📋 Projektübersicht

Eine **To-Do-Anwendung**, die mit 🟣 **Blazor WebAssembly (WASM)** entwickelt wurde, um Aufgaben zu erstellen, bearbeiten, löschen und als erledigt zu markieren. Das Frontend kommuniziert mit einer 🔵 **RESTful API** im Backend**, welche bereits für die [Android To do App](ibrazqrj/TodoApp: TodoApp based on C# and Java (Android Studio)) verwendet wurde. Die Anwendung unterstützt die Benutzer-[[#🔵 Authentifizierung & Autorisierung**]] von Azure Ad (MSAL).

🔧 Technologie-Stack

Frontend

- 🟣 **Blazor WebAssembly (WASM)**
- C# /.NET 9.0
- 🔵 **MudBlazor** (UI-Components)

Backend

- .NET Core Web API (C#)
- Entity Framework Core
- Microsoft SQL Server
- 🔐 **Authentifizierung mit Azure AD**

Tools & IDE

- Visual Studio 2022
- Azure Portal
- Swagger
- Azure DevOps

🔨 Projektstruktur

📂 BlazorToDoApp (Projekt-Root)

- **Connected Services** – Verbundene Dienste (z. B. API-Verbindungen).
- **Dependencies** – NuGet-Pakete und Abhängigkeiten.
- **Properties** – Projektkonfigurationen wie 📎 **Konfiguration ** `appsettings.json`.
- **wwwroot** – Statische Webressourcen (CSS, JS, Bilder).

📁 Custom Elements

- `CustomButton.razor` – Benutzerdefinierte Schaltfläche als Razor-Komponente.

📁 Handlers

- `ApiAuthorizationMessageHandler.cs` – API-Authentifizierung mit Zugriffstoken.

📁 Layout

- `IndexLayout.razor` – Layout der Startseite.
- `LoginDisplay.razor` – UI-Element für Login-Status.
- `MainLayout.razor` – Hauptlayout der App.
- `NavMenu.razor` – Navigation der Anwendung.
- `RedirectToLogin.razor` – Umleitung zur Login-Seite.

📁 Pages

Enthält die Seiten der App:

- `AddTodo.razor` – Neue Aufgaben hinzufügen.
- `Authentication.razor` – Authentifizierungsseite.
- `BaseTodoComponent.cs` – Basisklasse für Todo-Funktionen.
- `EditTodo.razor` – Aufgaben bearbeiten.
- `Home.razor` – Startseite der App.
- `Index.razor` – Standard-Einstiegsseite.
- `RecycleBin.razor` – Gelöschte Aufgaben (Papierkorb).
- `Todos.razor` – Todo-Liste anzeigen.

📁 Resources

- `Translations.cs` – Logik für Sprachverwaltung.
- **Übersetzungsdateien (.resx):**
  - `Translations.de-DE.resx` – Deutsch.

- `Translations.en-US.resx` – Englisch.
- `Translations.fr-FR.resx` – Französisch.
- `Translations.it-IT.resx` – Italienisch.

📁 Services

Backend-Logik der App:

- `HttpTodoService.cs` – API-Kommunikation für Todos.
- `ITodoService.cs` – Interface für Todo-Dienst.
- `LanguageService.cs` – Sprachwechsel-Verwaltung.
- `MemoryTodoService.cs` – In-Memory-Mock-Dienst.
- `TodoItem.cs` – Modellklasse für Aufgaben.

📝 Wichtige Dateien im Root

- `_Imports.razor` – Globale Razor-Namespaces.
- `App.razor` – Hauptkomponente mit Routing.
- `Program.cs` – Startpunkt der Blazor-Anwendung.
- `.gitignore` – Dateien, die Git ignoriert.
- `README.md` – Projektdokumentation.

🌟 Hauptfunktionen

🗹 CRUD-Operationen

- Erstellen, Lesen, Bearbeiten und Löschen der Todos

🗹 Authentifizierung

- Azure AD Login für einen sicheren Zugriff.

🗹 Pagination / Virtualization

- Infinite Scrolling mit Virtualize

🗹 Alerts

- Snackbar-Meldungen bei erfolgreichen und nicht erfolgreichen Ergebnissen mit Hilfe von Mudblazor

🗹 Sprachauswahl

- Translations mithilfe der .resx Files um die Sprachauswahl (Detusch, Englisch, Französisch und Italienisch) anzubieten.

🗹 Active & Recycle Bin Todos

- Die Anzeigen werden nach abgeschlossenen und aktive Todos gefiltert.

🔐 Authentifizierung mit Azure AD

➕ Azure App-Registrierung & Berechtigungen

Die App-Registration für die API war bereits vorhanden, weil dieselbe für die Android App verwendet wurde. Für den 🟣 **Blazor WebAssembly (WASM)** Client habe ich eine eigene 🟣 **App-Registration** erstellt. In der Registration für den Client habe ich die von der exposten API Registration die Berechtigungen übergeben und die URI hinzugefügt.

URI: `https://localhost:7032/authentication/login-callback`
Permission: `user_access`

📎 Konfiguration: `appsettings.json`

```
  "AzureAd": {
    "Authority": "https://login.microsoftonline.com/6ee5ce8d-f608-4339-b50f-39b
    "ClientId": "7d1c2066-7544-4f5a-8a6a-f35e5a82b4b3",
    "ValidateAuthority": true,
    "RedirectUri": "https://localhost:7032/authentication/login-callback",
    "PostLogoutRedirectUri": "https://localhost:7032"
  },
    "Api": {
    "BaseUrl": "http://localhost:5085/api/",
    "ApiAccess": "api://6f1ef8a4-18ee-4983-9d4e-b44abe56f234/api_access",
    "LoginMode": "redirect"
  }
```

📎 Konfiguration: `Program.cs`

```
builder.Services.AddMsalAuthentication(options =>
{
    var conf = builder.Configuration;

    builder.Configuration.Bind("AzureAd", options.ProviderOptions.Authenticatio
    options.ProviderOptions.DefaultAccessTokenScopes.Add(conf.GetValue<string>(
    options.ProviderOptions.LoginMode = conf.GetValue<string>("Api:LoginMode");
});
```

