MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Sebastian Uhlík

Informační systém pro pořádání akcí

Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce: RNDr. Martin Pergel, Ph.D.

Studijní program: Informatika

Studijní obor: IPSS

Praha 2021

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V …..... dne..... podpis

Název práce: Informační systém pro pořádání akcí

Autor: Sebastian Uhlík

Katedra / Ústav: Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce: RNDr. Martin Pergel, Ph.D., Katedra softwaru a výuky informatiky

Abstrakt:

Vlastní webové stránky jsou dnes pro volnočasové organizace nezbytnost. Obsahem této práce je nástroj usnadňující jejich vytvoření, který zároveň nese logický základ zpřístupňující zajímavou aplikační funkčnost i bez pokročilých programátorských znalostí a dovedností.
Jedná se o middleware s REST-like architekturou, napsaný v jazyce C# pomocí frameworků ASP.NET Core a Entity Framework Core.
Nabízí komplexní a provázanou databázi, ukázkový frontend, automaticky generovanou PHP knihovnu a dokumentaci, popisující volání všech jednotlivých endpointů pomocí PHP funkcí.
Práce se zabývá těmi aspekty vývoje webové aplikace, které vedou k jeho zjednodušení.

Klíčová slova:

Middleware, Entity Framework Core, ASP.NET Core, tvorba webové stánky, volnočasová organizace

Title: Information system for event organising

Author: Sebastian Uhlík

Department: Department of Software and Computer Science Education

Supervisor: RNDr. Martin Pergel, Ph.D.

Abstract:

Owning a web page is a neccessity for the youth organizations. This thesis is a tool created to simplify the creation of such a page. It contains advanced functionality which enables its user to build a modern web page without the programming knowledge neccessary.
From implementation point of view, it is a middleware with REST-like architecture, writen in C# language, utilizing the ASP.NET Core and Entity Framework Core.
It offers a database with complex entity relations, example frontend, automatically generated PHP library and a documentation describing usage of each endpoint using a PHP function.
The thesis focuses on those aspects of the web development that lead to its simplification.

Keywords: Middleware, Entity Framework Core, ASP.NET Core, web page creation, youth organization

1 Struktura práce

1 Struktura práce 1
2 Úvod 3
2.1 Úvod do problematiky 3
2.2 Specifikace problému 3
2.3 Význam práce 3
2.4 Určení práce 3
2.5 Cíl práce 4
3 Analýza práce 5
3.1 Implementační cíle 5
3.2 Databáze 5
3.2.1 Entity 5
3.2.2 Katalogy 5
3.2.3 CatalogCreator 6
3.3 Logická vrstva 8
3.3.1 Standardní operace 8
3.3.2 Vnímání kontextu 8
3.3.3 Speciální operace 8
3.4 Volba jazyka 8
3.4.1 C# 8
3.4.2 Python 8
3.4.3 PHP 8
3.5 Vývojové nástroje 9
4 Databázový model 10
4.1 Entity 10
4.2 Vazby 11
5 Popis API 12
5.1 Web API obecně 12
5.1.1 Co je to Web API 12
5.1.2 Cesta http requestu 12
5.2 Web API v detailech – Child API 13
5.2.1 Message a Faker 13
5.2.2 Validace 15
5.2.3 Controller 16
5.2.4 Service 18

5.2.5 Response....18
5.3 Web API katalogu....19
5.4 Swagger dokumentace....19
6 PHP knihovna....22
6.1 Motivace....22
6.2 Kontext knihovny....23
6.2.1 PHP Klient....23
6.2.2 Automatická generace....23
6.3 Dokumentace knihovny....24
7 Ukázkové použití....25
7.1 Popis frontendu....25
7.2 Využití PHP knihovny....26
8 Uživatelská příručka....27
8.1 Jak aplikaci lokálně spustit....27
8.2 Dokumentace....27
8.2.1 Swagger dokumentace API....27
8.2.2 Markdown dokumentce funkcí PHP knihovny....27
8.3 Jak naplnit katalogy....27
8.3.1 Seed a funkce na DevSupportControlleru....27
8.3.2 Vlastní implementace CreateCatalogRequest....27
8.3.3 Catalog admin....28
9 Závěr....29
10 Seznam použitých pramenů....30
11 Seznam použitých zkratek....32
12 Seznam obrázků....33
13 Seznam ukázek kódu....33
14 Přílohy....33

2 Úvod

2.1 Úvod do problematiky

Úspěch volnočasové organizace dnes z velké části závisí na tom, jak je schopna se prezentovat. To v důsledku znamená, že taková organizace, ať už sebemenší, potřebuje vlastní webovou stránku. Díky ní může zveřejňovat chystané akce, fotky z akcí minulých nebo třeba dokumenty potřebné pro účastníky.

2.2 Specifikace problému

Většina takových skupin mezi svými členy nemá programátora, který by takovou stránku dokázal sám vytvořit. Napsat pár řádků kódu v jazyce HTML dokáže sice po pár minutách snahy v podstatě každý, ale existuje široká škála problémů, které člověk bez pokročilých znalostí programování skoro jistě nevyřeší. Jedná se například o práci s databází účastníků akcí a členů organizace, autorizace přihlášek, automatizované odesílání emailů s informacemi nebo prostě jen ochrana svěřených dat.

Nejmenší z těchto organizací se v podstatě bez výjimky uchylují k jednomu z portálů pro tvorbu webových stránek zdarma. V jejich případě je to často i nejlepší volba, protože pokročilejší funkce zmíněné v minulém odstavci nepotřebují.

Velké organizace jako třeba Junák nebo Pionýr se zase bez výhrad uchylují k práci profesionálních programátorů. Ať už kvůli komplexitě potřebných stránek, nutnosti chránit tisíce osobních údajů a následkům případného neúspěchu nebo čistě proto, že web organizace takového významu musí, lidově řečeno, „nějak vypadat“.

2.3 Význam práce

Existuje ještě třetí skupina organizací, kterou budu označovat jako „střední“. Na mysli tím mám skupiny s desítkami členů, pro které využití databáze má smysl a schopnost efektivně zpracovávat přihlášky účastníků je pro jejich administrátory životně důležitá.
Bohužel i tato skupina často volí cestu stránek zdarma, protože si profesionální služby nemohou dovolit (což není nutně pravidlem) nebo je náročnost úkolu odradí od pokusů o vlastní tvorbu.

Dle dostupných informací neexistuje žádný bezplatný portál nebo systém, který by se na tuto problematiku zaměřoval a změnit to je hlavní motivací mojí práce.

2.4 Určení práce

Tato aplikace je zamýšlena jako nástroj pro člena volnočasové organizace, který dostal na starost tvorbu jejích webových stránek. Její cíl je poskytnout mu rozhraní s fungující aplikační logikou a se specializovanou databází.

Je psána tak, aby jejímu uživateli stačilo nahrát ji na server s podporou pro .NET Core, napojit na server s databází (což může a nemusí být jiný) a dodat frontend kompatibilní s normou REST[1]. Pro tento úkol poskytuje uživateli podrobnou dokumentaci svých endpointů a funkcí, které jsou mu k dispozici. Její součástí jsou také ukázkový frontend

[1] „Representational state transfer“ (REST) je druh softwarové architektury zaměřené na world wide web… (Wikipedia, 2021)

napsaný v jazyce PHP a automaticky generovaná PHP knihovna umožňující a dokumentující volání všech endpointů našeho API[2] pomocí ukázkových funkcí.

Využití aplikace předpokládá určitou počítačovou gramotnost a schopnost vyhledávat informace. Doporučuje prostudování poskytnuté dokumentace a ukázkového frontendu. Dále samozřejmě to, aby si její uživatel zařídil webový hosting a vlastní doménu pro svůj výsledný produkt. Na oplátku nabízí možnost vytvořit si vlastní webové stránky s použitím nejmodernějších technologií, které jsou nyní dostupné, bez nutnosti studovat databáze, bezpečnostní protokol https[3] a mnohá další komplexní témata.

2.5 Cíl práce

Cílem této práce je vyřešit některé zajímavé aspekty implementace takové aplikace. Primárně se proto soustředí na vytvoření dostatečně rozsáhlé a univerzální databáze, logické nadstavby k jejím jednotlivým entitám a knihovny, která usnadní vytvoření zmiňovaného frontendu.

[2] API „Aplication programing interface“ je typ softwarového interface, navrženého pro komunikaci s jinými logickými jednotkami softwaru. (Wikipedia, 2021)
[3] HTTP a HTTPS jsou protokoly pro přenos hyper textu.

3 Analýza práce

3.1 Implementační cíle

Za hlavní dva cíle implementace této aplikace můžeme považovat vytvoření databáze a logické vrstvy nad ní. Právě v databázi a jejích entitách bude spočívat většina specializace na volnočasové organizace, kterou tato práce slibuje. Úkolem logické nadstavby je pak poskytnout funkčnost, kterou lze u takových organizací předpokládat.
V poslední řadě je pak potřeba vytvořit k aplikaci ukázkový frontend a demonstrovat tak její použitelnost pro stanovený cíl.

3.2 Databáze

Jelikož je aplikace určena pro organizátory volnočasových organizací obecně, je třeba navrhnout databázi univerzálně. Musí podporovat vytváření událostí, registraci rodičů, vytváření profilů jejich jednotlivých dětí, registraci účasti dětí na událostech, vytvoření a používaní hierarchie v rámci organizace, a další.

3.2.1 Entity

S ohledem na myšlenku z předchozího odstavce, do databáze jistě patří entity reprezentující rodiče, dítě, událost a přihlášku na událost. Dále je třeba přidat entitu, která by umožnila nahrávání dokumentů, které s událostmi souvisí[4]. V neposlední řadě je třeba přidat entitu reprezentující organizátora a mohla by přijít k užitku i entita vyjadřující fakturu.

3.2.2 Katalogy

Kromě samotných entit, lze specializaci aplikace ještě zvýšit přidáním katalogů. To jsou objekty, jejichž záznamy (obvykle) nenesou žádné kontextové informace, ale pouze svou hodnotu. Samy o sobě nemají katalogy žádný význam. Ten získávají, až při vazbě s entitou, protože tím na ní reprezentují nějakou konkrétní hodnotu.

Například pomocí implementace následujících katalogů lze vytvořit robustní podporu pro zadávání adresy.

