BLAZOR APP PAGINATION CLIENT

HOST.CSHTML

@page "/"

@namespace BlazorAppPaginationClient

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

@{

Layout = null;

}

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>BlazorAppPaginationClient</title>

<**base** href="~/" />

<link rel="stylesheet" href="css/bootstrap/bootstrap.min.css" />

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

<link href="BlazorAppPaginationClient.styles.css" rel="stylesheet" />

<!-- Font Awesome CDN -->

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-beta3/css/all.min.css">

<script src="https://cdnjs.cloudflare.com/ajax/libs/jspdf/2.5.1/jspdf.umd.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/jspdf-autotable/3.5.25/jspdf.plugin.autotable.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/xlsx/0.17.3/xlsx.full.min.js"></script>

<script src="js/site.js"></script>

<script src="\_framework/blazor.server.js"></script>

</head>

<body>

<**component** **type**="typeof(App)" **render-mode**="ServerPrerendered" />

<div id="blazor-error-ui">

<**environment** **include**="Staging,Production">

An error has occurred. This application may no longer respond until reloaded.

</**environment**>

<**environment** **include**="Development">

An unhandled exception has occurred. See browser dev tools for details.

</**environment**>

<a href="" class="reload">Reload</a>

<a class="dismiss">🗙</a>

</div>

<script>

// Функција за извоз на табела во хоризонтален PDF

function exportTableToPdf() {

const { jsPDF } = window.jspdf;

const doc = new jsPDF('landscape'); // Поставување ориентација на страницата на "landscape"

// Додавање наслов или друга содржина

doc.text("Taxi Rides List", 14, 16); // Може да го промениш на што ти треба

// Конвертирање на HTML табела во PDF

doc.autoTable({

html: '.table', // Променете го селекторот за табелата ако е потребно

startY: 30, // Поставете го почетниот Y координат на табелата

margin: { top: 20 }, // Поставете маргини ако треба

theme: 'grid', // Визуелна тема за табелата

columnStyles: {

0: { cellWidth: 10 },

1: { cellWidth: 20 },

2: { cellWidth: 20 },

3: { cellWidth: 20 },

6: { cellWidth: 20 },

5: { cellWidth: 20 },

4: { cellWidth: 20 },

7: { cellWidth: 20 },

8: { cellWidth: 20 },

9: { cellWidth: 20 },

10: { cellWidth: 20 },

11: { cellWidth: 20 },

12: { cellWidth: 20 },

13: { cellWidth: 20 },

14: { cellWidth: 20 },

15: { cellWidth: 20 },

16: { cellWidth: 20 }

}

});

// Преземање на генерираното PDF

doc.save('taxi\_rides\_list.pdf');

}

// Функција за извоз во Excel

function exportToExcel(paginatedData, filename = 'taxi\_rides.xlsx') {

if (!paginatedData || paginatedData.length === 0) {

console.error("No data to export");

return;

}

// Претвори го податокот во Excel формат

const ws = XLSX.utils.json\_to\_sheet(paginatedData);

const wb = XLSX.utils.book\_new(); // Креирај нов Excel документ

XLSX.utils.book\_append\_sheet(wb, ws, 'Taxi Rides'); // Додади го листот со податоци

XLSX.writeFile(wb, filename); // Преземи го Excel фајлот

}

// Функција за извоз во CSV

function exportToCsv(data, filename = 'taxi\_rides.csv') {

if (!data || data.length === 0) {

alert("No data to export.");

return;

}

// Задавање на хедер на CSV фајлот

const header = ['medallion', 'hashLicense', 'pickupTime', 'dropOffTime', 'duration', 'distance', 'pLongitude', 'pLatitude', 'dLongitude', 'dLatitude', 'paymentType', 'fareAmount', 'surcharge', 'tax', 'tipAmount', 'tollsAmount', 'totalAmount'];

// Преобразување на податоците во низи (row data)

const rows = data.map(item => [

item.medallion,

item.hashLicense,

item.pickupTime,

item.dropOffTime,

item.duration,

item.distance,

item.pLongitude,

item.pLatitude,

item.dLongitude,

item.dLatitude,

item.paymentType,

item.fareAmount,

item.surcharge,

item.tax,

item.tipAmount,

item.tollsAmount,

item.totalAmount

]);

// Се додава хедерот како прв ред во CSV

const csvContent = [header, ...rows].map(e => e.join(",")).join("\n");

// Креирање на Blob и преземање на CSV фајл

const blob = new Blob([csvContent], { type: 'text/csv;charset=utf-8;' });

const link = document.createElement("a");

if (link.download !== undefined) { // Feature detection for download attribute

const url = URL.createObjectURL(blob);

link.setAttribute("href", url);

link.setAttribute("download", filename);

document.body.appendChild(link);

link.click();

document.body.removeChild(link);