⚠️ Wichtige Klassen & Logik

🗂️ Datenmodell `TodoItem.cs`

Das 🔵 **Datenmodell** dient dazu, die Struktur der Daten für eine To-Do-Aufgaben in der To Do App zu definieren. Hier wird mit Id, Title, Description, Time, IsComplete und isGrouped eine "Vorlage" dargestellt, welche Eigenschaften und Regeln ein Todo hat

```
public class TodoItem
{
     public int Id { get; set; }
     [Required(ErrorMessage = "Title is required!")]
     [StringLength(100, ErrorMessage = "Title cannot exceed 100 characters!")]
     public string Title { get; set; } = string.Empty;
     [Required(ErrorMessage = "Description is required!")]
     public string Description { get; set; } = string.Empty;
     [Required]
     public DateTime Time { get; set; }
     public bool IsComplete { get; set; }
     public bool IsGrouped { get; set; } = false;
}
```

🔗 API-Service `HttpTodoService.cs`

Im API-Service habe ich die verschiedenen asynchronen Methoden geschrieben, um die CRUD-Operationen auszuführen. Im folgendem Beispiel wird eine Liste von TodoItem-Objekte von der API abruft:

GetTodosAsync

Holt alle Todos von der Datenbank durch die API indem eine Request mit den nötigen Parametern gemacht wird und HttpMethod.Get verwendet wird. Falls die response Erfolgreich war wird der zurückgegebene Content in eine List von TodoItems gespeichert.

```
public async Task<List<TodoItem>?> GetTodosAsync(int offset = 0, int limit = 10
{
    try
    {
        var completed = isCompleted != null ? $"&completed={isCompleted}" : "";
        var request = new HttpRequestMessage(HttpMethod.Get, $"todo?offset={off
        var response = await _httpClient.SendAsync(request);

        if (response.IsSuccessStatusCode)
        {
            return await response.Content.ReadFromJsonAsync<List<TodoItem>>();
        }
        throw new HttpRequestException("API call failed");
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Fehler beim Abrufen der Todos: {ex.Message}");
        throw;
    }
}
```

- *Zuerst wird in der variable completed der Wert zugewiesen indem geprüft wird ob isCompleted nicht null ist und wen es wahr ist, wird ein String erstellt der den Wert von isCompleted in den string &completed={isCompleted} einfügt und wenn es nicht wahr ist, wird ein leerer String zurückgegeben.*
- *Nachdem wird in der variable request der Wert zugewiesen indem eine HTTP-Anfrage mit der GET-Methode gemacht wird um eine URL zu konstruieren die Query-Parameter behinhaltet.*
- *Sobald die HTTP-Anfrage erledigt ist, wird in der variable response die vorherige konstruierte URL an den httpClient gesendet und durch await, wird gewartet ob eine Antwort vom httpClient kommt.*
- *Wenn die Antwort erfolgreich ankommt, soll der Content von der Antwort von JSON in eine List aus TodoItems erstellt werden und hier wird auch gewartet bis eine Antwort kommt durch await.*
- *Wenn response aber unsuccessfull ist wird eine HttpRequestException rausgegeben. Wenn aber schon vorher ein Problem auftaucht geht es weiter bei catch und gibt genauso eine Exception raus.*

GetTodoCount

Die Methode zählt wie viele Todos noch aktiv sind, welche auch mit einer Http.Method.Get durchgeführt wird, welcher bei einem success in ints gespeichert wird.

```
public async Task<int?> GetTodoCount(bool completed = false)
{
    var request = new HttpRequestMessage(HttpMethod.Get, $"todo/count?completed
    var response = await _httpClient.SendAsync(request);

    if (response.IsSuccessStatusCode)
    {
        return await response.Content.ReadFromJsonAsync<int>();
    }

    return null;
}
```

- *In der variable request der Wert zugewiesen indem eine HTTP-Anfrage mit der GET-Methode gemacht wird um eine URL zu konstruieren die Query-Parameter behinhaltet.*
- *Danach wird der request an den httpClient gesendet und durch await wird auf die Antwort gewartet und als response gespeichert.*
- *Wenn response successfull ist gibt er ein int zurück und ansonsten gibt es null raus.*

CreateTodoAsnyc

Mit dieser methode wird ein Versuch gemacht ein Todo mit einer HttpClient.PostAsJsonAsync als neues Todo zu erstellen, falls es nicht klappt kommt es zu einer Exception.

```
public async Task<bool> CreateTodoAsync(TodoItem newTodo)
{
    try
    {
        var tokenResult = await _tokenProvider.RequestAccessToken();
        if (!tokenResult.TryGetToken(out var token))
        {
            return false;
        }

        _httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHea
        var response = await _httpClient.PostAsJsonAsync("http://localhost:5085

        return response.IsSuccessStatusCode;
```

```
    }
    catch (TaskCanceledException)
    {
        throw new HttpRequestException("Timed out.");
    }
}
```

- *In tokenResult wird vom Tokenprovider der Token gespeichert*
- *Wenn tokenResult nicht leer ist, wird versucht den Token aus dem Ergebnis zu extrahieren.*
- *Falls kein Token vorhanden ist wird die Funktion mit false beendet.*
- *Ansonsten wenn ein Token gefunden wird, wird der HTTP-Client so konfiguriert, dass er den Token im Authorization-Header mit dem Schema "Bearer" mitsendet.*
- *Danach wird eine POST-Anfrage an die API gesendet, um das neue Todo-Element zu erstellen.*
- *Danach wird von response entweder true oder false gegeben, jenachdem ob das erstellen des neuen Todos durchgeführt wurde.*
- *Falls während der Ausführung ein Fehler auftaucht, wird diese abgefangen und mit der Nachricht "Timed out" ausgelöst.*

GetTodoByAsync

Hier wird versucht ein Todo per id zu finden, wenn kein Ergebnis aufkommt, wird eine Exception ausgelöst.

```
public async Task<TodoItem?> GetTodoByIdAsync(int id)
{
    try
    {
        var tokenResult = await _tokenProvider.RequestAccessToken();
        if (!tokenResult.TryGetToken(out var token))
        {
            _httpClient.DefaultRequestHeaders.Authorization = new Authenticatio
        }

        var response = await _httpClient.GetAsync($"http://localhost:5085/api/T
        if (response.IsSuccessStatusCode)
        {
            return await response.Content.ReadFromJsonAsync<TodoItem>();
        }
        return null;
    }
    catch (TaskCanceledException)
    {
```

```
            throw new HttpRequestException("Timed out.");
        }
    }
```

