Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

DIPLOMOVÁ PRÁCE

Webový konfigurátor mikroskopu s využitím dynamicky generovaného UI

2023

Vojtěch Skopal

Vedoucí práce: Mgr. Radek Janoštík, Ph.D.

Studijní obor: Informatika, prezenční forma

Bibliografické údaje

Autor:	Vojtěch Skopal
Název práce:	Webový konfigurátor mikroskopu s využitím dynamicky generovaného UI
Typ práce:	diplomová práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2023
Studijní obor:	Informatika, prezenční forma
Vedoucí práce:	Mgr. Radek Janoštík, Ph.D.
Počet stran:	52
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Vojtěch Skopal
Title:	Web configurator of the microscope using dynamically generated UI
Thesis type:	master thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2023
Study field:	Computer Science, full-time form
Supervisor:	Mgr. Radek Janoštík, Ph.D.
Page count:	52
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Práce se zabývá tvorbou webové aplikace pro konfiguraci hardwarových komponent a softwaru elektronového mikroskopu před jeho instalací. V práci najdete uvedení do problematiky a vysvětlení základních pojmů. Dále popis způsobu generování uživatelského rozhraní a vyhodnocování konfiguračních pravidel. Nesmím zapomenout i na shrnutí architektury aplikace a použitých technologií. Poslední kapitoly práce jsou věnovány technické a uživatelské dokumentaci.

Synopsis

The thesis deals with the creation of a web application for configuring the hardware components and software of the electron microscope before its installation. In the thesis, you will find an introduction to the issue and an explanation of basic terms. Next, a description of how to generate the user interface and evaluate the configuration rules. I must not forget the summary of the architecture of the application and the technologies used. The last chapters of the work are devoted to technical and user documentation.

Klíčová slova: generování, uživatelského rozhraní; konfigurátor, mikroskop

Keywords: generation, user interface; configurator, microscope

Rád bych poděkovat Mgr. Radkovi Janoštíkovi, Ph.D. za příkladné vedení práce, ochotu a konzultace. Dále firmě Edhouse s.r.o za příležitost pracovat na diplomové práci právě pro ni. Jmenovitě děkuji Mgr. Janu Michnovi za konzultace technických požadavků a průběžnou zpětnou vazbu. Nakonec bych chtěl poděkovat i svojí manželce za podporu a motivaci při psaní této práce.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

1 Úvod **8**

2 Uvedení do problematiky **9**
2.1 Elektronový mikroskop . . . 9
2.2 Instalace mikroskopu . . . 9
2.2.1 Prostředí u zákazníka . . . 9
2.2.2 Vývojová laboratoř . . . 10
2.2.3 Simulace MPC . . . 10
2.3 Uživatelské role . . . 10
2.4 Konfigurační pravidla . . . 11
2.5 Aplikační módy . . . 11
2.6 Instalační proces . . . 12
2.7 Užitečné pojmy . . . 13

3 Generované uživatelského rozhraní **14**
3.1 Požadavky na uživatelské rozhraní . . . 14
3.2 Struktura vstupních dat . . . 14
3.3 Převedení dat do interních objektů . . . 16
3.4 Vizualizace v uživatelském rozhraní . . . 17

4 Konfigurační pravidla **18**
4.1 Struktura dat . . . 18
4.1.1 Jednořádková pravidla . . . 18
4.1.2 Skupinová pravidla . . . 19
4.2 Vyhodnocování pravidel . . . 19
4.3 Validace pravidel . . . 22

5 Architektura projektu a použité technologie **23**
5.1 Blazor . . . 23
5.1.1 Blazor Server . . . 24
5.1.2 Blazor WebAssembly . . . 24
5.2 Architektura a struktura projektu . . . 24
5.2.1 Back end . . . 24
5.2.2 Front end . . . 26
5.3 Podpora desktop verze . . . 28
5.3.1 Blazor WebView . . . 28
5.3.2 Blazor PWA . . . 29
5.3.3 Blazor a Electron . . . 29
5.3.4 Požadavky a zvolená varianta . . . 30

6 Technická dokumentace **32**
6.1 Požadavky pro vývoj a běh aplikace . . . 32
6.2 Sestavení aplikace a první spuštění . . . 33

6.3 Popis projektů . . . 34
6.3.1 Projekty webového serveru . . . 34
6.3.2 Projekty webového klienta . . . 36
6.3.3 Sdílené projekty . . . 37
6.3.4 Testovací projekty . . . 37
6.4 Detaily implementace . . . 38
6.4.1 Zpracování a ošetření chyb . . . 38
6.4.2 Logy z běhu aplikace . . . 38
6.5 Grafické komponenty třetích stran . . . 39
6.5.1 BlazorMonaco . . . 39
6.5.2 Blazored.Modal . . . 40
6.5.3 Blazored.Toast . . . 40
6.6 Vývojářské nástroje . . . 40
6.6.1 Visual Studio 2022 . . . 40
6.6.2 ReSharper . . . 41
6.6.3 Swagger . . . 41
6.6.4 Webové nástroje pro vývoj . . . 41
6.6.5 Redux DevTools . . . 41
6.7 Mock data . . . 42

7 Uživatelská dokumentace **43**
7.1 Přepínání módu aplikace . . . 43
7.2 Standardní mód . . . 44
7.3 Developer mód . . . 46
7.4 Administrátorský mód . . . 46
7.5 Edit/Standalone mód . . . 48

Závěr **49**

Conclusions **50**

A Obsah přiložených souborů **51**

Literatura **52**

Seznam obrázků

1 Úvodní obrazovka aplikace 44
2 Výběr typu mikroskopu 45
3 Shrnující obrazovka konfigurace 46
4 Administrátorský mód aplikace 47
5 Chybový dialog s popisem chyby v XML 47
6 Informační dialog o neuložených změnách v XML editoru 48

1 Úvod

Cílem této práce je vytvořit webovou aplikaci pro konfiguraci softwaru a hardwarových komponent elektronového mikroskopu. Hlavním aspektem aplikace je správné využití dynamicky generovaného UI na základě vstupních souborů s daty. Aplikace provádí uživatele konfiguračními kroky. Podporuje načítání a uložení zvolené konfigurace do Windows registrů.

Aplikace obsahuje tři módy spuštění. První dva módy nabízejí úplnou nebo omezenou sadu položek konfigurace. V posledním administrátorském módu lze provést editaci konfiguračních kroků. Ta se provádí přímo v aplikaci úpravou vstupních souborů s daty. Po dodržení formátu dat a úspěšném zpracování souborů, je uživatelské rozhraní s konfiguračními kroky automaticky přegenerováno a reflektuje použité úpravy. K webové aplikaci existuje i totožná desktopová verze. Zadané téma práce bylo řešeno ve spolupráci s firmou Edhouse s.r.o. Vytvořený software bude sloužit společnosti Thermo Fisher Scientific Brno s.r.o.

Je na místě zmínit, že existuje i původní okenní aplikace pro konfiguraci mikroskopu. Největším jejím úskalím nejsou ani tak použité starší technologie, ale především absence dynamicky generovaného uživatelské rozhraní. Pokud přijde nový požadavek na přidání nové komponenty nebo jinou změnu do této aplikace, vývojář je nucen změnit i uživatelské rozhraní. Tyto požadavky přicházejí poměrně často, protože komponenty i software mikroskopů je neustále rozšiřován. To vyžaduje instalaci na mikroskop, což znamená i rozšíření uživatelského rozhraní konfigurační aplikace. Další nevýhodou staré aplikace je pouze její desktop podpora.

Proto vznikla iniciativa vytvořit zcela novou aplikaci konfigurátoru a zmíněné nedostatky v ní napravit. Jednak přidat podporu pro generování uživatelského rozhraní, ale rovněž vytvořit aplikaci s podporou webové i desktop verze. Webová podpora aplikace bude mít využití hlavně do budoucna, kdy dojde k jejímu nasazení na server. K tomu jsou však potřeba ještě další změny v instalačním procesu.

Dalším vítaným rozšířením, které zjednoduší úpravy do konfigurátoru je jistě i XML editor. Uživatel je schopen změnit uživatelské rozhraní konfiguračních kroků a posléze si provedené změny zobrazit přímo v aplikaci. Nedílnou součástí aplikace jsou i konfigurační pravidla. Ty nám udávají jakým způsobem můžeme daný typ mikroskopu nakonfigurovat, a které komponenty nesmíme zapomenout vybrat. Konfigurační pravidla můžeme opět editovat v administrátorském módu aplikace.

Jednou z posledních funkcí je spuštění aplikace jen v režimu pro čtení. Ten neumožňuje navolenou konfiguraci nainstalovat na mikroskop, ale pouze ji uložit do textového souboru. Při příštím spuštění aplikace pak máme na výběr, jestli chceme načíst uloženou konfiguraci z registrů nebo právě z tohoto textového souboru.

2 Uvedení do problematiky

Oblast elektronové mikroskopie je sama o sobě velkým tématem. I samotná instalace, konfigurace a vývoj mikroskopu obecně obsahuje spoustu nových pojmů, proto mi dovolte některé z nich krátce vysvětlit.

