Ziel

Ein lauffähiger Prototyp als **ASP.NET Core 8 WebApp** mit **.NET Backend** und **MSSQL (Docker)**, um **Veranstaltungen anzulegen** und daraus **JSON-Dateien im gewünschten Format** zu exportieren.

Features

- Veranstaltungen erstellen, bearbeiten, löschen
- Übersichtsliste mit Export-Buttons
- Export einzelner Events oder aller Events als JSON (Download)
- Einfach anpassbares **Export-Schema** (zentrale Stelle)
- Persistenz via **EF Core** auf **SQL Server** (läuft in Docker)

Projektstruktur (Vorschlag)

```
DnaAustria.Events/
├─ DnaAustria.Events.csproj
├─ appsettings.json
├─ Program.cs
├─ Controllers/
│  └─ EventsController.cs
├─ Data/
│  └─ AppDbContext.cs
├─ Models/
│  └─ Event.cs
├─ Services/
│  └─ EventExportService.cs
├─ Pages/
│  ├─ _ViewImports.cshtml
│  ├─ _ViewStart.cshtml
│  ├─ Index.cshtml
│  └─ Events/
│     ├─ Index.cshtml
│     ├─ Create.cshtml
│     ├─ Edit.cshtml
│     └─ Delete.cshtml
├─ wwwroot/
│  ├─ css/site.css
│  └─ js/events.js
└─ docker-compose.yml (im Repo-Wurzelordner)
```

> Hinweis: Für einen schnellen Start sind Razor Pages gewählt (kein separates Frontend-Build nötig). Falls gewünscht, kann man später auf React/Angular umsteigen – das Backend bleibt gleich.

Docker: MSSQL starten

docker-compose.yml (im Repo-Wurzelordner):

```
version: "3.9"
services:
  mssql:
    image: mcr.microsoft.com/mssql/server:2022-latest
    container_name: dna_mssql
    environment:
      - ACCEPT_EULA=Y
      - MSSQL_SA_PASSWORD=YourStrong!Passw0rd
    ports:
      - "1433:1433"
    volumes:
      - mssql_data:/var/opt/mssql
    healthcheck:
      test: ["CMD", "/opt/mssql-tools/bin/sqlcmd", "-S", "localhost", "-U",
"sa", "-P", "YourStrong!Passw0rd", "-Q", "SELECT 1"]
      interval: 10s
      timeout: 5s
      retries: 10
volumes:
  mssql_data:
```

Start:

```
docker compose up -d
```

Projektdateien

1) `DnaAustria.Events.csproj`

```
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>net8.0</TargetFramework>
```

```
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Microsoft.EntityFrameworkCore" Version="8.0.5" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer"
Version="8.0.5" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Tools"
Version="8.0.5">
      <PrivateAssets>all</PrivateAssets>
      <IncludeAssets>runtime; build; native; contentfiles; analyzers;
buildtransitive</IncludeAssets>
    </PackageReference>
  </ItemGroup>
</Project>
```

2) `appsettings.json`

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=localhost,1433;Database=DnaAustriaEvents;User
Id=sa;Password=YourStrong!Passw0rd;TrustServerCertificate=True;"
  },
  "Export": {
    "Provider": "Default"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

3) `Models/Event.cs`

```
using System.ComponentModel.DataAnnotations;

namespace DnaAustria.Events.Models;

public class Event
{
    public int Id { get; set; }
```

```
    [Required, MaxLength(200)]
    public string Title { get; set; } = string.Empty;

    [MaxLength(2000)]
    public string? Description { get; set; }

    [Required]
    public DateTime StartUtc { get; set; }

    [Required]
    public DateTime EndUtc { get; set; }

    [MaxLength(200)]
    public string? LocationName { get; set; }

    [MaxLength(200)]
    public string? City { get; set; }

    [MaxLength(100)]
    public string? Organizer { get; set; }

    [MaxLength(200)]
    public string? TargetAudience { get; set; }

    [MaxLength(400)]
    public string? TagsCsv { get; set; }
}
```