- *In tokenResult wird vom Tokenprovider der Token gespeichert*
- *Wenn tokenResult nicht leer ist, wird versucht den Token aus dem Ergebnis zu extrahieren*
- *Falls der Token nicht extrahiert werden kann, wird ein Header gesetzt mit dem "Bearer" Schema*
- *Danach wird eine GET-Anfrage an die URL der API geschickt, wobe idie ID des gewünschten Todo-Elements in die URL eingefügt wird.*
- *Wenn der HTTP-Statuscode OK ist, wird der INhalt der Antwort in ein Objekt vom Typ TodoItem deserialisiert und zurückgegeben*
- *Falls aber der HTTP-Statuscode ein Fehler liefert, wird null zurückgegeben.*
- *Wenn die Anfrage aufgrund eines Timeouts abgebrochen wird, gibt der catch die Exception raus.*

UpdateTodoAsync

Bei einer Änderung eines Todos wird diese Methode aufgerufen, damit die Änderung gespeichert wird

```
public async Task<bool> UpdateTodoAsync(TodoItem updatedTodo)
{
    var tokenResult = await _tokenProvider.RequestAccessToken();
    if (tokenResult.TryGetToken(out var token))
    {
        _httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHea
    }

    var response = await _httpClient.PutAsJsonAsync($"http://localhost:5085/api
    return response.IsSuccessStatusCode;
}
```

- *In tokenResult wird vom Tokenprovider der Token gespeichert.*
- *Wenn tokenResult nicht leer ist, wird versucht den Token aus dem Ergebnis zu extrahieren.*
- *Falls der Token nicht extrahiert werden kann, wird ein Header gesetzt mit dem "Bearer" Schema.*
- *Danach wird eine PUT-Anfrage an die URL der API geschickt wobei die Id des todoItems in die URL eingefügt wird.*
- *Wenn response success als Antwort bekommt, wird es als OK zurückgegeben.*

DeleteTodoAsync

Mit dieser Methode wird ermöglich ein Todo komplett zu löschen.

```
public async Task<bool> DeleteTodoAsync(int id)
{
    var tokenResult = await _tokenProvider.RequestAccessToken();
    if (tokenResult.TryGetToken(out var token))
    {
        _httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHea
    }

    var response = await _httpClient.DeleteAsync($"http://localhost:5085/api/To
    return response.IsSuccessStatusCode;
}
```

- *In tokenResult wird vom Tokenprovider der Token gespeichert.*
- *Wenn tokenResult nicht leer ist, wird versucht den Token aus dem Ergebnis zu extrahieren.*
- *Falls der Token nicht extrahiert werden kann, wird ein Header gesetzt mit dem "Bearer" Schema.*
- *Danach wird eine DELETE-Anfrage an die URL der API mit der dazugehörigen id in der URL geschickt.*
- *Wenn repsonse success als Antwort bekommt, wird es als OK zurückgegeben.*

DeleteAllBinTodosAsnyc

Durch dieser Methode werden alle Todos die bereits im Recycle Bin durch den "Filter" isCompleted = true vorhanden sind, komplett gelöscht.

```
public async Task<bool> DeleteAllBinTodosAsync(bool? isCompleted = true)
{
    var tokenResult = await _tokenProvider.RequestAccessToken();
    if (tokenResult.TryGetToken(out var token))
    {
        _httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHea
    }

    var completed = isCompleted != null ? $"completed={isCompleted.ToString().T
    var response = await _httpClient.DeleteAsync($"http://localhost:5085/api/To

    return response.IsSuccessStatusCode;
}
```

- *In tokenResult wird vom Tokenprovider der Token gespeichert.*
- *Wenn tokenResult nicht leer ist, wird versucht den Token aus dem Ergebnis zu extrahieren.*
- *Falls der Token nicht extrahiert werden kann, wird ein Header gesetzt mit dem "Bearer" Schema.*
- *In completed wird überprüft ob der Parameter isCompleted nicht null ist. Falls nicht, wird ein Query-String mit "completed=true" erstellt. Wenn aber isCompleted null ist, bleibt der String leer*
- *Danach wird eine DELETE-Anfrage an die API gesendet, wobei die URL den optionalen Query-Parameter enthält.*
- *Das Ergebnis der Anfrage wird anhand des Stautscodes überprüft und es wird true gegeben wenn der Löschvorgang erfolgreich war, ansonsten false.*

🖥️ Frontend `BaseTodoComponent.cs`

Mit Hilfe des folgenden Codes, werden die API Calls ausgeführt. Für das Frontend wird die Logik zum laufen gebracht.

```
public class BaseTodoComponent : ComponentBase
{
    // [Inject] protected HttpTodoService TodoService { get; set; } = default!;
    [Inject] protected NavigationManager Navigation { get; set; } = default!;
    [Inject] protected IStringLocalizer<Translation> Localizer { get; set; } =
    [Inject] protected IAccessTokenProvider TokenProvider { get; set; } = defau
    [Inject] protected ITodoService TodoService { get; set; } = default!;
    [Inject] ISnackbar Snackbar { get; set; } = default!;


    protected List<TodoItem>? todos;
    protected Virtualize<TodoItem>? virtualizeComponent;
    protected bool isLoading = true;
    protected int offset = 0;
    protected int limit = 10;
    protected bool isCompleteFilter = false;
```

- *Mit den Inject Attributen werden die benötigten Services injiziert um die Methoden ausführen zu können.*
- *Genauso werden Lokale Variablen erstellt.*
- *NavigationManager dient für die Navigation innerhalb der Anwendung*

- *IStringLocalizer Bietet die Lokalisierung bzw. die Übersetzung der Seiten in verschiedene Sprachen.*
- *IAcessTokenProvider versorgt die Komponente mit Zugriffstokens für die Authentifizierung*
- *TodoService stellt Methoden für die Todo-bezogenen Operationen bereit.*
- *Snakcbar ermöglicht das Anzeigen von Benachrichtigungen.*
- *List todos ist eine Liste, die Todo-Elemente speichert.*
- *Virtualize virtualizeComponent ist eine Referenz zum Virtualizeobjekt, das zur Darstellung von Datenmengen dient.*
- *bool isLoading gibt an ob gerade Daten geladen werden.*
- *int offset und limit steuern die Paginierung (Startindex und Anzahl der geladenen Elemente).*
- *bool isCompleteFilter ist ein Filter um beispielsweise nur abgeschlossene oder aktive Todos anzuzeigen.*

```
    // Bei Initialisierung
    protected override async Task OnInitializedAsync()
    {
        try
        {
            todos = await TodoService.GetTodosAsync(isCompleted: isCompleteFilt
            offset += limit;
            isLoading = false;
        }
        catch (HttpRequestException)
        {
            Snackbar.Add(@Localizer["Connection failed"], Severity.Error);
            isLoading = false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Fehler beim Abrufen der Daten: {ex.Message}");
            isLoading = false;
        }
    }
```