2.1 Elektronový mikroskop

Rozlišujeme dva základní typy elektronových mikroskopů. Prvním z nich je tzv. transmisní elektronový mikroskop, který je někdy označován pouze zkratkou TEM. Druhým typem je rastrovací elektronový mikroskop. Ten můžeme najít pod zkráceným názvem SEM. My se budeme především zabývat konfigurací mikroskopu prvního typu, tzn. TEM. Mimo toto základní rozdělení existují ještě různé kombinace těchto typů.

Elektronový mikroskop typu TEM obsahuje bezpočet hardwarových komponent, různých rozšíření a specializované aplikace, které je nutné nainstalovat. Jde například o různé detektory, optické nastavení, typy kamer a jejich filtrů a mnoho dalšího. Dále u tohoto typu mikroskopu rozlišujeme dvě základní produktové rodiny, do kterých patří specifické řady mikroskopů a jejich generace. Nejedná se obecně o rozdělení pro elektronové mikroskopy tohoto typu , ale o rozdělení podle produktu společnosti Thermo Fisher Scientific. K ovládání mikroskopu a používání specializovaných aplikací slouží tzv. MPC. Na něho se instaluje software a provádí další nastavení.

2.2 Instalace mikroskopu

V praxi se setkáme z několika přístupy nastavení a instalací mikroskopu. Jedná se zejména o různá prostředí, kde chceme provést prvotní instalaci mikroskopu.

2.2.1 Prostředí u zákazníka

Klasickým příkladem je prostředí u zákazníka, který si elektronový mikroskop zakoupil. Po sestavení hardwarových komponent a částí mikroskopu servisními techniky, následuje zprovoznění a instalace softwaru na MPC. K instalaci a nastaveni komponent použije servisní technik instalační iso. Z něho si spustí instalační program. Jako první se automaticky nainstaluji chybějící prerekvizity. Potom proběhne kontrola volného místa na disku a jako následně se spustí aplikace konfigurátor.

Servisní technik vybere daný typ mikroskopu, hardwarové komponenty a software, který potřebuje nainstalovat a spustí instalaci. Je pravda, že pokud technik instaluje mikroskop přímo u zákazníka, tak má většinu komponent i softwaru už předvybranou. To do jisté míry zajištující konfigurační pravidla, ale i zadání produktového čísla mikroskopu před zahájením jeho konfigurace. Funkcionalita

pro zadávání produktových čísel mikroskopů do aplikace konfigurátor, však není součástí její nové verze. Je v plánu ji však přidat do budoucna.

Po spuštění instalace se automaticky začnou instalovat všechny vybrané komponenty a jejich software. Po dokončení celé instalace je ještě nutné restartovat MPC. Po tomto kroku je už mikroskop a jeho software plně nastaven a je možné ho začít používat.

2.2.2 Vývojová laboratoř

Dalším prostředím, kde se setkáme z potřebou nainstalovat mikroskop je vývojová laboratoř. Zde se testují a sestavují nové typy mikroskopů, testuje se i jejich software a nová nastavení. Je proto žádoucí mít možnost v instalačním programu a především v aplikaci konfigurátor vyzkoušet různé kombinace hardwarových komponent a softwaru. Například pověřený pracovník v laboratoři potřebuje z nějaké důvodu vyměnit určitý detektor u mikroskopu za jiný typ. Jako první půjde vyměnit hardwarovou komponentu detektoru a následně musí překonfigurovat mikroskop. V konfigurátoru musí zvolit daný typ detektoru, který použil.

2.2.3 Simulace MPC

Při vývoji softwaru a testování se často setkáme s použitím simulace MPC. Jedná se o simulační obraz reálného MPC. Je možné si ho u sebe nainstalovat jako virtuální PC. Abychom na něm mohli testovat náš software je opět nutné provést instalaci prerekvizit a samotného mikroskopu. V tomto případě si můžeme vybrat jaký typ mikroskopu chceme nainstalovat. Záleží na tom, na čem chceme testovat daný software. Stejně tak si můžeme v aplikaci konfigurátor libovolně zvolit komponenty a další předinstalovaný software. Pokud chceme něco po instalaci změnit, například doinstalovat software nebo změnit konfiguraci, použijeme k tomu opět instalační program a aplikaci konfigurátor.

2.3 Uživatelské role

Na základě výše popsaných pracovních prostředí můžeme zavést základní uživatelské role pro aplikaci konfigurátor. Definujme proto následující role:

- **Standardní uživatel** - Servisní technik nebo jiný uživatel, který instaluje mikroskop běžným způsobem.
- **Developer** - Vývojář mikroskopu v laboratoři.
- **Administrátor** - Správce aplikace konfigurátor. Jde o nově zavedenou roli sloužící pro vývojáře aplikace.

2.4 Konfigurační pravidla

Pokud se více zaměříme na jednotlivé hardwarové komponenty, je třeba dodat, že ne všechny jsou mezi sebou kompatibilní. Případně ne všechny jsou podporované pro daný typ mikroskopu. Dále existují určité závislosti mezi komponentami. Pokud je určitá komponenta vybraná je nutné vybrat i tu komponentu, na které závisí. Nebo naopak, pokud je vybrána daná komponenta, není už možné zvolit tu, která s ní není kompatibilní. Setkáme se i s pravidly definující minimální zvolenou sadu komponent a naopak i maximální počet komponent společného typu. Aby jsme se v pravidlech mohli lépe orientovat je vhodné si je přesně definovat, proto zaveďme následující konfigurační pravidla:

- **Requires** - Komponenta A vyžaduje komponentu B. Komponenta B je předvybrána nebo je vynuceno ji vybrat.
- **Incompatible** - Komponenta A je nekompatibilní s komponentou B. Pokud je komponenta A vybrána, komponentu B nelze vybrat.
- **CompatibleOnly** - Komponenta B je kompatibilní pouze s komponentou A. Pokud komponenta A není vybrána, nelze vybrat ani komponentu B.
- **RequiresSelection** - Pokud je vybrána komponenta A, je požadováno vybrat i komponentu B.
- **Not** - Komponenta nesmí být vybrána. Pokud je vybrána, je vyžadováno její výběr zrušit.
- **AtMost** - Je dovoleno vybrat maximálně N komponent stejné skupiny.
- **AtLeast** - Je požadováno vybrat minimálně N komponent stejné skupiny.
- **Equivalent** - Komponenta A je ekvivalentní s komponentou B. Pokud je vybrána komponenta A, je vybrána i komponenta B. Pokud komponenta A není vybrána, není vybrána ani komponenta B.
- **Exclusive** - Komponenta A vylučuje komponentu B. Když je vybrána komponenta A nejde vybrat komponenta B. Platí i obráceně.
- **ConditionalDefault** - Komponenta je ve výchozím stavu předvybrána, ale její výběr lze zrušit.

2.5 Aplikační módy

Aplikace konfigurátor obsahuje tři základní aplikační módy a jeden speciální mód, který jde s nimi kombinovat. Aplikační mód je režim aplikace, který je určený pro danou uživatelskou roli. Jde o režim spuštění aplikace, přepínání módů v již spuštěné aplikaci není běžným scénářem. Proto přepnutí módu nelze udělat z uživatelského rozhraní, ale pouze pomocí nastavení parametrů při spuštění aplikace.

Rozlišujeme tyto módy:

- **Standard** - Výchozí mód aplikace pro standardního uživatele. Hodnoty konfigurace jsou načítány a ukládány do registrů.
- **Developer** - Rozšíření standardního módu určeného pro vývojáře mikroskopu v laboratoři. Aplikace obsahuje navíc komponenty, které nejsou dostupné ve standardním módu. Některá konfigurační pravidla jsou zde ignorována.
- **Administrátor** - Aplikační mód pro vývojáře aplikace. Obsahuje XML editor pro editaci konfiguračních kroků a jednotlivých komponent. Je zde možnost přidat novou komponentu nebo smazat již existující. Dále XML editor podporuje i editaci konfiguračních pravidel. V tomto módu nejsou hodnoty konfigurace načítány z registrů, ale pracuje se pouze s výchozími hodnotami. Konfigurační pravidla jsou aplikovány pouze v režimu XML editoru, který slouží pro editaci pravidel.
- **Edit/Standalone mode** - Speciální mód, který lze kombinovat pouze s módy standard a developer. Uživatel si může vybrat jestli vstupní hodnoty konfigurace načte z registrů nebo z textového souboru. Nakonfigurované hodnoty pak nelze uložit do registrů, ale pouze do souboru. Tento mód tedy neslouží pro běžné použití při instalaci mikroskopu, ale pouze pro různé scénáře, kdy je potřeba si vyzkoušet specifickou sadu konfiguračních hodnot.

