Khoi Phung’s Product Description Document

This paper is submitted as the requirement for a Product Description Document for process:

19-MOT-IA-HRS-94589 - CS-03 Technical Advisor Process, Application Development.

|  |  |  |  |
| --- | --- | --- | --- |
| **Key information** | **URL** | **User Name** | **Password** |
| **Website** | <https://tc-cs03-app.azurewebsites.net> | tc@tc | Canada11!! |
| **API** | <https://tc-cs03-api.azurewebsites.net> |  |  |
| **OpenAPI Documentation** | <https://tc-cs03-api.azurewebsites.net/swagger/v1/swagger.json> |  |  |
| **API Swagger Page** | <https://tc-cs03-api.azurewebsites.net/swagger> |  |  |
| **GitHub Repo** | <https://github.com/happytobe/cs03-API>  <https://github.com/happytobe/cs03-webapp> |  |  |

Contents

[Introduction 3](#_Toc23478720)

[User Interface 3](#_Toc23478721)

[What is .NET Core? 3](#_Toc23478722)

[What is Blazer? 3](#_Toc23478723)

[Blazor Server 4](#_Toc23478724)

[Blazor WebAssembly 5](#_Toc23478725)

[Why Choose Blazor Server? 5](#_Toc23478726)

[Is it possible to write in Blazor Server now, and switch to Blazor WebAssembly later? 6](#_Toc23478727)

[Why host it in the cloud? 6](#_Toc23478728)

[Why choose Azure? 6](#_Toc23478729)

[User Interface Login Functionality 7](#_Toc23478730)

[Database 7](#_Toc23478731)

[Why choose SQL Server? 7](#_Toc23478732)

[How are the databases used? 7](#_Toc23478733)

[RESTful Web API 7](#_Toc23478734)

[What is an API? 7](#_Toc23478735)

[What is a Web API? 7](#_Toc23478736)

[What is a RESTful Web API? 7](#_Toc23478737)

[Database access 8](#_Toc23478738)

# Introduction

The requirement was to create an online feedback system for users to write a review. This system was to be split into 3 distinct layers:

* User interface
* RESTful Web API
* Database

These layers, and their underlying technology, will be discussed below.

# User Interface

The user interface was written in .NET Core 3.0 Blazor Server. The application is hosted in the Microsoft Azure Cloud as an App Service.

## What is .NET Core?

.NET Core is a Microsoft supported, free and open-source framework for writing software that can be deployed on web, mobile and desktop. It also has the ability to run on Windows, Linux and Mac operating systems.

.NET offers performance, reliability and security, as well as a large ecosystem of .NET libraries.

Choosing.NET Core 3.0, the latest stable version of .NET Core, allows for maximum future proofing with what was available as of the beginning of application development.

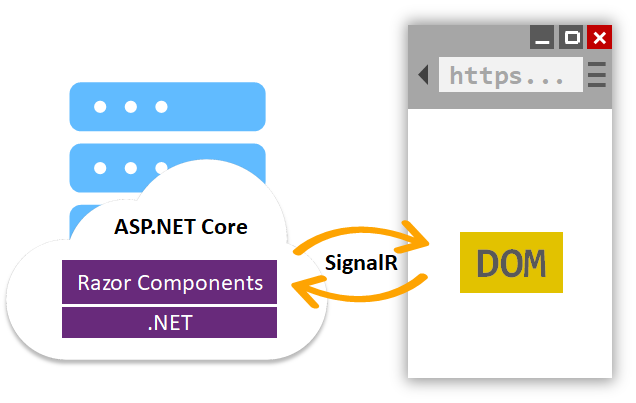
## What is Blazer?

Blazor is a programming language that allows one to create a rich web user interface using only HTML and C#, without having to rely on JavaScript (although it is still possible to inject JavaScript if desired).

There are 2 flavours of Blazor:

* Blazor Server
* Blazor WebAssembly

### Blazor Server



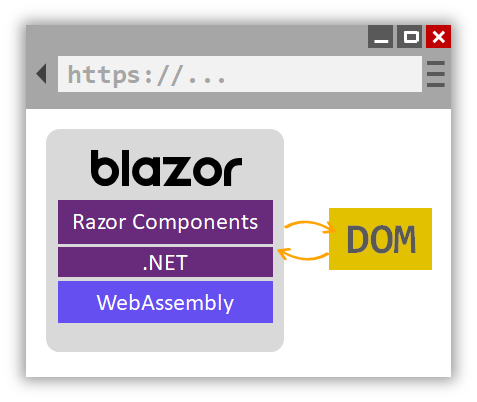
Formerly officially referred to as Server-side Blazor, and with good reason – it does all the processing on the server, and then sends a representation of that processing to be displayed in the browser via Razor Components stored in a Document Object Model (DOM).