- *Bei Initialisierung wird als aller erstes die GET-Methode ausgeführt indem der isCompleteFilter als isComplete übergeben wird. (Der Filter wird in Recycle Bin und in Todos gesteuert).*
- *offset wird mit limit addiert und isLoading wird auf false gesetzt, weil die Todos dann nicht mehr laden.*

- *Wenn was nicht richtig laufen würde, ob es ein Timeout oder ein problem bei der Verbindung zur API entsteht, wird eine Snackbar oder ein CW angegeben und isLoading ebenfalss auf false gesetzt und die Exception ausgegeben.*

```
    // Buttons um Todo zu editieren oder ein neues hinzufügen.
    protected void OpenEditTodo(int id) => Navigation.NavigateTo($"/edittodo/{i
    protected void OpenAddTodo() => Navigation.NavigateTo("/addtodo");
```

- *Bei diesen Methoden wird mithilfe von Navigation bei ausführung dieser Methoden die jeweilige Page aufgerufen.*

```
    // isComplete true : false
    protected async Task UpdateTodoStatus(int id, bool isComplete)
    {
        try
        {
            var todo = todos?.FirstOrDefault(t => t.Id == id);
            if (todo == null) return;

            todo.IsComplete = isComplete;

            var success = await TodoService.UpdateTodoAsync(todo);

            if (success)
            {
                todos = todos?.Where(t => t.Id != id).ToList();
                if (virtualizeComponent != null)
                {
                    await virtualizeComponent.RefreshDataAsync();
                }
                await InvokeAsync(StateHasChanged);
                if (todo.IsComplete == true)
                {
                    Snackbar.Add(@Localizer["Sucessfully moved your todo into t
                }
                else
                {
                    Snackbar.Add(@Localizer["Sucessfully moved your todo into t
                }
            }
            else
            {
                Console.WriteLine($"Fehler: Todo mit ID {id} konnte nicht aktua
            }
```

```
        }
        catch (HttpRequestException)
        {
            Snackbar.Add(@Localizer["Connection failed"], Severity.Error);
        }
    }
```

- *Mit FirstOrDefault wird in der lokalen Todo-Liste nach einem Element mit der passenden ID gesucht. Falls kein Element gefunden wird, wird die Methode beendet.*
- *Wenn todo null ist, gibt er auch null zurück.*
- *Ansonsten wird UpdateTodoAsync aufgrufen und die variable als parameter übergeben um das Todo zu aktualisieren.*
- *Wenn die Aktualisierung geklappt hat, wird der Status angepasst sein und wird aus der Lokalen Liste entfernt und in die andere Ansicht verschoben.*
- *Wenn virtualizeComponent nicht null ist wird Virtualize aktualisiert um die Änderungen zu sehen.*
- *Mit InvokeAsync wird dann die UI aktualisiert.*
- *Wenn der Status von false auf true gesetzt wird, gibt er eine Snackbar Notification raus genauso auch umgekehrt, dass es geklappt hat, falls es aber nicht klappt weil die ID nicht gefunden werden kann, gibt es eine Fehlermeldung.*
- *Falls im Durchlauf ein Problem entsteht wird der catch ausgelöst.*

```
    // Todos in der Virtualization mit Gruppierung anzeigen
    protected async ValueTask<ItemsProviderResult<TodoItem>> LoadTodos(ItemsPro
    {
        try
        {
            Console.WriteLine($"Loading Todos: StartIndex={request.StartIndex},

            var total = await TodoService.GetTodoCount(isCompleteFilter) ?? 0;
            var result = await TodoService.GetTodosAsync(request.StartIndex, re

            Console.WriteLine($"Loaded {result.Count} Todos from DB: {string.Jo

            var groupedTodos = new List<TodoItem>();
            var groupedByCategory = result.OrderBy(t => t.Time).GroupBy(t => Ge

            foreach (var group in groupedByCategory)
            {
                Console.WriteLine($"Processing group: {group.Key} with {group.C

                groupedTodos.Add(new TodoItem
```

```
                {
                    Id = -1,
                    Title = group.Key,
                    IsGrouped = true
                });

                foreach (var item in group)
                {
                    Console.WriteLine($"Adding Todo: {item.Title}");
                    groupedTodos.Add(item);
                }
            }

            Console.WriteLine($"Final grouped list contains {groupedTodos.Count

            return new ItemsProviderResult<TodoItem>(groupedTodos, total + grou
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Fehler in LoadTodos: {ex.Message}");
            return new ItemsProviderResult<TodoItem>(Array.Empty<TodoItem>(), 0
        }
    }
```

- *Zuerst wird in total das Ergebnis von GetTodoCount mit dem isCompleteFilter gespeichert und wenn das Ergebnis 0 ist, gibt es null zurück.*
- *Danach wird in result das Ergebnis von GetTodosAsync mit den jeweiligen parametern gespeichert.*
- *Es wird eine neue Liste aus TodoItems erstellt namens groupedTodos.*
- *Danach wird result zuerst nach dem Datum sortiert und mit Hilfe der Methode GetTodoCategory() in Gruppen eingeteilt. Die Variable groupedByCategory enthält also mehrere Gruppen, wobei jede Gruppe eine Kategorie repräsentiert*
- *Für jede Gruppe in groupedByCategory wird zuerst ein Gruppenkopf eingefügt. Ein neues TodoItem wird erstellt, die Id -1 wird zugewiesen, der Titel wird dann der Gruppenname und isGrouped wird auf true gesetzt. Anschliessend werden alle Todos der aktuellen Gruppe der Liste groupedTodos hinzugefügt.*
- *Danach wird ein neues Objekt vom Typ ItemsProviderResult erstellt und zurückgegeben.*
- *Falls während dem gesamten Prozess eine Exception auftritt, wird diese im catch-Block abgefangen.*

```
    protected async Task LoadNextItems()
    {
```

```
        todos?.AddRange(await TodoService.GetTodosAsync(offset, limit, isComple
        offset += limit;
        StateHasChanged();
    }
```

- *Hier wird von der aktuellen lokalen Liste die ganzen Todos per GET-Methode dazu addiert.*
- *Danach erhöhen wir offset mit dem Limit und aktualisieren die UI zum schluss mit StateHasChanged()*

```
    protected async Task DeleteTodo(int id)
    {
        var success = await TodoService.DeleteTodoAsync(id);
        if (success)
        {
            todos = todos.Where(t => t.Id != id).ToList();
            await virtualizeComponent.RefreshDataAsync();
            Snackbar.Add(@Localizer["Sucessfully deleted your todo!"], Severity
        }
        else
        {
            Snackbar.Add(@Localizer["Couldn't delete your todo. Please try agai
        }
    }
```

