

Vývoj Blazor balíčku pro správu a zobrazení školního rozvrhu

Filip Chromečka

Bakalářská práce
2025



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

Ústav informatiky a umělé inteligence

Akademický rok: 2024/2025

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	Filip Chromečka
Osobní číslo:	A22068
Studijní program:	B0613A140020 Softwarové inženýrství
Forma studia:	Prezenční
Téma práce:	Vývoj Blazor balíčku pro správu a zobrazení školního rozvrhu
Téma práce anglicky:	Development of a Blazor Package for School Timetable Management

Zásady pro vypracování

- V teoretické části popište webový framework Blazor. Popište také způsob komunikace s JavaScriptem (JS Interops).
- Podrobně definujte požadavky na balíček pro správu školního rozvrhu.
- Implementujte řešení v jazyce C# (.NET).
- Řádně otestujte pomocí libovolného UT frameworku (např. bUnit).
- Publikujte hotové řešení do správce balíčků nuget.org a popište kroky nutné k publikaci.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. MAYO, Joe. C# Cookbook. O'Reilly Media, 2021. ISBN 9781492093671.
2. GUPTA, Arun. Building End-to-End Apps with C# 11 and .NET 7. Bpb Publications, 2023. ISBN 9789355516374.
3. WRIGHT, Toi B. a HANSELMAN, Scott. Blazor WebAssembly by Example. Packt Publishing, 2021. ISBN 9781800563933.
4. ENGSTRÖM, Jimmy. Web Development with Blazor. Packt Publishing, 2021. ISBN 9781800202580.
5. FRAIN, Ben. Responsive Web Design with HTML5 and CSS. Packt Publishing, 2022. ISBN 9781803231723.

Vedoucí bakalářské práce: **Ing. Adam Ulrich**
Ústav matematiky

Datum zadání bakalářské práce: **27. října 2024**
Termín odevzdání bakalářské práce: **2. června 2025**



L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 17. listopadu 2024

PROHLÁŠENÍ AUTORA BAKALÁŘSKÉ PRÁCE

Beru na vědomí, že

- odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb., v platném znění bez ohledu na výsledek obhajoby;
- bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému a bude dostupná k nahlédnutí;
- jedno vyhotovení bakalářské práce v listinné podobě bude ponecháno Univerzitě Tomáše Bati ve Zlíně k uložení;
- na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- podle § 60 odst. 1 autorského zákona má Univerzita Tomáše Bati ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- podle § 60 odst. 2 a 3 mohu užít své dílo – bakalářskou práci – nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tj. k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá; neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji, že

- jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem řádně citoval; v případě publikace výsledků budu uveden jako spoluautor;
- při tvorbě této práce jsem využil nástroj generativního modelu AI ChatGPT; <https://chatgpt.com> za účelem formální úpravy textu. Obsah výstupu jsem řádně zkontroloval a přebírám za něj plnou zodpovědnost;
- odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou obsahově totožné.

Ve Zlíně, dne

.....
podpis autora

ABSTRAKT

Tato bakalářská práce se zabývá tvorbou knihovny pro správu a zobrazení rozvrhu v .NET frameworku Blazor, který je popsán v teoretické části. V praktické části jsou definovány klíčové požadavky na knihovnu, implementace, testování a publikace pro veřejné použití. Práce také popisuje různé způsoby použití knihovny.

Klíčová slova: Blazor, rozvrh, balíček

ABSTRACT

This bachelor thesis deals with the creation of a library for managing and displaying the timetable in the .NET framework Blazor, which is described in the theoretical part. The practical part defines the key requirements for the library, implementation, testing and publishment for public use. The thesis also describes different ways of using the library.

Keywords: Blazor, timetable, package

Rád bych poděkoval panu Ing. Adamovi Ulrichovi za jeho cenné rady a připomínky v průběhu psaní této bakalářské práce.

OBSAH

ÚVOD.....	11
I TEORETICKÁ ČÁST	12
1 BLAZOR.....	13
1.1 HOSTING MODELS.....	13
1.2 KOMPONENTY	13
1.3 ŽIVOTNÍ CYKLUS	14
1.3.1 OnInitialized a OnInitializedAsync.....	14
1.3.2 OnParametersSet a OnParametersSetAsync	14
1.3.3 OnAfterRender a OnAfterRenderAsync	14
1.4 KOMUNIKACE.....	14
1.4.1 Parametry	14
1.4.2 EventCallbacky	15
1.5 DIREKTIVY	15
1.6 RENDERFRAGMENTY	15
1.7 INTEGRACE S JAVASCRIPTEM.....	16
II PRAKTICKÁ ČÁST.....	17
2 NÁVRH	18
2.1 POŽADAVKY.....	18
2.2 ZOBRAZENÍ UDÁLOSTÍ.....	18
2.2.1 Denní zobrazení.....	19
2.2.2 Týdenní zobrazení	19
2.2.3 Měsíční zobrazení	20
2.3 NAVIGACE PO ČASOVÉ OSE.....	21
2.4 SPRÁVA UDÁLOSTÍ	21
2.4.1 Tvorba události.....	21
2.4.2 Aktualizace události	21
2.4.3 Odstranění události.....	21
2.5 IMPORT A EXPORT UDÁLOSTÍ.....	21
3 IMPLEMENTACE	23
3.1 STRUKTURA	23
3.2 GENERIČNOST	24
3.3 STRUKTURA ROZVRHU	26
3.4 ZOBRAZENÍ UDÁLOSTÍ.....	26
3.4.1 Služby.....	27
3.4.2 Vykreslení	28
3.5 SPRÁVA UDÁLOSTÍ	29

3.5.1	Drag and drop	31
3.5.2	Přidané vlastností.....	32
3.6	DATOVÁ VÝMĚNA	33
3.6.1	Export	33
3.6.2	Import	35
3.7	LOKALIZACE.....	36
3.8	TESTOVÁNÍ.....	37
3.9	PUBLIKACE.....	38
3.10	REPOZITÁŘ	39
4	POUŽITÍ	40
4.1	PŘEDPOKLADY	40
4.2	TVORBA PROJEKTU	40
4.3	MINIMAL SETUP.....	40
4.4	KONFIGURACE OBECNÉHO NASTAVENÍ.....	43
4.5	ZPRACOVÁNÍ AKCÍ.....	44
4.6	IMPORT A EXPORT.....	45
4.7	DODATEČNÉ VLASTNOSTI.....	46
4.7.1	Předpřipravené komponenty.....	47
4.7.2	Validace.....	48
4.7.3	Vlastní vstupní komponenta.....	49
4.8	LOKALIZACE.....	50
	ZÁVĚR.....	51
	SEZNAM POUŽITÉ LITERATURY	53
	SEZNAM OBRÁZKŮ	54
	SEZNAM UKÁZEK KÓDU	55
	SEZNAM TABULEK	56
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	57

ÚVOD

Jak již název napovídá, tato práce se věnuje frameworku Blazor. Ten si postupně získává pozornost jako pomůcka pro vývoj webových aplikací, ale jeho popularita stále zaostává za dominantními frameworky, kupříkladu React. Tento nedostatek popularity se odráží v omezené nabídce balíčků a knihoven, které by usnadnily a zrychlily vývoj. Cílem této práce je překlenout tuto mezeru vytvořením veřejnosti dostupné, modulární a konfigurovatelné knihovny, která umožní zobrazování a správu událostí v rozvrhu napříč různými typy projektů.

Práce je strukturovaná do specifických částí, kdy je nejprve obecně popsán samotný framework Blazor. Poté jsou definovány požadavky na knihovnu. V části implementace jsou popsány vybrané části knihovny, testování a publikace. Nakonec je popsáno podrobné využití knihovny v různých případech.

I TEORETICKÁ ČÁST

1 BLAZOR

Blazor je moderní framework vyvinutý společností Microsoft určený pro vývoj SPA. Umožňuje vývojářům využít znalosti jazyka C# a .NET ekosystému k vytváření dynamických a interaktivních uživatelských rozhraní bez nutnosti použít JavaScript [1].

1.1 Hosting models

Aplikaci využívající tento framework je možné vykreslovat různými způsoby, které umožňují flexibilitu v nasazení a optimalizaci výkonu dle specifických potřeb projektu:

- **Server-Side Rendering (SSR)**, který umožňuje vykreslení statických stránek na straně serveru,
- **Interactive Server**, kde veškerá aplikační logika běží na serveru a klient obdrží pouze strukturu DOM prostřednictvím SignalR, což je technologie využívající WebSokety pro obousměrnou komunikaci v reálném čase,
- **Interactive WebAssembly**, který představuje WebAssembly aplikaci běžící zcela na straně klienta bez jakékoli komunikace se serverem. Tento přístup je ideální pro aplikace, které vyžadují bezproblémovou funkčnost i při výpadku připojení k internetu nebo pro zamezení zátěže serveru,
- **Interactive Auto**, což je režim, ve kterém si framework automaticky vybere nejefektivnější způsob interaktivity pro danou komponentu [1;2;3].

1.2 Komponenty

Komponenty představují základní stavební bloky aplikace. Jedná se o soubory s příponou .razor, které automaticky dědí ze třídy ComponentBase.

Razor komponenta se skládá ze tří hlavních částí:

- Direktivy, představující vestavěná makra ovlivňující chování,
- Razor syntax, který kombinuje HTML s C# pro vykreslení UI,
- code blok, ve kterém je definována logika komponenty skrze C# kód

Pro lepší přehlednost kódu lze komponentu rozdělit do samostatných souborů dle struktury:

- Razor syntax,
- code-behind,

- kaskádové styly,
- JavaScript.