}

}

</script>

</body>

</html>

APP.RAZOR

<**Router** **AppAssembly**="@typeof(App).Assembly">

<**Found** **Context**="routeData">

<**RouteView** **RouteData**="@routeData" **DefaultLayout**="@typeof(MainLayout)" />

<**FocusOnNavigate** **RouteData**="@routeData" **Selector**="h1" />

</**Found**>

<**NotFound**>

<**PageTitle**>Not found</**PageTitle**>

<**LayoutView** **Layout**="@typeof(MainLayout)">

<p role="alert">Sorry, there's nothing at this address.</p>

</**LayoutView**>

</**NotFound**>

</**Router**>

SITE.JS

// Функција за извоз на CSV

function saveAsFile(filename, content) {

console.log('Starting CSV export');

var blob = new Blob([content], { type: 'text/csv;charset=utf-8;' });

var link = document.createElement("a");

link.href = URL.createObjectURL(blob);

link.download = filename;

link.click();

console.log('CSV export complete');

}

// Функција за извоз на Excel

function saveExcel(filename, data) {

console.log('Starting Excel export');

var ws = XLSX.utils.aoa\_to\_sheet(data);

var wb = XLSX.utils.book\_new();

XLSX.utils.book\_append\_sheet(wb, ws, "Taxi Rides");

XLSX.writeFile(wb, filename);

console.log('Excel export complete');

}

// Функција за извоз на PDF

function savePdf(filename, content) {

console.log('Starting PDF export');

const { jsPDF } = window.jspdf;

const doc = new jsPDF();

doc.text(content, 10, 10);

doc.save(filename);

console.log('PDF export complete');

}

LAUNCHSETTINGS.JSON

{

"iisSettings": {

"windowsAuthentication": false,

"anonymousAuthentication": true,

"iisExpress": {

"applicationUrl": "http://localhost:47748"

}

},

"profiles": {

"BlazorAppPaginationClient": {

"commandName": "Project",

"dotnetRunMessages": true,

"launchBrowser": true,

"applicationUrl": "http://localhost:5204",

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

}

},

"IIS Express": {

"commandName": "IISExpress",

"launchBrowser": true,

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

}

}

}

}

APPSETTINGS.JSON

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*"

}

NAVMENU.RAZOR

<div class="top-row ps-3 navbar navbar-dark">

<div class="container-fluid">

<a class="navbar-brand" href="">BlazorAppPaginationClient</a>

<button title="Navigation menu" class="navbar-toggler" @onclick="ToggleNavMenu">

<span class="navbar-toggler-icon"></span>

</button>

</div>

</div>

<div class="@NavMenuCssClass" @onclick="ToggleNavMenu">

<nav class="flex-column">

<!-- Овој линк ќе го води до вашата страница "PaginatedData.razor" -->

<div class="nav-item px-3">

<**NavLink** class="nav-link" href="taxiridesapp">

<span class="oi oi-list-rich" aria-hidden="true"></span> Taxi Rides

</**NavLink**>

</div>

</nav>

</div>

@code {

private bool collapseNavMenu = true;

private string? NavMenuCssClass => collapseNavMenu ? "collapse" : null;

private void ToggleNavMenu()

{

collapseNavMenu = !collapseNavMenu;

}

}

PAGINATEDDATA.RAZOR

@page "/taxiridesapp"

@using BlazorAppPaginationClient.Models

@using System.Text

@inject IJSRuntime JSRuntime

@inject HttpClient Http

<h3>Taxi Rides</h3>

<div class="controls-container">

<div class="search-box">

<label>Search: </label>

<input type="text" @bind="SearchText" @bind:event="oninput" placeholder="Search..." class="search-input" />

</div>

<div class="page-size-selector">

<label>Page Size: </label>

<select @bind="PageSize" @bind:event="onchange" class="dropdown">

<option value="10">10</option>

<option value="20">20</option>

<option value="50">50</option>

<option value="100">100</option>

</select>

</div>

<div class="total-records">

<strong>Total Records: @FilteredRecords</strong>

</div>

</div>

@if (TaxiRecords == null)

{

<p>Loading...</p>

}

else

{

<table class="table">

<thead>

<tr>

<th>#</th>

@foreach (var property in typeof(TaxiRecord).GetProperties())

{

<th @onclick="() => SortByColumn(property.Name)" class="sortable">

@property.Name @GetSortIndicator(property.Name)

</th>

}

</tr>

</thead>

<tbody>

@foreach (var (ride, index) in PaginatedRides.Select((r, i) => (r, i + 1 + ((\_currentPage - 1) \* PageSize))))