2.6 Instalační proces

1. Jakmile si uživatel úspěšně nakonfiguruje mikroskop, tzn. splní všechna pravidla a dostane se v průvodci konfigurace až nakonec, může spustit instalaci.
2. Nakonfigurované komponenty se ve tvaru klíč-hodnota uloží do registrů na MPC.
3. Spustí se instalační program, který si jako první načte uložené hodnoty v registrech.
4. Pomocí nich vyselektuje hardwarové komponenty a software, který je třeba nainstalovat. U hardwarových komponent se převážně jedná o instalaci ovladačů a dalšího nastavení.
5. Instalační program sekvenčně spustí instalátory jednotlivých komponent a softwaru. Vše běží automaticky není už potřeba žádný zásah uživatele.

2.7 Užitečné pojmy

- **Konfigurace** - Nakonfigurovaná sada vybraných komponent. Je reprezentována jako slovník ve tvaru klíč-hodnota.
- **Feature** - Společné označení pro hardwarovou komponentu nebo softwaru jednoznačně určeného pomocí unikátního jména.
- **Konfigurační položka** - Označení featury z hlediska generovaní uživatelského rozhraní.

3 Generované uživatelského rozhraní

Generované uživatelské rozhraní můžeme v různých typech aplikací udělat více způsoby. Záleží vždy na možnostech daného typu aplikace a technologii, kterou používáme pro tvorbu uživatelského rozhraní. V principu jsou však všechny generované uživatelské rozhraní podobné.

Potřebujeme na základě nějakého vstupu uživatelské rozhraní dynamicky vygenerovat. Může se jednat o datový vstup nebo vstup od uživatele. Datovým vstupem můžeme označit pevně stanovený způsob nebo strukturu dat, podle které chceme uživatelské rozhraní vygenerovat. Naopak generování na základě uživatelského vstupu odpovídá vytvoření uživatelského rozhraní podle dat, které zadal uživatel. Zjednodušeně lze říci, že oba způsoby jde vlastně sloučit do jednoho, protože i v druhém případě opět generujeme uživatelské rozhraní ze vstupních dat.

V naší aplikaci budeme uživatelské rozhraní generovat na základě dat ze souboru. Základní koncept generování uživatelského rozhraní ze souboru lze shrnout v následujících krocích:

1. Načtení, zpracování souboru a uložení získaných dat do příslušných datových objektů.

2. Vytvoření systému komponent, z kterých se bude generovat uživatelské rozhraní.

3. Vygenerování uživatelského rozhraní pomocí komponent a podle vstupních objektů s daty.

3.1 Požadavky na uživatelské rozhraní

Uživatelské rozhraní, které chceme generovat, se skládá z jednotlivých konfiguračních stránek. Každá stránka obsahuje konfigurační položky, které jsou zobrazeny samostatně nebo jsou zahrnuty do sdružujících skupin. Dále jsou konfigurační položky na stránce rozděleny do několika sloupců. Máme dva typy konfiguračních položek:

- **Enumerace** - Seznam hodnot vymezující kategorii komponent, z které může být zvolena jen jedna komponenta.

- **Booleovská hodnota** - Hodnota True/False, která symbolizuje výběr dané komponenty.

3.2 Struktura vstupních dat

Vstupem pro vygenerování uživatelského rozhraní je XML soubor. Označme tento soubor jako FeatureList. Pro každou rodinu mikroskopů budeme mít i jeden speciální FeatureList. Je to z toho důvodu, že každá rodina mikroskopů se liší v

sadě konfigurační položek. Byla by možná i varianta mít pouze jeden FeatureList pro obě rodiny a odlišné položky v něm vhodně parametrizovat. Tuto variantu jsem však vyloučil, protože by to zhoršilo rychlou orientaci v datech. Vstupní XML soubor obsahuje následující elementy:

- **FeatureList** - Kořenový element XML souboru, který v sobě obsahuje jednotlivé elementy stránek konfigurace.
- **FeaturePage** - Definuje stránku konfigurace a její položky, které jsou do tohoto elementu zanořené. Obsahuje pouze povinný atribut pojmenování stránky, který nemusí být unikátní.
- **FeatureEnumeration** - Definuje konfigurační položku typu enumerace. Obsahuje povinný atribut pojmenování unikátní v rámci elementů stejného typu a povinný atribut definující počáteční hodnotu enumerace.
- **EnumerationValue** - Definuje jednu z hodnot enumerace. Obsahuje povinný atribut pojmenování, který musí být unikátní pouze v rámci jednotlivých hodnot dané enumerace.
- **FeatureBool** - Definuje konfigurační položku typu booleovská hodnota. Obsahuje povinný atribut pojmenování unikátní v rámci elementů stejného typu a povinný atribut definující povinnou počáteční hodnotu True/False.
- **FeatureGroup** - Jedná se o sdružující element, do kterého je možné zanořit další elementy definující konfigurační položky (FeatureEnumeration, FeatureBool). Dále je do tohoto elementu možné zanořit další sdružující elementy a do nich opět zanořit elementy definující konfigurační položky. Tímto elementem můžeme vytvořit jakýsi strom zanoření společných skupin a podskupin konfiguračních položek. Element opět obsahuje povinný atribut pojmenování unikátní v rámci elementů stejného typu.
- **FeatureGeneralization** - Sdružuje názvy konfiguračních položek do generalizace. Obecně se jedná o komponenty splňující nějakou společnou vlastnost. Například typy mikroskopů, které všechny podporují pouze daný detektor nebo třeba jen specifický typ kamery. Element obsahuje povinný atribut pojmenování unikátní v rámci elementů stejného typu. Nejedná se o element, který by byl nějak vizualizován v uživatelském rozhraní. Slouží primárně pro seskupení konfigurační položek do společné generalizace. Využití těchto dat najdeme převážně při definování konfiguračních pravidel, kdy nám společná generalizace dává možnost se odkazovat skupinově k více konfiguračním položkám. Jednotlivé položky generalizace definujeme pomocí elementů Child s atributem názvu položky. Do generalizace můžeme zapsat položky v těchto tvarech:
  - **Enumerační hodnota**: Název komponenty typu enumerační hodnota. Je zadávána ve tvaru: [název enumerace].[název enumerační hodnoty].

- **Booleovská hodnota**: Název komponenty typu booleovská hodnota.
- **Enumerace**: Název celé enumerace.
- **Generalizace**: Název generalizace sdružující několik komponent dohromady. Tento tvar nám umožní sdružit generalizace do jedné společné.

Kromě povinných atributů mohou elementy FeatureEnumeration, EnumerationValue, FeatureBool a FeatureGroup obsahovat tyto nepovinné atributy:

- **UIName** - Definuje zástupné jméno konfigurační položky, které je použito pouze pro zobrazeni v uživatelském rozhraní.
- **Column** - Udává číslo sloupce, ve kterém se konfigurační položka na stránce zobrazí. Aktuálně je podporovaný rozsah 1 až 3.
- **FeatureVisibility** - Definuje v jakém módu aplikace je konfigurační položka zobrazena. Pokud tento atribut není nastaven, je položka zobrazena ve standardním módu aplikace.

Elementy FeatureEnumeration a FeatureGroup mohou navíc obsahovat ještě tyto nepovinné atributy:

- **NameIsHidden** - Skryje název sdružující skupiny v uživatelském rozhraní.
- **BorderIsHidden** - Skryje rámeček sdružující skupiny v uživatelském rozhraní.

3.3 Převedení dat do interních objektů

Ještě předtím než načteme vstupní XML soubor a začneme zpracovávat data, je nutné se zamyslet jakým způsobem budeme vstupní data ukládat. Musíme data uložit v takové formě, abychom z nich byli schopni vygenerovat uživatelské rozhraní a to včetně grafických znázornění všech podskupin na konfiguračních stránkách.

Navíc musíme data uložit do formy vhodné pro přenos na webového klienta, kde během přenosu dochází k serializaci dat do JSON formátu a po přenesení dat k jejich následné deserializaci. To byl hlavní důvod proč jsme nemohli ke každému typu elementu ze XML souboru jednoduše vytvořit příslušný objekt a ten pak dát přidat do společného seznamu všech konfiguračních položek na stránce. Pokud bychom nemuseli pracovat s JSON formátem, tak by to samozřejmě šlo a použili bychom k tomu třeba datovou kolekci ArrayList. Dala by se ještě zvážit možnost použití datové struktury slovníku i pro převod do JSON formátu, kde by klíčem byl typ konfigurační položky a hodnotou pak samotný objekt, který by tento typ reprezentoval. Jistě by se našlo i spoustu jiných řešení.

Nakonec jsem se však rozhodl rozdělit konfigurační položky podle typů elementů v XML souboru do jednotlivých seznamů a takto je přenášet na webového

klienta. Hlavním důvodem byl zřejmě lepší přehled všech použitých dat pro generování a i o něco lepší filtrování v jednotlivých seznamech. V neposlední řadě mi to dalo možnost se odstínit od rozdělení konfiguračních položek podle jednotlivých stránek, ale pohlížet na ně více skupinově pouze podle jejich typů.