Díky tomuto postupu lze oddělit zodpovědnost jednotlivých částí komponenty do samostatných souborů a zároveň ji ponechat jako jeden kus [1;4].

1.3 Životní cyklus

Životní cyklus zahrnuje řadu událostí, které umožňují tvorbu specifického chování při různých fázích existence komponenty. K tomu jsou implementovány tři základní metody. Každá z nich má svou synchronní i asynchronní podobu, přičemž synchronní varianty jsou prováděny před asynchronními [1].

1.3.1 OnInitialized a OnInitializedAsync

Tyto metody se volají při inicializaci komponenty, podobně jako konstruktor u normálních tříd. Jsou vyvolány pouze jednou při prvním vykreslení komponenty, tudíž jsou vhodné pro nastavení počátečního stavu či načtení potřebných dat [1].

1.3.2 OnParametersSet a OnParametersSetAsync

Tato metoda je vyvolána v potomkovi, změní-li se hodnota parametru jemu předaném z komponenty rodičovské. Umožňuje tak reagovat na změny a aktualizovat svůj stav [1].

1.3.3 OnAfterRender a OnAfterRenderAsync

Je vyvolána po každém vykreslení komponenty nebo její části. Zahrnuje příznak **firstRender** indikující prvotní vykreslení komponenty. Vhodná je pro operace, které vyžadují, aby bylo UI již plně vykresleno [1]. Příkladem je integrace s JavaScriptem.

1.4 Komunikace

Efektivní komunikace mezi komponentami je klíčová pro vytváření složitých aplikací. K tomuto účelu v Blazoru slouží parametry a event callbacky.

1.4.1 Parametry

Rodičovské komponenty předávají své hodnoty potomkům prostřednictvím parametrů, které jsou označeny atributem **Parameter**. Tyto parametry jsou z pohledu potomka neměnné a slouží k přijímání vstupních dat nebo nastavení konfigurace komponenty.

V situacích, kdy je nutné předat stejný objekt více úrovněmi komponentového stromu, lze využít kaskádované parametry označené atributem **CascadingParameter**. Rodičovská komponenta deklaruje tag **CascadingValue**, což umožní veškerým podřízeným komponentám přijmout objekt tímto tagem předávaným, aniž by bylo nutné explicitně předávat parametr skrz každou úroveň komponentového stromu [3;5].

1.4.2 EventCallbacky

Pro umožnění komunikace rodičovských a podřazených komponent při událostech, jako je interakce uživatele, se využívá struktury **EventCallback**. Podřazená komponenta může pomocí **EventCallback** vyvolat metody definované v rodičovské komponentě, čímž informuje rodiče o změně stavu nebo jiné relevantní události [3].

1.5 Direktivy

Direktivy, označované symbolem **@**, jsou speciální instrukce používané v komponentách:

- **@using**, která specifikuje zahrnutí jmenného prostoru,
- **@inherits**, pro možnosti dědit z jiné komponenty,
- **@typeparam**, díky které je možné komponentu definovat generickým typem,
- **@attribute**, která dovoluje aplikovat atributy stejným způsobem jako na třídy.

Kromě výše uvedených základních direktiv nabízí Blazor také speciální direktivy pro definici nad HTML prvky:

- **@bind**, umožňující obousměrné datové vazby mezi HTML prvky a proměnnými v komponentě,
- **@on[event]**, kterým lze zpracovat konkrétní události na HTML prvku [3].

1.6 Renderfragmenty

RenderFragment je komplexní typ využíváný k definování opakovaně použitelných částí UI. Obvykle jsou využity s atributem **Parameter**, aby bylo možné do komponenty vkládat dynamický markup [3].

1.7 Integrace s JavaScriptem

Ikdyž je Blazor propagován jako náhrada JavaScriptu, existují scénáře, kdy se jeho použití nelze vyhnout, například při potřebě integrace JS knihovny nebo nativní webové API.

Pomocí atributu **JSInvokable** lze zpřístupnit .NET metodu pro volání z JavaScriptu. JS kód může tyto metody zavolat pomocí **DotNet.invokeMethodAsync**.

Pro opačný směr komunikace, tj. volání JS funkcí z .NET kódu, Blazor poskytuje injektovatelné rozhraní **IJSRuntime**. Skrze metody **Invoke** a **InvokeAsync** je možné synchronně i asynchronně volat JS funkce, předávat argumenty a přijímat hodnoty [6].

II PRAKTICKÁ ČÁST

2 NÁVRH

V této části práce jsou definovány požadavky na chování a funkce knihovny. Jsou zde také zobrazeny prvotní návrhy uživatelského rozhraní (UI) vypracované v programu Figma.

2.1 Požadavky

Obecnými požadavky na knihovnu je:

- Zobrazení událostí v různých pohledech,
- správu událostí,
- import a export dat.

Pro zajištění využitelnosti knihovny napříč různými typy aplikací, jako školní rozvrhy, plánovače pracovních směn nebo osobní kalendáře, umožní z pohledu konfigurace:

- Specifikaci zobrazených dnů a měsíců,
- úpravu časového rámce dne,
- změnu jazyka UI,
- vlastní implementaci importu a exportu,
- využití vlastní třídy události,
- nastavování vlastností nad rámec povinných.

2.2 Zobrazení událostí

Události lze zobrazit ve třech režimech: denním, týdenním a měsíčním. Knihovna umožňuje konfigurovat, které dny a měsíce se zobrazí. Díky tomu lze vytvořit rozvrh specifický pro potřeby, jako je pracovní týden (pondělí až pátek) nebo školní rok (září až červen).

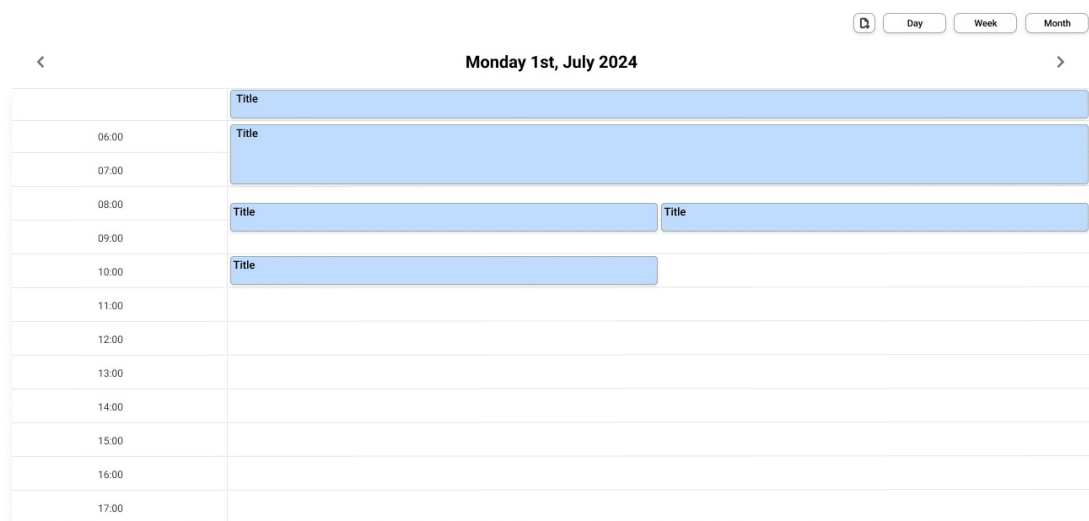
Dále je možné přizpůsobit časový rámec zobrazení, tedy stanovit počáteční a koncový čas. Tento časový rámec má vliv na generování mřížky, kde ovlivňuje počet jednotlivých časových buněk zobrazovaných v rozvrhu.

2.2.1 Denní zobrazení

Zobrazené UI je rozděleno na:

- Časový sloupec, který zobrazuje jednotlivé hodiny dle konfigurace časového rámce,
- sloupec s událostmi, který obsahuje časové buňky, ve kterých se jednotlivé události zobrazí.

První buňka ve sloupci s událostmi je určena pro mimo-rámcové události. Tyto události trvají více než jeden den nebo částečně spadají mimo definovaný časový rámec.

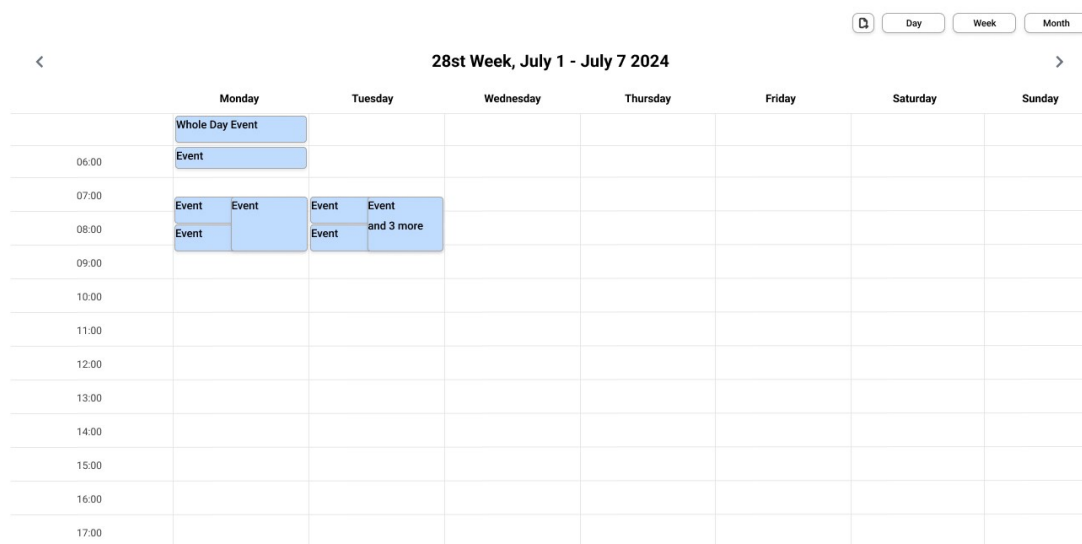


Obrázek 1: Návrh UI denního zobrazení v programu Figma

(zdroj: vlastní)

2.2.2 Týdenní zobrazení

Týdenní zobrazení je dosti podobné dennímu, rozdíl spočívá v počtu sloupců s událostmi, kdy pro každý definovaný den rozvrhu je zobrazen daný sloupec. Nad sloupcem je zobrazen název specifického dne, kdy se lze kliknutím na něj přepnout do denního zobrazení.