> *Země* – seznam podporovaných zemí (např. Česká republika, Slovenská republika)
> *Kraj* – seznam krajů v podporovaných zemích, každý záznam obsahuje id země, do které náleží
> *Okres* – seznam okresů v podporovaných zemích, každý záznam obsahuje id kraje, do kterého patří
> *PSČ* – seznam poštovních směrovacích čísel v podporovaných zemích, každý záznam obsahuje id okresu, do kterého patří

Díky takové implementaci lze libovolné zadání adresy zjednodušit na pouhé manuální vyplnění města, ulice a čísla popisného. Ze seznamu pak stačí vybrat příslušné poštovní směrovací číslo a s použitím vazeb definovaných mezi jednotlivými katalogy se zbytek adresy vyplní automaticky.

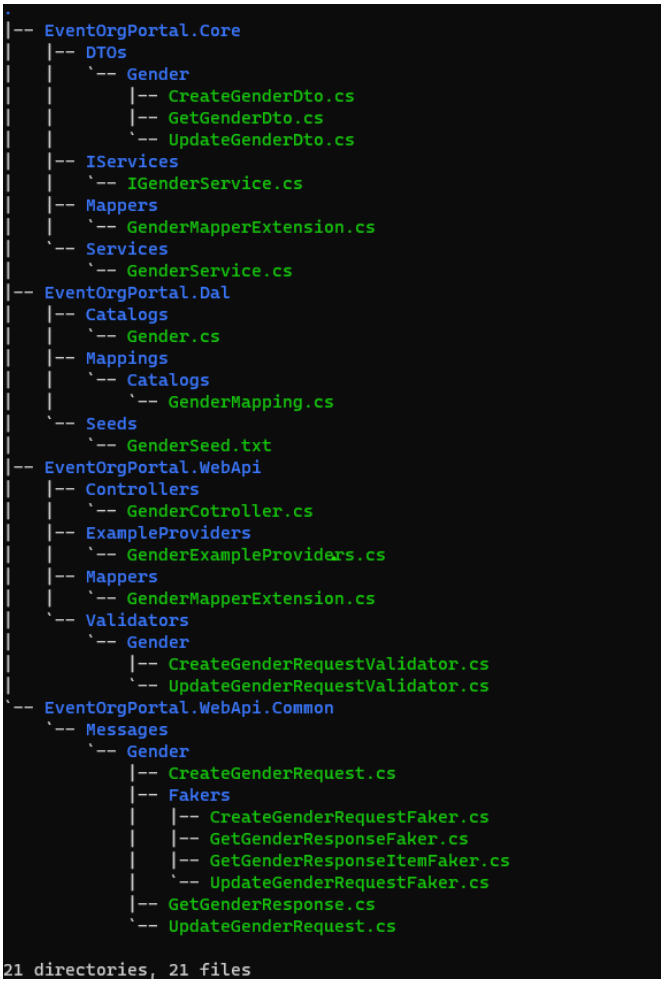
Jako další a typičtější katalogy, tedy katalogy bez kontextuálních vazeb, si můžeme představit například pohlaví, kód zdravotní pojišťovny, alergii nebo třeba schopnost dítěte plavat.

[4] Například posudek o zdravotní způsobilosti dítěte podle vyhlášky č.106/2001 Sb. v platném znění (ČESKO, 2001) nebo lékařské potvrzení zdravotní způsobilosti pracovat s dětmi podle požadavků §10 a §12 zákona č. 258/2000 Sb. o ochraně veřejného zdraví (ČESKO, 2000) (ČESKO, 2000).

Výhodou použití katalogu je, že organizátor si v jeho rámci může nadefinovat libovolné hodnoty a uložit je do databáze. Ty budou následně k dispozici při práci s jakýmkoli formulářem, který položky katalogu obsahuje.

3.2.3 CatalogCreator

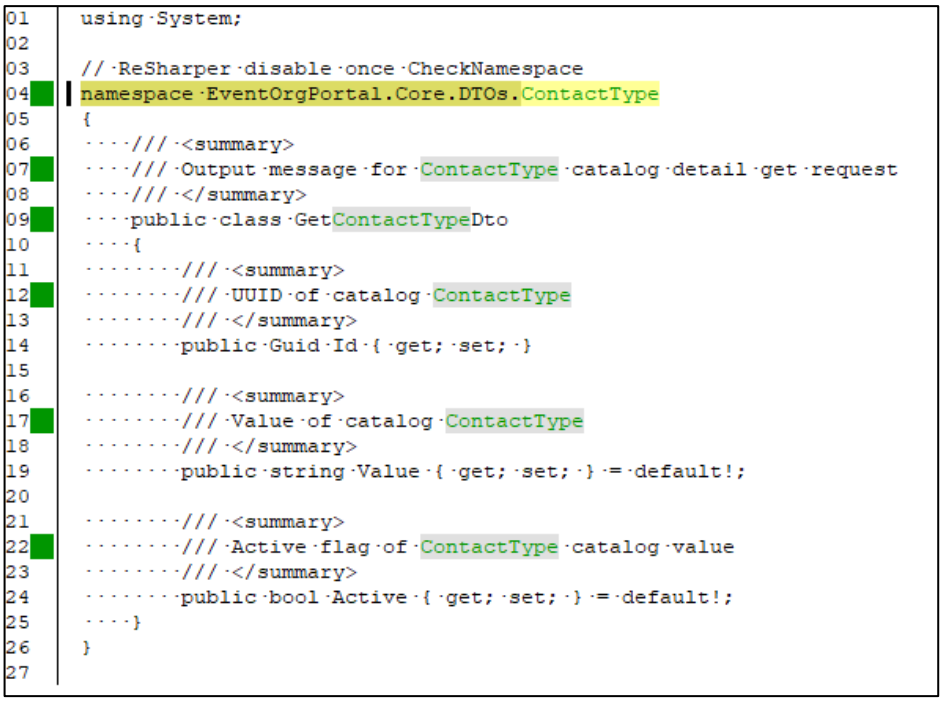
Abychom získali představu, o implementaci katalogu, prohlédněme si následující obrázek.



Obrázek 1 - implementace katalogu Gender

Na jeho spodním okraji si můžeme všimnout, že jeden katalog se rozkládá v celkem 21 různých souborech. Představa manuálního vytváření těchto souborů a doplňování adresářové struktury nás může přivést k úvahám o tom, jestli to nejde dělat lépe, a to především ve chvíli, kdy takových katalogů chceme vyrobit zhruba 20.

Když si navíc pročteme kód v jednotlivých souborech z tohoto obrázku zjistíme, že implementace dvou různých katalogů (chápejme těch základních, které nemají žádný kontext) se od sebe liší pouze v hodnotě svého jména.



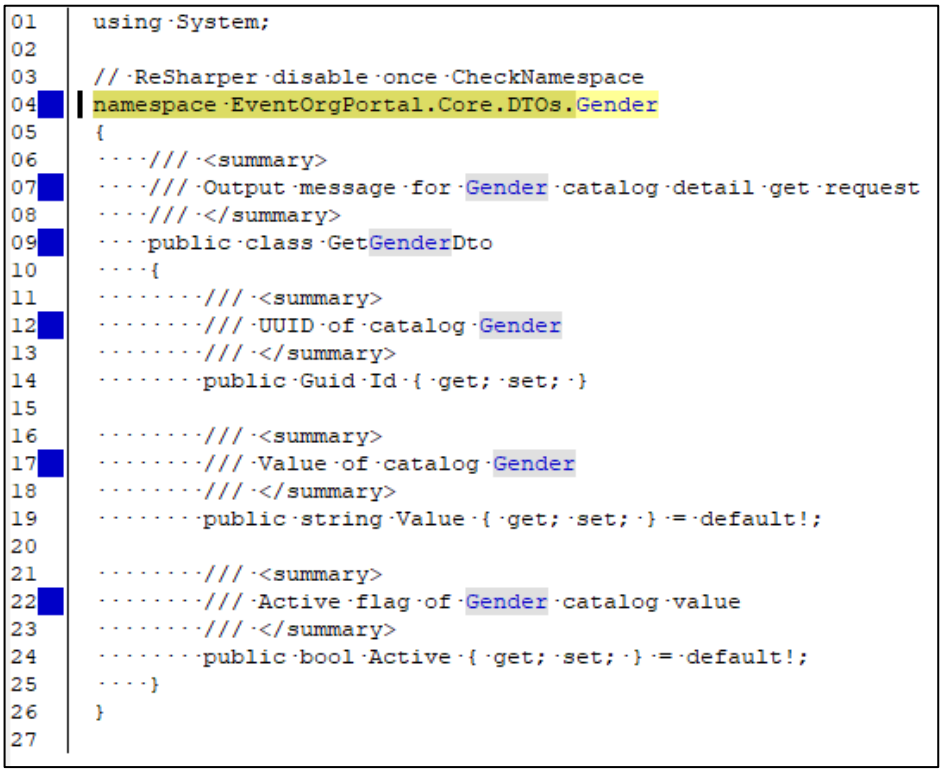
```
using System;

// ReSharper disable once CheckNamespace
namespace EventOrgPortal.Core.DTOs.ContactType
{
    /// <summary>
    /// Output message for ContactType catalog detail get request
    /// </summary>
    public class GetContactTypeDto
    {
        /// <summary>
        /// UUID of catalog ContactType
        /// </summary>
        public Guid Id { get; set; }

        /// <summary>
        /// Value of catalog ContactType
        /// </summary>
        public string Value { get; set; } = default!;

        /// <summary>
        /// Active flag of ContactType catalog value
        /// </summary>
        public bool Active { get; set; } = default!;
    }
}
```

Obrázek 2 - GetContatctTypeDto v aplikaci diff



```
using System;

// ReSharper disable once CheckNamespace
namespace EventOrgPortal.Core.DTOs.Gender
{
    /// <summary>
    /// Output message for Gender catalog detail get request
    /// </summary>
    public class GetGenderDto
    {
        /// <summary>
        /// UUID of catalog Gender
        /// </summary>
        public Guid Id { get; set; }

        /// <summary>
        /// Value of catalog Gender
        /// </summary>
        public string Value { get; set; } = default!;

        /// <summary>
        /// Active flag of Gender catalog value
        /// </summary>
        public bool Active { get; set; } = default!;
    }
}
```

Obrázek 3 - GetGenderDto v aplikaci diff

Na základě těchto úvah a pozorování vznikla myšlenka skriptu CatalogCreator - příloha 2) CatalogCreator.zip - Zdrojový kód a šablona.

Tento skript automaticky vytváří katalogy podle šablony, pro kterou je použit katalog Gender. Všechny soubory umisťuje na správné místo v adresářové struktuře a nahrazuje slova „Gender“ a „gender“ příslušnými variantami jména nově generovaného katalogu.

