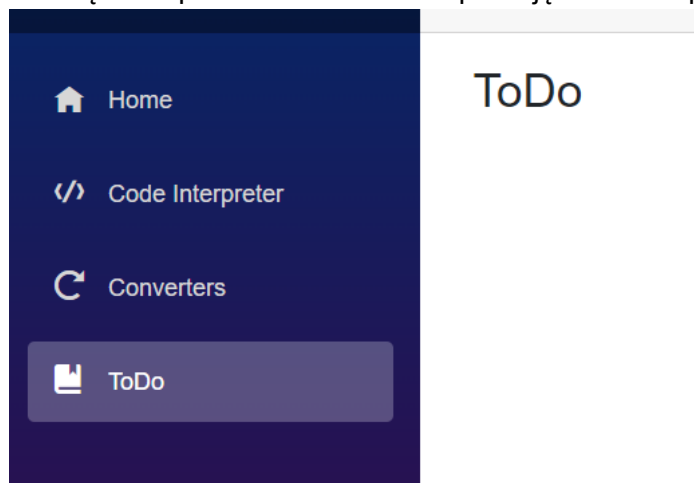
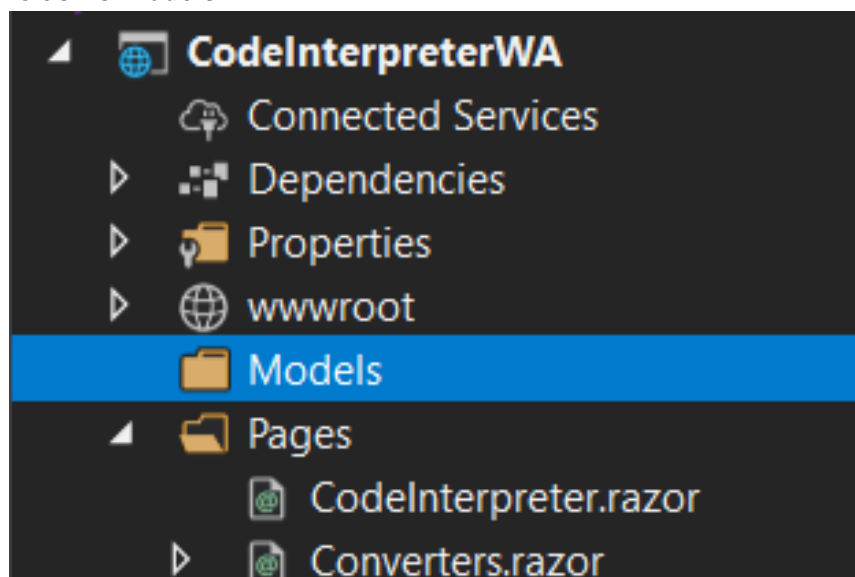


1. Proszę dodać nowy komponent o nazwie **ToDo**. Komponent ten będzie pozwalał na tworzenie oraz zarządzanie listą zadań do zrobienia.
2. Następnie proszę zdefiniować routing dla nowego komponentu. **@page "/todo"**.
3. Podobnie jak w poprzednich krokach proszę dodać nowy komponent *ToDo* do panelu nawigacji. Korzystając z dostępnych darmowych ikon na stronie <https://useiconic.com/open> proszę wybrać jedną i przypisać ją do nowo utworzonej nawigacji.
4. Proszę skompilować i uruchomić aplikację. W moim przypadku wygląda to tak:



5. Kolejnym krokiem jest dodanie folderu o nazwie **Models**, w tym miejscu jak sama nazwa wskazuje będziemy przechowywali różnego rodzaju modele w naszej aplikacji. W kroku **35** (poprzednie zadanie) utworzyliśmy model dla przeliczników, ale dla uproszczenia umieściliśmy go bezpośrednio w komponencie. Nie jest to dobra praktyka dlatego od teraz nasze modele będziemy przetrzymywali w folderze *Models*.



6. Do folderu *Models* proszę dodać nową klasę o nazwie **ToDoItemModel** która będzie reprezentować pojedynczy element z naszej listy zadań do zrobienia. Dodatkowo dodajmy atrybuty które pozwolą nam na walidację danych wprowadzonych przez użytkownika.

```
namespace CIWA.Models
{
    7 references
    public class ToDoItemModel
    {
        [StringLength(20, ErrorMessage = "Title is too long. Max 20 characters")]
        9 references
        public string Title { get; set; }
        5 references
        public bool IsDone { get; set; }
    }
}
```

7. Następnym krokiem jest wykorzystanie pliku **_Imports.razor**. Plik ten pozwala na implementację dyrektyw zapisanych w tym pliku do wszystkich komponentów. Proszę dodać dyrektywę **@using** wskazującą nowy folder z modelami. Od tego momentu każdy z komponentów powinien mieć dostęp do wszystkich modeli z tego folderu.

```
@using CodeInterpreterWA.Models
```