Obrázek 2: Návrh UI týdenního zobrazení v programu Figma

(zdroj: vlastní)

2.2.3 Měsíční zobrazení

Měsíční zobrazení poskytuje celkový přehled plánovaných událostí na celý měsíc. Pro vyplnění mřížky jsou zobrazeny i dny z předchozího a následujícího měsíce. Tyto dny události nezobrazují.



Obrázek 3: Návrh UI měsíčního zobrazení v programu Figma

(zdroj: vlastní)

2.3 Navigace po časové ose

Navigace po časové ose je umožněna šipkami v záhlaví, které uživateli dovolují přecházet mezi dny, týdny nebo měsíci. Pokud je zvolené datum neplatné, například při přechodu na den, který není v konfiguraci, automaticky se přechází na nejbližší platné datum.

2.4 Správa událostí

V této sekci jsou popsány postupy a pravidla pro tvorbu, úpravy a odstranění událostí, včetně specifik chování pro skupinové události.

Při tvorbě a aktualizaci musí událost splňovat následující pravidla:

- Vyplněný název události,
- validní den a měsíc dle konfigurace.

2.4.1 Tvorba události

Novou událost lze vytvořit pomocí tlačítka v záhlaví rozvrhu nebo kliknutím do prázdné buňky. Tím se otevře modální okno pro zadání názvu a času trvání události.

Při vytváření je dostupná volba opakovatelnosti jednorázové, denní, týdenní, měsíční, nebo vlastní opakování po dnech). V případě opakované události se vybírá datum, do kdy se událost opakuje.

2.4.2 Aktualizace události

Kliknutím na událost se otevře okno pro zobrazení detailů a možnost aktualizace. U skupinových událostí lze měnit celou skupinu, pouze budoucí události, nebo jednotlivou událost.

2.4.3 Odstranění události

Událost lze odstranit z detailu události příslušným tlačítkem v zápatí. U skupinové události má uživatel navíc možnost smazat celou skupinu, budoucí události, nebo jednu konkrétní událost.

2.5 Import a export událostí

Import událostí lze započít kliknutím na příslušné tlačítko v záhlaví rozvrhu. Tím je vyvoláno otevření standartního dialogu prohlížeče pro výběr souboru. Po vybrání se zobrazí

modálové okno ke specifikaci chování importu a to buď přidání do aktuálního seznamu událostí nebo jeho přepsání.

Pro export událostí stačí pouze kliknout na tlačítko a soubor s událostmi je automaticky stažen. Exportován je celý obsah seznam událostí.

3 IMPLEMENTACE

V této části je detailně popsána implementace řešení, doplněná praktickými ukázkami konkrétních realizací. Implementace je realizována v projektu typu **Razor Class Library** a nese název **Blazor.Timetable**. Ikony použité napříč knihovnou jsou poskytnuty knihovnou **Tabler**.

3.1 Struktura

Zdrojový kód je organizován do několika jmenných prostorů, které jsou strukturovány dle jejich funkčního zaměření. Díky tomu je knihovna přehlednější a odpovědnost je patřičně oddělena. Hlavní jmenné prostory jsou:

- Common,
- Components,
- Models,
- Resources,
- Services.

Hlavní komponenty, které v knihovně figurují, popisuje Tabulka 1.

Tabulka 1: Seznam podstatných komponent

Komponenta	Popis
Timetable	Hlavní komponenta, organizuje celý rozvrh
Options	Záhlaví s tlačítky pro export a změnu zobrazení
Navigation	Navigační část rozvrhu pro přechod po časové ose
GridItem	Buňka ve mřížce
TimetableEvent	Událost v buňce
ModalContainer	Obal modálových oken
EventModal	Správa události
GroupMoveModal	Dialog pro výběr možnosti přesunu
ImportModal	Dialog pro výběr možnosti importu

(zdroj: vlastní)

Mezi podpůrné komponenty patří komponenty ikon, jako například **CalendarDayIcon** a vstupní komponenty jako **InputDateTime**.

3.2 Generičnost

Jelikož jedním z požadavků je možnost použít v knihovně vlastní třídu události, je nutné knihovně umožnit práci s generickým typem události. K tomu je potřeba znát stromy výrazů, neboli selektory, pro povinné vlastnosti události [6].

Z těchto selektorů se skrze reflexi vytvoří gettery a settery vlastností za běhu programu, což řeší metody **CreateGetter** a **CreateSetter** třídy **PropertyHelper**, viz Ukázka kódu 1.

```

public static Func<TObject, TProperty?> CreateGetter<TObject, TProperty>
(Expression<Func<TObject, TProperty?>> expression) where TObject : class
{
    if (expression.Body is not MemberExpression
        && expression.Body is not UnaryExpression { Operand:
MemberExpression })
        throw new ArgumentException(
            "Expression must point to a property", nameof(expression));

    var compiledExpression = expression.Compile();

    return obj => Equals(obj, default(TObject))
        ? default
        : compiledExpression(obj);
}

public static Action<TObject, TProperty?> CreateSetter<TObject, TProperty>
(Expression<Func<TObject, TProperty?>> expression) where TObject : class
{
    var memberExpr = expression.Body switch
    {
        MemberExpression me => me,
        UnaryExpression ue when ue.Operand is MemberExpression me => me,
        _ => throw new ArgumentException(
            "Expression must point to a property", nameof(expression)),
    };

    if (memberExpr.Member is not PropertyInfo propertyInfo)
        throw new ArgumentException(
            "Member is not a property", nameof(expression));

    var parameter = expression.Parameters[0];
    var valueParameter = Expression.Parameter(typeof(TProperty), "value");

    var valueToAssign = valueParameter as Expression;

    if (propertyInfo.PropertyType != typeof(TProperty))
        valueToAssign = Expression.Convert(
            valueParameter, propertyInfo.PropertyType);

    var member = Expression.MakeMemberAccess(parameter, propertyInfo);
    var assign = Expression.Assign(member, valueToAssign);
    var lambda = Expression.Lambda<Action<TObject, TProperty?>>(
        assign, parameter, valueParameter);

    return lambda.Compile();
}

```

Ukázka kódu 1: Pomocné metody CreateGetter a CreateSetter

Při inicializaci hlavní komponenty Timetable jsou do ní předané selektory pomocí této třídy přetvořeny a uloženy ve třídě **PropertyAccessors**, aby nebylo potřeba tuto náročnou operaci opakovat. Napříč knihovnou je poté pro přístup k vlastnostem události použita totožná instance **PropertyAccessors**.

3.3 Struktura rozvrhu

Třídy pro reprezentaci rozvrhu jsou vytvořeny hierarchicky, kde třída **Grid** představuje rozvrh obsahující seznam sloupců a titulků řádků pro denní a týdenní zobrazení, viz Ukázka kódu 2. Pro zajištění inicializace veškerých vlastností nezbytných při následném zobrazení je využíván modifikátor **required**. Díky němu lze vynutit inicializaci dané vlastnosti při tvorbě třídy [7]. Sloupce, představující jednotlivé dny týdne, jsou charakterizovány výčtovým typem **DayOfWeek**, indexem a seznamem časových buněk, které dále obsahují informace o událostech. Uvnitř událostí je používána třída **PropertyAccessors** pro simulaci getterů a setterů.

```
internal sealed class Grid<TEvent> where
    TEvent : class
{
    public required string Title { get; init; }
    public ICollection<Column<TEvent>> Columns { get; init; } = [];
    public IEnumerable<string> RowTitles { get; init; } = [];

    public CellItem<TEvent>? FindItemById(Guid itemId)
    {
        return Columns.SelectMany(col => col.Cells)
            .SelectMany(cell => cell.Items)
            .FirstOrDefault(item => item.Id == itemId);
    }

    public Cell<TEvent>? FindCellByEventId(Guid eventId)
    {
        return Columns.SelectMany(col => col.Cells)
            .FirstOrDefault(cell =>
                cell.Items.Any(e => e.Id == eventId));
    }

    public Cell<TEvent>? FindCellById(Guid cellId)
    {
        return Columns.SelectMany(col => col.Cells)
            .FirstOrDefault(cell => cell.Id == cellId);
    }
}
```

Ukázka kódu 2: Generická třída rozvrhu

3.4 Zobrazení událostí

Proces zobrazení události na UI lze popsat v následujících krocích:

1. Komponenta na základě konfigurace zavolá příslušnou zobrazovací službu,
2. tato služba vytvoří a adekvátně naplní třídu mřížky událostmi spadajícími do aktuálního zobrazení,
3. obsah této třídy je vykreslen do rozvrhu.

3.4.1 Služby

Pro každý typ zobrazení je implementována zobrazovací služba, která má za úkol ze seznamu všech událostí vyfiltrovat specifické události spadající do aktuálního zobrazení a vytvořit z nich třídu reprezentující rozvrh.

