:

Pre-Boil Specific Gravity: Target: 0 Actual: 0

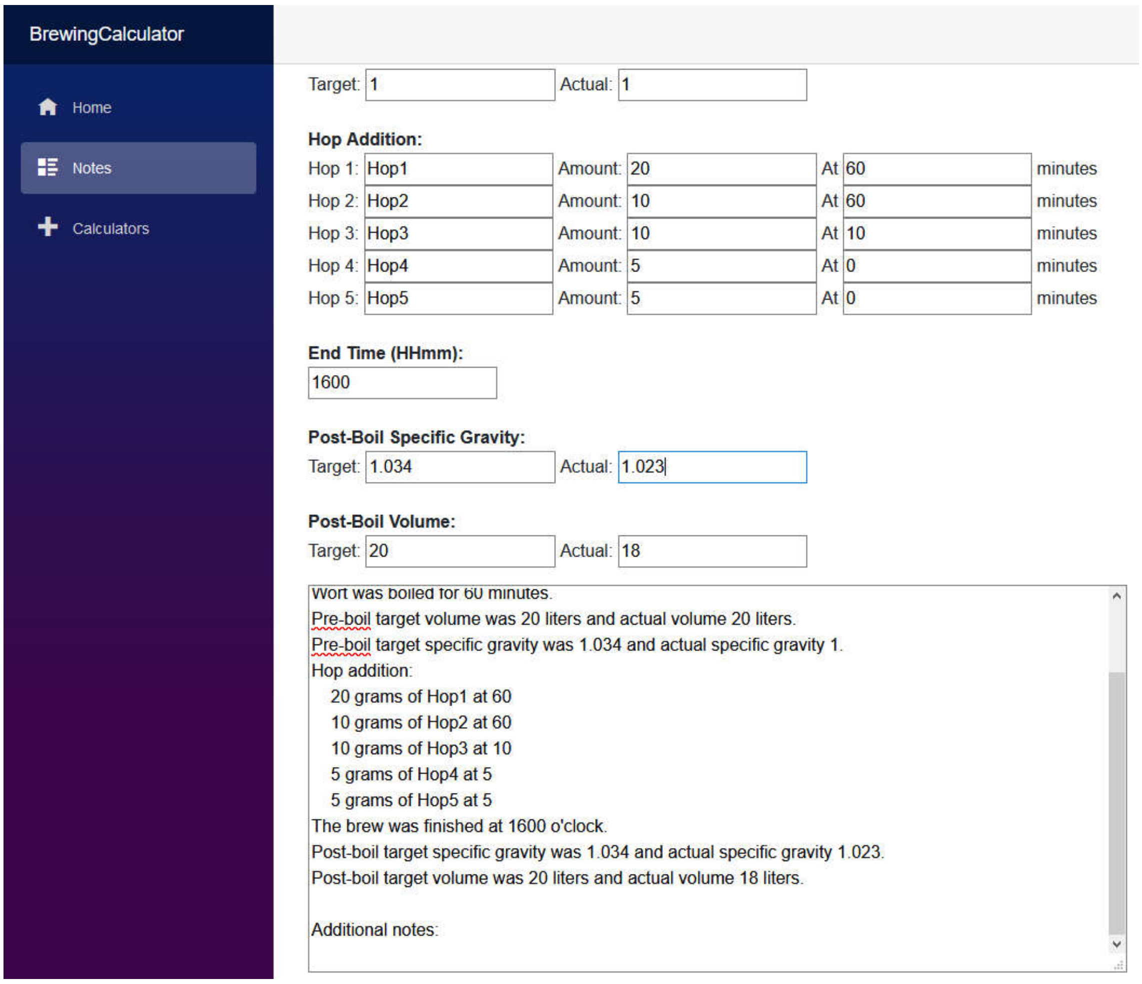
Hop Addition:

Hop 1:		Amount:	0	At	0	minutes
Hop 2:		Amount:	0	At	0	minutes
Hop 3:		Amount:	0	At	0	minutes
Hop 4:		Amount:	0	At	0	minutes
Hop 5:		Amount:	0	At	0	minutes

End Time (HHmm): 0

Figure 12. Screenshot of Brewing Notes page of application.