- *In success wird die die DeleteTodoAsync Methode aufgerufen und id als parameter übergeben*
- *Wenn success erfolgt werden die todos welche die selben Id's haben in eine Liste erstellt*
- *Danach wird auf einen Datarefresh gewartet und zum schluss eine Snackbar Notification angezeigt*
- *Wenn das nicht klappt, gibts eine Snackbar Notification mit einer Fehlermeldung.*

```
    protected async Task DeleteAllBinTodos()
    {
        try
        {
            var success = await TodoService.DeleteAllBinTodosAsync(isCompleteFi
            if (success)
            {
                todos = todos.Where(t => t.IsComplete != isCompleteFilter).ToLi
                await virtualizeComponent.RefreshDataAsync();
                Snackbar.Add(@Localizer["Sucessfully cleared your recycling bin
            }
```

```
            else
            {
                Console.WriteLine("Couldn't delete all bin tasks");
            }
        }
        catch (HttpRequestException)
        {
            Snackbar.Add(@Localizer["Something went wrong... Please try again!"
        }
    }
```

- *Als erstes wird in success die DeleteAllBinTodosAsync mit dem isCompleteFilter gespeichert und wenn success ausgelöst wird, wird nach isComplete mit dem isCompleteFilter verglichen und in eine Liste gepackt.*
- *Danach erwartet man vom virtualizeComponent die Daten zu aktualisieren, um zusätzlich nachdem die Antwort kommt eine Snackbar Notification für die Bestätigung auszulösen.*
- *Falls aber keine Antwort von DeleteAllBinTodosAsync kommt in der Konsole eine Meldung.*
- *Sollte im Verlauf ein Fehler auftreten wird eine Snackbar ausgelöst mit einer Fehlermeldung.*

```
    protected string GetTodoCategory(DateTime todoDate)
    {
        var today = DateTime.Today;
        var tomorrow = today.AddDays(1);
        var yesterday = today.AddDays(-1);

        if (todoDate.Date == today) return Localizer["TODAY"];
        if (todoDate.Date == tomorrow) return Localizer["TOMORROW"];
        if (todoDate.Date == yesterday) return Localizer["YESTERDAY"];
        if (todoDate.Date > tomorrow) return Localizer["PENDING"];
        return Localizer["PREVIOUS"];
    }
}
```

- *In GetTodoCategory bekommen wir als Parameter den todoDate für time (TodoItem Datum)*
- *Hier entstehen 3 variablen today, tomorrow und yesterday. Die Variablen erhalten die dazugehörigen Values durch DateTime.Today.*
- *Wenn beispielsweise Gruppiert wird, geht das Programm durch die if statements und gibt die richtigen Values zurück die er benötigt.*

🖥️ Frontend `Todos.razor` `RecycleBin.razor`

Bei den beiden Razor komponenten, wird das Frontend UI aufgebaut und zusätzlich die Logik von BaseTodoComponen.cs eingebunden um jeweils die Methoden ausführen zu können.

```
@page "/todos"
@inherits BaseTodoComponent
@using BlazorToDoApp.Services
@using Microsoft.AspNetCore.Components.WebAssembly.Authentication
@using BlazorToDoApp.Resources
@using Microsoft.Extensions.Localization
@inject IAccessTokenProvider TokenProvider
@inject ISnackbar Snackbar

@* Required *@
<MudThemeProvider />
<MudPopoverProvider />

@* Needed for dialogs *@
<MudDialogProvider />

@* Needed for snackbars *@
<MudSnackbarProvider />

<AuthorizeView>
    <Authorized>
        <div class="twelve">
            <h1>@Localizer["MY TODOS"] 📝</h1>
        </div>

        @if (isLoading == true)
        {
            <p>@Localizer["Todos are loading..."]</p>
        }
        else if (todos == null)
        {
            <p>@Localizer["No todos found."]</p>
        }
        else if (todos.Count == 0)
        {
            <p>@Localizer["No todos found."]</p>
        }
        else
        {
            <div>
```

```
                <table class="table">
                    <thead>
                        <tr>
                            <th>@Localizer["Todo"]</th>
                            <th>@Localizer["Description"]</th>
                            <th>@Localizer["Date"]</th>
                            <th>@Localizer["Is Complete?"]</th>
                            <th style="text-align: right;">@Localizer["Actions"
                        </tr>
                    </thead>
                    <tbody>
                        <Virtualize @ref="virtualizeComponent" Context="todo" I
                            @if (todo.IsGrouped)
                            {
                                <tr class="header-row">
                                    <td colspan="5"><strong>@todo.Title</strong
                                </tr>
                            }
                            else if (!todo.IsComplete)
                            {
                                <tr>
                                    <td>@todo.Title</td>
                                    <td>@todo.Description</td>
                                    <td>@todo.Time.ToString("dd.MM.yyyy")</td>
                                    <td>@(todo.IsComplete ? "✔" : "❌")</td>
                                    <td style="text-align: right;">
                                        <button class="action-button" @onclick=
                                        <button class="action-button" @onclick=
                                    </td>
                                </tr>
                            }
                        </Virtualize>
                    </tbody>
                </table>
            </div>
        }

        <button id="bottone6" @onclick="OpenAddTodo">➕</button>
    </Authorized>
    <NotAuthorized>
        <p>@Localizer["Unauthorized"]<a href="authentication/login">@Localizer[
    </NotAuthorized>
</AuthorizeView>


@code {
```

```
    protected override async Task OnInitializedAsync()
    {
        isCompleteFilter = false;
        await base.OnInitializedAsync();
    }
}
```

```
@page "/recyclebin"
@inherits BaseTodoComponent
@using BlazorToDoApp.Services
@using Microsoft.AspNetCore.Components.WebAssembly.Authentication
@using BlazorToDoApp.Resources
@using Microsoft.Extensions.Localization
@inject IAccessTokenProvider TokenProvider
@inject ITodoService TodoService

@* Required *@
<MudThemeProvider />
<MudPopoverProvider />

@* Needed for dialogs *@
<MudDialogProvider />

@* Needed for snackbars *@
<MudSnackbarProvider />

<AuthorizeView>
    <Authorized>
        <div class="twelve">
            <h1>@Localizer["RECYCLE BIN"] 🗑️</h1>
        </div>

        @if (isLoading == true)
        {
            <p>@Localizer["Todos are loading..."]</p>
        }
        else if ((todos ?? []).Count == 0)
        {
            <p>@Localizer["No todos found."]</p>
        }
        else
        {
            <div>
                <table class="table">
                    <thead>
```

```
                        <tr>
                            <th>@Localizer["Todo"]</th>
                            <th>@Localizer["Description"]</th>
                            <th>@Localizer["Date"]</th>
                            <th>@Localizer["Is Complete?"]</th>
                            <th style="text-align: right;">@Localizer["Actions"
                        </tr>
                    </thead>
                    <tbody>
                        <Virtualize @ref="virtualizeComponent" Context="todo" I
                            @if (todo.IsGrouped)
                            {
                                <tr class="header-row">
                                    <td colspan="5"><strong>@todo.Title</strong
                                </tr>
                            }
                            else if (todo.IsComplete)
                            {
                                <tr>
                                    <td>@todo.Title</td>
                                    <td>@todo.Description</td>
                                    <td>@todo.Time.ToString("dd.MM.yyyy")</td>
                                    <td>@(todo.IsComplete ? "✔" : "❌")</td>
                                    <td style="text-align: right;">
                                        <button class="action-button" @onclick=
                                        <button class="action-button" @onclick=
                                        <button class="action-button" @onclick=
                                    </td>
                                </tr>
                            }
                        </Virtualize>
                    </tbody>
                </table>
            </div>
        }

        <button id="bottone6" @onclick="DeleteAllBinTodos">🔥</button>
    </Authorized>
    <NotAuthorized>
        <p>@Localizer["Unauthorized"]<a href="authentication/login">@Localizer[
    </NotAuthorized>
</AuthorizeView>

@code {
    protected override async Task OnInitializedAsync()
    {
```

```
        isCompleteFilter = true;
        await base.OnInitializedAsync();
    }
}
```