8. Teraz proszę powrócić do komponentu **ToDo** i w sekcji **@code** proszę dodać listę typu *ToDoItemModel* o nazwie **_todos**. Do listy proszę dodać dwa domyślnie zadania

```
private IList<ToDoItemModel> _todos = new List<ToDoItemModel>
{
    new ToDoItemModel { IsDone = false, Title = "Poczytać o Blazor" },
    new ToDoItemModel { IsDone = true, Title = "Utworzyć przelicznik $ na PLN" }
};
```

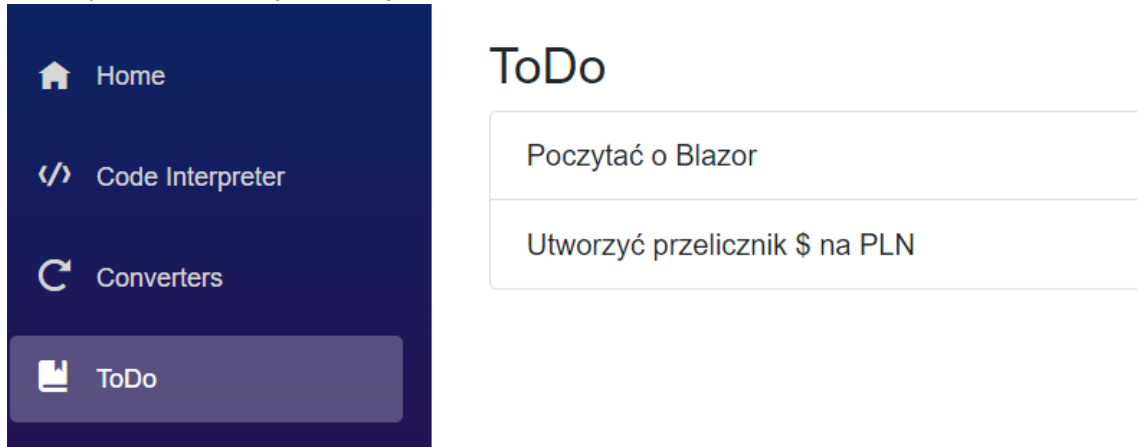
9. Kolejnym krokiem jest wyświetlenie zadań do zrobienia. W tym celu proszę dodać następujący kod do komponentu *ToDo*.

```
@page "/todo"

<h3>ToDo</h3>

<ul class="list-group">
    @foreach(var todo in _todos)
    {
        <li class="list-group-item">@todo.Title</li>
    }
</ul>
```

10. Proszę sprawdzić czy aplikacja działa poprawnie.



11. Musimy umożliwić użytkownikowi dodawanie nowych zadań. W tym celu proszę dodać przycisk oraz input. Akcja kliknięcia przycisku powinna uruchomić metodę **Add** którą dodasz w następnym kroku. Aby pobrać wartość wpisaną przez użytkownika należy powiązać input ze zmienną o nazwie **_todo**. Zmienną utworzysz w następnym kroku.

```
<h3>ToDo</h3>

<div class="col-md-4 mb-3">
    <input class="form-control" placeholder="Todo item" @bind="_todo" />
    <button class="btn btn-primary mt-1" @onclick="Add">Add</button>
</div>
```

12. W tym kroku proszę utworzyć prywatną zmienną typu string o nazwie **_todo**, oraz metodę o nazwie **Add**, której zadaniem jest dodanie nowego zadania do listy oraz wyczyszczenie zawartości zmiennej **todo**.

```

string _todo;

private void Add()
{
    if (!string.IsNullOrEmpty(_todo))
    {
        _todos.Add(new TodoItemModel { Title = _todo });
        // for resetting todo input
        _todo = string.Empty;
    }
}

```

13. W tym kroku dodamy obsługę walidacji wprowadzonych danych przez użytkownika. W tym celu musimy nieco zmodyfikować kod dodający formularz (EditForm).

```

<div class="col-md-4 mb-3">
    <input class="form-control" placeholder="Please put todo item..." @bind="_todo" />
    <button class="btn btn-primary mt-1" @onclick="Add"><span class="oi oi-plus mr-2" aria-hidden="true"></span>Add</button>
</div>

<EditForm Model="@_todo" OnValidSubmit="@Add" class="col-md-6 mb-3">
    <DataAnnotationsValidator />
    <ValidationSummary />

    <InputText class="form-control" id="title" @bind-Value="_todo.Title" />

    <button type="submit" class="btn btn-primary mt-1" @onclick="Add" disabled="@(!context.Validate())">
        <span class="oi oi-plus mr-2" aria-hidden="true"></span>Add
    </button>
</EditForm>

```

```

TodoItemModel _todo = new TodoItemModel();

private void Add()
{
    if (!string.IsNullOrEmpty(_todo.Title))
    {
        _todos.Add(new TodoItemModel { Title = _todo.Title });
        // for resetting todo input
        _todo = new TodoItemModel();
    }
}

```

14. Teraz aby umożliwić użytkownikowi zaznaczenie które zadania zostały już zrobione proszę zmienić kod wyświetlający zadania w następujący sposób:

```
<ul class="list-group">
  @foreach (var todo in todos)
  {
    <li class="list-group-item">
      @todo.Title
      <div class="float-right custom-control custom-checkbox">
        <input type="checkbox" class="form-check-input" id="doneCheckBox" @bind="todo.IsDone">
        <label class="form-check-label" for="doneCheckBox">Done</label>
      </div>
    </li>
  }
</ul>
```

