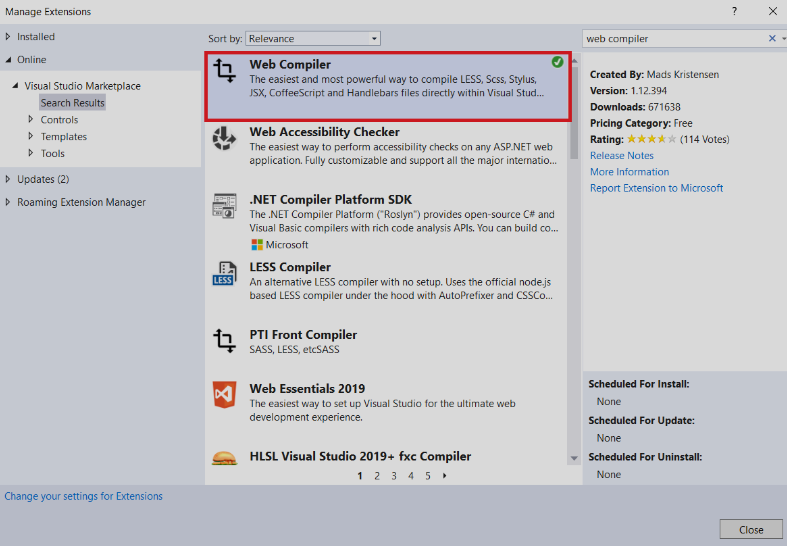
# Cel projektu

Głównym założeniem jest przetestowanie możliwości blazora oraz stworzenie prostych kontrolek dla tej technologii. Dodatkowo projekt sprawdza możliwości pracy z SCSS, CSS i JS. Stworzone kontrolki posłużyły do opisania procesu tworzenia i publikacji pakietów nuget.

# Instalacja Web Compilera

W kontrolkach wykorzystano style zdefiniowane zdefiniowane w formacie SCSS. Aby style mogły zadziałać prawidłowo należy przekonwertować je do formatu CSS. Aby to osiągnąć należy w Visual Studio doinstalować wtyczkę Web Compiler.

Należy wejść w Extensions -> Manage Rxtensions. Następnie znaleźć i zainstalować rozszerzenie.



Następnie należy zamknąc Visual Studio. Okno z instalacją powinno automatycznie się pojawić.

W projekcie powinien znajdować się główny plik zawierający style. Dla przykładu App.scss.

Aby skompilować plik należy kliknąć na niego prawym przyciskiem myszy i wybrać opcję Web Compiler -> Compile File. Spowoduje to wygenerowanie pliku compilerconfig.json zawierającego ustawienia kompilowania plików SCSS.

Wszystkie pliki które zostaną wysłane do klienta należy umieścić w folderze wwwroot. Dlatego miejsce docelowe skompilowanego pliku powinno się znajdować w wwwroot. Przykład:

[

{

"outputFile": "wwwroot/css/site.css",

"inputFile": "App.scss"

}

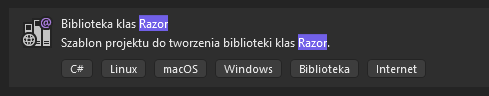
]

Ostatnim krokiem jest zaimportowanie stylu w index.html

<link href="css/site.css" rel="stylesheet" />

\*\*\* W przypadku korzystania z izolacji CSS nie ma wymogu przenoszenia styli do wwwroot

# Tworzenie biblioteki

W celu utworzenia komponentów opublikowanych w repozytorium nuget należy stworzyć nowy projekt “Biblioteka klas Razor”.

# Tworzenie komponentu

W celu utworzenia kontrolki należy dodać nowy składnik Razor. To wystarczy by po zaimportowaniu biblioteki do innego projektu móc bez problemu używać kontrolki.

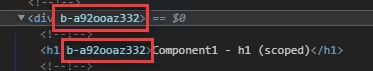
Jedną z ważniejszych rzeczy jest zarządzanie stylami oraz kodem Java Script. Blazer umożliwia specjalne mechanizmy pozwalające na wyizolowanie i automatyczne udostępnianie takich miejsc dla konkretnego komponentu.

## Izolacja CSS

Tworząc plik css dla składnika Razor o nazwie pasującej do szablonu: <nazwa komponentu>.razor.css C# automatycznie umieści go w bibliotece. Tak zdefiniowane style będą wpływały tylko na ten konkretny komponent. W momencie kompilacji blazor generuje nowy plik css na podstawie tego w komponencie, a następnie umieszcza go pod ścieżką “\_framework/scoped.styles.css”. Każdy styl zawiera specjalny atrybut który został specjalnie wygenerowany dla tworzonego komponentu.