4) `Data/AppDbContext.cs`

```
using DnaAustria.Events.Models;
using Microsoft.EntityFrameworkCore;

namespace DnaAustria.Events.Data;

public class AppDbContext(DbContextOptions<AppDbContext> options) :
DbContext(options)
{
    public DbSet<Event> Events => Set<Event>();
}
```

5) `Services/EventExportService.cs`

```
using DnaAustria.Events.Models;
using System.Text.Json;
```

```
using System.Text.Json.Serialization;

namespace DnaAustria.Events.Services;

public class EventExportService
{
    private static readonly JsonSerializerOptions JsonOptions = new()
    {
        WriteIndented = true,
        DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull
    };

    // Hier das gewünschte Zielschema anpassen
    // Aktuelles Default-Format (Beispiel für Entdecke.DNAustria – bitte bei
Bedarf Felder/Keys anpassen)
    public object MapToExport(Event e) => new
    {
        id = e.Id,
        title = e.Title,
        summary = e.Description,
        start = e.StartUtc.ToString("o"),
        end = e.EndUtc.ToString("o"),
        location = new { name = e.LocationName, city = e.City },
        organizer = e.Organizer,
        target_audience = e.TargetAudience,
        tags = (e.TagsCsv ?? string.Empty)
            .Split(',', StringSplitOptions.RemoveEmptyEntries |
StringSplitOptions.TrimEntries)
    };

    public string ToJson(Event e)
        => JsonSerializer.Serialize(MapToExport(e), JsonOptions);

    public string ToJson(IEnumerable<Event> events)
        => JsonSerializer.Serialize(events.Select(MapToExport), JsonOptions);
}
```

6) `Controllers/EventsController.cs`

```
using DnaAustria.Events.Data;
using DnaAustria.Events.Models;
using DnaAustria.Events.Services;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace DnaAustria.Events.Controllers;
```

```
[ApiController]
[Route("api/[controller]")]
public class EventsController(AppDbContext db, EventExportService exporter) :
ControllerBase
{
    [HttpGet]
    public async Task<ActionResult<IEnumerable<Event>>> GetAll()
        => await db.Events.OrderBy(e => e.StartUtc).ToListAsync();

    [HttpGet("{id:int}")]
    public async Task<ActionResult<Event>> GetById(int id)
        => await db.Events.FindAsync(id) is { } e ? e : NotFound();

    [HttpPost]
    public async Task<ActionResult<Event>> Create(Event input)
    {
        if (input.EndUtc <= input.StartUtc)
            return BadRequest("Ende muss nach Beginn liegen.");

        db.Events.Add(input);
        await db.SaveChangesAsync();
        return CreatedAtAction(nameof(GetById), new { id = input.Id }, input);
    }

    [HttpPut("{id:int}")]
    public async Task<IActionResult> Update(int id, Event input)
    {
        if (id != input.Id) return BadRequest();
        if (!await db.Events.AnyAsync(e => e.Id == id)) return NotFound();
        if (input.EndUtc <= input.StartUtc) return BadRequest("Ende muss nach
Beginn liegen.");

        db.Entry(input).State = EntityState.Modified;
        await db.SaveChangesAsync();
        return NoContent();
    }

    [HttpDelete("{id:int}")]
    public async Task<IActionResult> Delete(int id)
    {
        var e = await db.Events.FindAsync(id);
        if (e is null) return NotFound();
        db.Events.Remove(e);
        await db.SaveChangesAsync();
        return NoContent();
    }
```

```
    // EXPORT: Einzelnes Event als JSON-Datei
    [HttpGet("{id:int}/export")]
    public async Task<IActionResult> ExportOne(int id)
    {
        var e = await db.Events.FindAsync(id);
        if (e is null) return NotFound();
        var json = exporter.ToJson(e);
        return File(System.Text.Encoding.UTF8.GetBytes(json), "application/
json", $"event_{id}.json");
    }

    // EXPORT: Alle Events als JSON-Datei
    [HttpGet("export")]
    public async Task<IActionResult> ExportAll()
    {
        var all = await db.Events.OrderBy(e => e.StartUtc).ToListAsync();
        var json = exporter.ToJson(all);
        return File(System.Text.Encoding.UTF8.GetBytes(json), "application/
json", "events.json");
    }
}
```