3.3 Logická vrstva

Primárním úkolem logické vrstvy je poskytnout funkcionalitu nad objekty z databáze.

3.3.1 Standardní operace

Jako standartní funkce chápeme ty, které nejsou nijak ovlivněny zaměřením práce. Jedná se o standartní funkce pro vytvoření, úpravu, zobrazení a smazání záznamu v rámci nějaké databázové tabulky. Jedná se o naprostý základ jakékoli aplikace pracující s databází.

3.3.2 Vnímání kontextu

Zajímavější součástí aplikace jsou pak funkce, které pracují s databázovým kontextem. Jedná se především o různé druhy validace dat zadávaných uživatelem. První z nich, je ověření fyzické podoby dat. Použití správného formátu, počtu znaků nebo třeba vyplnění povinných položek ve formuláři. Další způsob validace spočívá v ověření logické správnosti dat. Například pro zadaný vedlejší (vazební) klíč se jedná a ověření existence záznamu entity (na kterou vzniká vazba vzniká).

3.3.3 Speciální operace

Za speciální považujeme ty operace, které vznikly jako přímý důsledek zaměření práce. Je to skupina funkcí, jejichž primárním cílem je, co nejvíce zpříjemnit používání aplikace k jejímu účelu. Příkladem takové funkce je zobrazení podmnožiny informací o záznamu entity za účelem zobrazení seznamu takových záznamů. U entity, která nese 20 a více různých informací bude taková možnost obzvláště cenná. Dalším příkladem je připravená filtrace záznamů entit, jako třeba ty děti, které jedou na konkrétní akci, všechny děti určitého rodiče nebo v rámci katalogů všechna města konkrétního okresu, či třeba všechny skupiny v rámci organizační jednotky.

3.4 Volba jazyka

Volba jazyků byla jednoduchá. Vybral jsem ty, se kterými mám nejvíce zkušeností a jejichž propojení za účelem vytvoření webové aplikace mě zajímalo. V případě jazyka pro tvorbu frontendu jsem zvolil jednoduše ten, který se zdálo nejjednodušší použít.

3.4.1 C#

Jedná se o moderní, objektově orientovaný a robustní jazyk s nesčetnou komunitou vývojářů a vysoce kvalitní dokumentací. Obsahuje mnoho knihoven a frameworků, které rozšiřují jeho funkcionalitu a posouvají hranice toho, co je pomocí něj ještě možno vytvořit. Jako několik příkladů za všechny uveďme následující:

System.Reflection – knihovna umožňující práci se zdrojovým kódem
ASP.NET Core – framework pro tvorbu moderních webových aplikací
Entity Framework Core – framework umožňující práci s databází přímo ze C# projektu pomocí knihovny LINQ[5]

3.4.2 Python

Ideální jazyk pro tvorbu jednoduchých skriptů, jako je například CatalogCreator. Jeho použití pro práci s textem je kromě jednoduchosti opodstatněné i optimalizací této činnosti.

3.4.3 PHP

Rozšířený jazyk, který nabízí jednouchou tvorbu ukázkového frontendu.

[5] Knihovna umožňující operace nad daty pomocí query (Microsoft, 2021).

3.5 Vývojové nástroje

Pro vývoj bude použita aplikace *Visual Studio 2019*. Kromě ní následující nástroje, které několika slovy čtenáři přiblížím.

Docker – nástroj využívající virtualizaci různých operačních systémů pro spouštění služeb[6] v navzájem oddělených paměťových blocích, takzvaných kontejnerech
Swagger – nástroj pro vývoj a dokumentaci webových API, umožňuje posílat data na jednotlivé přístupové body těchto API a testovat tak jejich funkčnost, dále dovoluje export souboru pro vytvoření interaktivní dokumentace
Postman – i toto je nástroj pro vývoj webového API, oproti Swaggeru obsahuje funkce, které testování usnadňují (jako například možnost tvorby automatických testů)

[6] Příkladem takové služby může být databázový server, nebo server umožňující spuštění webové aplikace.

4 Databázový model

Databázový model dodržuje principy popsané v analýze práce (3.2 Databáze strana 5). Místo psaného textu jej proto budu reprezentovat spíše schématy.

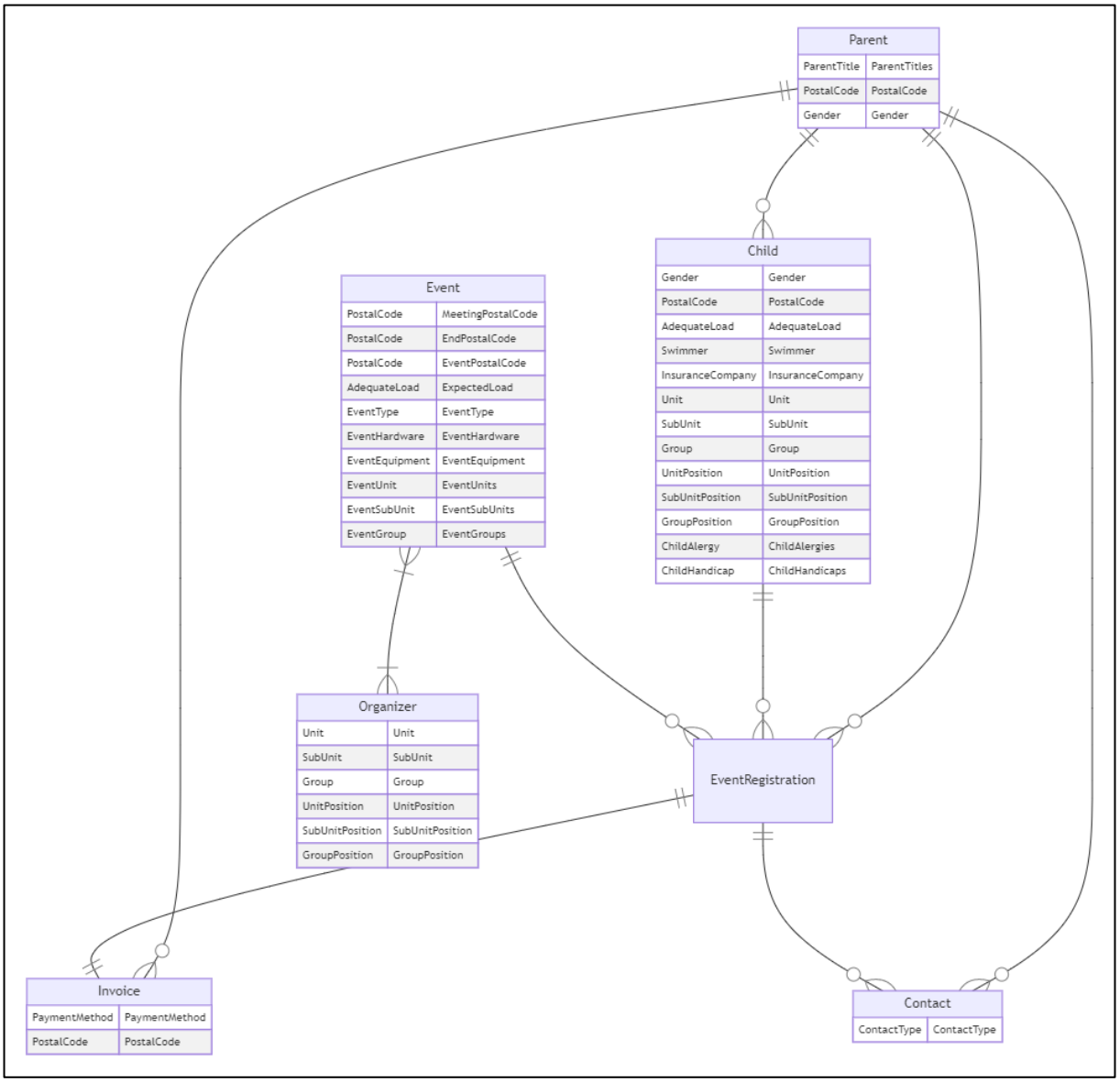
4.1 Entity

V následujícím obrázku můžeme vidět seznam entit doplněný o jejich jednotlivé datové položky. Mnoho z těchto položek je volitelných, což znamená, že entity je možno používat vyplněné jen na nutné minimum, a v případě zájmu ukládat více informaci, není nutné do implementace entit zasahovat.

Obrázek 4 - Přehled databázových entit

4.2 Vazby

Obrázek 5, který uvidíme nyní se na rozdíl od předchozího zaměřuje na vazby mezi jednotlivými entitami. Položky, které jsou na entitách vypsány představují katalogy a konkrétně ukazují, se kterými entitami se mohou pojit. Vzhledem k jejich četnosti a možnému napojení jednoho katalogu na více entit najednou, snad čtenář ocení toto vyjádření více, než čistě čárové schéma.



Obrázek 5 - Schéma vazeb

5 Popis API

5.1 Web API obecně

5.1.1 Co je to Web API

Již mnohokráte jsme se setkali s termínem API. Vzhledem ke kontextu práce se navíc vždy jednalo o webové aplikační rozhraní. Obecně je to soubor funkcí a tříd, které jsou při práci s aplikací programátorovi k dispozici.
Abych mohl tento pojem vysvětlit podrobněji, vydefinuji nyní několik dalších:

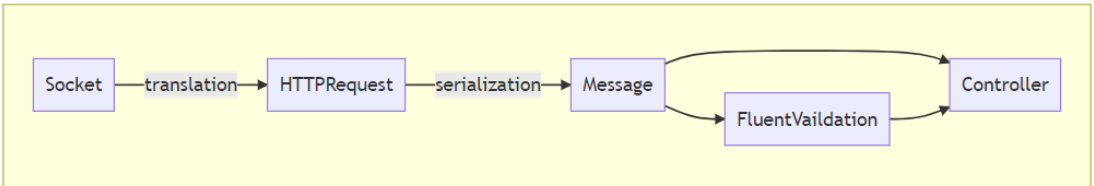
> *Endpoint* – „Koncový bod“ pro komunikaci pomocí http protokolu. Jedná se o pomyslnou hranici, na které končí Web API, a odkud je vyvolávána endpointu příslušná služba.
>
> *Controller* – „Řadič“ je specializovaná třída nesoucí definice všech endpointů příslušného API. Pro každý tento endpoint obsahuje metodu, která volá již zmiňovanou přiřazenou službu.
>
> *Message* – „Zpráva“ je struktura implementovaná tak, aby přenášela veškeré informace z položky body příchozích http dotazů (http request) a odchozích odpovědí (http response).