Analogiczna operacja zachodzi dla kodu html



Takie rozwiązanie pozwala uniknąć ręcznego importowania stylów za pomocą <link> ponieważ blazor robi to za nas w tle.

## Izolacja JS

Analogicznie jak w przypadku z css blazor oferuje mechanizm izolacji dla kodu java script czyniąc go niedostępnym w global scopie, a dostępnym tylko w komponencie. By to osiągnąc plik musi spełniać schemat <nazwa komponentu>.razor.js

Działanie jest bardzo proste. Ładujemy referencje do pliku w naszym komponencie, a następnie na niej wywołujemy metody java script. Przykład takiej izolacji może wyglądać następująco:



Metody zdefiniowane w pliku java script muszą zostać poprzedzone słówkiem export.

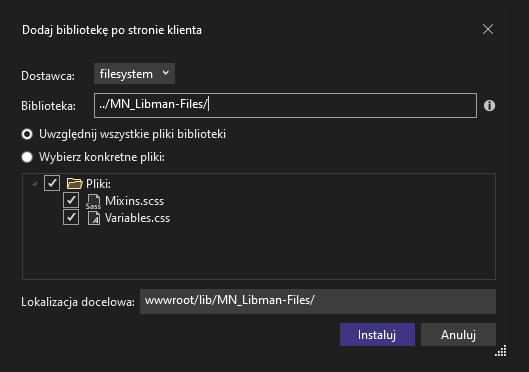
# Libman

Libman, jest lekkim narzędziem służącym do pozyskiwania bibliotek po stronie klienta. Repozytorium nuget służy do przechowywania oraz udostępniania skomplikowanych bibliotek i bardzo ciężko zmusić je do udostępniania plików scss.

Jako że cały projekt jest oparty na tych plikach, ich część wspólna została umieszczona w osobnym katalogu “MN\_Libman-Files”. Dzięki takiemu podejściu kod kontrolek importuje potrzebne pliki do folderu wwwroot a następnie za pomocą @import style danej kontrolki mają dostęp do globalnych mixinów. Jako że pracujemy na plikach scss, podczas kompilacji do css zostają przeniesione tylko style używane przez naszą kontrolkę. Co pozwala uniknąć udostępniania niepotrzebnego kodu.

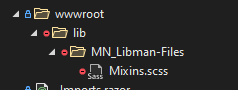
Aby zaimportować biblioteki należy prawym przyciskiem myszy nacisnąć na projek => dodaj => Biblioteka po stronie klienta.

W Visual studio pojawi się okno. które umożliwi wybór źródła biblioteki, folder docelowy, oraz konkretne pliki do importu.

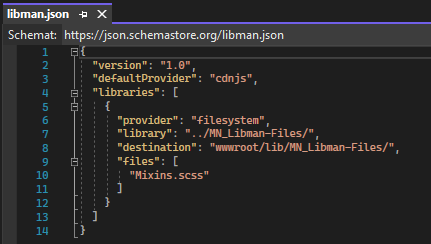


W celu uproszczenia przykładu wykorzystywane pliki zostały umieszczone w repozytorium i odwołujemy się do nich po ścieżce.

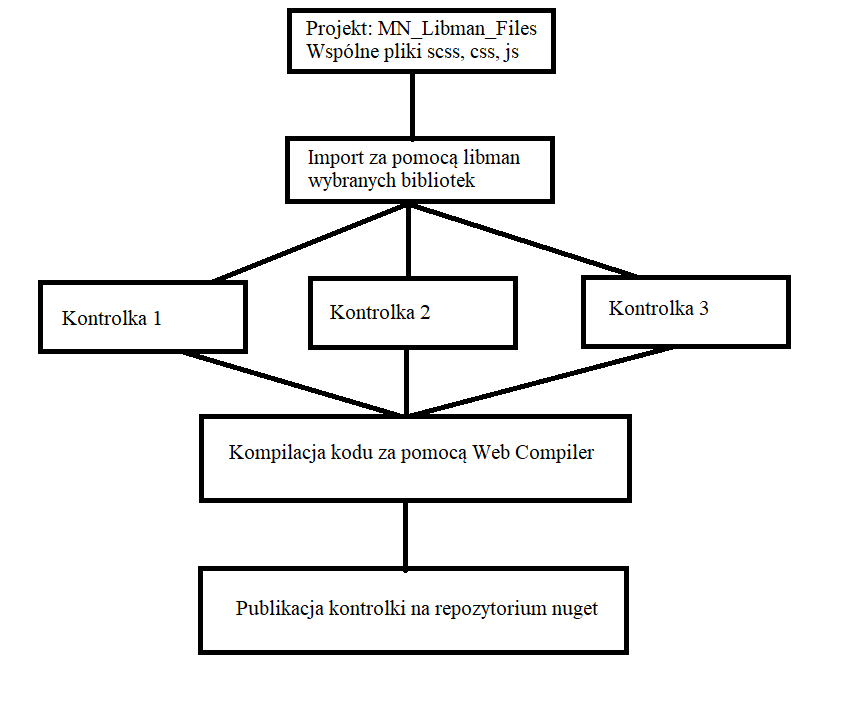
Tak zaimportowane pliki będą widoczne w wybranej ścieżce