💬 Language-Switcher

Damit die Switcherfunktion auch so funktioniert wie gewünscht, habe ich einen Service `LanguageService.cs` erstellt und die Logik dafür implementiert.

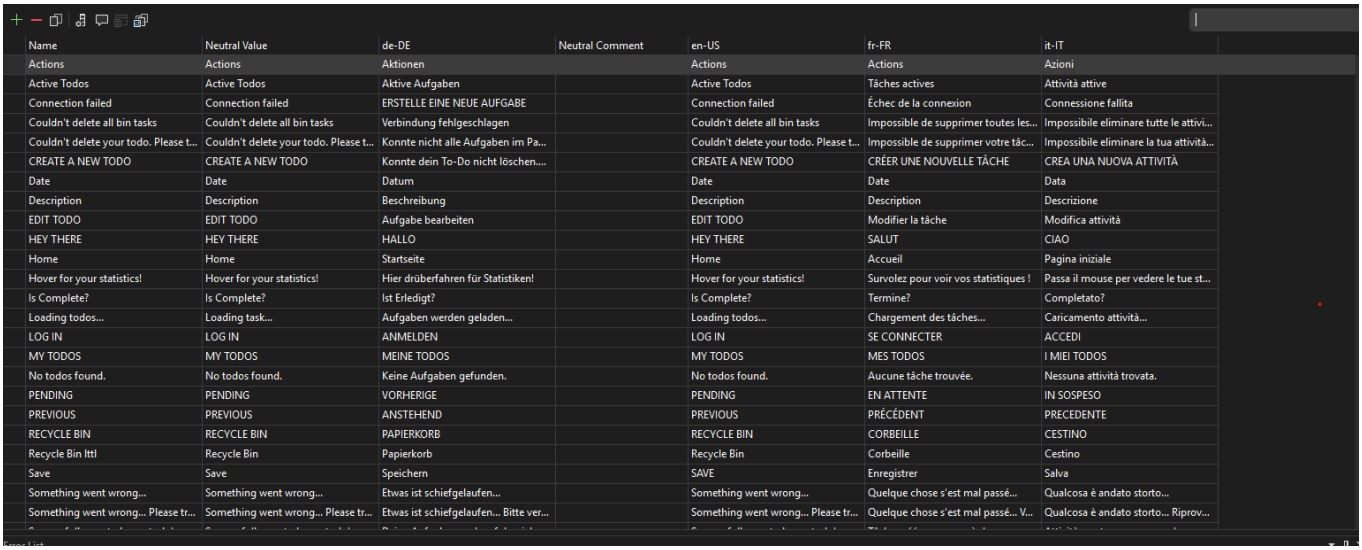
```
public class LanguageService
{
    private readonly NavigationManager _navigationManager;
    private readonly IJSRuntime _jsRuntime;

    public CultureInfo CurrentCulture { get; private set; }

    public LanguageService(NavigationManager navigationManager, IJSRuntime jsRu
    {
        _navigationManager = navigationManager;
        _jsRuntime = jsRuntime;
        CurrentCulture = CultureInfo.DefaultThreadCurrentCulture ?? new Culture
    }

    public async Task SetLanguage(string culture)
    {
        await _jsRuntime.InvokeVoidAsync("blazorCulture.set", culture);
        SetCulture(culture, true);
    }

    private void SetCulture(string culture, bool reloadPage)
    {
        var newCulture = new CultureInfo(culture);
        CultureInfo.DefaultThreadCurrentCulture = newCulture;
        CultureInfo.DefaultThreadCurrentUICulture = newCulture;
        CurrentCulture = newCulture;

        Console.WriteLine($"Sprache wurde gesetzt auf: {newCulture.Name}");

        if (reloadPage)
        {
            _navigationManager.NavigateTo(_navigationManager.Uri, forceLoad: tr
```

```
        }
    }
}
```

Damit der Wechsel der Sprache auch über die ganze Webapp funktioniert, habe ich ein get & set Script in `index.html` definiert um die aktuelle Sprache zu erkennen und bei einem Wechsel die ausgesuchte Sprache zu setzen, welches lokal im Browser mit localStorage gespeichert wird.

```
<script>
    window.blazorCulture = {
        get: () => window.localStorage['BlazorCulture'],
        set: (value) => window.localStorage['BlazorCulture'] = value
    };
</script>
```

Mit Hilfe von Globalization ist es möglich die verschiedenen Cultures bzw. Sprachen zu verwenden. Mit einer Dummy Klasse `Translations.cs` die keinen Inhalt oder auch keine Logik beinhaltet und die verschiedenen .resx für die Übersetzung, macht es das Übersetzen einfach. Für jeweils eine Sprache wird eine .resx erstellt um in einer vereinfachten Tabellenansicht die Übersetzungen einzupflegen.