Nyní mohu, již konkrétněji říci, že naše webová aplikace zpřístupňuje endpointy na controllerech příslušných jednotlivým entitám existujícím v databázi. Pro takovou entitu jsou to standardně endpointy pro její vytvoření, aktualizaci, zobrazení a smazání, ve zkratce CRUD.

Controllery existují i pro některé systémové služby, jako je například serverový čas. Jejich endpointy většinou plný CRUD neimplementují, protože možnost vytvořit nebo smazat systémový čas nedává moc dobrý smysl.

5.1.2 Cesta http requestu

K lepšímu pochopení toho, k čemu webové API slouží, popíši datovou cestu, kterou prochází příchozí http komunikace.



Obrázek 6 - Cesta http requestu

Když je http komunikace přijata příslušným socketem, je nejprve znovu sestaven původní http request. Z jeho položek header a body je následně pomocí serializace sestaven message, přičemž o používané serializační metodě rozhoduje položka Conent-Type, z již zmiňovaného headeru.

V závislosti na druhu příchozí žádosti, musí message buďto nejprve projít validací, nebo je předán controlleru přímo. (Validací zde chápejme kontrolu přenášených dat z fyzického hlediska. Tedy zda jsou vyplněny všechny povinné položky zprávy, zda nepřesahují povolenou délku nebo naopak, zda jsou neprázdné atp.)

V controlleru pak cesta komunikace končí tím, že je zavolána obslužná metoda příslušného endpointu a potenciálně i systémová služba.

V tomto místě API pomyslně končí. Protože se ale jedná o velmi důležitý koncept, podíváme se v následující podkapitole ještě za tuto hranici, a na konkrétním příkladu popíši jeho použití.

5.2 Web API v detailech – Child API

Jako ukázkové API jsem si vybral to, které přísluší entitě dítěte (Child) z naší databáze. Vybral jsem ho, protože patří mezi ta složitější a má několik endpointů, jež u jiných entit nenajdeme.

5.2.1 Message a Faker

API nese následující Message struktury:

CreateChildRequest – Standardní message pro vytvoření dítěte. Od CreateRequestů ostatních entit se mírně liší v tom, že neumožňuje vyplnit některé z položek, které výsledná entita v databázi má. Je to proto, že tento message přísluší endpointu, který se zavolá, když si rodič přidá do svého profilu nové dítě. Není žádoucí, aby rodič v takové chvíli mohl manuálně určit číslo registrace, číslo členské průkazky dítěte nebo třeba sám zařadit dítě do nějaké skupiny v rámci struktury naší organizace.

UpdateChildRequest – Message pro úpravu dítěte má stejná omezení jako ten pro jeho vytvoření. Nelze připustit úpravu některých údajů ze strany někoho, kdo není jedním z vedoucích organizace. Message proto umožňuje úpravu pouze u vybraných údajů.

GetChildResponse – Tento message se od jiných GetRequestů nijak neliší. Slouží pro zobrazení veškerých údajů o konkrétním dítěti.

UpdateChildOrgRequest – Speciální message určený členům organizace. Umožňuje úpravu všech položek dítěte.

GetChildInfoResponse – Speciální message sloužící pro přenos pouze základních informací, jako jsou identifikátor, jméno, příjmení a přezdívka. Existuje proto, aby bylo možné se podívat na seznam všech dětí na profilu rodiče, nebo třeba na ty, které jsou přihlášeny na konkrétní akci, bez nutnosti zobrazovat tabulku s asi 40 sloupci.

```
/// <summary>
/// Output message for Child entity info get request
/// </summary>
public class GetChildInfoResponseItem
{
    /// <summary>
    /// UUID of the Child entity
    /// </summary>
    public Guid Id { get; set; }

    /// <summary>...</summary>
    public string FirstName { get; set; } = default!;

    /// <summary>...</summary>
    public string? MiddleName { get; set; }

    /// <summary>...</summary>
    public string LastName { get; set; } = default!;
}
```

Ukázka kódu 1 - GetChildInfo message

Pro každý výše uvedený message existuje příslušně nakonfigurovaný náhodný generátor - faker. Faker (neboli „fake data generator“, česky generátor náhodných dat) je knihovna poskytující pseudonáhodně generovaná data, podle specifického modelu. Model může být zaměřený na celá či desetinná čísla, stejně dobře jako na názvy států, lidská jména, či slovo nebo odstavec Lorem ipsum[7].

Fakery, v tomto případě konkrétně z knihovny Bogus (Chavez, 2021), se využívají pro vytváření messagů bez nutnosti ručně vyplňovat jednotlivé jeho položky, což například u message *GetChildResponse*, který nese položek 44, velmi oceníme. Díky této automatické generaci můžeme message snadno testovat a nádavkem získat ještě jejich konkrétní příklady do naší dokumentace.

```
public class GetChildInfoResponseItemFaker : IFaker<GetChildInfoResponseItem>
{
    public Bogus.Faker<GetChildInfoResponseItem> BogusGenerator { get; }

    public GetChildInfoResponseItemFaker()
    {
        BogusGenerator = new AutoBogus.AutoFaker<GetChildInfoResponseItem>()
            .RuleFor(a => a.Id, f => f.Random.Guid())
            .RuleFor(a => a.FirstName, f => f.Person.FirstName.ToString())
            .RuleFor(a => a.MiddleName, f => f.Person.UserName.ToString())
            .RuleFor(a => a.LastName, f => f.Person.LastName.ToString());
    }

    public GetChildInfoResponseItem Generate()
    {
        return BogusGenerator.Generate();
    }
}
```

Ukázka kódu 2 - Definice GetChildInfo message fakeru

[7] Lorem ipsum (zkráceně lipsum) je označení pro standardní pseudolatinský text užívaný v grafickém designu a navrhování jako demonstrativní výplňový text… (Wikipedie, 2020)

5.2.2 Validace

Validaci provádíme pouze pro *Create & UpdateRequest*. Tím je zaručeno že do databáze se dostanou pouze korektně vyplněné záznamy, a proto není třeba žádná kontrola na *GetResponse.*

Samotný proces není složitý. Pro každý message existuje jemu příslušný validátor s předdefinovanými pravidly pro jeho jednotlivé položky, které se jedna po druhé proti těmto pravidlům zkontrolují.
Nezajímáme se při tom o správnost příchozích dat, ale o jejich korektnost. Za chybu tedy považujeme, když vytvářené dítě „Petr Malý“ nebude mít vyplněné datum narození, nikoli to, že se pravděpodobně nejmenuje „Petr“.

V následující ukázce můžeme vidět, jak definice pravidel vypadá. Zdůraznil bych zvláště pravidlo „NotNull()“, protože jeho přítomnost v této ukázce jednoznačně určí, které položky jsou povinné (ty co pravidlo mají) a které ne.

```
/// <summary>
///
/// </summary>
public class CreateChildRequestValidator : AbstractValidator<CreateChildRequest>
{
    /// <summary>
    ///
    /// </summary>
    public CreateChildRequestValidator()
    {
        RuleFor(e => e.FirstName).NotNull().MaximumLength(150);
        RuleFor(e => e.MiddleName).MaximumLength(150);
        RuleFor(e => e.LastName).NotNull().MaximumLength(150);
        RuleFor(e => e.NickName).MaximumLength(150);
        RuleFor(e => e.BirthDate).NotNull();
        RuleFor(e => e.Mobile).MaximumLength(25);
        RuleFor(e => e.EmailAddress).MaximumLength(320);
        RuleFor(e => e.AddressLine1).NotNull().MaximumLength(150);
        RuleFor(e => e.AddressLine2).MaximumLength(150);
        RuleFor(e => e.AddressLine3).MaximumLength(150);
        RuleFor(e => e.AddressLine4).MaximumLength(150);
        RuleFor(e => e.InsuranceNumber).NotNull().MaximumLength(150);
        RuleFor(e => e.Medicals).MaximumLength(2000);
        RuleFor(e => e.ParentId).NotNull();
        RuleFor(e => e.GenderId).NotNull();
        RuleFor(e => e.AdequateLoadId);
        RuleFor(e => e.SwimmerId).NotNull();
        RuleFor(e => e.PostalCodeId).NotNull();
        RuleFor(e => e.InsuranceCompanyId).NotNull();
        RuleFor(e => e.AlergyIds);
        RuleFor(e => e.HandicapIds);
    }
}
```

Ukázka kódu 3 - CreateChild message validator

5.2.3 Controller

Hlavní činnost controlleru spočívá v zavolání obslužné funkce endpointu, který přísluší příchozímu http requestu. Určení této funkce se provádí podle URL[8], takže každý endpoint musí mít jednoznačně nastavený svůj atribut route[9].

V následující ukázce přesto můžeme vidět použití tohoto atributu pouze na samotné třídě *ChildController*. To je proto, že jednotlivé endpointy tento atribut z controlleru dědí a apendují za něj svůj typ. (V ukázce *[HttpGet("ByParent")]*) Zbylé atributy se používají při generování dokumentace a PHP frontend knihovny (kapitola 6. PHP knihovna, strana 22).

```
/// <summary> Controller defining Child entity endpoints. </summary>
[ApiController]
[ApiVersion(ApiConstants.Version10)]
[Route("api/v{version:apiVersion}/[controller]")]
public class ChildController : ControllerBase
{
    private readonly ILogger<ChildController> _logger;
    private readonly IChildService _childService;

    /// <summary> ... </summary>
    public ChildController(ILogger<ChildController> logger, IChildService
childService)
    {
        _logger = logger;
        _childService = childService;
    }

    /// <summary>  Get list of existing Child values ... </summary>
    /// <returns>Https request</returns>
    /// <response code="200">Child list successfully found</response>
    /// <response code="400">The parent does not exist</response>
    [HttpGet("ByParent")]
    [ProducesResponseType(StatusCodes.Status200OK, Type =
typeof(GetChildResponse))]
    [ProducesResponseType(StatusCodes.Status400BadRequest, Type =
typeof(ValidationProblemDetails))]
    public IActionResult GetChildByParentList([FromQuery] Guid parentId)
    {
        try
        {
            var childList =
_childService.GetChildrenInfoByParent(parentId).ToMessage();
            return Ok(childList);
        }
        catch (KeyDoesNotExistException)
        {
            return BadRequest($"Parent with id '{parentId}' does not exist!");
        }
    }
...
```

Ukázka kódu 4 - ChildController

Endpointy na Child controlleru můžeme rozdělit do dvou skupin. První obsahuje ty, které nemají žádný speciální účel nebo vlastnosti. Řadíme do ní *DeleteChildItem*, *GetChildItem*,

[8] …, běžně webová adresa je řetězec znaků, který slouží k přesné specifikaci umístění zdrojů informací na internetu (Wikipedie, 2021)
[9] Atribut route definuje URL řetězec a informace pro obslužnou funkci endpointu. (TutorialsTeacher.com, 2020)

CreateChildItem a *UpdateChildItem* i když u posledních dvou existuje drobná výjimka v tom, že nezpřístupňují všechny položky entity. Pro úplnost jsou funkce těchto endpointů popořadě smazat dítě, získat všechny položky dítěte, vytvořit nové dítě a upravit již existující dítě.
Druhá skupina je pro nás zajímavější a na její členy se podíváme podrobněji.