7) `Program.cs`

```
using DnaAustria.Events.Data;
using DnaAustria.Events.Services;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddRazorPages();
builder.Services.AddControllers();

builder.Services.AddDbContext<AppDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

builder.Services.AddScoped<EventExportService>();

var app = builder.Build();

if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Error");
    app.UseHsts();
}
```

```
app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.MapControllers();
app.MapRazorPages();

app.Run();
```

8) Razor Pages – UI

`Pages/_ViewImports.cshtml`

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

`Pages/_ViewStart.cshtml`

```
@{
    Layout = null;
}
```

`Pages/Index.cshtml`

```
@page
@{
    Response.Redirect("/Events");
}
```

`Pages/Events/Index.cshtml`

```
@page
@model EventsIndexModel
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>Veranstaltungen</title>
```

```
  <link rel="stylesheet" href="/css/site.css" />
</head>
<body>
  <main class="container">
    <header>
      <h1>Veranstaltungen</h1>
      <a class="btn" href="/Events/Create">Neu anlegen</a>
      <a class="btn secondary" href="/api/Events/export">Alle als JSON</a>
    </header>

    <table>
      <thead>
        <tr>
          <th>Titel</th>
          <th>Start</th>
          <th>Ende</th>
          <th>Ort</th>
          <th>Aktionen</th>
        </tr>
      </thead>
      <tbody>
      @foreach (var e in Model.Items)
      {
        <tr>
          <td>@e.Title</td>
          <td>@e.StartUtc.ToLocalTime().ToString("yyyy-MM-dd HH:mm")</td>
          <td>@e.EndUtc.ToLocalTime().ToString("yyyy-MM-dd HH:mm")</td>
          <td>@e.LocationName</td>
          <td>
            <a class="btn small" href="/Events/Edit?id=@e.Id">Bearbeiten</a>
            <a class="btn small danger" href="/Events/Delete?id=@e.Id">Löschen</
a>
            <a class="btn small secondary" href="/api/Events/@e.Id/
export">JSON</a>
          </td>
        </tr>
      }
      </tbody>
    </table>
  </main>
</body>
</html>
```

Pages/Events/Create.cshtml

```
@page
@model CreateEventModel
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>Event anlegen</title>
  <link rel="stylesheet" href="/css/site.css" />
</head>
<body>
  <main class="container">
    <h1>Event anlegen</h1>
    <form method="post">
      <label>Titel<input asp-for="Item.Title" required maxlength="200" /></
label>
      <label>Beschreibung<textarea asp-for="Item.Description"
maxlength="2000"></textarea></label>
      <div class="grid">
        <label>Start (UTC)<input asp-for="Item.StartUtc" type="datetime-local"
required /></label>
        <label>Ende (UTC)<input asp-for="Item.EndUtc" type="datetime-local"
required /></label>
      </div>
      <div class="grid">
        <label>Ort<input asp-for="Item.LocationName" maxlength="200" /></label>
        <label>Stadt<input asp-for="Item.City" maxlength="200" /></label>
      </div>
      <div class="grid">
        <label>Veranstalter<input asp-for="Item.Organizer" maxlength="100" /></
label>
        <label>Zielgruppe<input asp-for="Item.TargetAudience" maxlength="200" /
></label>
      </div>
      <label>Tags (Komma-getrennt)<input asp-for="Item.TagsCsv"
maxlength="400" /></label>
      <div class="actions">
        <button class="btn" type="submit">Speichern</button>
        <a class="btn secondary" href="/Events">Abbrechen</a>
      </div>
    </form>
  </main>
```

```
</body>
</html>
```