In Figure 12 the note taking tools are seen. Note taking happens by filling out the specific forms. Data entered to these forms will be automatically added to the “story box” under the forms (See Figure 13). In this box it is also possible to write additional notes of the brew.



Target:	1	Actual:	1			

Hop Addition:

Hop 1:	Hop1	Amount:	20	At	60	minutes
Hop 2:	Hop2	Amount:	10	At	60	minutes
Hop 3:	Hop3	Amount:	10	At	10	minutes
Hop 4:	Hop4	Amount:	5	At	0	minutes
Hop 5:	Hop5	Amount:	5	At	0	minutes

End Time (HHmm):
1600

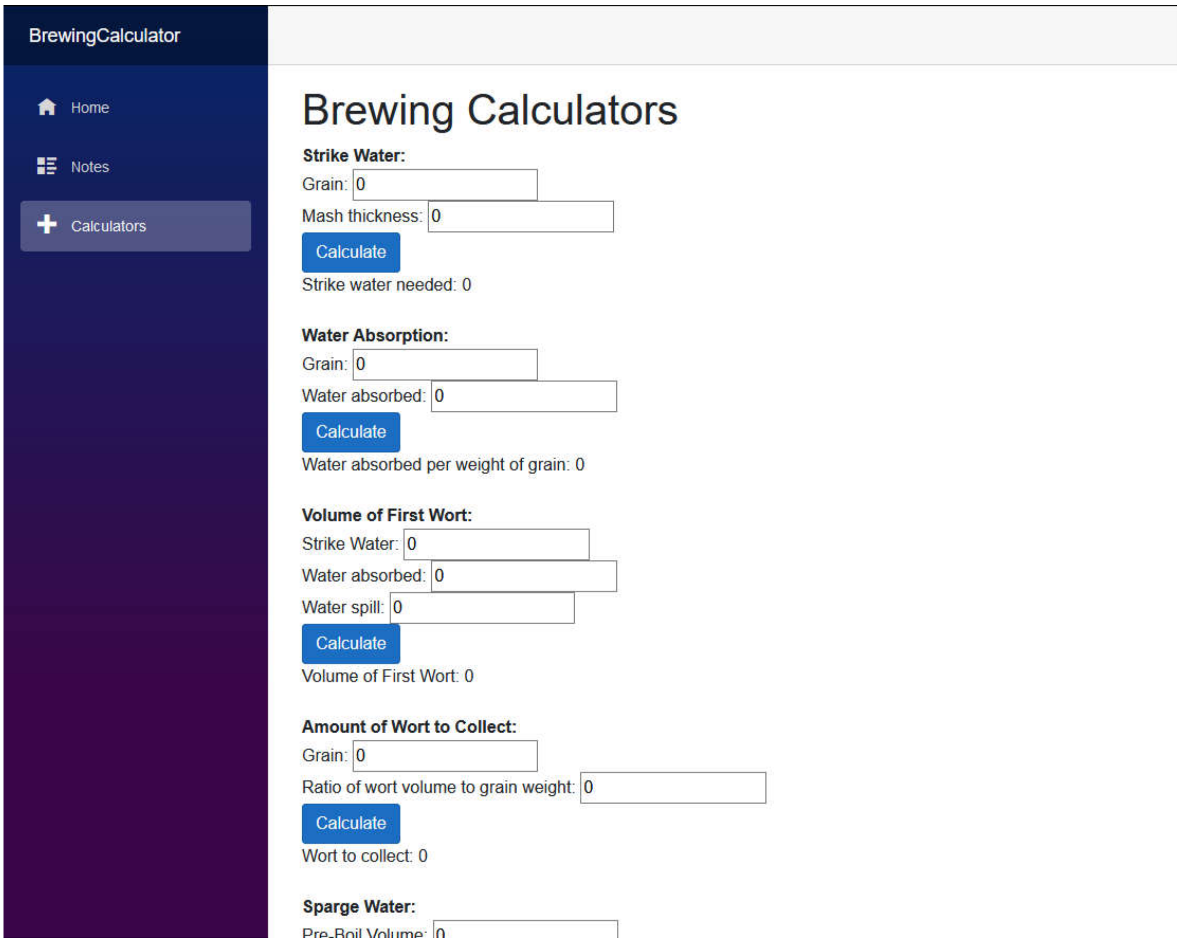
Post-Boil Specific Gravity:
Target: 1.034 Actual: 1.023