UpdateChildOrgItem – Tento endpoint byl vytvořen s myšlenkou na vedoucí organizace, kterým dává přístup k položkám určujícím pozici a funkci dítěte v rámci organizační hierarchie. Kromě toho jim dovoluje upravit i identifikaci registrační číslo a číslo jeho členské průkazky.

GetChildByOrganizationList – Umožňuje vyfiltrovat zobrazení všech členů na členy některé organizační jednotky nebo ty s nějakou konkrétní funkcí. Pro tento účel musí z http requestu obdržet položku *filterValue,* tedy hodnotu, pomocí které filtrace proběhne, a *filterType*, která určí položku pro filtraci vybírá jednu z těchto položek entity:

```
public enum ChildFilter
{
    [EnumMember(Value = "none")]
    None,
    [EnumMember(Value = "unitId")]
    UnitId,
    [EnumMember(Value = "subUnitId")]
    SubUnitId,
    [EnumMember(Value = "groupId")]
    GroupId,
    [EnumMember(Value = "unitPositionId")]
    UnitPostionId,
    [EnumMember(Value = "subUnitPositionId")]
    SubUnitPostionId,
    [EnumMember(Value = "groupPositionId")]
    GroupPositionId,
}
```

Ukázka kódu 5 - filtrace dle org. hierarchie

GetChildByParentList – Získá seznam dětí konkrétního rodiče. Vzhledem k myšlence účtů rodičů, kde si tito vytvoří profily svých dětí, je existence tohoto endpointu samozřejmostí. Pomocí dotazu na něj získá rodič snadno a rychle přehled o profilech svých dětí. (Samozřejmě pouze tehdy, když bude endpoint dodaným frontendem takto využit.)

5.2.4 Service

Třída Service obsahuje definice funkcí, které přímo manipulují s databázovým kontextem[10]. Proto je důležité, aby data, se kterými pracujeme na této (Service) úrovni, dávala v kontextu databáze smysl. Tím se dostáváme k takzvané kontextové validaci, jež takovou funkčnost zajišťuje.

Nejsnáze si kontextovou validaci vysvětlíme na příkladu. Představme si hypotetickou situaci, kdy našemu API přijde request na vytvoření nového profilu dítěte. Request se zpracuje a projde všemi fázemi naší datové cesty až do controlleru. Tam je z message vytvořen takzvaný DTO[11]. Odtud se nejprve zavolá *ValidateChild* metoda na naší service a v případě, že validace proběhne v pořádku, je DTO předán metodě *CreateChild* a záznam je přidán do databáze.
Úkolem metody *ValidateChild* je ověřit integritu vytvářených vazeb. Entita Child má vazby na mnohé další entity, ale pro jednoduchost uvažujme pouze vazbu N:1 na entitu Parent (Tedy vazbu, která určuje, že každý 1 rodič může mít N dětí.). *CreateChildDto* na sobě nese položku *ParentId*, která obsahuje primární klíč konkrétní entity Parent. Validace tedy spočívá v tom, že se zkonstruuje databázový dotaz, který ověří, zda existuje entita Parent s příslušným klíčem.
Tento způsob validace je dostatečný, protože jako primární klíče používáme náhodně generované GUIDy a duplicita jakýchkoli dvou klíčů přes všechny entity je tak vyloučena. Na vysvětlenou, GUIDy jsou 128 bitové řetězce, které lze, co se praktického použití týče, považovat za unikátní (Wikipedia, 2021).

Co se týče konkrétní implementace třídy *ChildService,* není třeba ji rozvádět, protože neobsahuje nic neobvyklého. Pro každý z endpointů controlleru má jednu obslužnou metodu a pro endpointy *CreateChildItem, UpdateChildItem* a *UpdateChildOrgItem* má validační metody, které definují, jaké z položek entity Child je třeba zkontrolovat z hlediska kontextu databáze.

5.2.5 Response

Http response je prostředek komunikace serveru ke klientovi. Jeho účelem je poskytnout klientovi data, o která žádal, informovat ho o provedení požadované akce, nebo o chybě, ke které během provádění požadované akce došlo (IBM, 2021).

Jak plyne z úvodního odstavce, response se generuje, ke každému příchozímu requestu. Dochází k tomu v Controlleru poté, co je vyhodnocen výsledek validace, případně volání obslužné funkce na service.

[10] Databázový kontext je nejdůležitější třída v rámci Entity Framework Core. Reprezentuje napojení na databázi a umožňuje provádět nad ní CRUD operace, přímo použitím C# kódu (EntityFrameworkTutorial.net, 2020).
[11] „Data transfer object“ Jedná se o třídu určenou čistě pro přenášení dat. (Obvykle mezi dvěma oddělenými logickými celky programu.)

Můžeme se setkat s těmito odpověďmi z následujících důvodů:

200 OK – Response typ, se kterým se vrací výsledek úspěšného *Get* requestu.
201 Created – Informuje o úspěšném vytvoření záznamu v databázi. Nese URL vytvořeného záznamu a časový údaj jeho vytvoření.
204 No Content – Odpověď po korektním vymazání záznamu z databáze.
400 Bad Request – Response typ informující o chybě v žádosti. Nese data exaktně popisující původ chyby. Vzniká při chybě ve validaci message nebo při kontextové validaci záznamu.
404 Not Found – Vzniká, když v databázi není nalezen požadovaný záznam.
409 Conflict – Tato chybová odpověď se vrací, pokud již v databázi existuje záznam s identickou položkou, která je použita jako index. e.g. email na entitě Parent.

5.3 Web API katalogu

Strukturou API se katalogy příliš neliší od ostatních entit. Odlišnosti můžeme najít v existenci endpointů pro výpis všech záznamů, které některé z entit nedefinují. Další rozdíl spočívá v tom, že seznamy vzniklé zavoláním těchto endpointů, umožňují API katalogů extenzivně filtrovat.

Příklad takové filtrace můžeme najít na katalogu PostalCode. Pro zavolání endpointu *GetPostalCodeList* vyžadujeme, aby uživatel vyplnil alespoň první číslici z hledaného poštovního směrovacího čísla. Jedná se o, na první pohled, drobný detail, který ale fakticky omezí statisíce potenciálních hodnot pouze na desetitisíce.
S dalším druhem filtrace se pak setkáme na katalogu District. Okresy, si můžeme nechat filtrovat buďto podle země nebo podle regionu, do kterého náleží. Jedná se o optimalizaci na čtení (vyhledávání). Když budeme hledat například okres Mimoň, aniž bychom v ten moment znali jeho jméno a mohli ho vyplnit manuálně, ale budeme vědět třeba to, že leží v Libereckém kraji, množina okresů, kterou bude před nalezením toho správného potřeba zobrazit, se výrazně zmenší.

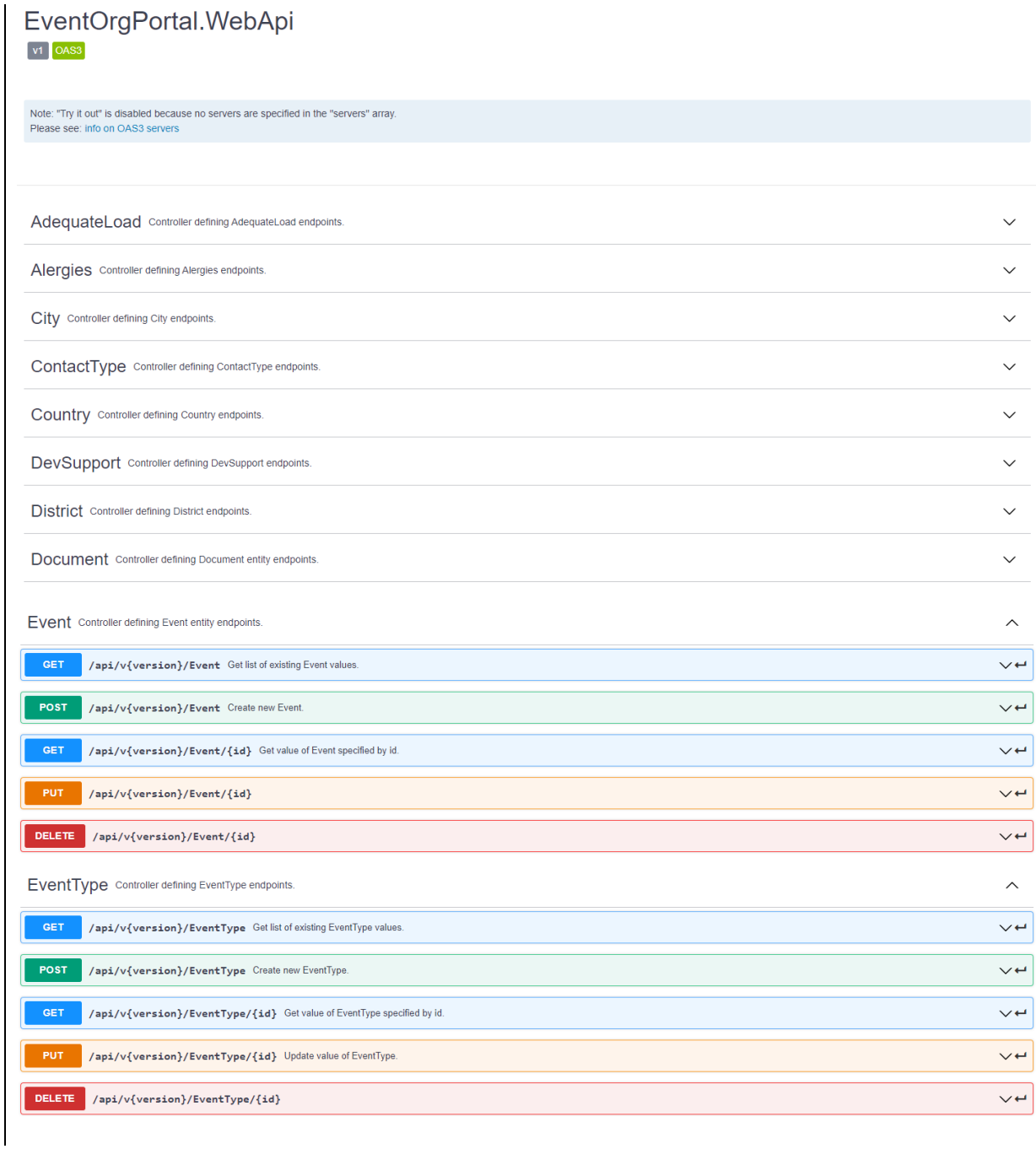
O katalozích můžeme říct, že obsahují minimální objem dat a jejich záznamy nemají samy o sobě význam. Ten získají tím, že se na ně naváže nějaká entita, u níž jednoznačně určí platnou hodnotu. Například pohlaví, u rodiče, nebo kód zdravotní pojišťovny u dítěte.