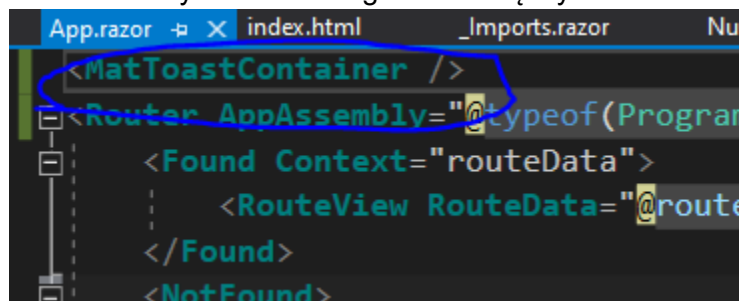
15. Na koniec [dodajmy darmowe komponenty od Material Design \(MatBlazor\)](#), wykorzystamy komponent Toast który będzie się wyświetlał po każdym poprawnym dodaniu elementu do listy ToDo. Najpierw proszę zainstalować MatBlazor przy pomocy nuget. Następnie w pliku _Imports.razor dodać @using MatBlazor, pozwoli nam to na wykorzystanie zainstalowanych komponentów w naszych komponentach. Teraz musimy dodać pliki statyczne do index.html

```
<script src="_content/MatBlazor/dist/matblazor.js"></script>
<link href="_content/MatBlazor/dist/matblazor.css" rel="stylesheet" />
```

Następnym krokiem jest rejestracja serwisu w Program.cs

```
builder.Services.AddMatBlazor();
builder.Services.AddMatToaster(config =>
{
    config.Position = MatToastPosition.TopCenter;
    config.NewestOnTop = true;
    config.ShowCloseButton = true;
    config.MaximumOpacity = 95;
    config.VisibleStateDuration = 3000;
});
```

Teraz musimy zdefiniować gdzie ma się wyświetlać wiadomość zrobimy to globalnie w pliku App.razor.



```
App.razor  X index.html _Imports.razor Nu
<MatToastContainer />
<Router AppAssembly="@typeof(Program).Assembly" />
  <Found Context="routeData">
    <RouteView RouteData="@routeData" />
  </Found>
  <NotFound />
```

16. Aby wyświetlić wiadomość musimy ją utworzyć. Od tego momentu każde poprawne dodanie nowego zadania do listy ToDo powinno wyświetlić informacje na ekranie. Proszę pamiętać o wstrzyknięciu serwisu odpowiedzialnego na Toaster messages.

```
1 @page "/todo"
2 @inject IMatToaster Toaster
3
4 <h3>ToDo</h3>
5
6 if (!string.IsNullOrEmpty(_todo.Title))
7 {
8     _todos.Add(new ToDoItemModel { Title = _todo.Title });
9     Toaster.Add($"ToDo Item '{_todo.Title}' has been added successfully", MatToastType.Success, "Add");
10    // for resetting todo input
11    _todo = new ToDoItemModel();
12 }
```

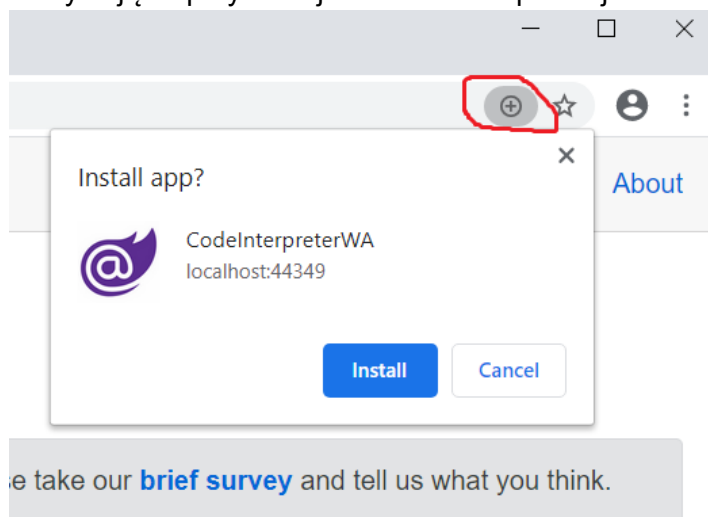
17. Proszę skompilować i uruchomić aplikację oraz sprawdzić czy działa poprawnie

Poczytać o Blazor ☐ Done

Utworzyć przelicznik \$ na PLN ☒ Done

18. Ponieważ podczas tworzenia projektu zaznaczyliśmy opcję PWA teraz możemy zainstalować naszą aplikację i korzystać z niej jak z natywnej aplikacji systemu. Proszę zainstalować naszą aplikację

korzystając z przycisku jak na obrazku poniżej



19. Proszę uruchomić aplikację i sprawdzić czy działa poprawnie.