Všechny zobrazovací služby jsou implementací rozhraní **IDisplayService**. To definuje vlastnost **DisplayType**, která určuje, ke kterému typu zobrazení se daná implementace váže a metodu **CreateGrid** pro generaci instance třídy rozvrhu, viz Ukázka kódu 3. Metoda přijímá seznam generických událostí, konfigurační třídu rozvrhu **TimetableConfig**, datum a pomocnou třídu **PropertyAccessors** pro přístup k vlastnostem událostí.

```
internal interface IDisplayService
{
    DisplayType DisplayType { get; }

    Grid<TEvent> CreateGrid<TEvent>(
        IList<TEvent> events,
        TimetableConfig config,
        DateOnly date,
        PropertyAccessors<TEvent> props)
        where TEvent : class;
}
```

Ukázka kódu 3: Rozhraní IDisplayService

Jednotlivé implementace, tj. **DailyService**, **WeeklyService** a **MonthlyService**, jsou zaregistrovány do kontejneru služeb, do hlavní komponenty **Timetable** injektovány a při aktualizaci rozvrhu je zavolána příslušná implementace dle aktuálního typu zobrazení viz Ukázka kódu 4.

```
private void UpdateGrid()
{
    var displayService = DisplayServices.FirstOrDefault(s => s.DisplayType ==
        _timetableManager.DisplayType)
        ?? throw new NotSupportedException($"Implementation for
{nameof(DisplayType)}: '{_timetableManager.DisplayType}' not found.");

    _timetableManager.Grid = displayService.CreateGrid(Events, TimetableConfig,
        _timetableManager.CurrentDate, _eventProps);
    StateHasChanged();
}
```

Ukázka kódu 4: Metoda pro aktualizaci rozvrhu dle typu zobrazení

3.4.2 Vykreslení

Rozvrh je vykreslován v hlavní komponentě **Timetable** formou **CSS** gridu, což je způsob dvoudimenzionálního vykreslení uživatelského rozhraní umožňující přesné umístění prvků na stránce [8].

Pro jednotlivé buňky mřížky je vytvořena komponenta **GridItem**, která umožňuje specifikovat umístění a rozsah položky pomocí parametrů **RowIndex**, **ColumnIndex** a **Span**. Parametr **Direction** určuje směr rozšíření. Komponenta také podporuje zpracování události kliknutí přes parametr **OnClick** a umožňuje vložení vlastního obsahu skrze **RenderFragment ChildContent**. Část komponenty **Timetable**, která řeší vykreslení jednotlivých časových buněk, zobrazuje Ukázka kódu 5.

```
@foreach(var column in _timetableManager.Grid.Columns)
{
    foreach (var cell in column.Cells)
    {
        var style = "timetable-body-cell grid-border"
            + (cell.Type == CellType.Disabled ? " disabled" : "");
        <GridItem CssClass=@style
            Direction=SpanDirection.Vertical
            RowIndex=@(cell.RowIndex + 1)
            SlotId=@(cell.Type != CellType.Disabled ? cell.Id : null)
            OnClick="@(() => HandleOpenCreateModal(cell.DateTime))"
            ColumnIndex=@(!isMonthView ? column.Index + 1 : column.Index)
        />
    }
}
```

Ukázka kódu 5: Vykreslení časových buněk

Zbytek rozvrhu je zobrazován po sloupcích, kde se v prvním řádku vykreslí buňka s názvem dne, jedná-li se o denní či týdenní zobrazení. V tomto případě jsou rovněž v prvním sloupci vykresleny titulky časových buněk, vztahující se k danému řádku.

Pro zobrazení a správné umístění událostí v buňce je využita generická komponenta **TimetableEvent**, která se vkládá jako potomek komponentě **GridItem**. Parametry se jí konfigurují vlastnosti jako pořadí **Order**. V případě kdy se překrývá více událostí ve stejné časové buňce, také směr rozšíření **Direction** podobně jako u komponenty **GridItem**. Ukázka kódu 6 zobrazuje kód k vykreslení komponenty události.

```

@foreach (var cell in column.Cells)
{
    foreach (var cellItem in cell.Items)
    {
        var direction = isMonthView || cell.Type == CellType.Header
            ? SpanDirection.Horizontal : SpanDirection.Vertical;
        <GridItem Offset=@cell.Items.Count
            Direction=@direction
            ColumnIndex=@(isMonthView ? column.Index : column.Index + 1)
            RowIndex=@(cell.Type == CellType.Header
                ? 2 : cell.RowIndex + 1)
            Span=@cellItem.Span>
            <TimetableEvent CellItem=@cellItem
                OnEventUpdated="(UpdateAction<TEvent> props)
                    => HandleEventUpdatedAsync(props)"
                OnEventDelete="(DeleteAction<TEvent> props)
                    => HandleEventDeletedAsync(props)"
                AdditionalFields=@AdditionalFields
                Order=@cell.Items.IndexOf(cellItem)
                Direction=@direction />
        </GridItem>
    }
}

```

Ukázka kódu 6: Vykreslení události v časové buňce

3.5 Správa událostí

Správu samotnou deleguje komponenta **Timetable** třídě **TimetableManager**. Ta definuje metody pro tvorbu, úpravu či odstranění jednorázových a skupinových událostí. UI správy je implementováno v komponentě **EventModal**, která slouží jak pro tvorbu tak i úpravu viz Ukázka kódu 7.

```

private async Task SaveAsync()
{
    if (!Validate())
        return;

    if (State == EventModalState.Create)
    {
        var createProps = new CreateAction<TEvent>
        {
            Repetition = SelectedRepeatability,
            RepeatUntil = RepeatUntil.ToDateTimeOnly(),
            RepeatDays = RepeatAfterDays,
            EventDescriptor = EventDescriptor
        };
        await OnCreate.InvokeAsync(createProps);
    }
    else
    {
        var updateProps = new UpdateAction<TEvent>
        {
            Original = OriginalEventDescriptor,
            New = EventDescriptor,
            Scope = SelectedScope
        };
        await OnUpdate.InvokeAsync(updateProps);
    }

    ModalService.Close();
}

```

Ukázka kódu 7: Metoda pro uložení

Při zavolání callbacku **OnCreate**, viz Ukázka kódu 8, se argument rozloží a předá do metody **CreateEvents** třídy **TimetableManager**, kde se vytvoří události dle požadavku. Výstupný seznam zpracuje komponenta **Timetable** a rozvrh aktualizuje. Podobný postup je proveden i v případě úpravy či odstranění událostí. Názorný příklad tvorby zobrazuje Ukázka kódu 9.

```

private async Task HandleEventDeletedAsync(
    DeleteAction<TEvent> deleteProps)
{
    if (deleteProps.Scope == ActionScope.Single)
    {
        var deletedEvent = _timetableManager.DeleteEvent(
            Events, deleteProps);
        await OnEventDeleted.InvokeAsync(deletedEvent);
    }
    else
    {
        var deletedEvents = _timetableManager.DeleteGroupEvent(
            Events, deleteProps);
        await OnGroupEventDeleted.InvokeAsync(deletedEvents);
    }
    UpdateGrid();
}

```

Ukázka kódu 8: Metoda pro odstranění událostí ze seznamu

```

public IList<TEvent> CreateEvents(
    EventDescriptor<TEvent> templateDescriptor,
    Repeatability repetition,
    DateOnly? repeatUntil, int? repeatDays = null)
{
    var rootDesc = templateDescriptor.DeepCopy();
    var result = new List<EventDescriptor<TEvent>> { rootDesc };
    if (repetition != Repeatability.Once)
    {
        var groupId = Guid.NewGuid().ToString();
        rootDesc.GroupId = groupId;

        var baseFrom = rootDesc.DateFrom;
        var baseTo = rootDesc.DateTo;
        var i = 1;
        while (true)
        {
            var nextFrom = repetition switch
            {
                Repeatability.Daily => baseFrom.AddDays(i),
                Repeatability.Weekly => baseFrom.AddDays(7 * i),
                Repeatability.Monthly => baseFrom.AddMonths(i),
                Repeatability.Custom when repeatDays.HasValue
                    => baseFrom.AddDays(repeatDays.Value * i),
                _ => throw new ArgumentException("Invalid repetition"),
            };

            if (nextFrom.ToDateOnly() > repeatUntil)
                break;
            var desc = rootDesc.DeepCopy();
            desc.DateFrom = nextFrom;
            desc.DateTo = baseTo + (nextFrom - baseFrom);
            desc.GroupId = groupId;
            result.Add(desc);
            i++;
        }
    }

    return [... result.Select(d => d.Event)];
}

```

Ukázka kódu 9: Metoda třídy TimetableManager pro tvorbu událostí

3.5.1 Drag and drop

Pro implementaci funkcionality drag-and-drop událostí mezi časovými buňkami v rozvrhu je využita knihovna **interact.js** viz interactjs.io, jelikož v Blazoru samotném je obtížné tuto funkcionality implementovat.