Post-Boil Volume:
Target: 20 Actual: 18

```
Wort was boiled for 60 minutes.
Pre-boil target volume was 20 liters and actual volume 20 liters.
Pre-boil target specific gravity was 1.034 and actual specific gravity 1.
Hop addition:
    20 grams of Hop1 at 60
    10 grams of Hop2 at 60
    10 grams of Hop3 at 10
    5 grams of Hop4 at 5
    5 grams of Hop5 at 5
The brew was finished at 1600 o'clock.
Post-boil target specific gravity was 1.034 and actual specific gravity 1.023.
Post-boil target volume was 20 liters and actual volume 18 liters.

Additional notes:
```

Figure 13. Screenshot of Brewing Notes page of application.

In the calculators page of the application, different brewing calculators are available. User can enter the values and press “calculate” button to get the results (see Figure 14).


BrewingCalculator

Home

Notes

Calculators

Brewing Calculators

Strike Water:

Grain: 0

Mash thickness: 0

Calculate

Strike water needed: 0

Water Absorption:

Grain: 0

Water absorbed: 0

Calculate

Water absorbed per weight of grain: 0

Volume of First Wort:

Strike Water: 0

Water absorbed: 0

Water spill: 0

Calculate

Volume of First Wort: 0

Amount of Wort to Collect:

Grain: 0

Ratio of wort volume to grain weight: 0

Calculate

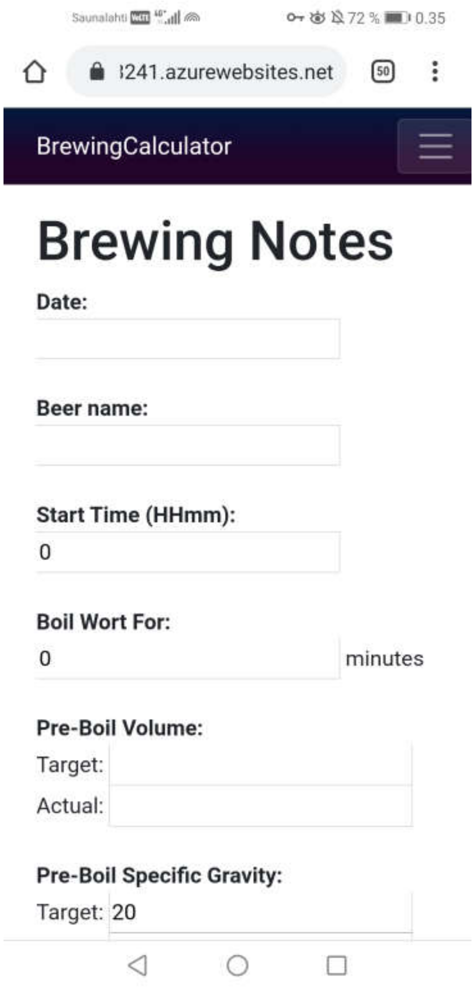
Wort to collect: 0

Sparge Water:

Pre-Boil Volume: 0

Figure 14. Screenshot of Calculators page of application.

Application also scales down automatically for phones, as seen on Figure 15.