Tím že jsme data rozdělili do jednotlivých seznamů podle jejich typů, jsme však ztratili jejich provázanost, kterou definovala struktura XML souboru. Mám na mysli určení na kterou konfigurační stránku jednotlivé položky patří nebo jak jsou strukturovány v podskupinách. Proto vzniklo několik pomocných objektů a identifikátorů, které zajišťují tuto provázanost, a pomoci kterých jsme schopni uživatelské rozhraní podle vstupních dat správně vygenerovat.

3.4 Vizualizace v uživatelském rozhraní

V uživatelském rozhraní potom systém do sebe vnořených grafických komponent zajišťuje vygenerování konfiguračního průvodce. Trochu složitějším bodem bylo vygenerování do sebe různě vložených podskupin konfiguračních položek. Hlavně z toho důvodu, že vstupní data dopředu neinformovala o tom kolik těchto zanoření bude. Tuto informaci by jistě šlo do dat přidat nebo ji uložit do interních objektů, jakmile by se sečetl počet zanoření.

Od této varianty jsem však upustil, protože jsem chtěl mít řešení generovaného uživatelského rozhraní co nejvíce obecné. Pro tento účel byla vytvořena grafická komponenta pro vizualizaci podskupiny konfiguračních položek, která se rekurzivně vkládá sama do sebe tak dlouho než se nedojde k poslednímu zanoření.

4 Konfigurační pravidla

Podobně jako jsme si definovali vstupní data pro vygenerování uživatelského rozhraní musíme se i více zaměřit na vstupní data pro konfigurační pravidla. Hlavním cílem je jejich dobrá čitelnost v datech, snadná editace a rozhodně i vytváření nových pravidel. Běžný scénář, který se opakuje při požadavku pro přidání nové komponenty do konfigurátoru je totiž následující. V první řadě je třeba rozšířit FeatureList o nové komponenty a zařadit je na požadovanou stránku konfigurace. Druhým krokem je dodefinování konfiguračních pravidel pro nově přidané komponenty, pokud samozřejmě tyto nové komponenty pravidla vyžadují.

4.1 Struktura dat

Zdrojem pro definici konfiguračních pravidel je opět XML soubor. Pro každou produktovou rodinu existuje vlastni definice pravidel. Motivaci pro toto rozdělení bylo jako v předchozím případě přehlednost a dobrá čitelnost dat. Pro každý typ pravidla existuje jeden XML element. Pravidla dělíme na jednořádková a skupinová.

4.1.1 Jednořádková pravidla

Jednořádková pravidla ve většině případů obsahují první povinný atribut pro tzv. hlavní komponentu a druhý povinný atribut pro komponentu, která je s ní ve vztahu daného pravidla. Třetím povinným atributem je textový popis, který se zobrazí v uživatelském rozhraní jako vysvětlení pravidla. Jako nepovinný je atribut pro určení ignorování pravidla ve vývojářském módu aplikace. Do konfiguračních pravidel můžeme zadávat komponenty v různých tvarech stejně jako u generalizace:

- **Enumerační hodnota** - Pravidlo se bude vztahovat pouze na tuto enumerační hodnotu.
- **Booleovská hodnota** - Pravidlo se bude vztahovat pouze na tuto booleovská hodnotu.
- **Enumerace** - Pravidlo se bude vztahovat na všechny hodnoty dané enumerace.
- **Generalizace** - Pravidlo se bude vztahovat na všechny komponenty obsažené v generalizaci a to i z hlediska průchodu do hloubky.

Mimo základní jednořádková pravidla, kde se atributy pro zadávání komponent liší jen v jejich názvech, máme i další speciální pravidla, která mají jinou strukturu. Oproti ostatním pravidlům obsahují tyto rozdíly v atributech:

- **Not** - Obsahuje pouze jeden atribut pro název komponenty. Do pravidla je možné zadat pouze enumerační hodnotu.

- **AtMost** - Obsahuje atribut pro název komponenty a číselný atribut pro definování maximálního počtu vybraných komponent. Pravidlo pracuje pouze s generalizacemi.
- **AtLeast** - Obsahuje atribut pro název komponenty a číselný atribut pro definování minimálního počtu vybraných komponent. Pravidlo pracuje pouze s generalizacemi.
- **ConditionalDefault** - Neobsahuje atribut pro popis pravidla, protože pravidlo slouží pro nastavování počátečních hodnot. Do pravidla je možné zadat pouze enumerační nebo booleovskou hodnotu.

4.1.2 Skupinová pravidla

Skupinová pravidla jsou rozšířením těch jednořádkových. Umožňují nám zadat do běžných pravidel skupinu komponent, na které se bude pravidlo vztahovat. Skupina komponent se může vztahovat pouze na jednu stranu pravidla. Jsou pravidla, kde můžeme skupinu komponent zadat jako hlavní komponenty nebo jako skupinu ovlivněných komponent. Více komponent můžeme do pravidla zadat pomocí elementu Feature a atributu názvu komponenty.

4.2 Vyhodnocování pravidel

Vyhodnocování všech pravidel probíhá zpravidla na začátku při načtení konfigurace podle vybrané produktové rodiny. Dále k němu však dochází při každé uživatelské změně vybraných položek konfigurace. Je nutné zkontrolovat všechna pravidla, jestli provedenou změnou od uživatele nedošlo k jejich porušení. Pokud k porušení pravidel došlo, je nutné uživatele správně upozornit, proč je vybraná konfigurace z hlediska pravidel nevalidní. Postupně jak uživatel řeší všechny konfigurační problémy jsou pravidla znovu vyhodnocována až dojde k validní konfiguraci a uživatel může pokračovat v instalaci.

Pravidla se vyhodnocují sekvenčně, tak jak jsou pod sebou zapsána ve vstupním XML souboru. I když dojde k porušení jednoho pravidla, nejprve se dokončí vyhodnocení všech ostatních pravidel a až potom je uživatel upozorněn. V každém pravidle se nejprve kontroluje zda byla vybrána hlavní komponenta. Pokud byla vybrána, pokračuje se ve vyhodnocování pravidla. U skupinových pravidel, se v některých pravidlech kontroluje výběr alespoň jedné komponenty nebo výběr úplně všech komponent ze skupiny. Pravidla se podle různých typů a tvarů komponent vyhodnocují následovně:

- **Requires** - Pravidlo povinného výběru se podle tvaru požadované komponenty vyhodnocuje takto:
  - **Enumerační hodnota**: Daná komponenta se předvybere. Jakýkoliv jiný výběr položky z dané enumerace bude zakázán.

  - **Booleovská hodnota**: Daná komponenta se předvybere. Bude zakázáno zrušit výběr této komponenty.
  - **Enumerace**: Pokud ani jedna položka z enumerace není vybrána nebo je zvolena pouze tzv. None hodnota, přidá se varování pro uživatele o nutnosti vybrat hodnotu z dané enumerace
  - **Generalizace**: Nejprve se rekurzivním průchodem do hloubky zkontroluje zda pravidlo bylo porušeno. Pokud ano, je vyžadován výběr alespoň jedné položky z dané generalizace. Přidá se popis pravidla jako upozornění pro uživatele a dané položky generalizace budou v uživatelském rozhraní označeny.

- **Incompatible** - Pravidlo nekompatibilnosti se vyhodnocuje následovně:
  - **Enumerační hodnota**: Komponenta, na kterou odkazuje enumerační hodnota, se odvybere, pokud byla vybrána a enumerace se nastaví na hodnotu None. Bude zakázán další výběr dané enumerační hodnoty.
  - **Booleovská hodnota**: Komponenta, na kterou odkazuje booleovská hodnota, se odvybere a bude zakázán její další výběr.
  - **Enumerace**: Hodnota enumerace se nastaví na hodnotu None a bude zakázán další výběr všech jejich položek.
  - **Generalizace**: Položky generalizace se projdou do hloubky a pro každou enumerační a booleovskou hodnotu se nastaví jejich nekompatibilita.

- **CompatibleOnly** - Pokud není vybrána hlavní komponenta, je druhá komponenta nekompatibilní pro všechny ostatní komponenty z dané skupiny komponent. Pro nekompatibilní komponentu se provedou kroky podle pravidla nekompatibilnosti.

- **RequiresSelection** - Pokud požadovaná komponenta není vybrána, přidá se popis pravidla jako upozornění pro uživatele a označí se komponenty, které je nutné vybrat. Pro generalizaci se opět provede průchod do hloubky a v uživatelském rozhraní se označí všechny její položky.

- **Exclusive** - Pokud je vybrána první komponenta pravidla, druhá komponenta se nastaví jako s ní nekompatibilní. Pokud naopak je vybrána druhá komponenta, bude jako s ní nekompatibilní nastavena ta první.

- **Equivalent** - Pokud je vybrána první komponenta, pro druhou komponentu se použije pravidlo povinného výběru. Pokud první komponenta vybrána není použije se pro druhou komponentu pravidlo nekompatibilnosti.