{

<tr>

<td>@index</td>

@foreach (var property in typeof(TaxiRecord).GetProperties())

{

<td>@property.GetValue(ride)</td>

}

</tr>

}

</tbody>

</table>

<div class="pagination-container">

<button @onclick="PreviousPage" disabled="@(\_currentPage == 1)" class="pagination-button">Previous</button>

<span>Page @\_currentPage of @TotalPages</span>

<button @onclick="NextPage" disabled="@(\_currentPage >= TotalPages)" class="pagination-button">Next</button>

</div>

<div class="page-input-container">

<label>Enter Page: </label>

<input type="number" @bind="EnteredPage" class="page-input" />

<button @onclick="GoToPage" class="pagination-button">Go</button>

</div>

<div class="export-buttons">

<button class="btn btn-primary" @onclick="ExportToCsv">Export to CSV</button>

<button class="btn btn-success" @onclick="ExportToExcel">Export to Excel</button>

<button class="btn btn-danger" @onclick="ExportToPdf">Export to PDF</button>

</div>

}

<style>

.controls-container {

display: flex;

justify-content: space-between;

align-items: center;

margin-bottom: 15px;

}

.search-input, .dropdown, .page-input {

padding: 5px;

border-radius: 5px;

border: 1px solid #ccc;

}

.table {

width: 100%;

border-collapse: collapse;

margin-top: 10px;

}

.table th, .table td {

border: 1px solid #ddd;

padding: 8px;

text-align: left;

}

.sortable:hover {

cursor: pointer;

background-color: #f1f1f1;

}

.pagination-container, .page-input-container {

display: flex;

justify-content: center;

align-items: center;

margin-top: 10px;

gap: 10px;

}

.pagination-button {

padding: 5px 10px;

border: none;

background-color: #007bff;

color: white;

border-radius: 5px;

cursor: pointer;

}

.pagination-button:disabled {

background-color: #ccc;

cursor: not-allowed;

}

.export-buttons {

margin-top: 15px;

text-align: center;

}

</style>

@code {

private List<TaxiRecord> TaxiRecords = new();

private string SearchText = "";

private int \_currentPage = 1;

private int PageSize = 10;

private int EnteredPage;

private int TotalRecords = 0;

private int FilteredRecords => FilteredRides.Count;

private int TotalPages => (int)Math.Ceiling((double)FilteredRecords / PageSize);

private string SortColumn = nameof(TaxiRecord.pickupTime);

private bool SortAscending = true;

private List<TaxiRecord> FilteredRides =>

string.IsNullOrWhiteSpace(SearchText)

? TaxiRecords

: TaxiRecords.Where(r => typeof(TaxiRecord).GetProperties()

.Any(p => p.GetValue(r)?.ToString()?.Contains(SearchText, StringComparison.OrdinalIgnoreCase) == true)).ToList();

private List<TaxiRecord> PaginatedRides =>

FilteredRides

.OrderBy(r => SortAscending ? GetPropertyValue(r, SortColumn) : null)

.ThenByDescending(r => SortAscending ? null : GetPropertyValue(r, SortColumn))

.Skip((\_currentPage - 1) \* PageSize)

.Take(PageSize)

.ToList();

protected override async Task OnInitializedAsync()

{

try

{

// Get the data from your API or any other source

TaxiRecords = await Http.GetFromJsonAsync<List<TaxiRecord>>("http://localhost:5098/api/TaxiRides/GetTaxiRides2");

TotalRecords = TaxiRecords.Count;

}

catch (Exception ex)

{

Console.WriteLine($"Error: {ex.Message}");

}

}

private void NextPage()

{

if (\_currentPage < TotalPages) \_currentPage++;

}

private void PreviousPage()

{

if (\_currentPage > 1) \_currentPage--;

}

private void GoToPage()

{

if (EnteredPage >= 1 && EnteredPage <= TotalPages)

{

\_currentPage = EnteredPage;

}

}

private void SortByColumn(string columnName)

{

if (SortColumn == columnName)

{

SortAscending = !SortAscending;

}

else

{

SortColumn = columnName;

SortAscending = true;

}

}

private string GetSortIndicator(string columnName)

{

if (SortColumn == columnName)

{

return SortAscending ? "▲" : "▼";

}

return "";

}

private object GetPropertyValue(TaxiRecord ride, string propertyName)

{

var property = typeof(TaxiRecord).GetProperty(propertyName);

return property?.GetValue(ride, null);

}

// Export functionality (CSV, Excel, PDF)

private async Task ExportToCsv()

{

var csvContent = GenerateCsvContent(PaginatedRides);

await JSRuntime.InvokeVoidAsync("saveAsFile", "taxi\_rides.csv", csvContent);