Saunalahti 72 % 0.35

241.azurewebsites.net

BrewingCalculator

Brewing Notes

Date:

Beer name:

Start Time (HHmm): 0

Boil Wort For: 0 minutes

Pre-Boil Volume:

Target:

Actual:

Pre-Boil Specific Gravity:

Target: 20

Figure 15. Screenshot taken from phone.

Overall the functionality works as intended. This version of the project can be used to give brief presentation of the functionality, which is the intent of MVP.

4 Discussion

The goal of this final year project was achieved. A responsive web application was created and published, without using a single line of JavaScript. However, the application created was much simpler than originally planned. This partly due the fact that I did not have previous experience in web development and misjudged the amount of work that it would cause me to learn things. Just knowing C# was not enough in this case. I did not want to extend the deadline for my project and rather take it as a learning experience.

The application itself could be upgraded many ways. First the code itself could be cleaner and better organized. The app uses the basic layout at the moment, but it could be modified for a better functioning look.

Next step would be creating user database and allowing users to create custom views for their note taking and calculating tools. The note taking tool could be editable in a way that you could choose the ready-made forms and create a custom report for user’s own preferences. The calculator page could have more calculators and an option to choose which one’s are visible for the user. That would improve the user experience as not everyone will use everything you have to offer.

The next step would be changing this application to a full recipe formulation application. This would allow user to create a recipe by choosing the malts, hops and yeasts and the application would automatically calculate the times and amounts you need to know. Social features could be added, and recipes could be made shareable with other users.

These upgrades would of course mean that the application would turn from Blazor WebAssembly to Blazor Server project as we would need better access to the databases and backend. However, this all could be done with Blazor.

Publishing the software was made really easy. Just by creating an account and few guided steps and your application was online. Azure itself is a huge monolith of services and it would require another thesis to just go through all the options within the app services alone. You get a good dashboard for managing and diagnosing your application. Of course, if you would like to really release your application, the free version will run out of resources quickly and you will need to pay and scale up the service.

I learned a lot during this final year project. I had to deep dive into the field of web development as I only had brief basic understanding about it. I learned a lot about cloud services and how you can use cloud to publish your application. It was also a good practice to use some agile methods while developing the application.

Finding information was sometimes problematic, as the Blazor WebAssembly is still in preview. There were not that many trusted references available, although Microsoft has a good documentation.

5 Conclusion

Developing a responsive web application without using JavaScript is now possible. Blazor is (by the time of writing this thesis) still in preview stage. However, during my brief familiarization process, I did not come across things that you could not do with Blazor. There is even a possibility to use JavaScript with Blazor, so there really are no limits to what is possible to achieve.

There are of course the performance issues that could cause issues to Blazor to really breakthrough, but those are minor issues. More pressing issue is that, will developers take this in their toolbox. JavaScript has been an industry standard for a long time and competing against it will be hard.

Only the time will tell. Whatever happens after the Blazor is officially released in May 2020, one thing is for sure: There is alternative way to do things in web development for a first time in many years.

References

1 Sharma, Ankit. Blazor Quick Start Guide [online]. October 2018. URL: https://learning.oreilly.com/library/view/blazor-quick-start/9781789344141. Accessed 11 April 2020.

2 MDN Web Docs. WebAssembly [online]. 2020. URL: https://developer.mozilla.org/en-US/docs/WebAssembly. Accessed 11 April 2020.

3 WebAssembly. Roadmap [online]. 2015. URL: https://webassembly.org/roadmap/. Accessed 11 April 2020.

4 Rourke, Mike. Learn WebAssemly [online]. September 2018. URL: https://learning.oreilly.com/library/view/learn-webassembly/9781788997379/?ar. Accessed 11 April 2020.

5 Ragupathi, Mugilan T.S. Learning ASP.NET Core MVC Programming [online] November 2016. URL: https://learning.oreilly.com/library/view/learning-aspnet-core/9781786463838/?ar. Accessed 11 April 2020.