- **Not** - Pravidlo jako vstup bere pouze jednu komponentu typu enumerační hodnota. Pokud enumerační hodnota je vybrána, přidá se popis pravidla

jako upozornění pro uživatele a všechny ostatní položky enumerace budou v uživatelském rozhraní označeny. Pravidlo se často používá jako negace None hodnoty pro enumeraci, u které vyžadujeme výběr jedné z jejich položek.

- **AtMost** - Pravidlo podporuje jako vstupní komponentu pouze generalizaci. Nejprve se zkontroluje zda nebyl dovršen maximální počet vybraných komponent z dané generalizace. Pokud se tak stalo, zbylé položky z generalizace už nepůjde vybrat.

- **AtLeast** - Pravidlo opět podporuje jako vstupní komponentu pouze generalizaci. Nejprve se zkontroluje, zda byl vybrán alespoň minimální počet požadovaných komponent z dané generalizace. Pokud ne, přidá se do uživatelského rozhraní popis pravidla jako upozornění pro uživatele, že musí vybrat ještě některé další komponenty z generalizace.

- **ConditionalDefault** - Pravidlo jako vstup podporuje pouze enumerační nebo booleovskou hodnotu. Pokud je vybrána první komponenta, pokračuje se ve vyhodnocování pravidla. Zkontroluje se zda druhá výchozí komponenta ještě nikdy nebyla změněna uživatelem nebo že nebyla zakázána jiným pravidlem či zda není od nějakého pravidla vyžadováno ji vybrat. Pokud ani jedna podmínka nenastala, tato výchozí komponenta se vybere.

Skupinová pravidla:

- **RequiresOneOfThem** - Jako skupinu komponent můžeme zadat komponenty na druhé straně pravidla. Vyhodnocuje se, zda byla alespoň jedna ze skupiny komponent vybrána. Pokud ne, vybere se první komponenta ze skupiny.

- **EveryOfThemRequires** - Jako skupinu komponent můžeme zadat komponenty na první straně pravidla. Pokud byla vybrána kterákoliv komponenta ze skupiny, použije se pravidlo povinného výběru pro požadovanou komponentu.

- **IncompatibleWithAllOfThem** - Skupina komponent je opět na druhé straně pravidla. Použije se pravidlo nekompatibilnosti pro všechny komponenty ze skupiny.

- **EveryOfThemIncompatible** - Pokud byla vybrána kterákoliv komponenta ze skupiny, použije se pravidlo nekompatibilnosti s komponentou na druhé straně pravidla.

- **CompatibleOnlyAnyOfThem** - Komponenta na druhé straně pravidla je kompatibilní pouze s komponentami ze skupiny.

- **RequiresSelectionOneOfThem** - Je požadováno vybrat alespoň jednu komponentu ze skupiny.

- **RequiresSelectionWithAllOfThem** - Je požadováno vybrat komponentu na druhé straně pravidla, pokud byly vybrány všechny komponenty ze skupiny.
- **AtLeastNumberOfThem** - Pravidlo se vyhodnocuje stejně jako jednořádkové pravidlo AtLeast s tím rozdílem, že můžeme zadat skupinu komponent typu generalizace.

4.3 Validace pravidel

Pravidlo je nevalidní, pokud zadaná komponenta neexistuje. FeatureList vůbec neobsahuje název dané komponenty nebo byla komponenta zadaná ve špatném tvaru. V tomto případě se přeruší vyhodnocování zbývajících pravidel, sada pravidel je nevalidní.

Uživatel dostane upozornění o tom, že komponenta použitá v pravidlu se nenalezla. Pokud například právě editoval pravidla v administrátorském módu aplikace, musí opravit název komponenty, aby mohl pokračovat dál.

5 Architektura projektu a použité technologie

Při výběru technologie pro webovou aplikaci, jsem musel zvážit některé požadavky, které na mě byly kladeny. Musel jsem vzít v potaz, že vytvořená aplikace může být ještě v budoucnu dále rozšiřována. Určitě do ní bude potřeba přidávat data nových komponent a jejich pravidel. Dále je v budoucnu v plánu nasadit aplikaci na webový server, až na to bude instalační proces připraven. S tím mohou být spojeny ještě další úpravy v aplikaci. Vytvořená aplikace a její další podpora se začlení pod vývojářský tým, který má na starost celý instalační proces. V tomto vývojářské týmu je hlavní znalost v technologii .NET a zkušenost s podporou a rozšiřováním staré okenní aplikace napsané v této technologii.

Původně jsem zvažoval sáhnout po nějakém tradičním javascriptovém frameworku jako je Angular, React nebo třeba Vue.js, ale nakonec jsem musel zvolit webovou technologii z prostředí .NET. A to hlavně z toho důvodu, aby členové z vývojářského týmu se mohli snadněji seznámit a proniknout do nově vytvořené aplikace. Zároveň jsem chtěl zvolit moderní technologii a vyvarovat se starším webovým technologiím v prostředí .NET. Po zvážení všech možnosti jsem se rozhodl vytvořit aplikaci v technologii Blazor.

5.1 Blazor

Dříve než se zaměříme na samotný Blazor, musíme trochu více vysvětlit co je to tzv. WebAssembly. Tradiční webové aplikace jsou v poslední době na straně front endu nejčastěji psány pomocí javascriptových nebo typescriptový frameworků. Jedná se například o Angular nebo třeba React. [1]

Můžeme říci, že WebAssembly je nový webový standard, který může být v prohlížeči reprezentován jako bytecode nebo může být i v binárním formátu. Obecně z hlediska rychlosti zpracování a spuštění by tato reprezentace měla být lepší než javascriptový kód. Neznamená to ale, že bychom mohli javascript plně nahradit za WebAssembly. Stále budeme potřebovat javascript pro přístup k DOMu a na další specifické operace. Většina moderních prohlížečů už dnes podporuje WebAssembly a i z hlediska bezpečnosti je na tom WebAssembly podobně jako javascript. [1]

Blazor je moderní webový framework, který umožňuje vyvíjet webové aplikace pomocí C# a Razoru. Do Razor souborů můžeme definovat HTML šablonu se speciálními Razor značkami, které nám umožňují provolávat C# funkce. Samotný C# kód může být obsažen přímo v Razor souboru nebo může být oddělen v samostatném souboru za použití tzv. partial class.

Z hlediska .NET spadá Blazor pod skupinu ASP.NET Core. Výše bylo zmíněno, že WebAssembly nedokáže pracovat s DOMem. Blazor tento problém řeší tak, že si udržuje svůj virtuální DOM a ten pravidelně synchronizuje se skutečným DOMem. Proto tuto synchronizaci je využit právě javascript. Blazor nabízí dvě možnosti jak aplikace vyvíjet. [1]

5.1.1 Blazor Server

V tomto případě aplikace neběží jako WebAssembly, ale je spuštěna na serveru a komunikaci s prohlížečem zprostředkovává technologie SignalR. Prohlížeč zde zastupuje pouze roli zobrazovače, aplikace v něm neběží. V případě otevření aplikace ve více instancích prohlížeče, má každý prohlížeč potažmo uživatel také vlastní instanci aplikace. Mohlo by se zdát, že to musí představovat určitou výkonnostní zátěž na server, ale není to tak hrozné. Samozřejmě záleží na dostupnosti a spojení na server. Rozhodně bude patrný rozdíl mezi spuštění serveru jako localhost a mezi nasazením aplikace například na vzdálený Windows server. Pokud však zvážíme otázku bezpečnosti, tak rozhodně aplikace, která běží pouze na serveru, na tom bude dobře. [1]

5.1.2 Blazor WebAssembly

Runtime webové aplikace je postavený na WebAssembly, zkompilovaná data běží přímo v prohlížeči. Teoreticky nám pro spuštění této aplikace budou stačit i statické soubory hostované na webového serveru. A to je jeden z hlavních důvodů proč k aplikaci v Blazor WebAssembly potřebujeme mít právě i webový server, který bude tuto aplikaci hostovat. Návrhově jde tedy o tradiční pojetí webové aplikace jako server a klient. [1]

5.2 Architektura a struktura projektu

Z důvodu vlastních preferencí i klasického pojetí webové aplikace jako server a klient jsem se rozhodl pro vývoj aplikace použít právě Blazor WebAssembly. Klientská část aplikace je tedy postavena na této technologii a pro webový server je použita technologie ze skupiny ASP.NET Core, která je určena pro tento typ projektu. Pro klientskou část byl kladen důraz na zachování konceptu tenkého klienta. Všechna hlavní logika je umístěna na webovém serveru. Komunikace a přenos dat mezi klientem a webovým serverem probíhá standardně pomocí REST architektury. Pokud budeme brát klientskou část pouze jako systém komponent pro zobrazení dat bez žádné další logiky, můžeme říci, že v projektu je zachován i koncept MVC architektury.