Každá událost a časová buňka má unikátní identifikátor. Při tažení události se díky pozici dopadu zjistí identifikátor cílové buňky, a ten se společně s identifikátorem události použije pro volání .NET metody. Ta pak v aktuální instanci rozvrhu identifikuje a aktualizuje příslušnou událost a její novou lokaci dle datumu nové buňky. V případě skupinové události je zobrazeno modálové okno pro výběr možnosti přesunu. Ukázka kódu 10 zobrazuje JS funkci vyvolanou při upuštění přetahované události do časové buňky.

```

async end(event) {
  event.preventDefault();
  const target = event.target;
  const closestSlot = findClosestSlot(target);
  target.style.zIndex = '';
  target.style.cursor = 'pointer';

  if (!closestSlot) {
    resetPosition(target);
    return;
  }

  const eventId = target.getAttribute(eventArgument);
  const targetSlotId = closestSlot.getAttribute(slotArgument);
  const originalSlotId = target.getAttribute(originalSlotArgument);

  if (targetSlotId === originalSlotId) {
    resetPosition(target);
    return;
  }

  await objRef.invokeMethodAsync('MoveEventAsync', eventId, targetSlotId)
    .catch(error => {
      console.error("Error moving event: ", error);
    });
}

```

Ukázka kódu 10: Funkce pro vyvolání změny události

3.5.2 Přidané vlastnosti

Jelikož má knihovna umožnit přidání vlastních vlastností při tvorbě či úpravě události, je nezbytné definovat rozšiřitelný systém vstupů s možností validace.

Validace je provedena při změně hodnoty vstupu či při potvrzení v komponentě **EventModal**. Aby bylo možné z této komponenty validovat dodatečné vlastnosti, je vytvořena kaskádová hodnota, která poskytuje vstupním komponentám metodu pro přidání vlastní validační metody do seznamu v **EventModal**. Díky tomu lze všechny validační metody, například při tvorbě, spustit naráz a případně zamezit dokončení, pokud některá z nich obsahuje chybu.

Vstupní komponenty dědí z generického základu **BaseInput**, který definuje kompletní rozhraní umožňující registraci při provedení metody životního cyklu **OnParametersSet**. Díky tomu pouze stačí vstupní komponentě definovat validační funkci a o zbytek se postará sama. Část relevantního kódu abstraktní komponenty **BaseInput** zobrazuje Ukázka kódu 11. Validační funkce se poté spouští skrze metodu **Validate**.

```

[CascadingParameter(Name = "RegisterValidation")] public Action<Func<bool>>?
RegisterValidation { get; set; }
[CascadingParameter(Name = "StateChanged")] public Action? StateChanged { get;
set; }

protected virtual TType BindProperty
{
    get => _getter(Model) ?? default!;
    set
    {
        if (EqualityComparer<TType>.Default.Equals(_getter(Model), value))
            return;

        _setter(Model, value);
        ValueChanged.InvokeAsync(value);
        ValidateInput();

        StateChanged?.Invoke();
    }
}

protected override void OnParametersSet()
{
    if (Selector is null)
        throw new InvalidSetupException("Selector is not set");

    RegisterValidation?.Invoke(ValidateInput);

    _getter = PropertyHelper.CreateGetter(Selector!);
    _setter = PropertyHelper.CreateSetter(Selector!);
}

private bool ValidateInput()
{
    if (Validate is null)
        return true;

    ErrorMessage = Validate(_getter(Model)!);
    return string.IsNullOrEmpty(ErrorMessage);
}

```

Ukázka kódu 11: Část kódu komponenty BaseInput

3.6 Datová výměna

V této části je popsán proces datové výměny, který zahrnuje import a export událostí ve výchozí implementaci pro formát CSV.

3.6.1 Export

Export událostí řeší výchozí implementace **CsvExportTransformer**. Ta s pomocí třídy **CsvGenerator** vytvoří normalizovaný seznam řádků a v transformeru se tento seznam převede do třídy datového proudu. Jelikož je stažení exportovaného souboru realizováno skrze JS, musí tento transformer vracet proud formátu **DotNetStreamReference**, viz Ukázka kódu 12.

```

public sealed class CsvExportTransformer : IExportTransformer
{
    public IExportInfo Transform<TEvent>(
        IEnumerable<TEvent> records, IList<ISelector<TEvent>> properties)
        where TEvent : class
    {
        var csvContent = CsvGenerator.CreateCsvContent(
            records, properties);

        var memoryStream = new MemoryStream();
        using var writer = new StreamWriter(
            memoryStream, Encoding.UTF8, 1024, true);

        foreach (var line in csvContent)
        {
            writer.WriteLine(string.Join(CsvGenerator.Separator, line));
        }

        writer.Flush();
        memoryStream.Position = 0;

        return new ExportInfo
        {
            StreamReference = new DotNetStreamReference(
                memoryStream, true),
            FileExtension = "csv"
        };
    }
}

```

Ukázka kódu 12: Implementace export transformeru

Transformer je využit v komponentě **Options**, kde při kliknutí na tlačítko exportu je zavolána metoda **Transform**. Její výstup, tj. datový proud a název souboru, je předán do JS funkce **downloadFileFromStream**.

```

private async Task HandleExportClickedAsync()
{
    if (_jsModule is null) return;

    var transformResult = ExportConfig.Transformer.Transform(
        Events, ExportConfig.Selectors);
    var fileName = $"{ExportConfig.FileName}.{transformResult.FileExtension}";

    using var stream = transformResult.StreamReference;

    await _jsModule.InvokeVoidAsync("downloadFileFromStream", fileName, stream);
}

```

Ukázka kódu 13: Metoda pro vyvolání stažení souboru

JavaScript funkce **downloadFileFromStream** poté provede potřebné kroky ke stažení souboru z proudu.

```

export async function downloadFileFromStream(
  fileName, contentStreamReference) {
  try {
    const arrayBuffer = await contentStreamReference.arrayBuffer();
    const blob = new Blob([arrayBuffer],
      { type: 'application/octet-stream' });
    const url = URL.createObjectURL(blob);

    const anchorElement = document.createElement('a');
    anchorElement.href = url;
    anchorElement.download = fileName ?? '';
    document.body.appendChild(anchorElement);
    anchorElement.click();
    document.body.removeChild(anchorElement);
    URL.revokeObjectURL(url);
  } catch (error) {
    console.error('Error while exporting to file:', error);
  }
}

```

Ukázka kódu 14: Funkce pro stažení souboru z proudu

3.6.2 Import

Import událostí začíná v JavaScript funkci **promptFileSelect**. Tato funkce je zavolána kliknutím na tlačítko importu v komponentě **Options** stejným způsobem jako u funkce pro export. Jejím účelem je převod souboru na formát base64 a předání do komponentové metody **ReceiveFileBase64Async**.

```

export function promptFileSelect(dotNetRef, maxSize, allowedExtensions)
{
  const inp = document.createElement('input');
  inp.type = 'file';
  if (allowedExtensions && allowedExtensions.length)
    inp.accept = allowedExtensions.map(e => '.' + e).join(',');

  inp.onchange = async ev => {
    const file = ev.target.files[0];
    const buf = await file.arrayBuffer();
    const u8 = new Uint8Array(buf);

    let binary = '';
    for (let i = 0; i < u8.length; i++) {
      binary += String.fromCharCode(u8[i]);
    }
    const base64 = btoa(binary);
    dotNetRef.invokeMethodAsync('ReceiveFileBase64Async', base64);
  };
  inp.click();
}

```

Ukázka kódu 15: Funkce pro předání zpracovaného souboru

Metoda **ReceiveFileBase64Async**, kterou zobrazuje Ukázka kódu 16, pomocí metody **Transform** třídy **CsvImportTransformer** soubor zpracuje a vytvoří seznam událostí. Tento seznam, spolu s metodou pro dokončení importu, je předán do modálového okna. Zde

je možnost výběru buďto přepsání aktuálního seznamu událostí seznamem importovaným, nebo jeho rozšíření viz Ukázka kódu 17.

```
[JSInvokable]
public async Task ReceiveFileBase64Async(string base64)
{
    var content = Convert.FromBase64String(base64);
    var stream = new MemoryStream(content, writable: false);
    var items = ImportConfig.Transformer.Transform(
        stream, ImportConfig.Selectors);

    await stream.DisposeAsync();

    var parameters = new Dictionary<string, object>
    {
        { "ImportedEvents", items },
        { "OnSubmit", OnImport }
    };

    ModalService.Show<ImportModal<TEvent>>(parameters);
}
```

Ukázka kódu 16: Metoda pro zpracování obsahu souboru

```
private async Task HandleImportAsync(ImportAction<TEvent> props)
{
    if (props.Type == ImportType.Append)
    {
        foreach (var item in props.Events)
            Events.Add(item);
    }
    else
    {
        Events = props.Events;
    }

    UpdateGrid();
    await OnEventsImported.InvokeAsync(props);
}
```

Ukázka kódu 17: Metoda dokončení importu

3.7 Lokalizace

Pro usnadnění realizace překladu je vytvořena třída **Localizer**, která obaluje třídu **ResourceManager** a poskytuje přístup k lokalizovaným řetězcům prostřednictvím indexového přístupu. Pro šablonové texty je k dispozici metoda **GetLocalizedString**, která umožňuje jejich přímé nahrazení viz Ukázka kódu 18.