6 Microsoft. Docs. ASP.NET Overview [online]. September 2019. URL: https://docs.microsoft.com/en-us/aspnet/overview. Accessed 11 April 2020.

7 Himschoot, Peter. Blazor Revealed: Building Web Applications in .NET [online]. February 2019. URL: https://learning.oreilly.com/library/view/blazor-revealed-building/9781484243435/?ar. Accessed 11 April 2020.

8 Microsoft. Docs. Introduction to ASP.NET Core Blazor [online]. March 2020. URL: https://docs.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-3.1. Accessed 11 April 2020.

9 Litvinavicius, Taurius. Exploring Blazor: Creating Hosted, Server-side, and Client-side Applications with C# [online]. November 2019. URL: https://learning.oreilly.com/library/view/exploring-blazor-creating/9781484254462/?ar. Accessed 11 April 2020.

10 Microsoft Azure. What is Azure? [online]. 2020. URL: https://azure.microsoft.com/en-us/overview/what-is-azure/. Accessed 11 April 2020.

11 Wali, Mohamed. Learn Microsoft Azure [online]. December 2018. URL: https://learning.oreilly.com/library/view/learn-microsoft-azure/9781789617580/?ar. Accessed 11 April 2020.

12 Agile Alliance. Agile 101. What is Agile? [online] 2020. URL: https://www.agilealliance.org/agile101/. Accessed 11 April 2020.

13 Agile Alliance. Glossary. Iteration [online] 2020. URL: https://www.agilealliance.org/glossary/iteration/. Accessed 11 April 2020.

14 Krasadakis, George. How (and why) to write great User Stories [online]. June 2018. URL: https://www.freecodecamp.org/news/how-and-why-to-write-great-user-stories-f5a110668246/. Accessed 11 April 2020.

15 Agile Alliance. Glossary. Minimum Viable Product (MVP) [online]. 2020. URL: https://www.agilealliance.org/glossary/mvp/. Accessed 11 April 2020.

16 The Beer Connoisseur. Beer 101: The Fundamental Steps of Brewing [online] June 2016. URL: https://beerconnoisseur.com/articles/beer-101-fundamental-steps-brewing?page=2. Accessed 11 April 2020.

17 More Beer. How to Lauter for the Highest Extract Efficiency [online]. July 2012. URL: https://www.morebeer.com/articles/lautering_for_highest_efficiency. Accessed 11 April 2020.

18 Palmer, John. How to Brew. 2017. Brewers Publications. Boulder, Colorado.

19 Colby, Chris. Beer & Wine Journal. Useful Brewing Calculations. January 2014. URL: https://beerandwinejournal.com/brew-calcs/. Accessed 11 April 2020.

20 Codecademy. What Is an IDE? [online]. 2020. URL: https://www.codecademy.com/articles/what-is-an-ide. Accessed 11 April 2020.

21 Microsoft. Docs. Create and use ASP.NET Core Razor components [online] March 2020. URL: https://docs.microsoft.com/en-us/aspnet/core/blazor/components?view=aspnetcore-3.1. Accessed 11 April 2020.

22 Washington, Michael. Blazor Binding, Events and Parameters [online]. September 2019. URL: http://blazorhelpwebsite.com/Blog/tabid/61/EntryId/4350/Blazor-Binding-Events-and-Parameters.aspx. Accessed 11 April 2020.

23 Microsoft Azure. Publish your ASP.NET app to Azure [online]. 2020. URL: https://tutorials.visualstudio.com/aspnet-azure/publish. Accessed 11 April 2020.

24 Microsoft. Visual Studio Docs. Publish a Web app to Azure App Service using Visual Studio [online]. January 2019. URL: https://docs.microsoft.com/en-us/visualstudio/deployment/quickstart-deploy-to-azure?view=vs-2019. Accessed 11 April 2020.