5.2.1 Back end

Back end z hlediska architektury tvoří webový server zahrnující REST API, které je jediným přístupovým bodem k serveru. Toto API je zdokumentované pomocí technologie Swagger, která nabízí i možnost si API vyzkoušet. Jednotlivé přístupové body API jsou definovány podle MVC architektury v kontrolérech. Pokud se mezi serverem a klientem posílají složená data v objektech, jsou překládány do datových transakčních objektů, zkráceně do DTO. Data se tedy přenáší výhradně v těchto objektech a v kontrolérech potom probíhá mapování na doménové objekty. Pokud přijde URL request na některý přístupový bod API je obecně zpracován následovně:

1. Pokud URL nebo tělo requestu obsahuje datové transakční objekty, jsou namapovány na doménové objekty.
2. Zavolá se příslušná metoda na aplikační vrstvě a předají se ji vstupní parametry.
3. Vrácená doménová data jsou namapována na příslušné transakční objekty.
4. Namapovaná data jsou předána jako výsledek URL requestu.

Vykonání příslušné metody jako první probíhá v aplikační vrstvě. Z ní mohou být provolány pomocné metody dalších logických vrstev. Webový server je tedy rozdělen do několika vrstev:

- **API vrstva** - Definuje veřejné API serveru, které slouží jako vstupní brána pro komunikaci s klientem. Pro klienta tato část představuje jakýsi interface, s kterým může komunikovat. Vrstva obsahuje pouze definici kontrolérů bez další větší logiky. Kontroléry jsou rozděleny a pojmenovány podle logických částí aplikace. Pro dokumentaci REST API slouží technologie Swagger. Vizualizuje nám názvy kontrolérů, jejich metody s příslušnými parametry a strukturu datových objektů používaných v parametrech metod. Můžeme si pomocí této technologie celé API vyzkoušet.
- **Aplikační vrstva** - Jedná se o hlavní logickou vrstvu serveru, která spojuje ostatní vrstvy dohromady. Obsahuje několik aplikačních servis rozdělených podle logických částí aplikace. Každá serviska obsahuje aplikační metody, které mohou být provolány z API vrstvy. V těchto metodách je obsažena jen minimální logika. Najdeme v nich převážně jen provolání metod jednotlivých servisek z utility nebo datové vrstvy.
- **Utility vrstva** - Obsahuje několik projektů, v kterých najdeme servisky s podrobnější logikou. Jde například o zpracování vstupních XML souborů, implementaci systému pravidel a dalších pomocných servisek.
- **Datová vrstva** - Zde najdeme jak statická data vstupních XML souborů a definici jejich šablon tak i specifické datové servisky. Patří k nim například práce s Windows registry nebo s registrovými soubory.

Ve všech čtyřech vrstvách je použitý systém registrace komponent (v našem případě servisek) pomocí Dependency Injection, někdy označovaného pouze zkratkou DI. Tento koncept registrace komponent rozhodně není vhodný pro všechny typy aplikací. Obecně se jedná o návrhový model, kdy jednotlivé komponenty jsou od sebe odděleny natolik, že jedna neví co dělá ta druhá. Daná komponenta potřebuje jen získat určitá data nebo vykonat jednostrannou operaci, a tak pouze zavolá metodu jiné komponenty a nanejvýš ji předá vstupní parametry. Jedná se tak o aplikace, kde je zjevný nějaký izolovaný systém komponent. [2]

V našem případě komponenta představuje nějakou servisku, která slouží pro získání dat nebo vykonání specifické operace. Tyto servisky jsou použity pouze pro provolávání jejich metod. Rozhodně se z nich nevyčítají žádná data, ani neslouží k jejímu uložení. Data z nich lze získat pouze jako návratové hodnoty volaných metod.

Ke každé servisce musí existovat i její rozhraní. Serviska a rozhraní jsou pak společně registrovány v kontejneru aplikace. V aplikaci se potom pracuje jen s rozhraními daných servisek. Kontejner aplikace zajistí vytvoření nebo znovupoužití instance na danou servisku. Servisky tak pracují jen s kontejnerem aplikace a nemusejí se starat o vytváření instancí komponent, s kterými potřebují pracovat, ani je nezajímá co konkrétně jim potřebnou servisní komponentu poskytne. Od toho právě slouží kontejner aplikace, aby to odstínil a zabezpečil poskytnutí instance na správnou komponentu. Vše co serviska musí udělat je jen to, že si danou závislost předá ve svém konstruktoru. [2]

Tento návrhový vzor jsem vybral jednak proto, že jsem s ním již měl nějaké zkušenosti pro tento typ aplikací, ale taky kvůli tomu, že mi přesně pro tuto strukturu projektu připadalo vhodného ho použít. Použití tohoto návrhového vzoru by nám mělo zjednodušit práci tím, že se nemusíme tolik starat o závislosti komponent. Rozhodně nám DI také pomůže psát čitelnější kód. V neposlední řadě je tento způsob doporučován na větší projekty nebo na projekty, které mají tendenci se rozrůst do budoucna. [2]

Systém DI je implementovaný přímo v ASP.NET Core, pro naše účely je však použita knihovna Castle Windsor. Tuto technologii jsem zvolil, protože nabízí některé rozšířené možnosti DI a opět jsem s ní měl nějaké zkušenosti z jiných projektů. Pomocí této technologie se registrují nové servisky do kontejneru aplikace.

5.2.2 Front end

Front end aplikace tvoří v hlavní řadě webová aplikace napsaná v technologii Blazor WebAssembly. Dále tu pak máme pomocné projekty, které obsahují jen minimální logiku. Tuto prezentační část aplikace můžeme rozdělit do tří vrstev:

- **Data Access** - Zprostředkovává přístup k datům z webového serveru. Tuto vrstvu tvoří jeden projekt, který obsahuje několik servisek. Ty v sobě obsahují za pomoci HTTP klienta volání na přístupové body API. Každá serviska obsahuje vytvoření HTTP requestu a předání vstupních parametrů buď jako součást URL nebo jako obsah těla requestu. Po získání odpovědi na HTTP request má dále na starosti kontrolu návratového kódu a vyčtení vrácených dat. Data jsou v HTTP requestu mezi serverem a klientem přenášena v datové struktuře JSON. Je proto třeba zajistit serializaci dat při přenosu z klienta na server a následnou deserializaci dat při odpovědi ze serveru.

- **Store** - Představuje vnitřní stav webové aplikace. Blazor vnitřní stav aplikace neboli tzv. state management nijak neřeší, neposkytuje k tomu přímo

nějakou vlastní podporu. Uvádí jen možnosti jak vnitřní stav aplikace implementovat. Pro Blazor Server by jistě mohlo být řešením ukládat data přímo na serveru do vhodné datové reprezentace. U Blazor WebAssembly by zase byla možnost využít tzv. session, což je úložný aparát přímo v prohlížeči. Případně můžeme implementovat nějakou globální třídu a ukládat vnitřní stav aplikace přímo do paměti. Ani jedno řešení mi však nepřipadne pro naše účely vhodné a navíc představuje jistou míru implementace navíc. Proto jsem se rozhodl zvolit již hotové řešení podpory vnitřního stavu aplikace a vybral jsem technologii Fluxor.

Fluxor je Blazor knihovna, která je postavená na technologii Redux. Ten obsahuje podporu pro vnitřní stav aplikace a celý state management. Redux je hojně využíván v Reactu a do jisté míry na něm staví i Angular. Navíc i pro Fluxor lze do vývojářské konzole v prohlížeči přidat rozšíření na Redux devtools. Ty nám zajistí lepší sledování vnitřního stavu aplikace a rozšíří nám to možnosti ladění.

State, který definuje vnitřní stav aplikace, je hlavním stavebním kamenem Fluxor Storu. Můžeme mít jen jeden State pro celou aplikaci a nebo více definic State, rozdělených podle sekcí aplikace. State můžeme měnit pouze pomocí tzv. Reducerů, což jsou metody, které reagují na vyvolání akce.

Akce je vlastnoručně definovaný objekt, který může obsahovat vstupní parametry pro předání dat. Akci můžeme vyvolat z Razor komponenty pomocí rozhraní IDispatcher. Jakmile je akce vyvolána přejde se do její Reducer metody a změní se vnitřní stav aplikace podle definice. Dalším aparátem, který Fluxor nabízí, jsou tzv. efekty.

Efekt je definovaná metoda, která podobně jako Reducer metoda reaguje na vyvolané akce s tím rozdílem, že efekt nijak nepracuje s vnitřním stavem aplikace. Můžeme v něm vykonat jakoukoliv vlastní logiku. Primárně se používá pro asynchronní načtení dat ze serveru. Pro změnu vnitřního stavu aplikace můžeme po načtení dat pomocí rozhraní IDispatcher vyvolat akci, která bude napojena na nějakou Reducer metodu.

Jednou z dalších možností jak reagovat na akce je použití rozhraní IActionSubscriber. To můžeme použít v Razor komponentách a navázat na vyvolání akce další logiku.

Pořadí aktivace jednotlivých Fluxor aparátů po vyvolání akce je potom následující:

1. Jako první se zavolá Reducer metoda a změní se vnitřní stav aplikace.
2. Následuje vyvolání efekt metody.
3. Jako poslední se dostaneme do těla metody navázanou na rozhraní IActionSubscriber.