`Pages/Events/Edit.cshtml`

```
@page
@model EditEventModel
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>Event bearbeiten</title>
  <link rel="stylesheet" href="/css/site.css" />
</head>
<body>
  <main class="container">
    <h1>Event bearbeiten</h1>
    <form method="post">
      <input type="hidden" asp-for="Item.Id" />
      <label>Titel<input asp-for="Item.Title" required maxlength="200" /></
label>
      <label>Beschreibung<textarea asp-for="Item.Description"
maxlength="2000"></textarea></label>
      <div class="grid">
        <label>Start (UTC)<input asp-for="Item.StartUtc" type="datetime-local"
required /></label>
        <label>Ende (UTC)<input asp-for="Item.EndUtc" type="datetime-local"
required /></label>
      </div>
      <div class="grid">
        <label>Ort<input asp-for="Item.LocationName" maxlength="200" /></label>
        <label>Stadt<input asp-for="Item.City" maxlength="200" /></label>
      </div>
      <div class="grid">
        <label>Veranstalter<input asp-for="Item.Organizer" maxlength="100" /></
label>
        <label>Zielgruppe<input asp-for="Item.TargetAudience" maxlength="200" /
></label>
      </div>
      <label>Tags (Komma-getrennt)<input asp-for="Item.TagsCsv"
maxlength="400" /></label>
      <div class="actions">
```

```
        <button class="btn" type="submit">Speichern</button>
        <a class="btn secondary" href="/Events">Abbrechen</a>
      </div>
    </form>
  </main>
</body>
</html>
```

`Pages/Events/Delete.cshtml`

```
@page
@model DeleteEventModel
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>Event löschen</title>
  <link rel="stylesheet" href="/css/site.css" />
</head>
<body>
  <main class="container">
    <h1>Event löschen</h1>
    <form method="post">
      <input type="hidden" asp-for="Item.Id" />
      <p>Möchtest du <strong>@Model.Item.Title</strong> wirklich löschen?</p>
      <div class="actions">
        <button class="btn danger" type="submit">Löschen</button>
        <a class="btn secondary" href="/Events">Abbrechen</a>
      </div>
    </form>
  </main>
</body>
</html>
```

PageModels `Pages/Events/*.cshtml.cs`

```
using DnaAustria.Events.Data;
using DnaAustria.Events.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;
```

```
namespace DnaAustria.Events.Pages.Events;

public class EventsIndexModel(AppDbContext db) : PageModel
{
    public List<Event> Items { get; set; } = [];
    public async Task OnGet() => Items = await db.Events.OrderBy(e =>
e.StartUtc).ToListAsync();
}

public class CreateEventModel(AppDbContext db) : PageModel
{
    [BindProperty] public Event Item { get; set; } = new();
    public async Task<IActionResult> OnPostAsync()
    {
        if (!ModelState.IsValid) return Page();
        if (Item.EndUtc <= Item.StartUtc)
        {
            ModelState.AddModelError(string.Empty, "Ende muss nach Beginn
liegen.");
            return Page();
        }
        db.Events.Add(Item);
        await db.SaveChangesAsync();
        return RedirectToPage("/Events/Index");
    }
}

public class EditEventModel(AppDbContext db) : PageModel
{
    [BindProperty] public Event Item { get; set; } = new();
    public async Task OnGetAsync(int id)
    {
        Item = await db.Events.FindAsync(id) ?? new Event();
    }
    public async Task<IActionResult> OnPostAsync()
    {
        if (!ModelState.IsValid) return Page();
        if (Item.EndUtc <= Item.StartUtc)
        {
            ModelState.AddModelError(string.Empty, "Ende muss nach Beginn
liegen.");
            return Page();
        }
        db.Attach(Item).State = EntityState.Modified;
        await db.SaveChangesAsync();
        return RedirectToPage("/Events/Index");
    }
```

```
}

public class DeleteEventModel(AppDbContext db) : PageModel
{
    [BindProperty] public Event Item { get; set; } = new();
    public async Task OnGetAsync(int id)
    {
        Item = await db.Events.FindAsync(id) ?? new Event();
    }
    public async Task<IActionResult> OnPostAsync()
    {
        var entity = await db.Events.FindAsync(Item.Id);
        if (entity is not null)
        {
            db.Events.Remove(entity);
            await db.SaveChangesAsync();
        }
        return RedirectToPage("/Events/Index");
    }
}
```

