Projektdokumentation

Inhaltsverzeichnis

[Projektplan - 2 -](#_Toc56342015)

[Projektauftrag - 2 -](#_Toc56342016)

[Projektstrukturplan - 2 -](#_Toc56342017)

[Projektarbeitspaketspezifikation - 2 -](#_Toc56342018)

[Projektzeitplan - 2 -](#_Toc56342019)

[Projektdokumentation - 2 -](#_Toc56342020)

[Visual Studio - 2 -](#_Toc56342021)

[Installation - 2 -](#_Toc56342022)

[GIT - 2 -](#_Toc56342023)

[Konfiguration - 2 -](#_Toc56342024)

[Entwicklung - 2 -](#_Toc56342025)

[Projekterstellung - 2 -](#_Toc56342026)

[CSS-Framework - 2 -](#_Toc56342027)

[Verlinkungen - 3 -](#_Toc56342028)

[Login Mechanismus - 3 -](#_Toc56342029)

[Bilanzberechnung in C# - 3 -](#_Toc56342030)

[Google Maps Integration - 3 -](#_Toc56342031)

[Bilderbearbeitung - 3 -](#_Toc56342032)

[Verwendete Programme - 3 -](#_Toc56342033)

[Vorschau Bildbearbeitung - 3 -](#_Toc56342034)

[Vorher – Nachher Vergleich - 3 -](#_Toc56342035)

# Projektplan

## Projektauftrag

## Projektstrukturplan

## Projektarbeitspaketspezifikation

## Projektzeitplan

# Projektdokumentation

## Visual Studio

Für das Projekt wurde als Entwicklungsumgeben Visual Studio 2019 in der Community Edition ausgewählt, da diese Version des Visual Studio kostenlos ist, solange man nur nicht kommerzielle Projekt damit umsetzt. Ebenfalls wird für eine Blazor WebAssembly Webanwendung auch die .Net Core Runtime benötigt, welche automatisch bei der Installation von Visual Studio mit installiert wird.

Ein weiterer Punkt warum Visual Studio als Entwicklungsumgebung ausgewählt wurde ist, da in diesem Projekt das Version Control System GIT ist und jenes nativ von Visual Studio nach der Installation unterstützt wird.

### Installation

Bei der Installation ist es wichtig die Komponente „ASP.NET und Webentwicklung“ auszuwählen. Durch diese wird das neueste .Net Core Framework mit installiert welches wir für Blazor WebAssembly benötigen. Nach diesem Schritt muss nichts mehr bei der Installation beachtet werden und man kann sich durch den Installer klicke bis alles fertig installiert wurde.

### Konfiguration

## Entwicklungsumfeld

### Version Control System

In diesem Projekt wurde als Version Control System GIT verwendet. Der Vorteil von diesem System besteht darin das jede Änderung in logische Blöcke zusammengefasst werden kann. Solche Blöcke nennt man Commits. Diese Commits kann man sich mit Software wie zum Beispiel Github Desktop veranschaulichen. Dadurch besteht die Möglichkeit auf jeden logischen Block von vorher zurück zu setzten und somit von einem sauberen früheren Stand wieder weiter zu Arbeiten. Dies ist umso hilfreicher sollte man mit mehreren Personen an einem Projekt arbeiten.