Jednotlivé překlady jsou uloženy v souborech prostředků s příponou **.resx**, respektive **GlobalResource.resx** pro angličtinu a **GlobalResource.cs.resx** pro češtinu.

```
internal sealed class Localizer
{
    private readonly ResourceManager _resourceManager;

    public Localizer(string baseName, System.Reflection.Assembly assembly)
    {
        _resourceManager = new ResourceManager(baseName, assembly);
    }

    public string this[string key]
    {
        get => _resourceManager.GetString(key, CultureInfo.CurrentCulture) ?? key;
    }

    public string GetLocalizedString(string key, params object[] args)
    {
        return string.Format(this[key], args);
    }
}
```

Ukázka kódu 18: Třída Localizer

3.8 Testování

Vybrané části knihovny, například zobrazovací služby, jsou testovány pomocí jednotkových testů v samostatném projektu **Blazor.Timetable.Tests**. K testování je využit framework **xUnit**, který je navíc obohacen využitím knihovny **FluentAssertions**, díky které lze testy psát více intuitivním a snadno pochopitelným způsobem. Ukázka kódu 19 zobrazuje jednotkový test třídy **CsvGenerator**. Metoda je rozpoznána jako testovací díky atributu **Fact** [9]. Tabulka 2 poté informace o jednotlivých testovacích třídách.

```
[Fact]
public void CreateCsvContent_ShouldCreateCorrectDataRows()
{
    var testRecords = new List<TestEvent>
    {
        new() { Property1 = "Value1", Property2 = "Value2" },
        new() { Property1 = "Value3", Property2 = "Value4" }
    };
    var propertySelectors = new List<ISelector<TestEvent>>
    {
        new Selector<TestEvent, string>("Property1", e => e.Property1),
        new Selector<TestEvent, string>("Property2", e => e.Property2)
    };

    var result = CsvGenerator.CreateCsvContent(testRecords, propertySelectors);

    result.Should().HaveCount(3);
    result[1].Should().Equal(["Value1", "Value2"]);
    result[2].Should().Equal(["Value3", "Value4"]);
}
```

Ukázka kódu 19: Znázornění testovacího případu s využitím FluentAssertions

Tabulka 2: Přehled testovacích tříd

Testovací třída	Testovaná třída	Počet testovacích případů
PropertyHelperTests	PropertyHelper	25
DailyServiceTests	DailyService	22
WeeklyServiceTests	WeeklyService	11
MonthlyServiceTests	MonthlyService	3
CsvGeneratorTests	CsvGenerator	6
CsvImportTransformerTests	CsvImportTransformer	6

(zdroj: vlastní)

Při testování zobrazovacích služeb je kladen důraz na různé scénáře, které mohou při generaci nastat. Jedná se například o různé trvání událostí či konfigurace.

3.9 Publikace

Pro možnost publikace balíčku do platformy **NuGet**, která slouží k jejich uchování a distribuci, je nutné nejprve provést konfiguraci projektu. Toho lze docílit přidáním specifického nastavení do projektového souboru **Blazor.Timetable.csproj**:

- Možnost zabalení projektu do balíčku,
- unikátní identifikátor,
- název,
- verze,
- autor,
- cesta k licenčnímu souboru a souboru příručky **README**.

Po konfiguraci je možné balíček sestavit pomocí příkazu *dotnet pack*. Ten zabalí projekt do souboru s příponou **.nupkg** a umístí jej do výchozího adresáře **./bin**. Výsledný **.nupkg** soubor je připraven k publikaci a jeho nahrání lze snadno provést prostřednictvím webového rozhraní nuget.org [10;11].

3.10 Repozitář

Zdrojové kódy řešení jsou dostupné na platformě **GitHub**. Adresa repozitáře je <https://github.com/Filipxxd/Blazor.Timetable>.

Repozitář obsahuje dvě hlavní větve – **master** a **dev**. Větev **dev** je určena pro vývoj a testování, zatímco **master** slouží jako stabilní produkční větev, z níž se balíček publikuje.

Pro usnadnění opakovaných činností jsou vytvořeny automatizované procesy pomocí **GitHub Actions**. Ty se spouští při každém sloučení změn nebo příkazu push. Níže je Tabulka 3 s přehledem automatizovaných činností. [12]

Tabulka 3: Implementované procesy

Název	Ovlivněná větev	Popis
Build and Test	všechny	Provede sestavení projektu a spustí testy
Release package	master	Vytvoří a publikuje balíček

(zdroj: vlastní)

4 POUŽITÍ

Knihovna nabízí rozmanité možnosti konfigurace, a proto je důležité ukázat různé scénáře využití a přizpůsobení v konzumní aplikaci. Pro snadnější pochopení jsou implementace realizovány v předgenerované stránce **Home.razor**, která je standardně součástí nově vytvořených Blazor projektů.

4.1 Předpoklady

Pro vytvoření Blazor projektu a využití této knihovny je vyžadováno následující:

- Nainstalovaná runtime a **SDK** pro **.NET 8.0** a vyšší,
- **IDE**, ideálně Microsoft Visual Studio 2019 a vyšší.

4.2 Tvorba projektu

Následující kroky popisují postup při vytváření nového Blazor Server projektu pomocí vývojového prostředí Visual Studio:

- Otevření vývojového prostředí,
- vybrání možnosti **Vytvořit nový projekt** z hlavního menu,
- ze seznamu dostupných šablon je vybrání **Blazor Server App**,
- zadání názvu projektu,
- zvolení verze a nastavení render mode **Server**,
- dokončení tvorby.

4.3 Minimal setup

Nejprve je potřeba přidat do projektu samotnou knihovnu. Toho lze docílit příkazem *dotnet add package Blazor.Timetable*. Alternativně lze balíček do knihovny přidat skrze **NuGet Package Explorer**.

Dále do hlavní třídy aplikace **Program** zaregistrovat služby prostřednictvím rozšiřující metody `AddBlazorTimetable`. Aby bylo možné aplikovat kaskádové styly, je nutné aktivovat obsluhu statických souborů pomocí `UseStaticFiles`.

```
using Blazor.Timetable.Common.Extensions;  
  
var builder = WebApplication.CreateBuilder(args);  
  
// ...  
  
builder.Services.AddBlazorTimetable();  
  
var app = builder.Build();  
  
app.UseStaticFiles();  
  
// ...  
  
app.Run();
```

Ukázka kódu 20: Přidané metody ve třídě Program

V komponentě **App.razor** se do tagu hlavičky přidá link na soubor kaskádových stylů. Cesta se skládá z názvu projektu a přípony **styles.css**.

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

Ukázka kódu 21: Tag kaskádových stylů

Následně je potřeba vytvořit třídu samotné události, se kterou bude knihovna pracovat. Tato třída musí obsahovat povinné vlastnosti viz Tabulka 4. Příklad takové třídy představuje Ukázka kódu 22.

Tabulka 4: Povinné vlastnosti

Vlastnost	Datový typ
Název události	string
Začátek	DateTime
Konec	DateTime
Identifikátor skupiny	string?

(zdroj: vlastní)

```
public class Subject
{
    public string Name { get; set; } = string.Empty;
    public DateTime StartTime { get; set; }
    public DateTime EndTime { get; set; }
    public string? GroupId { get; set; }
}
```

Ukázka kódu 22: Třída události rozvrhu

V komponentě, která rozvrh zobrazuje, je přidána direktiva **@using** spolu s atributem **@rendermode** nastaveným na **InteractiveServer**. V kódové části se vytvoří seznam událostí, který se následně předá instanci komponenty **Timetable** pomocí direktivy **@bind-Events**. Jelikož je komponenta generická, musí se rovněž definovat typ události **TEvent** a spolu s ním selektory povinných vlastností.

```
@page "/"
@using Blazor.Timetable.Components
@rendermode InteractiveServer

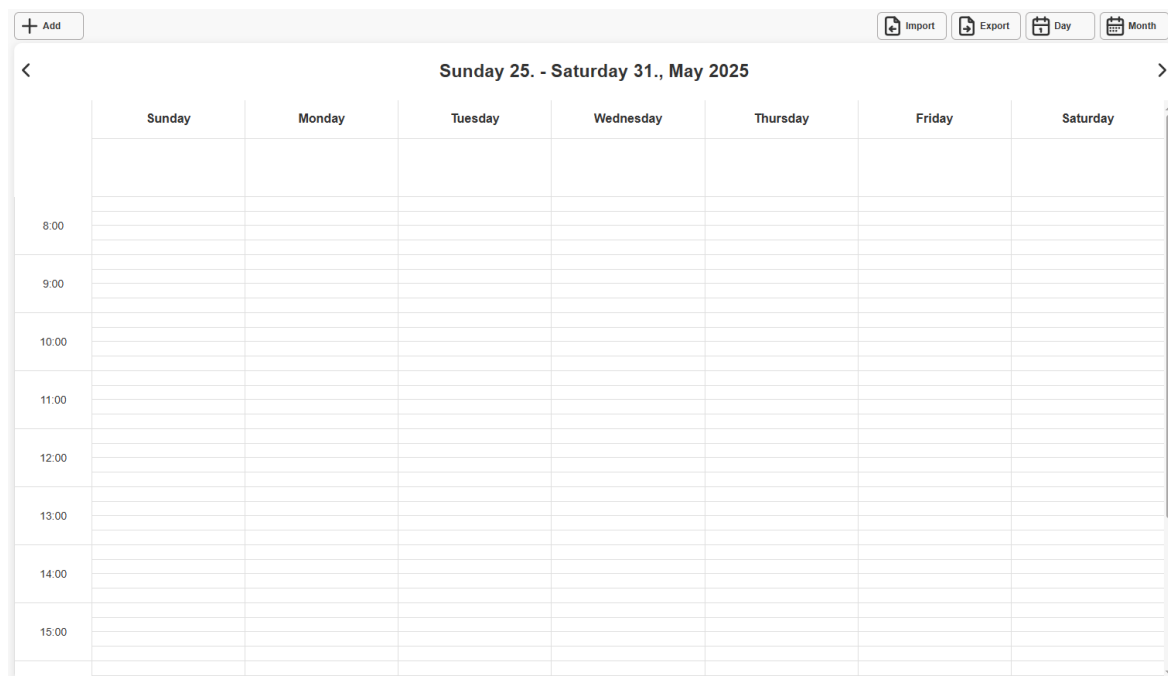
<PageTitle>Example page</PageTitle>

<Timetable @bind-Events=Subjects
    TEvent=Subject
    Title="e => e.Name"
    DateFrom="e => e.StartTime"
    DateTo="e => e.EndTime"
    GroupId="e => e.GroupId">
</Timetable>

@code{
    public IList<Subject> Subjects { get; set; } = [];
}
```

Ukázka kódu 23: Zdrojový kód komponenty Home.razor

Nyní je možné projekt spustit. Na hlavní stránce se zobrazí funkční modul knihovny ve výchozím nastavení, viz Obrázek 4. Při vytvoření, či následující úpravě, jsou změny automaticky propsány do seznamu událostí v komponentě **Home**.