Name	Neutral Value	de-DE	Neutral Comment	en-US	fr-FR	it-IT
Actions	Actions	Aktionen		Actions	Actions	Azioni
Active Todos	Active Todos	Aktive Aufgaben		Active Todos	Tâches actives	Attività attive
Connection failed	Connection failed	ERSTELLE EINE NEUE AUFGABE		Connection failed	Échec de la connexion	Connessione fallita
Couldn't delete all bin tasks	Couldn't delete all bin tasks	Verbindung fehlgeschlagen		Couldn't delete all bin tasks	Impossible de supprimer toutes les...	Impossibile eliminare tutte le attivi...
Couldn't delete your todo. Please t...	Couldn't delete your todo. Please t...	Konnte nicht alle Aufgaben im Pa...		Couldn't delete your todo. Please t...	Impossible de supprimer votre tâc...	Impossibile eliminare la tua attività...
CREATE A NEW TODO	CREATE A NEW TODO	Konnte dein To-Do nicht löschen....		CREATE A NEW TODO	CRÉER UNE NOUVELLE TÂCHE	CREA UNA NUOVA ATTIVITÀ
Date	Date	Datum		Date	Date	Data
Description	Description	Beschreibung		Description	Description	Descrizione
EDIT TODO	EDIT TODO	Aufgabe bearbeiten		EDIT TODO	Modifier la tâche	Modifica attività
HEY THERE	HEY THERE	HALLO		HEY THERE	SALUT	CIAO
Home	Home	Startseite		Home	Accueil	Pagina iniziale
Hover for your statistics!	Hover for your statistics!	Hier drüberfahren für Statistiken!		Hover for your statistics!	Survolez pour voir vos statistiques !	Passa il mouse per vedere le tue st...
Is Complete?	Is Complete?	Ist Erledigt?		Is Complete?	Termine?	Completato?
Loading todos...	Loading task...	Aufgaben werden geladen...		Loading todos...	Chargement des tâches...	Caricamento attività...
LOG IN	LOG IN	ANMELDEN		LOG IN	SE CONNECTER	ACCEDI
MY TODOS	MY TODOS	MEINE TODOS		MY TODOS	MES TODOS	I MIEI TODOS
No todos found.	No todos found.	Keine Aufgaben gefunden.		No todos found.	Aucune tâche trouvée.	Nessuna attività trovata.
PENDING	PENDING	VORHERIGE		PENDING	EN ATTENTE	IN SOSPESO
PREVIOUS	PREVIOUS	ANSTEHEND		PREVIOUS	PRÉCÉDENT	PRECEDENTE
RECYCLE BIN	RECYCLE BIN	PAPIERKORB		RECYCLE BIN	CORBEILLE	CESTINO
Recycle Bin !ttl	Recycle Bin	Papierkorb		Recycle Bin	Corbeille	Cestino
Save	Save	Speichern		SAVE	Enregistrer	Salva
Something went wrong...	Something went wrong...	Etwas ist schiefgelaufen...		Something went wrong...	Quelque chose s'est mal passé...	Qualcosa è andato storto...
Something went wrong... Please tr...	Something went wrong... Please tr...	Etwas ist schiefgelaufen... Bitte ver...		Something went wrong... Please tr...	Quelque chose s'est mal passé... V...	Qualcosa è andato storto... Riprov...

In `MainLayout.razor` (Dasselbe `index.razor`) verwenden wir in der Todo App buttons in einer custom List, um die Sprache mit einem @onClick zu wechseln und die Methode ChangeLanguage() aufzurufen.

```
<ul class="dropdown__items">
    <li><button @onclick='() => ChangeLanguage("en-US")'>English</button></li>
    <li><button @onclick='() => ChangeLanguage("de-DE")'>Deutsch</button></li>
    <li><button @onclick='() => ChangeLanguage("fr-FR")'>Français</button></li>
    <li><button @onclick='() => ChangeLanguage("it-IT")'>Italiano</button></li>
</ul>

@code {
    private void ChangeLanguage(string culture)
    {
        LanguageService.SetLanguage(culture);
        JS.InvokeVoidAsync("localStorage.setItem", "preferredLanguage", culture
    }
}
```

🚀 Lexikon

📚 Allgemeine Begriffe

🟣 Blazor WebAssembly (WASM)

Blazor WASM ist ein Clientseitiges Framework von Microsoft zur Entwicklung von Singe-Page-Webanwendungen mit C#, welcher den C#-Code direkt im Browser ausführt.

🟣 Razor-Komponenten (.razor)

Blazor-Dateien mit HTML- und C#-Code zur Erstellung von UI-Komponenten, wie zb. wie Todos.razor oder Recyclebin.razor

🟣 Dependency INjection (DI)

Entwurfsmuster, um Abhängigkeiten wie Services in Klassen zu injizieren, wie zb. @inject ITodoService TodoService in Razor-Komponenten.

🟣 Middleware

Software-Komponenten, die HTTP-Anfragen und -Antworten im Backen verarbeiten.

🟣 Entity Framework Core (EF Core)

ORM (Object-Relational Mapping) zur Datenbank-Kommunikation in C#, wie zb. Datenbankzugriff für TodoItem-Entitys.

🟣 App-Registration

Die App-Registration dient dazu, eine Anwendung in Azure AD zu registrieren. Dadurch wird der Anwendung eine Identität zugewiesen, sodass man sich Authentifizieren und Autorisieren kann und auf geschützte Ressourcen zugreifen kann. In der App-Registration werden Client-ID, Redirect-URIs und API-Berechtigungen definiert.

📘 Projektbezogene Keywords & Konzepte

🔵 Datenmodell

Eine Datenklasse, die eine einzelne Aufgabe darstellt, welche Eigenschaften eine Aufgabe hat.

🔵 RESTful API

Eine RESTful API ist eine Schnittstelle mit der zwei Systeme Informationen auf sichere Weise austauschen können. RESTful steht für "Representational State Transfer" und beschreibt einen Architekturstil für verteilte Systeme. In dem Fall wird durch eindeutige URLs Daten von einem System bei einem anderen System abgefragt und kann diese somit auch manipulieren durch HTTP CRUD-Operationen.

Methode	Endpunkt	Beschreibung
GET	/api/todo	Alle Todos Abrufen
GET	/api/todo/{id}	Todos nach ID abrufen
POST	/api/todo	Neues Todo erstellen
PUT	/api/todo/{id}	Bestehendes Todo aktualisieren
DELETE	/api/todo/{id}	Todo löschen

🔵 HttpTodoService (API-Service)

Der HttpTodoService dient zur Kommunikation mit der RESTful API, welcher die CRUD-Methoden (Create, Read, Update, Delete) beinhaltet, sodass man die TodoItems verwalten kann, welcher auf der Datenbank liegen.