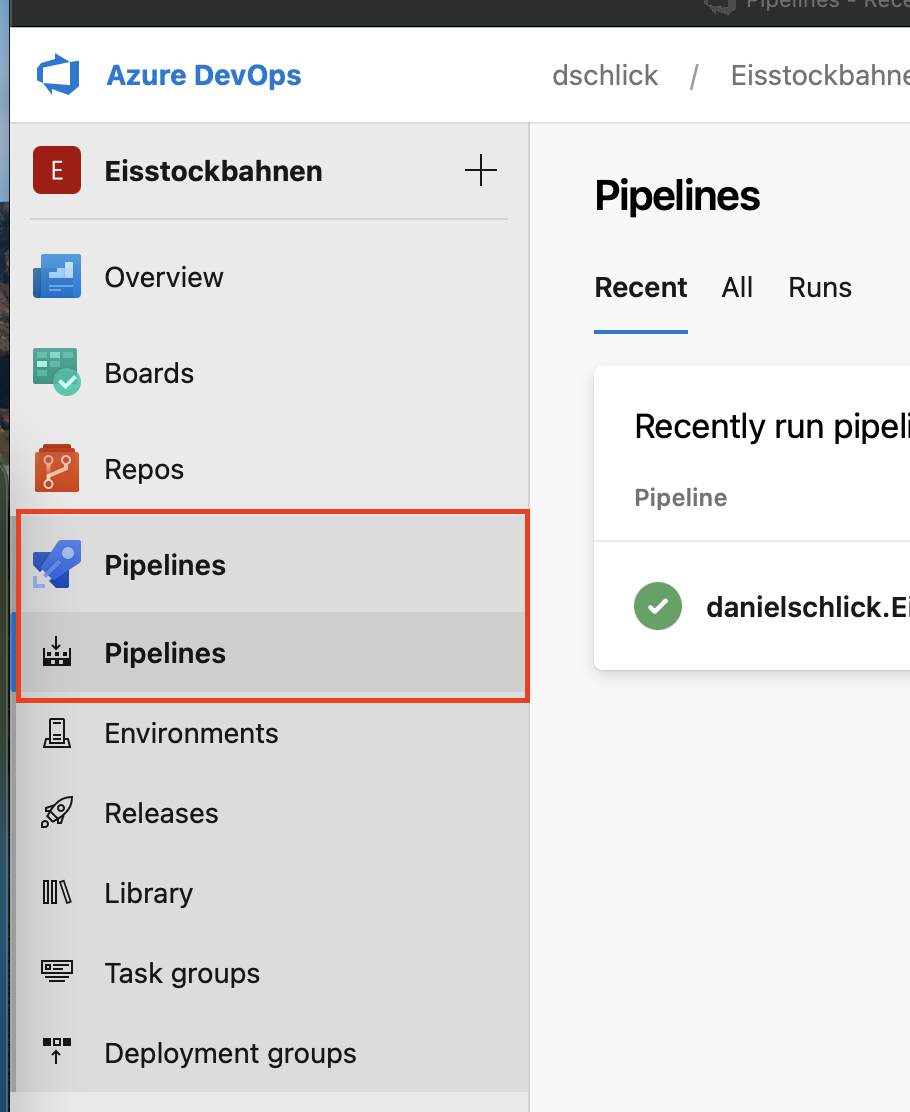
### Github

Bei diesem Projekt wird Github als Version Control Provider verwendet. Einen Version Control Provider verwende ich aus dem Grund das somit mein Projekt gesichert auf einem Server gespeichert ist. So kann im Fall eines technischen versagen das ganze Projekt auf einem anderen System weiter Entwickelt werden.

Github ist seit Jahren mit Marktführer und gehört dem Microsoft Konzern an. Dies vereinfacht die Einbindung in Azure Pipelines.

### Azure Pipelines

Azure Pipelines dienen hier als Rückgrat für den Continuous Integration/Continuous Deployment Prozess. Auf der Seite „<https://dev.azure.com/>“ kann man mit Hilfe eines Microsoft oder Github Kontos sich einloggen und ein Projekt erstellen. Bei diesem Projekt verwenden wir nur den Punkt „Pipelines“.



Mithilfe dieser Pipelines können wir nach jedem Check In in unser Github Repository automatisiert einen Build Vorgang anstoßen. Der Ablauf dieses Build Vorganges wird in sogenannten YAML (Yet Another Markup Language) beschrieben und liegt ebenfalls im Github Repository so das wir auch hier einen Versionsverlauf haben. In diesem Projekt habe ich folgendes YAML geschrieben.

trigger:

branches:

include:

- main

paths:

exclude:

- azure-pipelines.yml

# Erstellen von globalen Variablen

variables:

buildConfiguration: 'Release'

vmImageName: 'ubuntu-latest'

stages:

- stage: Build

displayName: Build stage

jobs:

- job: Build

displayName: Build

pool:

vmImage: $(vmImageName)

# Projekt bauen um Richtigkeit des source codes zu überprüfen

steps:

- task: DotNetCoreCLI@2

displayName: Build Client

inputs:

command: 'build'

projects: '\*\*/\*.csproj'

arguments: '--configuration $(buildConfiguration)'

- stage: Release

displayName: Release stage

jobs:

- job: Release

displayName: Release to Zip

pool:

vmImage: $(vmImageName)

# Release fähige Version von dem Projekt bauen (leichtgewichtiger und optimiert für produktiv Umgebung)

steps:

- task: DotNetCoreCLI@2

displayName: Publish the project

inputs:

command: 'publish'

publishWebProjects: true

arguments: '--output $(Build.ArtifactStagingDirectory)/blazor --configuration $(buildConfiguration)'

# zip archive mit gebauten dll's erstellen

- task: PublishBuildArtifacts@1

inputs:

ArtifactName: 'blazordist'

## Entwicklung

### Projekterstellung

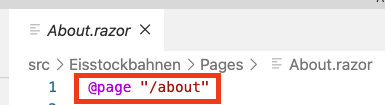
### CSS-Framework

Als CSS Framework wird in diesem Projekt materializecss verwendet. Dies ist ein sehr funktionsstarkes Framework welches schon sehr stark eingesetzt wird. Des Weiteren ist dieses Framework leichtgewichtiger als viele andere Konkurrenzprodukte wie zum Beispiel Bootstrap. Dies bringt den Vorteil das weniger Netzwerkverkehr erzeugt wird, welches ein schnelleres Laden der Webseite ermöglicht. Zudem kommt materializecss zur Gänze ohne jQuery aus welches aufgrund seiner Größe und schlechten Performance somit nicht die Geschwindigkeit der Projektseite beinträchtigen kann.

### Verlinkungen

Verlinkungen in BlazorWASM funktioniert nicht wie in HTML, hierfür wurde ein eigenes Konzept entwickelt.

Auf jeder Seite die man Verlinken will benötigt man das Property „@page“. Danach unter doppelten Anführungszeichen der Pfad unter welchem die Seite erreichbar sein soll. Im Falle der About Seite sieht die Verlinkung wie folgt aus.



Will man jedoch jetzt diese Seite über eine andere Blazor Seite aufrufen muss man das Blazor HTML Property „NavLink“ verwenden. Welches grundsätzlich gleich funktioniert wie das HTML „nav“ Element, da jedoch die Blazor Seiten kompiliert werden, muss man dies hier expliziert definieren.



### Login Mechanismus

### Bilanzberechnung in C#

### Google Maps Integration

## Bilderbearbeitung

### Verwendete Programme

### Vorschau Bildbearbeitung

### Vorher – Nachher Vergleich