W solucji zostanie również umieszczony plik libman.json który zawiera informacje w jaki sposób przywrócić bibliotekę



# Zasada działania projektu



# Publikacja kontrolki na repozytorium Nuget

## Generowanie tokenu dostępu

Aby wygenerować token dostępu należy wejść na repozytorium github. Następnie odszukać zakładkę Settings -> Developer Settings -> Personel Access Token, a następnie utworzyć token z odpowiednim poziomem dostępu. Raz utworzony token nie może być edytowany.

## nuget.config

Aby opublikować pakiet nuget należy utworzyć w projekcie plik nuget.config, a następnie wkleić do niego następujący tekst:

<?xml version="1.0" encoding="utf-8"?>

<configuration>

<packageSources>

<clear />

<add key="github" value="https://nuget.pkg.github.com/NAMESPACE/index.json" />

<add key="nuget.org" value="https://api.nuget.org/v3/index.json" />

</packageSources>

<packageSourceCredentials>

<github>

<add key="Username" value="USERNAME" />

<add key="ClearTextPassword" value="TOKEN" />

</github>

</packageSourceCredentials>

</configuration>

NAMESPACE- należy zastąpić nazwą naszego repozytorium

USERNAME - należy zastąpić nazwą użytkownika

TOKEN - należy zastąpić tokenem wygenerowanym w poprzednim kroku

nuget.org służy jako dostęp do standardowych pakietów. Bez niego nuget widział by tylko nasze biblioteki.

## tworzenie paczki nuget

Aby stworzyć paczkę należy prawym przyciskiem myszki nacisnąc na projekt, następnie wybrać Properties i w zakładce Package zaznaczyć opcję “generate package files during build operation”. Następnie wypełnić pola z odpowiednią wersją nazwą itp.

**UWAGA: pliki nuget należy tworzyć zawsze w trybie Relesse**

Alternatywną opcją jest stworzenie pliku z rozszerzeniem .nuspec. Zazwyczaj plik taki wygląda w następujący sposób: ASEMBLYNAME.nuspec.

Zawartość pliku:

<?xml version="1.0" encoding="utf-8"?>

<package xmlns="http://schemas.microsoft.com/packaging/2010/07/nuspec.xsd">

<metadata>

<id></id>

<version></version>

<description></description>

<authors></authors>

</metadata>

</package>

Plik zawiera podobne parametry jak w zakładce packages.

Podczas procesu budowania należy użyć wtedy komendy:

dotnet pack -c Release PROJECTNAME.csproj -o . /p:NuspecFile=NUSPECNAME.nuspec

PROJECTNAME - nazwa pliku z rozszerzeniem.csproj

NUSPECNAME- nazwa pliku .nuspec

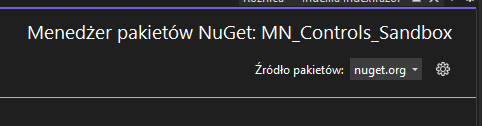
Ta opcja jest odrobinę bardziej skomplikowana bo wymaga ręcznego wskazania zależności bibliotek oraz plików (dll) jakie zostaną przeniesione do nugeta