Každý katalog má navíc položku Active, která zajišťuje určitou zpětnou kompatibilitu. Katalog totiž není vhodné mazat, pokud byl nějakou dobu používán a stále existují záznamy entit, které se na něj vážou. Pokud bychom katalog smazali, budou tyto entity poškozeny. Položka *Active* s hodnotou *false* určuje, že záznam v katalogu je „readonly“, tedy že může být zobrazen při výpisu položek nějaké entity, ale nelze jej už žádné další entitě přiřadit.

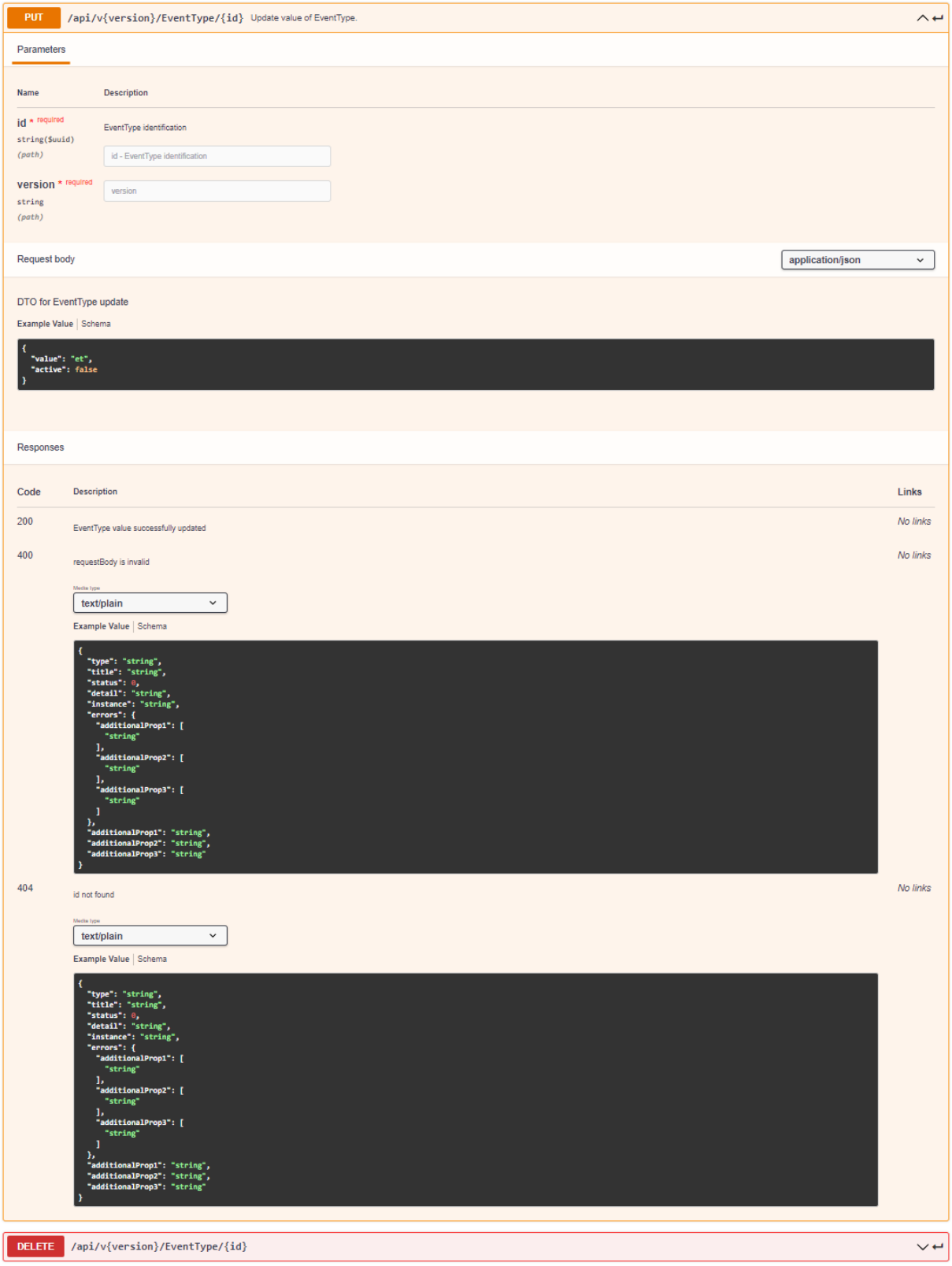
5.4 Swagger dokumentace

Jak jsme si mohli všimnout, ukázky (Ukázka kódu 1 strana 14) a (Ukázka kódu 4 strana 16) obsahují bloky komentujícího kódu. Kromě zpřehlednění vývoje aplikace mají tyto komentáře ještě jednu zásadní funkci, využívají se totiž k automatickému generování dokumentace, pomocí knihovny Swashbuckle (Microsoft, 2021), (Morris, 2021).

Tato knihovna při spuštění programu vygeneruje vývojářskou dokumentaci celého API pro platformu Swagger, která je interaktivní a velmi podrobná, jak můžeme vidět na následujících obrázcích (Obrázek 7 strana 22), (Obrázek 8 strana 23).



Obrázek 7 - Swagger dokumentace, endpointy API



Obrázek 8 - Swagger dokumentace, detail endpointu

Konfiguraci pro dokumentaci našeho projektu můžeme najít v rámci projektu v namespace *EventOrgPortal.WebAPI.Swagger*.
Jak si dokumentaci otevřít, se dozvíte v kapitole 8.2.1 Swagger dokumentace API.

6 PHP knihovna

V tuto chvíli již známe databázovou strukturu aplikace a vysvětlili jsme, a konkrétně si popsali, webové API a jeho funkce. Než však začneme s tvorbou ukázkového frontendu, představíme si ještě naši automaticky generovanou PHP knihovnu.

6.1 Motivace

Primární funkcí frontendu je zobrazit data, která si uživatel vyžádá a umožnit mu s nimi interagovat (Wikipedia, 2021). Z toho plyne, že frontend musí dokázat zmiňovaná data získat, a jednou z možností, jakou lze takové funkcionality v jazyce PHP dosáhnout, je použití rozšiřující knihovny Curl[12].

V následující ukázce kódu můžeme vidět velmi jednoduchý způsob odeslání http *GetPostalCode* requestu pomocí funkcí z knihovny Curl a výpisu získané odpovědi.

```php
// build request
$r = curl_init(https://localhost:62150/api/v1.0/PostalCode)
curl_setopt($r, CURLOPT_RETURNTRANSFER, true);
curl_setopt($r, CURLOPT_HEADER, 0);

// execute request
$data = curl_exec($r);
curl_close($r);

// print results
echo $data;
```

Ukázka kódu 6 - PHP GetPostalCode http request

Stejného výsledku dokážeme docílit následující metodou, která je ale výrazně přehlednější.

```php
class HTTPRequester {
    /**
     * @description Make HTTP-GET call
     * @param       $url
     * @param       array $params
     * @return      HTTP-Response body or an empty string if the request fails or
is empty
     */
    public static function HTTPGet($url, array $params) {
        $query = http_build_query($params);
        $ch    = curl_init($url.'?'.$query);
        curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($ch, CURLOPT_HEADER, false);
        $response = curl_exec($ch);
        curl_close($ch);
        return $response;
    }
    ...
}
```

Externí kód 1 - kód z webu StackOwerflow (mwatzer, 2018)

[12] Knihovna Curl přináší snadnou tvorbu a odesílání HTTP requestů, a integraci do webových API (Borboa, 2021).

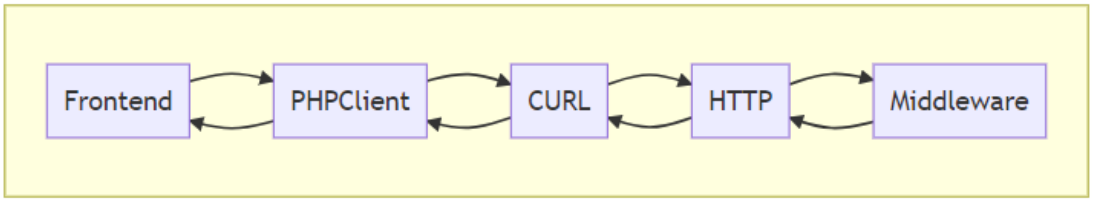
Struktura kódu z ukázky (Externí kód 1 - kód z webu StackOwerflow) může svou konstrukcí připomínat strukturu controlleru. Snadno si tak můžeme představit, že by se zobrazená třída nejmenovala *HTTPRequester* ale například *HTTPPostalCodeRequester* a kromě metody *HTTPGet* obsahovala ještě metody *HTTPCreate, HTTPUpdate* a *HTTPDelete*. Obsahovala by tak http request pro každý endpoint katalogu PostalCode a fakticky by jej tak zpřístupnila pro použití na frontendu.

6.2 Kontext knihovny

Množinu tříd, které vznikly aplikací principu z minulého odstavce, na každou entitu a katalog z naší databáze, nazýváme PHP klient.

6.2.1 PHP Klient

Hlavními funkcemi PHPClienta jsou vytváření http requestů na naše API a deserializace dat http responsů z formátu JSON[13]. Je proto důležité si pro jeho použití korektně zařadit jeho pozici v komunikaci mezi Frontendem a Middlewarem, kterou znázorňuje následující diagram.