9) `wwwroot/css/site.css` **(Minimal-Styling)**

```
:root { font-family: system-ui, sans-serif; }
.container { max-width: 960px; margin: 2rem auto; padding: 0 1rem; }
header { display: flex; gap: .5rem; align-items: center; justify-content: space-
between; margin-bottom: 1rem; }
.btn { background: #0b5ed7; color: white; text-decoration: none; padding: .
5rem .75rem; border-radius: .5rem; display: inline-block; }
.btn.secondary { background: #6c757d; }
.btn.danger { background: #dc3545; }
.btn.small { padding: .25rem .5rem; font-size: .9rem; }
.grid { display: grid; grid-template-columns: 1fr 1fr; gap: 1rem; }
label { display: grid; gap: .25rem; margin-block: .5rem; }
textarea, input { padding: .5rem; border: 1px solid #ccc; border-radius: .
5rem; }
.actions { display: flex; gap: .5rem; margin-top: 1rem; }
table { width: 100%; border-collapse: collapse; }
th, td { padding: .5rem; border-bottom: 1px solid #eee; text-align: left; }
```

Datenbank vorbereiten (EF Core)

> Migrationen können via CLI erstellt werden. Für den Prototyp reicht es, die DB beim ersten Start zu erzeugen.

Option A – Migrationen nutzen

```
dotnet tool install --global dotnet-ef
cd DnaAustria.Events
setx ASPNETCORE_ENVIRONMENT Development

# erste Migration erzeugen
 dotnet ef migrations add InitialCreate
# anwenden
 dotnet ef database update
```

Option B – SQL manuell (nicht empfohlen hier)

Direkt in SQL Server eine leere DB `DnaAustriaEvents` anlegen (EF erstellt Tabellen beim Update).

Export-Format anpassen

Die zentrale Stelle ist `Services/EventExportService.cs` (`MapToExport`). Dort können Keys/Felder so umbenannt/ergänzt werden, dass sie exakt dem geforderten **Entdecke.DNAustria**-Schema entsprechen.

Beispiel für alternative Keys:

```
public object MapToExport(Event e) => new {
  eventId = e.Id,
  name = e.Title,
  description = e.Description,
  time = new { startUtc = e.StartUtc, endUtc = e.EndUtc },
  venue = e.LocationName,
  city = e.City,
  categories = (e.TagsCsv ?? "").Split(',',
StringSplitOptions.RemoveEmptyEntries | StringSplitOptions.TrimEntries)
};
```

Start & Nutzung

1. **SQL Server** starten: `docker compose up -d`
2. **Projekt** bauen & starten:

```
dotnet build
dotnet run
```

3. Browser öffnen: `https://localhost:5001` (oder `http://localhost:5000`)
4. Über **Veranstaltungen → Neu anlegen** Einträge erfassen.
5. Export über Buttons in der Liste oder REST:
6. `GET /api/Events/{id}/export` (einzelnes Event)
7. `GET /api/Events/export` (alle Events)

Nächste Schritte (optional)

- **Validierung**: Zeitzonenhandling (Europe/Vienna) zusätzlich zu UTC
- **Auth**: Login (z. B. Azure AD / OpenID Connect) für Redaktion
- **Import**: CSV/Excel-Upload zu Events
- **Versionierung**: Export-Version in Dateiheader
- **CI/CD**: GitHub Actions + Dockerfile fürs Backend