When the user interacts with these Razor Components on the screen, the DOM captures the event and sends it back to the server for processing.

After processing, the server sends the modifications to the DOM back to the browser. This cycle repeats again and again for every user event fired from the browser.

Communication between the browser and server is via a real-time communication library called SignalR. For small-scale applications such as this one, SignalR is hosted with the Blazor Server application. As the load increases, it is possible to remove SignalR from the application server and host it separately, including in the cloud.

### Blazor WebAssembly



Also known as Client-side Blazor. This is appropriate, because all the processing is handled by the browser. The Razor components are compiled into .NET assemblies, which are downloaded to the browser along with the .NET runtime.

WebAssembly (abbreviated Wasm) is a sandboxed execution evironment within the browser that runs the .NET assembly and runtime. Updates between the applaction and the DOM are very fast because they are both housed within the browser.

The downside of client-side Blazor is that depending on the size of the application, it may consume a lot of bandwidth when downloading (which could be of concern to mobile users), and it may to be slow on the initial load.

### Why Choose Blazor Server?

As of the beginning of application development, Blazor Server was released with .NET Core 3.0 whereas Blazor WebAssembly is currently available for preview in .NET Core 3.1 release candidate (RC). Blazor Server was chosen because it runs on production ready .NET Core 3.0.

When the day comes that both Blazor offerings are stable, the questions to consider when deciding between the two should include:

|  |  |  |
| --- | --- | --- |
|  | **Blazor Server** | **Blazor WebAssembly** |
| Do you require instant updates from every twitch of the mouse? E.g. a drawing app or real-time gaming |  | × |
| Does the app require a lot of heavy processing? | × |  |
| Would you like the app to work offline? |  | × |
| Do you need to support older browsers that don’t support WebAssembly? | × |  |
| Will your clients be running on low-powered devices? | × |  |

### Is it possible to write in Blazor Server now, and switch to Blazor WebAssembly later?

Yes, because both Blazor offerings use Razor components, it is straight forward to switch from one to another.

### Why host it in the cloud?

We are launching a new application that:

1. Doesn’t generate revenue.
2. May not have a lot of activity.

By hosting in the cloud, we avoid having to buy any servers and will we not require any time, space nor place to physically locate the server. The cloud model allows us to launch our application with minimal upfront hardware costs, yet will allow us to scale when we need to.

### Why choose Azure?

Azure was chosen because Microsoft has datacenters within Canada. This is a requirement for some higher level Protected apps at Transport Canada, and thus eliminates other cloud providers who do not meet this geographical requirement. By pooling resources for all apps to be hosted together, there is a potential for cost savings.

### User Interface Login Functionality

User accounts and passwords are managed using ASP.NET Core Identity. This membership system can be configured to use support login using credentials from external providers including Facebook, Google, Microsoft Account, and Twitter, which is a good option to have for future enhancements.

## Database

The requirements called for data to be persisted in a database. An Azure cloud-hosted SQL Server database was chosen.

### Why choose SQL Server?

SQL server is an enterprise grade relational database management system with a long track record of proven performance and reliability.

### How are the databases used?

There are two databases. One is used by the API for storing user data. The other is used by the User Interface for storing user accounts and passwords. Both are Azure SQL Server Relational Databases.

## RESTful Web API

The RESTful Web API is written in ASP.NET Core 3.0. The numerous benefits of .NET Core 3.0 are already detailed above.

### What is an API?

API stands for Application Programming Interface. By definition, it is a set of subroutine definitions, protocols, and tools for building software and applications.

For a real-world comparison, it is akin to a menu at a restaurant, where is lists the dishes you may order, along with a description of each.

### What is a Web API?

Spoiler alert: it is an API that is accessed over the web. It is a concept (not a technology) that makes use of HTTP for data transfer.

### What is a RESTful Web API?

REST is an acronym for Representational State Transfer. It is defined by 6 guiding constraints that restrict how a server can process and respond to client requests. By operating within these constraints, a system should gain desirable functionalities, among them being performance, scalability, simplicity, and reliability.

### Database access

Database access is handled via Dapper, a lightweight client capable of connecting to many databases. This plays a small role in keeping the API layer decoupled from the database layer.