Obrázek 9 - Frontend-Middleware komunikace

6.2.2 Automatická generace

Jak již víme, struktura třídy z PHP klienta je velmi podobná struktuře controlleru. V kapitole 5.2.3 Controller, konkrétně v ukázce - ChildController jsme si zmínili, že třída Controller a její endpointy obsahují mnoho různých anotací. Znovu tyto anotace zmiňuji proto, že z nich dokážeme získat veškeré informace potřebné pro vytvoření funkce z následující ukázky:

```
namespace EventOrgClient {
    const url = "https://localhost:62150/";
    class Country
    {
        const route = url . "api/v1.0/Country";
        public static function GetList()
        {
            $response = HTTPRequester::HTTPGet(self::$route);
            return json_decode($response);
        }
    }
}
```

Ukázka kódu 7 - PHP Get na katalogu Country

[13] JSON je minimalistický formát pro přenos dat. (Wikipedia, 2021)

Samotné generování provádí třída Generator z namespace *EventOrgPortal.EfGenerator*. Používá k tomu knihovnu System.Reflection, která umožňuje získat veškeré informace o typech, třídách, metodách, … , z aktuální assembly[14] (Microsoft, 2021).

Vygenerovanou knihovnu lze nalézt v příloze 4)a ClientLib.php a HTTPRequester.php - Vygenerovaná PHP knihovna

6.3 Dokumentace knihovny

Abychom práci na frontendu maximálně zjednodušili, generujeme kromě všech PHP klientů ještě doplňující dokumentaci, která se zaměřuje na jejich jednotlivé metody, a pro každou z nich demonstruje, jak s její pomocí vytvořit http request a jaké odpovědi na tento request můžeme dostat.

Vygenerovanou dokumentaci lze nelézt v příloze 4)b ClientLibDoc.md - Použití endpointů API pomocí funkcí z knihovny.

[14] Assembly je kolekce typů a veškerých dalších zdrojů, které po sestavení vytvoří logický funkční celek (Microsoft, 2021).

7 Ukázkové použití

V této kapitole se zaměříme na implementaci ukázkového frontendu k našemu webovému api. Nebudeme se pokoušet o vytvoření celého webu ani pokročilého grafického zpracování. Místo toho se zaměříme na základní funkcionalitu a více otestujeme možnosti, které nabízejí naše katalogy.

7.1 Popis frontendu

Rodič se zaregistruje tak, že na úvodní stránce klikne na tlačítko „Registrace rodiče“. Bude přesměrován na stránku s registračním formulářem. Které položky z formuláře jsou povinné lze zjistit použitím přílohy 3) swagger.json - Zdrojový soubor pro swagger dokumentaci. Zpět na úvodní stránku se lze vrátit kliknutím na tlačítko „Hlavní stránka“.

Svůj profil si rodič zobrazí kliknutím na tlačítko „Profil rodiče“. Bude přesměrován na stránku, která ho vyzve k zadání jeho primárního klíče – tedy GUIDu, který byl vygenerován při jeho registraci.
Po jeho úspěšném zadání, zobrazí stránka výpis údajů jeho profilu a tabulku s přehledem jeho dětí. Pod tabulkou se nachází tlačítko, které mu umožní přidat do profilu dítě nové. Kliknutí na něj ho přesměruje na registrační formulář dítěte.

Registrace dítěte je identická registraci rodiče. Je třeba vyplnit formulář a odeslat jej.
Po úspěšném vytvoření nového dítěte, se toto zobrazí v přehledové tabulce na stránce profil rodiče.

Identickým způsobem jako profil rodiče a dítěte vytváříme i událost naší organizace.
Do registračního formuláře vyplníme všechny položky, které chceme uživatelům ukázat, minimálně však ty, které jsou povinné. (I tuto informaci samozřejmě získáme použitím přílohy 3) .)

Použitím tlačítka „Správa katalogů“ se dostaneme na stránku, která pro každý katalog obsahuje tabulku podobnou této.

Pohlaví

Id	Pohlaví	Aktivní		
f7745780-8887-40f0-831f-f9dd7f2879cc	Muž	Aktivní	Uložit	Smazat
a306d746-0b02-4704-96b1-2282005adac8	Žena	Aktivní	Uložit	Smazat
b32d02e2-5396-4289-9549-bf2f135eea7c	Nebinární	Aktivní	Uložit	Smazat
			Vytvořit	

Obrázek 10 - Administrace katalogu Gender

Tabulka zobrazuje všechny záznamy uložené v katalogu[15] a umožňuje každou z nich upravit, aktivovat nebo naopak deaktivovat a smazat. Dovoluje samozřejmě také vytvoření zcela nového záznamu.

[15] Je důležité zmínit, že použití takovéto stránky je vhodné, jelikož se jedná o ukázku funkcionality projektu. V hotové aplikaci by to mohlo být problematické v případě, že by katalogy byly skutečně používány a obsahovaly potenciálně i tisíce hodnot. Pro takový případ by samozřejmě byla na místě jiná implementace.

7.2 Využití PHP knihovny

Naše PHP knihovna z přílohy[16] posloužila při implementaci tohoto frontendu svému účelu dobře.

Voláním jejích funkcí se nám podařilo získat si veškerý databázový kontext, který jsme potřebovali. Jako dobrý teoretický příklad poslouží její využití pro stránku zajišťující administraci katalogů. Její funkce jsou zde použity nejen pro zobrazení, ale skrývají se ze všemi operacemi, které jsou nad záznamy katalogů dostupné.

Jako praktický příklad si prohlédněme následující ukázku kódu z vytvořujícího formuláře pro událost. Pomocí knihovních funkcí si ukládáme hodnoty několika katalogů, které jsou pak použity pro naplnění výběrových prvků ve formuláři. (Pro přehlednost jsou volání knihovních funkcí zvýrazněna modrou barvou.

```
<?php
    include_once "..\common.php";
    include_once "..\EventOrgClient.php";

/*
 * Functions for retrieving catalog values.
 */
    $adequate_loads = \EventOrgClient\AdequateLoadService::GetList();
    function fill_adequate_load_catalog($adequate_loads) {
        echo "<option value=\"\">...</option><br>";
        foreach ($adequate_loads->body->data as $item) {
            if ($item->active) {
                echo "<option value=\"{$item->id}\">{$item->value}</option><br>";
            }
        }
    }

    $event_types = \EventOrgClient\EventTypeService::GetList();
    function fill_eventType_catalog($eventTypes) {
        echo "<option value=\"\">...</option><br>";
        foreach ($eventTypes->body->data as $item) {
            if ($item->active) {
                echo "<option value=\"{$item->id}\">{$item->value}</option><br>";
            }
        }
    }

    $assigned_units = \EventOrgClient\UnitService::GetList();
    function fill_assigned_units_catalog($assigned_units) {
        foreach ($assigned_units->body->data as $item) {
            if ($item->active) {
                echo "<option value=\"{$item->id}\">{$item->value}</option><br>";
            }
        }
    }

...

?>
```

Ukázka kódu 8 - Úryvek z create_event.php

[16] 4)a ClientLib.php a HTTPRequester.php - Vygenerovaná PHP knihovna a zavaděč standartních http funkcí

8 Uživatelská příručka

8.1 Jak aplikaci lokálně spustit

Pro spuštění aplikace potřebujeme mít kromě Visual Studia nainstalovaný ještě nástroj Docker Desktop[17]. Pro ten obsahuje projekt konfigurační soubor „docker-compose“, který při spuštění vytvoří v Dockeru všechny potřebné kontejnery.
Aplikaci je tedy dostatečné spustit přímo z Visual Studia a zbytek se nainstaluje a nakonfiguruje automaticky.

Je důležité zmínit, že s každým sestavením kontejnerů Dockeru se socket, na kterém je aplikace dosažitelná, změní. Tuto změnu je potřeba reflektovat při volání aplikace z frontendu nebo z nástroje Postman tím, že v adrese „localhost:SOCKET_NUMBER/…“ nastavíme aktuální hodnotu.

8.2 Dokumentace

8.2.1 Swagger dokumentace API

Nabízejí se dvě možnosti přístupu k této dokumentaci. První spočívá ve spuštění projektu „docker-compose“, který je nakonfigurován tak, aby aktuálně vygenerovanou dokumentaci otevřel ve výchozím webovém prohlížeči.

Druhá je složitější, ale nabízí možnost čtení dokumentace bez nutnosti spuštění aplikace. To je možné díky použití nástroje SwaggerHub (https://swagger.io/tools/swaggerhub/). Nástroj nabízí možnost bezplatného vytvoření účtu a po přihlášení import souboru z přílohy 3) swagger.json - Zdrojový soubor pro swagger dokumentaci.

8.2.2 Markdown dokumentce funkcí PHP knihovny

Dokumentace[18] se generuje společně s knihovnou[19] spuštěním projektu *EventOrgPortal.FeGenerator*. Následně ji můžeme najít v cestě „PHP/doc/EventOrgClientDoc.md“.

8.3 Jak naplnit katalogy

8.3.1 Seed a funkce na DevSupportControlleru

Většina katalogů je předvyplněna z takzvaných seedů (textových souborů, ze kterých se vyplňuje obsah databáze při jejím sestavení). Příkladem je katalog Gener, jehož hodnoty jsou předdefinovány v souboru na cestě „EventOrgPortal.Dal/Seeds/GenderSeed.txt“.

Druhý způsob inicializace katalogů spočívá ve využití metod definovaných na DevSupportControlleru, které umožňují naplnění systému katalogů reprezentujících adresu a organizační hierarchii. Seedování těchto katalogů se provádí manuálně proto, aby se při inicializaci databáze nepřetěžoval Entity Framework.

8.3.2 Vlastní implementace CreateCatalogRequest

Jakákoli implementace frontendu, která je schopna poslat http create request na příslušný endpoint *CreateCatalogNameItem* umožní ukládat do tohoto katalogu data.

---

[17] Nástroj nezaručuje plnou podporu pro edici operačního systému Windows Home. (Zaručena pro verzi Professional).
[18] Příloha 4)b ClientLibDoc.md - Použití endpointů API pomocí funkcí z knihovny
[19] Příloha 4)a ClientLib.php a HTTPRequester.php - Vygenerovaná PHP knihovna a zavaděč standartních http funkcí

8.3.3 Catalog admin

S prací poskytovaný ukázkový frontend obsahuje stránku zaměřenou na administraci katalogů[20]. Ta mimo jiné umožňuje právě vkládání nových hodnot a lze ji tedy za tímto účelem použít. Jak už jsme si ale vysvětlili[21] využití této implementace ve velkých projektech sebou nese rizika.