🔵 MemoryService

Alternative zum HttpTodoService, der Todos nicht über die API abruft sondern im Arbeitsspeicher speichert. Dieser Service ist nützlich für verschiedene Tests ohne API.

🔵 ITodoService (Interface)

Die Schnittstelle definiert alle Todo-Operationen, welche in HttpTodoService und MemoryService implementiert werden, sobald man das Interface in den Klassen verwendet.

Virtualize-Komponente

Die Virtualize-Komponente dient für das virtuelle Scrolling. Aber was genau ist virtuelles Scrolling? - Virtualize lädt tatsächlich nur die Daten, die der Endbenutzer auch auf seinem Bildschirm sieht. Der rest wird nicht mehr geladen sein, sobald diese auch nicht mehr auf dem Bildschirm des Endbenutzers zu sehen sind, um Ressourcen zu sparen.

MudBlazor

Mit MudBlazor (UI-Komponentenbibliothek für Blazor) wird das benutzen von speziellen UI-Komponenten möglich! Beispiele dafür sind die MudSnackbars in diesem Projekt. Sobald ein Fehler auftaucht oder die API nicht reagiert, ein neues Todo erstellt wird oder Todos gelöscht werden, taucht eine Notification auf dem Bildschirm, welcher einen Fehler einblendet.

Snackbar:

```
Snackbar.Add("Verbindung zur API fehlgeschlagen!", Serverity.Error)
//Keyword             //Message                //Keyword.Keywordoption
```

```
@* Required *@
<MudThemeProvider />
<MudPopoverProvider />

@* Needed for dialogs *@
<MudDialogProvider />

@* Needed for snackbars *@
<MudSnackbarProvider />
// Diese Komponente werden in die Razorfiles eingebunden um die Notifications s
```

Authentifizierung & Autorisierung

Mit Microsoft MSAL wird ermöglicht das Login mit Microsoft-Accounts durchzuführen, welches in `Program.cs` konfiguriert wurde:

```
builder.Services.AddMsalAuthentication(options =>
{
    var conf = builder.Configuration;

    builder.Configuration.Bind("AzureAd", options.ProviderOptions.Authenticatio
    options.ProviderOptions.DefaultAccessTokenScopes.Add(conf.GetValue<string>(
```

```
    options.ProviderOptions.LoginMode = conf.GetValue<string>("Api:LoginMode");
});
```

Die Keywords AzureAd, Api:ApiAccess und Api:LoginMode wurden in `appsettings.json` definiert:

```
{
  "AzureAd": {
    "Authority": "https://login.microsoftonline.com/6ee5ce8d-f608-4339-b50f-39b
    "ClientId": "7d1c2066-7544-4f5a-8a6a-f35e5a82b4b3",
    "ValidateAuthority": true,
    "RedirectUri": "https://localhost:7032/authentication/login-callback",
    "PostLogoutRedirectUri": "https://localhost:7032"
  },
  "Api": {
    "BaseUrl": "http://localhost:5085/api/",
    "ApiAccess": "api://6f1ef8a4-18ee-4983-9d4e-b44abe56f234/api_access",
    "LoginMode": "redirect"
  }, ...
}
```

- ***Authority** ist die URL des Identitätsanbieters, also der Azure AD-Mandanten, an den sich die Webapplikation wendet um die Authentifizierung durchzuführen.*
- ***ClientId** Ist die eindeutige Kennung der Anwendung in Azure AD. Damit wird die Webapp identifiziert.*
- ***ValidateAuthority** gibt an, ob die Gültigkeit der angefragten Authority-URLs überprüft werden soll.*
- ***RedirectUri** ist die URL, an die Nutzer nach einer erfolreichen Anmeldung weitergeleitet werden.*
- ***PostLogoutRedirectUri** ist die URL, an die Nutzer nach dem Abmelden zurückgeleitet werden.*
- ***BaseUrl** beinhaltet die URL, unter der die API erreichbar ist*
- ***APIAccess** definiert den Zugriffsbereich, auch genannt Scope, für die API. - Die ist wichtig um Kontrollen für Berechtigungen der Webapp durchzuführen.*
- ***LoginMode** legt fest, wie der Anmeldeprozess angezeigt wird. Ob ein Popup-Fenster(popup) aufgeht oder zu der Anmeldeseite im selben Tag weitergeleitet wird (redirect).*

In Razor-Komponenten ist AuthorizeView öfters zu sehen. Mit AuthorizeView zeigt es jeweils ob der Benutzer angemeldet ist oder noch als "Gast" die Webapp öffnet die richtige Ansicht an und

bietet eine Grundsicherheit an um nicht ohne Login auf andere Seiten zuzugreifen.
`Index.razor`:

```
<AuthorizeView>
    <Authorized>
        <p>You allready are logged in! Redirecting...</p>
    </Authorized>
    <NotAuthorized>
        <div class="background"></div>
        <div class="card">
            <img class="logo" src="https://i.imgur.com/AocAEOJ.png" alt="Logo">
            <h2 class="welcome">TO DO APP</h2>
            <h6 class="welcome">@Localizer["Track your tasks."]</h6>
            <form class="form">
                <button @onclick="NavigateToLogin">@Localizer["LOG IN"]</button
            </form>
        </div>
        <div class="footer">
            <footer>
                <div class="text">
                    @@ibrazqrj
                </div>
                <img class="logo" src="https://i.imgur.com/XzvD7dv.png" alt="ib
            </footer>
        </div>
        <div class="languagePosition">
        <div class="dropdown">
            <input type="checkbox" id="dropdown">
            <label class="dropdown__face" for="dropdown">
                <div class="dropdown__icon">
                    <svg fill="#000000" viewBox="-1 0 19 19" xmlns="http://www.
                        <path d="M16.417 9.57a7.917 7.917 0 1 1-8.144-7.908 1.7
                    </svg>
                </div>
            </label>
            <ul class="dropdown__items">
                    <li><button @onclick='async () => ChangeLanguage("en-US")'>
                    <li><button @onclick='async () => ChangeLanguage("de-DE")'>
                    <li><button @onclick='async () => ChangeLanguage("fr-FR")'>
                    <li><button @onclick='async () => ChangeLanguage("it-IT")'>
            </ul>
            </div>
        </div>
    </NotAuthorized>
</AuthorizeView>
```

👥 Contributors

Ibrahim Zeqiraj

Projekt-Owner & Developer
GitHub