Pokud pro jakýkoliv se zmíněných aparátů neexistuje metoda navázána na

vyvolanou akci, bude se pokračovat kontrolou dalšího aparátu v definovaném pořadí.

- **Razor komponenty** - Jedná se o poslední vrstvu prezentační části aplikace. Razor komponenty tvoří v aplikaci jakousi hierarchii. Jsou do sebe navzájem zanořeny, a to od nejspecifičtějších komponent po ty nejobecnější. Například existuje komponenta pro kroky konfigurace, která má do sebe vnořené komponenty pro konfigurační stránky a ty zase používají další komponenty pro jednotlivé prvky uživatelského rozhraní. Platí pravidlo, že komponenty, které leží nejníže v této struktuře, obsahují jen minimální logiku a označujeme je jako UI komponenty.

  Razor komponenty pracují s vnitřním stavem aplikace. Můžeme v nich dělat tyto operace:

  1. Vyčítat hodnoty z vnitřního stavu aplikace pomocí definice State. V případě více definic State se musíme odkazovat na ten konkrétní, který obsahuje námi požadovaná data.
  2. Provolat nějakou námi definovanou akci, která změní vnitřní stav aplikace. K tomu použijeme rozhraní IDispatcher.
  3. Navázat nějakou logiku na konkrétní vyvolanou akci pomocí rozhraní IActionSubscriber.

5.3 Podpora desktop verze

V Blazoru podporu desktop verze, někdy též označovanou jako podporu hybridní verze, můžeme vyřešit třemi základními přístupy. Obecně potřebujeme, abychom mohli naši webovou aplikaci bez větší námahy převést na desktop. Současně chceme zachovat i podporu webové verze a způsob vývoje aplikace. Rozhodující je i na jaké platformě chceme desktop verzi používat. Dalšími kritérii je například podpora offline módu, vlastní instalátor aplikace nebo spustitelnost aplikace pouze z příkazové řádky. Po zvážení všech podmínek, které požadujeme, si můžeme vybrat z těchto možností.

5.3.1 Blazor WebView

Jedna z prvních možností je použít Blazor WebView. V podstatě jde o pohled z webové aplikace vložený do standardní desktop aplikace. V první řadě je nutné převést Blazor aplikaci na Razor knihovnu. U Blazor WebAssembly to znamená jen drobné úpravy projektu, naopak pro aplikaci vytvořenou jako Blazor Server je nutné oddělit Razor komponenty do vedlejší knihovny. Druhým krokem je vytvoření desktop aplikace, která bude sloužit jen jako obálka pro Razor knihovnu. Z .NET technologií pro desktop aplikace a s cílenou platformou pro Windows máme na výběr z WPF nebo Windows.Forms. Pokud bychom chtěli zvolit multiplatformní přístup, je pro nás jasná volba .NET MAUI. Posledním

krokem je použití komponenty Blazor WebView z nuget balíčku a napojení na Razor knihovnu. [3]

Výsledkem bude standardní desktop aplikace s možností spuštění z příkazové řádky pomocí EXE souboru. Budeme však hlavně postrádat podporu offline módu. Celkově je toto řešení vhodné spíše pro méně náročné aplikace. Z architektonického hlediska jde však o již nemoderní přístup při řešení podpory hybridní verze aplikace.

5.3.2 Blazor PWA

Blazor PWA neboli progresivní webová aplikace je moderní přístup při řešení podpory hybridní verze. V dnešní době již poměrně hodně webových aplikací dostupných z prohlížeče nabízí i možnost si stáhnout a lokálně si nainstalovat PWA verzi. Stačí k tomu jen v prohlížečích Chrome nebo Edge v nabídce vpravo nahoře si PWA verzi nainstalovat. Aplikace se nainstaluje pod Chrome případně Edge aplikace, podle toho z kterého prohlížeče jsme instalaci spustili.

Technicky jde o aplikaci postavené na Chromium, které umožňuje spuštění a běh webové aplikace. Po instalaci lze aplikaci spustit i z příkazové řádky. Stačí k tomu znát ID aplikace, které jí bylo přiřazeno při instalaci, a způsob jak aplikaci spustit. Obojí bychom měli vyčíst ze zástupce na PWA aplikaci. [3]

Blazor WebAssembly přímo podporuje PWA verzi aplikace. Jde přímo vytvořit projekt už s touto podporou, případně lze pomocí drobných úprav podporu PWA aplikace přidat. PWA aplikace obsahuje podporu offline verze, která lze v Blazor projektu nakonfigurovat. Základní nastavaní PWA aplikace jako je její název, popis a další nastavitelné parametry můžeme udělat pomocí konfiguračního JSON souboru, který se označuje jako manifest.

Menší nevýhodu bych viděl v instalaci aplikace. Bez akce uživatele, který v prohlížeči potvrdí, že chce PWA aplikaci nainstalovat, vlastně není jiný způsob jak rovnou desktop verzi spustit bez provedení tohoto kroku. Dá se říct, že jde pouze o jakousi obálku nebo zástupce webové aplikace, který sám o sobě nepředstavuje plnohodnotnou desktop aplikaci. Nenalezl jsem jednoduchý způsob jak PWA aplikaci spustit bez instalace v prohlížeči nebo jak zabalit potřebné soubory pro její spuštění. Naopak přednost tohoto řešení je v rychlosti instalace a spuštění aplikace. PWA aplikace působí svižně a vlastně nepoznáte větší rozdíly oproti webové verzi.

5.3.3 Blazor a Electron

Electron je technologie, která umožňuje webovou aplikaci napsanou ať už pomocí javascriptových frameworků nebo i Blazoru zabalit do desktop aplikace. Pro Blazor je nutné pro podporu Electronu použít NuGet balíček Electron.NET a doplnit potřebné úpravy do projektu. Lze to udělat jak pro Blazor Server tak i pro Blazor WebAssembly. Dále je nutné si globálně nainstaloval i NuGet balíček ElectronNET.CLI, který nám umožní zavolat příkaz pro sestavení Electron aplikace z příkazové řádky. [3]

Při sestavení se nejprve přeloží celý projekt a doinstalují se potřebné NPM balíčky pro vytvoření Electron aplikace. Nakonec probíhají ještě nějaké optimalizace a samotné sestavení. Výsledkem je samostatný instalátor aplikace, pomocí kterého můžeme aplikaci nainstalovat kam potřebujeme. Nebo můžeme použít už rozbalenou strukturu, která obsahuje spustitelný EXE soubor a všechny přeložené DLL soubory a další závislosti aplikace. Takto sestavená Electron aplikace v sobě obsahuje i Chrominum, které opět zajišťuje spuštění a běh webové aplikace. Při prvním sestavení Electron aplikace může celý proces trvat delší dobu. [3]

Balíček Electron.NET podporuje široké nastavení aplikace. Můžeme nastavit velikost okna, název aplikace nebo i cestu kam chceme výsledné soubory sestavení uložit. Dále je možné zakázat některé chování okna desktop aplikace, povolit či zakázat klávesové zkratky. Máme možnost Electron aplikaci věrohodně připodobnit klasickým desktop aplikacím. Nastavení můžeme udělat pomocí konfiguračního souboru electron.manifest.json nebo přímo v kódu před vytvoření okna pro electron aplikaci. Aplikace lze pomocí EXE souboru standardně spustit z příkazové řádky.

První spuštění aplikace může zabrat delší dobu, u opětovného spuštění je už toto zpoždění skoro zanedbatelné. Tento problém však jistě vyřeší vhodná načítající obrazovka aplikace neboli splash obrazovka, která jde do Electronu nakonfigurovat tak, aby byla zobrazena než dojde k samotnému načtení hlavního okna aplikace.

5.3.4 Požadavky a zvolená varianta

Požadavky na desktop verzi vycházející z potřeb integrování aplikace do instalačního procesu jsou následující:

1. Potřebujeme sestavit potřebné soubory pro spuštění desktop verze v rámci buildu aplikace. Můžeme k tomu klidně využít speciální nástroj a příkaz, který se pak připojí k buildu. Výsledné soubory při budoucím nasazení aplikace budeme chtít přidat na instalační ISO.

2. Desktop aplikaci musí jít spustit z příkazové řádky pomocí spouštějícího souboru. Nemusí se jednat přímo o EXE soubor, ale klidně o nějaký skript, který to zajistí.

3. Dále požadujeme podporu zadávání volitelných argumentů z příkazové řádky při spuštění aplikace. Ty nám budou sloužit k přepínání módů aplikace.

Během analýzy a implementace jsem vyzkoušel všechny tři základní přístupy podpory hybridní verze, které jsou pro Blazor vhodné. Jako první jsem začínal s Blazor WebView. Od tohoto řešení jsem však odstoupil, protože mi nevyhovoval nemoderní návrh z pohledu architektury. Měl jsem také obavy ze svižnosti aplikace a nemožnosti budoucí podpory offline režimu. Z hlediska architektury bych se víceméně musel vzdát pojetí webové aplikace jako server a klient a všechnu

logiku přemístit pod Razor knihovnu. To byl asi hlavní důvod proč jsem v tomto řešení dále nepokračoval.