[20] Příloha 4)c catalog-admin.php - Nástroj pro správu katalogů
[21] V odstavci následujícím po obrázku Obrázek 10 na straně 25

9 Závěr

Navrhl a vytvořil jsem databázi, jejíž objekty jsou zaměřeny na potřeby volnočasových organizací. Mezi objekty existuje mnoho vazeb, které používání databáze usnadní.

Vytvořil jsem 22 katalogů. Řadí se mezi ně katalogy jednoduché, jako například Gender nebo EventType, a také komplexní, jako jsou skupiny Country, Region, District, City, PostalCode a Unit, SubUnit a Group.

Napsal jsem skript pro generování katalogů, s jehož pomocí je možno vytvořit další rozšiřující katalogy v rámci minut namísto hodin.

Dokončil jsem jednotlivá webová API entit a katalogů. Kromě základních funkcí CRUD jsem u těch, kde to bylo vhodné, implementoval funkce speciální, jako například předdefinovanou filtraci nebo získávání podmnožiny dat.

Oproti implementačním cílům jsem navíc napsal automatický generátor PHP knihovny a její dokumentaci pomocí C# knihovny System.Reflection. Knihovna definuje funkci pro každý endpoint každého controlleru a umožňuje tak jejich jednoduché zavolání.

Pomocí této knihovny jsem vytvořil ukázkový frontend, který demonstruje funkčnost celé mojí aplikace. Po grafické stránce je tento frontend primitivní, ale jakékoli jeho potenciální vylepšení je mimo rozsah této práce.

Aplikace je napsána tak, aby byla snadno rozšiřitelná. V případě zájmu o další katalogy, lze pomocí CreatorScriptu snadno vyhovět. Problematická by nebyla ani implementace dalších entit či funkcí.

Dále bych rád zmínil, čemu jsem se v práci nevěnoval, a proč. Vyhýbal jsem se implementaci funkcí, které by nepřinesly nic nového. Kupříkladu autorizace je známý koncept a existuje mnoho návodů a způsobů, jak jej docílit. Mohl jsem také implementovat napojení aplikace na platební bránu nebo třeba automatické odesílání emailů, ale z hlediska problematiky popsané v úvodu práce by to nemělo velký užitek.

V poslední řadě se chci zmínit o entitách Organizer a Contact uvedených v databázovém modelu. Organizer ztratil svůj význam pro tuto práci s rozhodnutím neimplementovat autorizaci. Neimplementoval jsem proto jeho API, nicméně přesto je součástí databáze a je jedním z možných rozšíření práce. Entita Contact už s myšlenkou na rozšíření práce vznikla. Její funkcí je přiřadit k dítěti dodatečnou kontaktní osobu pro účast na konkrétní akci. Stejně jako Organizer postrádá API, ale součástí databáze už je.

Definovat konečnost této práce je náročný úkol. Vždy se totiž dá vymyslet další entita, katalog či funkce, která by mohla být její součástí. Reflektuje nicméně základní modely výrazně větších projektů a v konečném důsledku by se mohla stát veřejně použitelným projektem.

10 Seznam použitých pramenů

Borboa, Zach. 2021. php-curl-class. *Github.* [Online] 17. 6 2021. [Citace: 18. 7 2021.] https://github.com/php-curl-class/php-curl-class.

ČESKO. 2000. § 10 zákona č. 258/2000 Sb., o ochraně veřejného zdraví a o změně některých souvisejících zákonů. *Zákony pro lidi.cz.* [Online] 11. 8 2000. [Citace: 20. 7 2021.] https://www.zakonyprolidi.cz/cs/2000-258#p10.

—. 2000. § 12 zákona č. 258/2000 Sb., o ochraně veřejného zdraví a o změně některých souvisejících zákonů. *Zákony pro lidi.cz.* [Online] 11. 8 2000. [Citace: 20. 7 2021.] https://www.zakonyprolidi.cz/cs/2000-258#p12.

—. 2001. fragment #f5146442 vyhlášky č. 106/2001 Sb., Ministerstva zdravotnictví o hygienických požadavcích na zotavovací akce pro děti. *Zákony pro lidi.cz.* [Online] 26. 3 2001. [Citace: 20. 7 2021.] https://www.zakonyprolidi.cz/cs/2001-106#f5146442.

EntityFrameworkTutorial.net. 2020. Context Class in Entity Framework. *EntityFrameworkTutorial.* [Online] 2020. [Citace: 17. 7 2021.] https://www.entityframeworktutorial.net/basics/context-class-in-entity-framework.aspx.

Chavez, Brian. 2021. Bogus. *github.com.* [Online] 33.0.2, 21. 2 2021. [Citace: 13. 7 2021.] https://github.com/bchavez/Bogus.

IBM, Corporation. 2021. HTTP responses. *IBM.* [Online] 2021. [Citace: 17. 7 2021.] https://www.ibm.com/docs/en/cics-ts/5.3?topic=protocol-http-responses.

Microsoft. 2021. Assemblies in .NET. *Microsoft Docs.* [Online] 2021. [Citace: 20. 07 2021.] https://docs.microsoft.com/en-us/dotnet/standard/assembly/.

—. 2021. Get started with Swashbuckle and ASP.NET Core. *Microsoft Docs.* [Online] 2021. [Citace: 18. 7 2021.] https://docs.microsoft.com/en-us/aspnet/core/tutorials/getting-started-with-swashbuckle?view=aspnetcore-5.0&tabs=visual-studio.

—. 2021. Language Integrated Query (LINQ) (C#). *Microsoft Docs.* [Online] 2021. [Citace: 20. 7 2021.] https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/.

—. 2021. System.Reflection Namespace. *Microsoft Docs.* [Online] 2021. [Citace: 20. 7 2021.] https://docs.microsoft.com/en-us/dotnet/api/system.reflection?view=net-5.0.

Morris, Richard. 2021. Swashbuckle.AspNetCore. *Github.* [Online] v6.1.1, 16. 3 2021. [Citace: 18. 7 2021.] https://github.com/domaindrivendev/Swashbuckle.AspNetCore.

mwatzer. 2018. How do I send a POST request with PHP? *Stack Overflow.* [Online] 6. 3 2018. [Citace: 18. 7 2021.] https://stackoverflow.com/a/45494300.

TutorialsTeacher.com. 2020. Routing in MVC. *TutorialsTeacher.* [Online] 2020. [Citace: 14. 7 2021.] https://www.tutorialsteacher.com/mvc/routing-in-mvc.

Wikipedia, contributors. 2021. API. *Wikipedia, The Free Encyclopedia.* [Online] 22. 7 2021. [Citace: 22. 7 2021.] https://en.wikipedia.org/w/index.php?title=API&oldid=1034884558. 1034884558.

—. 2021. Front-end web development. *Wikipedia, The Free Encyclopedia.* [Online] 11. 7 2021. [Citace: 18. 7 2021.] https://en.wikipedia.org/w/index.php?title=Front-end_web_development&oldid=1033074480. 1033074480.

—. 2021. JSON. *Wikipedia, The Free Encyclopedia.* [Online] 16. 7 2021. [Citace: 19. 7 2021.] https://en.wikipedia.org/w/index.php?title=JSON&oldid=1033820857. 1033820857.

—. 2021. Representational state transfer. *Wikipedia, The Free Encyclopedia.* [Online] 24. 6 2021. [Citace: 22. 7 2021.] https://en.wikipedia.org/w/index.php?title=Representational_state_transfer&oldid=1030195274. 1030195274.

—. 2021. Universally unique identifier. *Wikipedia, The Free Encyclopedia.* [Online] 12. 7 2021. [Citace: 17. 7 2021.] https://en.wikipedia.org/w/index.php?title=Universally_unique_identifier&oldid=1033220213. 1033220213.

Wikipedie, přispěvatelé. 2020. Lorem ipsum. *Wikipedie: Otevřená encyklopedie.* [Online] 9. 8 2020. [Citace: 13. 7 2021.] https://cs.wikipedia.org/w/index.php?title=Lorem_ipsum&oldid=18911372. 18911372.

—. 2021. Uniform Resource Locator. *Wikipedie: Otevřená encyklopedie.* [Online] 13. 5 2021. [Citace: 2021. 7 14.] https://cs.wikipedia.org/w/index.php?title=Uniform_Resource_Locator&oldid=19824605. 19824605.

11 Seznam použitých zkratek

REST - Representational state transfer

API - Application Programming Interface

CRUD - Create, Read, Update, Delete

URL - Uniform Resource Locator

DTO - Data Transfer Object

GUID - GUID a UUID Globally Unique Identifier Universally Unique Identifier

Curl - Client URL

JSON - JavaScript Object Notation

12 Seznam obrázků

Obrázek 1 - implementace katalogu Gender 6
Obrázek 2 - GetContatctTypeDto v aplikaci diff 7
Obrázek 3 - GetGenderDto v aplikaci diff 7
Obrázek 4 - Přehled databázových entit 10
Obrázek 5 - Schéma vazeb 11
Obrázek 6 - Cesta http requestu 12
Obrázek 7 - Swagger dokumentace, endpointy API 20
Obrázek 8 - Swagger dokumentace, detail endpointu 21
Obrázek 9 - Frontend-Middleware komunikace 23
Obrázek 10 - Administrace katalogu Gender 25

13 Seznam ukázek kódu

Ukázka kódu 1 - GetChildInfo message 14
Ukázka kódu 2 - Definice GetChildInfo message fakeru 14
Ukázka kódu 3 - CreateChild message validator 15
Ukázka kódu 4 - ChildController 16
Ukázka kódu 5 - filtrace dle org. hierarchie 17
Ukázka kódu 6 - PHP GetPostalCode http request 22
Ukázka kódu 7 - PHP Get na katalogu Country 23
Ukázka kódu 8 - Úryvek z create_event.php 26

14 Přílohy

1) EventOrgPortal.zip - Zdrojový kód projektu
2) CatalogCreator.zip - Zdrojový kód a šablona
3) swagger.json - Zdrojový soubor pro swagger dokumentaci
4) PHP.zip – PHP knihovna, dokumentace a ukázkový frontend
    a. ClientLib.php a HTTPRequester.php - Vygenerovaná PHP knihovna a zavaděč standartních http funkcí
    b. ClientLibDoc.md - Použití endpointů API pomocí funkcí z knihovny
    c. catalog-admin.php - Nástroj pro správu katalogů