## Publikacja

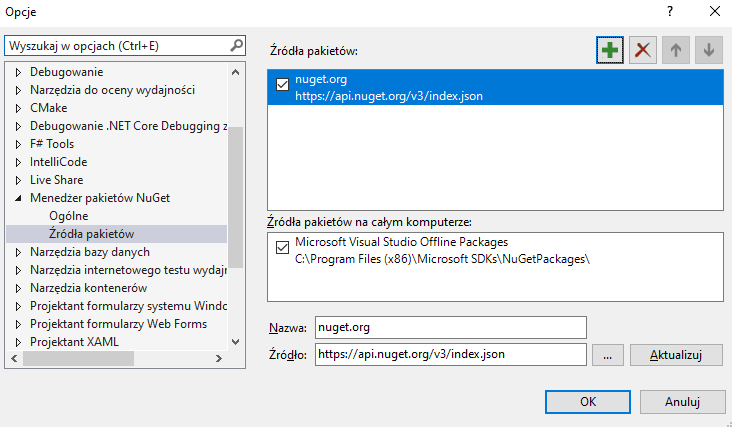
Folder “NugetScripts” zawiera skrypty pozwalające na automatyczne opublikowanie nowego pakietu

# Dodanie nowego źródła nuget

Pakiety nugetowe znajdują się już w repozytorium. Aby móc je pobrać należy otworzyć okno odpowiedzialne za zarządzanie pakietami nuget. W prawym górnym rogu znajduje się ikona koła zębatego odpowiadająca za opcje



Po naciśnięciu przycisku pojawi się okno pozwalające na zarządzanie źródłami pakietów. W prawym górnym rogu znajduje się zielony przycisk krzyża umożliwiający dodanie nowego źródła.



W polu źródło należy podać taki sam url jak w pliku nuget.config. Następnie kliknąć aktualizuj i na samym końcu ok.

Prawidłowa konfiguracja pozwoli na dostęp do pakietów:

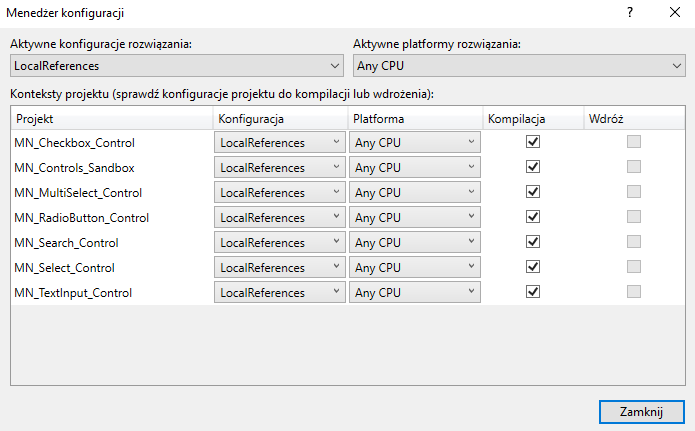
# 

# Rozwój oraz debugowanie własnych pakietów

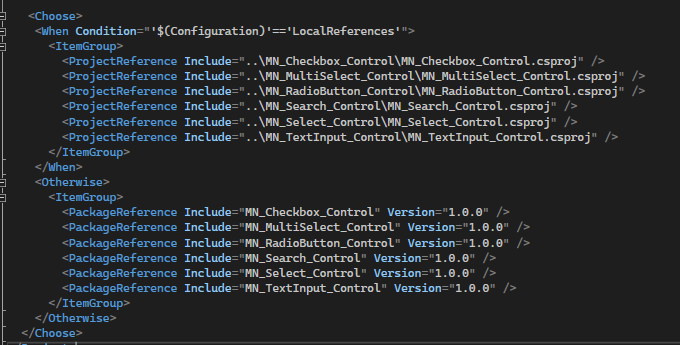
Pakiety są zazwyczaj zwykłymi bibliotekami klas. Taki projekt wymaga uruchomienia bądź testowania z jakiegoś nadrzędnego projektu. Aby rozwijać taki kod należy dodać referencje do projektu na podstawie ścieżki, a po zakończeniu wprowadzania zmian opublikować nowy pakiet. Analogicznie w razie jakiegoś błędu. Ciągłe dodawanie i usuwanie referencji jest niepotrzebną stratą czasu. Aby tego uniknąć należy dodać nowy profil kompilacji. Domyślnie aplikacja posiada 2 tryby: Debug i Release. Z rozwijanego dropdownu można dodać nowy profil:



Na rozwijanej liście “Aktywne konfiguracje rozwiązania” znajduje się opcja “nowy” po wybraniu której powstanie nowa konfiguracja. Należy ją nazwać na przykład “LocalReferences”



Następnie w pliku .csproj należy dodać sekcję Chose w której zdefiniujemy referencje za pomocą ścieżki oraz pakietu:



Na podstawie wcześniej zdefiniowanej konfiguracji można się przełączać między źródłami bibliotek co umożliwia debugowanie na podstawie źródeł, a nie zdekompilowanego kodu.

# Uruchomienie Projektu