Obrázek 4: Výchozí UI komponenty

(zdroj: vlastní)

4.4 Konfigurace obecného nastavení

K úpravě obecného nastavení knihovny slouží konfigurační třída **TimetableConfig**. Příklad konfigurace zobrazuje Ukázka kódu 24.

Pro stanovení časového rámce se využívají parametry **TimeFrom** a **TimeTo**, které určují rozsah ovlivňující denní a týdenní zobrazení prostřednictvím řádků v mřížce. Parametr **TimeTo** musí být větší než **TimeFrom**, s minutovou částí zaokrouhlenou na čtvrt hodinové intervaly. K zobrazení celodenního časového úseku lze parametr **TimeFrom** nastavit na půlnoc a **TimeTo** na 23 hodin a 59 minut.

Specifikované dny se v denním a týdenním zobrazení zobrazují jako sloupce v rozvrhu. Tyto dny a měsíce rovněž ovlivňují navigaci, jelikož lze přejít pouze na předem definované časové úseky.

```

<Timetable @bind-Events=Subjects
    TEvent=Subject
    Title="e => e.Name"
    DateFrom="e => e.StartTime"
    DateTo="e => e.EndTime"
    GroupId="e => e.GroupId"

    TimetableConfig="new(){
        DisplayType=DisplayType.Day,
        Days=[DayOfWeek.Monday, DayOfWeek.Tuesday,
            DayOfWeek.Wednesday, DayOfWeek.Thursday,
            DayOfWeek.Friday],
        TimeFrom=new(8, 0),
        TimeTo=new(16, 0)
    }">
</Timetable>

```

Ukázka kódu 24: Příklad konfigurace komponenty

4.5 Zpracování akcí

Pro implementaci vlastních reakcí na události v knihovně, jako je například úprava události, mohou být využity parametry viz Tabulka 5. Tyto parametry přijímají **EventCallbacky** s datovým typem odpovídajícím kontextu akce, což znamená, že je nezbytné, aby metoda do něj předaná přijímala potřebné argumenty se shodným datovým typem. Příklad přidání vlastní reakce zobrazuje Ukázka kódu 25.

Tabulka 5: Parametry pro předání vlastních reakcí

Parametr	Datový typ	Popis
OnEventCreated	TEvent	Tvorba události
OnGroupEventCreated	IList<TEvent>	Tvorba skupinové události
OnEventChanged	TEvent	Aktualizace události
OnGroupEventChanged	IList<TEvent>	Aktualizace skupinové události
OnEventDeleted	TEvent	Odstranění události
OnGroupEventDeleted	IList<TEvent>	Odstranění skupinové události
OnNextClicked	void	Navigace po časové ose kupředu
OnPreviousClicked	void	Navigace po časové ose zpět
OnDisplayTypeChanged	DisplayType	Změna aktuálního typu zobrazení
OnEventsImported	ImportAction<TEvent>	Import událostí

(zdroj: vlastní)

```

<T timetable @bind-Events=Subjects
    TEvent=Subject
    Title="e => e.Name"
    DateFrom="e => e.StartTime"
    DateTo="e => e.EndTime"
    GroupId="e => e.GroupId"

    OnEventChanged=HandleEventChanged>
</T timetable>

@code{
    public IList<Subject> Subjects { get; set; } = [];

    void HandleEventChanged(Subject subject)
    {
        // Zde je možné změny zpracovat
    }
}

```

Ukázka kódu 25: Vlastní reakce na úpravu události

4.6 Import a export

Ve výchozí konfiguraci jsou import a export událostí implementovány pro formát **CSV** a pracuje pouze s povinnými vlastnostmi. Toto chování může být upraveno pro každou operaci zvlášť pomocí konfiguračních tříd. Tabulka 6 a Tabulka 7 tato rozhraní zobrazuje.

Je-li formát **CSV** nevyhovující, je možné vytvořit vlastní implementace transformerů díky rozhraním **IImportTransformer** a **IExportTransformer**. Tyto vlastní implementace se poté předají do konfiguračních tříd do vlastnosti **Transformer**.

Tabulka 6: Rozhraní třídy ExportConfig

Vlastnost	Datový typ
FileName	string
Transformer	IExportTransformer
Selectors	IList<ISelector<TEvent>>

(zdroj: vlastní)

```

ExportConfig=@(new(){
    FileName="EventExport",
    Transformer=new CsvExportTransformer(),
    Selectors=[
        new Selector<Subject, string>("Subject", e => e.Name),
        new Selector<Subject, DateTime>("From", e => e.StartTime),
        new Selector<Subject, DateTime>("To", e => e.EndTime),
        new Selector<Subject, string?>("Group", e => e.GroupId),
        new Selector<Subject, string>("Desc", e => e.Description),
    ]
})

```

Ukázka kódu 26: Příklad konfigurace exportu

Tabulka 7: Rozhraní třídy ImportConfig

Vlastnost	Datový typ
AllowedExtensions	string[]
Transformer	IImportTransformer
MaxFileSizeBytes	long
Selectors	IList<ISelector<TEvent>>

(zdroj: vlastní)

```

ImportConfig=@(new(){
    AllowedExtensions = ["csv"],
    MaxFileSizeBytes = 5_000_000,
    Transformer = new CsvImportTransformer(),
    Selectors = [
        new Selector<Subject, string>("Subject", e => e.Name),
        new Selector<Subject, DateTime>("From", e => e.StartTime),
        new Selector<Subject, DateTime>("To", e => e.EndTime),
        new Selector<Subject, string?>("Group", e => e.GroupId),
        new Selector<Subject, string>("Desc", e => e.Description),
    ]
})

```

Ukázka kódu 27: Příklad konfigurace importu

4.7 Dodatečné vlastnosti

Dodatečné vlastnosti jsou vlastnosti třídy události nad rámec povinných, se kterými má knihovna pracovat, tj. umožnit například při tvorbě jejich vyplnění. Tyto vlastnosti jsou přidány předáním jejich selektorů do komponenty **Timetable** prostřednictvím parametru **AdditionalProps**. Následně jsou požadované vstupy vytvořeny pomocí render fragmentu **AdditionalFields**, který integruje požadované vstupní komponenty. K dispozici jsou předpřipravené komponenty uvedené viz Tabulka 8, případně lze implementovat komponenty vlastní

4.7.1 Předpřipravené komponenty

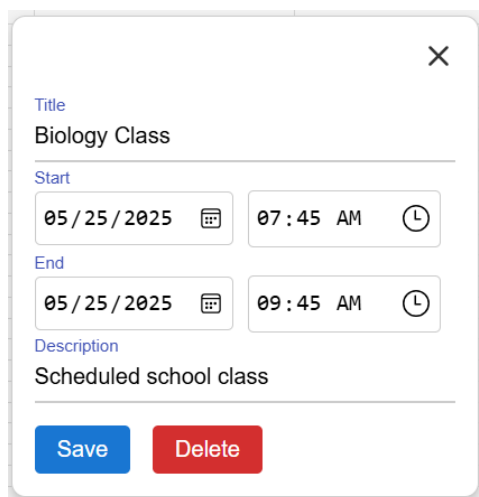
Pro zajištění integrace s knihovnou jsou poskytnuty předpřipravené komponenty. Ty umožňují zápis, úpravu a validaci událostí.

Tabulka 8: Předpřipravené komponenty

Název komponenty	Datový typ	Popis
Input	string	Vstup pro text
InputDateTime	DateTime	Vstup pro datum a čas
Dropdown	T	Generický výberovník

(zdroj: vlastní)

Příklad implementace zahrnuje vlastnost **Description** ve třídě **Subject**. Selektor této vlastnosti je přidán do **AdditionalProps** a render fragment **AdditionalFields** obsahuje instanci komponenty **Input** s odpovídajícími parametry. Objekt **context**, představující aktuální instanci události, je předán jako **Model** této komponenty pro umožnění navázání titulku ke vstupu. Výsledný kód viz Ukázka kódu 28.



The image shows a UI form for a school class. It has a title 'Biology Class', start and end date/time fields, and a description field 'Scheduled school class'. At the bottom are 'Save' and 'Delete' buttons.

Obrázek 5: UI dodatečné vlastnosti

(zdroj: vlastní)

```

<Timestep @bind-Events=Subjects
    TEvent=Subject
    Title="e => e.Name"
    DateFrom="e => e.StartTime"
    DateTo="e => e.EndTime"
    GroupId="e => e.GroupId"
    AdditionalProps=@([x => x.Description])>
<AdditionalFields>
    <Input Model="@context"
        Label="Description"
        Selector="x => x.Description" />
</AdditionalFields>
</Timestep>

```

Ukázka kódu 28: Přidání vlastnosti Description

4.7.2 Validace

Validace je realizována prostřednictvím parametru **Validate**, do kterého je možné specifikovat metodu prováděnou při změně hodnoty vlastnosti a před uložením události.