}

private async Task ExportToExcel()

{

var excelContent = GenerateExcelContent(PaginatedRides);

await JSRuntime.InvokeVoidAsync("saveExcel", "taxi\_rides.xlsx", excelContent);

}

private async Task ExportToPdf()

{

var pdfContent = GeneratePdfContent(PaginatedRides);

await JSRuntime.InvokeVoidAsync("savePdf", "taxi\_rides.pdf", pdfContent);

}

private string GenerateCsvContent(List<TaxiRecord> data)

{

var sb = new StringBuilder();

sb.AppendLine("Medallion,Hash License,Pickup Time,Drop-off Time,Duration,Distance,Pickup Longitude,Pickup Latitude,Drop-off Longitude,Drop-off Latitude,Payment Type,Fare Amount,Surcharge,Tax,Tip Amount,Tolls Amount,Total Amount");

foreach (var item in data)

{

sb.AppendLine($"{item.medallion},{item.hashLicense},{item.pickupTime},{item.dropOffTime},{item.duration},{item.distance},{item.pLongitude},{item.pLatitude},{item.dLongitude},{item.dLatitude},{item.paymentType},{item.fareAmount},{item.surcharge},{item.tax},{item.tipAmount},{item.tollsAmount},{item.totalAmount}");

}

return sb.ToString();

}

private object GenerateExcelContent(List<TaxiRecord> data)

{

var sheet = new List<List<object>> {

new List<object> { "Medallion", "Hash License", "Pickup Time", "Drop-off Time", "Duration", "Distance", "Pickup Longitude", "Pickup Latitude", "Drop-off Longitude", "Drop-off Latitude", "Payment Type", "Fare Amount", "Surcharge", "Tax", "Tip Amount", "Tolls Amount", "Total Amount" }

};

foreach (var item in data)

{

sheet.Add(new List<object> { item.medallion, item.hashLicense, item.pickupTime, item.dropOffTime, item.duration, item.distance, item.pLongitude, item.pLatitude, item.dLongitude, item.dLatitude, item.paymentType, item.fareAmount, item.surcharge, item.tax, item.tipAmount, item.tollsAmount, item.totalAmount });

}

return sheet;

}

private string GeneratePdfContent(List<TaxiRecord> data)

{

var content = "Taxi Ride Data:\n\n";

foreach (var item in data)

{

content += $"Medallion: {item.medallion}, Hash License: {item.hashLicense}, Pickup: {item.pickupTime}, Drop-off: {item.dropOffTime}, Total Amount: {item.totalAmount}\n";

}

return content;

}

}

PROGRAM.CS

var builder = WebApplication.CreateBuilder(args);

// Додавање на Razor Pages и Blazor Server

builder.Services.AddRazorPages();

builder.Services.AddServerSideBlazor(); // Essential for Blazor Server apps

//builder.Services.AddSingleton<PdfExportService>();

// Додавање на HttpClient за користење во Blazor компонентите

builder.Services.AddScoped<HttpClient>(sp =>

new HttpClient { BaseAddress = new Uri("http://localhost:5098") } // Постави ја твојата API база адреса тука

);

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

else

{

app.UseExceptionHandler("/Error");

app.UseHsts();

}

app.UseStaticFiles();

app.UseRouting();

app.MapBlazorHub();

app.MapFallbackToPage("/\_Host"); // This should be added

app.Run();

TAXIRECORD.CS

namespace BlazorAppPaginationClient.Models

{

public class TaxiRecord

{

public string medallion { get; set; } = string.Empty; // Број на лиценцата

public string hashLicense { get; set; } = string.Empty; // Хеш од лицето + лиценцата

public DateTime pickupTime { get; set; } // Време на подигнување

public DateTime dropOffTime { get; set; } // Време на симнување

public int duration { get; set; } // Времетраење во секунди

public float distance { get; set; } // Растојание во милји

public float pLongitude { get; set; } // Географска должина на појдовната точка

public float pLatitude { get; set; } // Географска ширина на појдовната точка

public float dLongitude { get; set; } // Географска должина на дојдовната точка

public float dLatitude { get; set; } // Географска ширина на дојдовната точка

public string paymentType { get; set; } = string.Empty; // Тип на плаќање (кеш, картичка, не познато)

public float fareAmount { get; set; } // Основа цена во USD

public float surcharge { get; set; } // Доплата во USD

public float tax { get; set; } // Данок во USD

public float tipAmount { get; set; } // Бакшиш во USD

public float tollsAmount { get; set; } // Такси за патишта во USD

public float totalAmount { get; set; } // Вкупен износ во USD

}

}