Metoda musí přijímat datový typ odpovídající vlastnosti a vrátet nullable string s chybovou hláškou k zobrazení. Návrátová hodnota **null** značí platnost hodnoty. Příklady implementace validace zobrazuje Ukázka kódu 29 a výsledné UI Obrázek 6.

```

<Input Model="@context"
    Label="Description:"
    Selector="x => x.Description"
    Validate=ValidationExample />

@code {
    private string? ValidationExample(string? description)
    {
        if (string.IsNullOrEmpty(description))
            return "Description cannot be empty";

        if (description.Length > 100)
            return "Cannot exceed 100 characters";

        return null;
    }
}

```

Ukázka kódu 29: Validace dodatečné vlastnosti

The screenshot shows a modal form with a close button (X) in the top right corner. The form contains the following elements:

- Title:** A text input field containing 'Biology Class'.
- Start:** Two input fields for date and time. The date is '05/25/2025' and the time is '07:45 AM'. Both have calendar and clock icons respectively.
- End:** Two input fields for date and time. The date is '05/25/2025' and the time is '09:45 AM'. Both have calendar and clock icons respectively.
- Description:** A text input field with a red border and a red error message below it: 'Description cannot be empty'.
- Buttons:** Two buttons at the bottom: a blue 'Save' button and a red 'Delete' button.

Obrázek 6: UI validace dodatečné vlastnosti

(zdroj: vlastní)

4.7.3 Vlastní vstupní komponenta

V případě kdy předpřipravené komponenty neposkytují požadovanou funkcionalitu je možné implementovat komponentu vlastní. Základem pro tyto implementace slouží třída **BaseInput**, která umožňuje využití předdefinovaného rozhraní pro přiřazení vlastnosti a její validaci při změně aktuální hodnoty.

Pro ukázkou je vytvořena komponenta **ExampleNumberInput**, která umožňuje práci s celými čísly. Nejprve se skrze atribut **@inherits** podědí základ **BaseInput**, kde typovými argumenty jsou třída události **Subject** a struktura **int**. Poté je vytvořeno samotné UI s využitím vlastností ze základu jako **BindProperty**, **Label**, **HasError**, **ErrorMessage** a **Id**.

```
@using Blazor.Timetable.Components.Shared.Forms
@inherits BaseInput<Subject, int?>

<div>
    <input type="number" @bind=BindProperty id=@Id />
    <label for=@Id>@Label</label>
</div>

@if (HasError)
{
    <div>
        ErrorMessage
    </div>
}
```

Ukázka kódu 30: Komponenta ExampleNumberInput

Pro demonstraci použití je do třídy události **Subject** přidána **int** hodnota pro maximální počet osob – **MaxOccupancy**. Přidání do komponenty **Timetable** probíhá stejně jako u předpřipravených komponent.

```
<ExampleCustomInput Model=@context  
    Label="Maximum occupancy:"  
    Selector="x => x.MaxOccupancy" />
```

Ukázka kódu 31: Použití komponenty `ExampleNumberInput`

4.8 Lokalizace

Knihovnou jsou nabízeny dva jazyky uživatelského rozhraní: čeština a angličtina (výchozí). Přepnutí jazyka je umožněno metodou **Localize**, která se řetězí za **AddBlazorTimetable** při registraci ve třídě **Program**. Metoda přijímá pouze jeden argument, kterým je výčtový typ **Language** s hodnotami **Czech** a **English**. Příklad změny jazyka zobrazuje Ukázka kódu 32.

```
builder.Services.AddBlazorTimetable().Localize(Language.Czech);
```

Ukázka kódu 32: Změna jazyka knihovny

ZÁVĚR

Cílem této bakalářské práce bylo vytvoření veřejně dostupné, modulární a konfigurovatelné knihovny pro správu a zobrazení školního rozvrhu v rámci .NET frameworku Blazor.

V teoretické části byly podrobně popsány základy Blazoru a praktická realizace zahrnovala definování klíčových požadavků, implementaci funkcionalit, testování a publikaci balíčku na platformě NuGet. Rovněž byla zdokumentována podrobná konfigurace a způsoby použití knihovny v konzumní aplikaci.

Výsledná knihovna splňuje stanovené cíle a poskytuje efektivní řešení pro správu a zobrazování rozvrhů v aplikacích postavených na frameworku Blazor.

SEZNAM POUŽITÉ LITERATURY

1. WRIGHT, Toi B. a HANSELMAN, Scott. Blazor WebAssembly by Example. Packt Publishing, 2021. ISBN 9781800563933
2. *ASP.NET Core Blazor render modes*. Online. 2024. Dostupné z: <https://learn.microsoft.com/en-us/aspnet/core/blazor/components/render-modes?view=aspnetcore-9.0>. [cit. 2025-05-25].
3. ENGSTRÖM, Jimmy. Web Development with Blazor. Packt Publishing, 2021. ISBN 9781800202580.
4. *ASP.NET Core Razor components*. Online. 2024. Dostupné z: <https://learn.microsoft.com/en-us/aspnet/core/blazor/components/?view=aspnetcore-9.0>. [cit. 2025-05-25].
5. *ASP.NET Core Blazor cascading values and parameters*. Online. 2024. Dostupné z: <https://learn.microsoft.com/en-us/aspnet/core/blazor/components/cascading-values-and-parameters?view=aspnetcore-9.0>. [cit. 2025-06-01].
6. MAYO, Joe. C# Cookbook. O'Reilly Media, 2021. ISBN 9781492093671.
7. GUPTA, Arun. Building End-to-End Apps with C# 11 and .NET 7. Bpb Publications, 2023. ISBN 9789355516374.
8. FRAIN, Ben. Responsive Web Design with HTML5 and CSS. Packt Publishing, 2022. ISBN 9781803231723.
9. *Unit testing C# in .NET using dotnet test and xUnit*. Online. 2025. Dostupné z: <https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-csharp-with-xunit>. [cit. 2025-06-01].
10. *Publish NuGet packages*. Online. 2023. Dostupné z: <https://learn.microsoft.com/en-us/nuget/nuget-org/publish-a-package>. [cit. 2025-06-01]
11. *Package readme on NuGet.org*. Online. 2024. Dostupné z: <https://learn.microsoft.com/en-us/nuget/nuget-org/package-readme-on-nuget-org>. [cit. 2025-06-01].
12. *Understanding GitHub Actions*. Online. 2025. Dostupné z: <https://docs.github.com/en/actions/about-github-actions/understanding-github-actions>. [cit. 2025-06-01].

SEZNAM OBRÁZKŮ

Obrázek 1: Návrh UI denního zobrazení v programu Figma.....	19
Obrázek 2: Návrh UI týdenního zobrazení v programu Figma	20
Obrázek 3: Návrh UI měsíčního zobrazení v programu Figma.....	20
Obrázek 4: Výchozí UI komponenty	43
Obrázek 5: UI dodatečné vlastnosti	47
Obrázek 6: UI validace dodatečné vlastnosti	49

SEZNAM UKÁZEK KÓDU

Ukázka kódu 1: Pomocné metody CreateGetter a CreateSetter	25
Ukázka kódu 2: Generická třída rozvrhu	26
Ukázka kódu 3: Rozhraní IDisplayService	27
Ukázka kódu 4: Metoda pro aktualizaci rozvrhu dle typu zobrazení	27
Ukázka kódu 5: Vykreslení časových buněk	28
Ukázka kódu 6: Vykreslení události v časové buňce	29
Ukázka kódu 7: Metoda pro uložení	30
Ukázka kódu 8: Metoda pro odstranění událostí ze seznamu	30
Ukázka kódu 9: Metoda třídy TimetableManager pro tvorbu událostí	31
Ukázka kódu 10: Funkce pro vyvolání změny události v Blazoru	32
Ukázka kódu 11: Část kódu komponenty BaseInput	33
Ukázka kódu 12: Implementace export transformeru	34
Ukázka kódu 13: Metoda pro vyvolání stažení souboru	34
Ukázka kódu 14: Funkce pro stažení souboru z proudu	35
Ukázka kódu 15: Funkce pro předání zpracovaného souboru	35
Ukázka kódu 16: Metoda pro zpracování obsahu souboru	36
Ukázka kódu 17: Metoda dokončení importu	36
Ukázka kódu 18: Třída Localizer	37
Ukázka kódu 19: Znázornění testovacího případu s využitím FluentAssertions	37
Ukázka kódu 20: Přidané metody ve třídě Program	41
Ukázka kódu 21: Tag kaskádových stylů	41
Ukázka kódu 22: Třída události rozvrhu	42
Ukázka kódu 23: Zdrojový kód komponenty Home.razor	42
Ukázka kódu 24: Příklad konfigurace komponenty	44
Ukázka kódu 25: Vlastní reakce na úpravu události	45
Ukázka kódu 26: Příklad konfigurace exportu	46
Ukázka kódu 27: Příklad konfigurace importu	46
Ukázka kódu 28: Přidání vlastnosti Description	48
Ukázka kódu 29: Validace dodatečné vlastnosti	48
Ukázka kódu 30: Komponenta ExampleNumberInput	49
Ukázka kódu 31: Použití komponenty ExampleNumberInput	50
Ukázka kódu 32: Změna jazyka knihovny	50

SEZNAM TABULEK

Tabulka 1: Seznam podstatných komponent	23
Tabulka 2: Přehled testovacích tříd.....	38
Tabulka 3: Implementované procesy	39
Tabulka 4: Povinné vlastnosti	41
Tabulka 5: Parametry pro předání vlastních reakcí	44
Tabulka 6: Rozhraní třídy ExportConfig	45
Tabulka 7: Rozhraní třídy ImportConfig	46
Tabulka 8: Předpřipravené komponenty	47

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API	Application programming interface
CSV	Comma-separated values
CSS	Cascading style sheet
DOM	Document object model
IDE	Integrated development environment
JS	JavaScript
SDK	Software development kit
SPA	Single page application
UI	User interface