Při zvolení za hlavní technologii Blazor WebAssembly a uspořádání struktury aplikace jako server a klient, bylo jako další nejrychlejší řešení vyzkoušet PWA. To Blazor WebAssembly přímo podporuje a stačí k tomu jen několik málo úprav. S tímto řešením jsem v průběhu implementace vystačil poměrně dlouho. Musel jsem však najít cestu jak obejít nutnost instalace PWA aplikace z prohlížeče. Zkoušel jsem nějak zautomatizovat tento proces, vytvořit nějaký skript co by to dokázal a sloužil by zároveň jako spouštěč aplikace. Nemohl jsem však stále přijít na vhodné řešení, vždy mě to vedlo na potřebnou akci od uživatele v prohlížeči. Některá vyzkoušená řešení mi připadala dosti kostrbatá a nesplňovala všechny požadavky, proto jsem nakonec musel zavrhnout i variantu s PWA aplikací.

Použití Electronu jsem záměrně nechal až nakonec. Na první pohled je sice zjevné, že splňuje všechny požadavky, které jsou nutné pro podporu desktop verze, ale pro samotný projekt použíti Electronu znamená další závislosti. A to hlavně závislost na NPM balíčcích, což znamená mít nainstalovanou požadovanou verzi Node.js společně s NPM. Tuto novou závislost jsem zvažoval hlavně z důvodu budoucí integrace a nasazení aplikace ve firemním prostředí. Nakonec po kratší analýze se ukázalo, že by instalace NPM balíčků neznamenala větší problém. Dokonce již ve firemním prostředí existovaly projekty, co s nimi již pracovaly. Menší obavy jsem ještě měl z doby trvání vytvoření Electron aplikace a náročnosti běhu aplikace na systém. Dále jsem na podporu Electronu chtěl přejít poměrně pozdě, když už jsem měl většinu funkcionality v aplikaci implementovanou, a proto jsem si nebyl jistý, zda bude v Electronu vše fungovat, a to včetně použití knihoven třetích stran.

Zvoleni Electronu pro podporu hybridní verze se nakonec ukázalo jako správná varianta. Desktop aplikace šla dobře přizpůsobit všem požadavkům, včetně podmínek pro chování okna aplikace. Měl jsem menší problémy s fungováním komponenty pro XML editor, ale nakonec se ukázalo, že bylo nutné zakázat některá nastavení Electron aplikace a vše se vyřešilo. Myslím si, že Electron bude plnohodnotným řešením desktop verze pro nasazení do produkce a integrováním do instalačního procesu do doby, než bude možné přejít na webovou verzi aplikace.

6 Technická dokumentace

Technická dokumentace k projektu slouží jako souhrn nejdůležitějších informací z technického hlediska. Jedná se pouze o základního průvodce k projektu pro lepší pochopení a seznámení se s použitými technologiemi a s implementací samotnou. Zdrojové soubory projektu u důležitých částí aplikace obsahují dokumentaci se základním popisem metod a jejich argumentů. V projektu dále najdete i README soubor se základními informacemi.

6.1 Požadavky pro vývoj a běh aplikace

Aplikace byla vyvíjena v prostředí Visual Studio 2022. Pro správné fungování projektu bude nutné si toho prostředí nainstalovat společně s moduly pro webový vývoj v ASP.NET Core ve verzi pro .NET 6. Po splnění této podmínky by projekt měl jít bez větších problémů načíst ve vývojovém prostředí.

Samozřejmostí je jistě i fungování projektu pouze na systému Windows. Jedním z hlavních důvodů je právě nutná závislost na Windows registrech. Pro správné fungování projektu bude nutné zkontrolovat a případně povolit čtení a zápis do registrů pro přihlášený uživatelský účet nebo obecně pro všechny. Jedná se o registry pod touto cestou:

- **Computer/HKEY_LOCAL_MACHINE/SOFTWARE**

K nastavení můžete použít systémový nástroj Registry Editor. Alternativou je spustit aplikaci jako administrátor. Pro něj by zápis a čtení pro tyto registry již měl být povolen. Následující závislosti projektu jsou již spojeny pouze s podporou desktop verze. K sestavení Electron aplikace je nutné globálně nainstalovat NuGet balíček ElectronNET.CLI v požadované verzi:

- **dotnet tool install –global ElectronNET.CLI –version 23.6.1**

Po instalaci tohoto nástroje bude ještě nutné provést příkaz pro obnovení dotnet nástrojů a restartovat příkazovou řádku nebo powershell terminál, ve kterém jsme příkaz spouštěli.

Dále je potřeba mít nainstalovaný program Node.js v minimální verzi 16.17.0. Instalaci tohoto programu proveďte včetně instalace NPM. Tento program najdete mezi instalačními soubory v přiložených souborech práce nebo je volně ke stažení na internetu. Pokud již z nějakého jiného důvodu používáte program Node.js, ale v nižší verzi a potřebujete se mezi verzemi programu přepínat, tak mohu jen doporučit nainstalovat i program NVM. Ten slouží pro jednoduché přepínání a instalování nových verzí Node.js.

Jako poslední si zkontrolujte, zda máte ve Visual Studiu nainstalovanou i podporu pro .NET 5 a případně ji doinstalujte. Balíček ElectronNET.CLI tuto verzi z nějakého důvodu ještě používá, i když i podle dokumentace by už měl být plně převeden na .NET 6.

6.2 Sestavení aplikace a první spuštění

Při prvním otevření a načtení celé solution ve Visual Studiu, by se měli automaticky stáhnout všechny použité NuGet balíčky v jednotlivých projektech. Pro samotnou kompilaci aplikace by již neměli být potřeba žádné další závislosti. Jakmile se nám podaří úspěšně aplikaci zkompilovat, můžeme vyzkoušet její první spuštění. Celou aplikaci bohužel nejde zkompilovat pro hardwarovou architekturu x86 nebo x64. Je to z toho důvodu, že projekt v Blazor WebAssembly pro svou správnou funkci vyžaduje architekturu ARM32. Nejvhodnějším řešením je tak všechny projekty kompilovat jako AnyCPU a nechat výběr architektury na kompilátoru.

Pro spuštění aplikace v základní webové verzi přímo z Visual Studia nás však čeká ještě jedna překážka. Bude nutné vytvořit lokální SSL certifikát, pokud jste tak již neučinili například při vývoji jiné aplikace, která používá webový server postavený na technologii ASP.NET Core. Vývojové prostředí by nám tuto možnost mělo automaticky nabídnout při prvním spuštění aplikace. Platnost certifikátu je po dobu 30 dnů, potom je nutné ho obnovit.

Když chceme aplikaci spustit v prohlížeči ve webové verzi, vybereme ve vývojovém prostředí hlavní projekt Configurator.Server.Api a spustíme ho. Webový server i se zabalenou klientskou částí bude ve výchozím stavu naslouchat na těchto URL adresách:

- **https://localhost:7061**
- **http://localhost:5061**

V projektu je nastaveno, že se nám při spuštění z vývojového prostředí i automaticky otevře náš výchozí prohlížeč s URL adresou spuštěné aplikace.

Pokud chceme spustit aplikaci v desktop verzi, bude k tomu potřeba se řídit těmito pokyny. Musíme sestavit Electron aplikaci neboli zabalit všechny binární soubory do spustitelného programu. V terminálu powershellu případně v příkazové řádce se přepneme do složky projektu Configurator.Server.Api a použijeme tento příkaz:

- **dotnet electronize build /target win /PublishSingleFile false /PublishReadyToRun false**

Provede se kompilace celého projektu a vytvoří se Electron aplikace. Spustitelný program najdeme pod hlavní bin složkou aplikace, konkrétně:

- **bin/Desktop/win-unpacked/Configurator.Server.Api.exe**

Cesta, kam se vytvoří spustitelný EXE soubor, jde samozřejmě i změnit v souboru electron.manifest.json, který najdete v hlavním projektu aplikace.

6.3 Popis projektů

Aplikace je z hlediska implementace rozčleněna do několika projektů. Projekty jsou rozděleny do 3 skupin a to podle základního návrhu aplikace. Máme projekty, které patří k webovému serveru, projekty pro webového klienta a sdílené projekty, které využívá jak server tak klient. Název skupiny je součástí názvu daného projektu a je přítomen i v názvu jmenného prostoru projektu.

6.3.1 Projekty webového serveru

- **Configurator.Server.Api** - Jde o hlavní projekt webového serveru a zároveň i spouštěč celé aplikace. Obsahuje hlavní soubor pro vytvoření webového serveru v ASP.NET Core včetně registrace kontrolérů, vytvoření dokumentace API ve Swaggeru, zavedení dependency injection a připojení mapovacích tříd pro transakční datové objekty. Dále soubor obsahuje konfiguraci logování aplikace a nastavení hlavní třídy pro zpracování výjimek na vrstvě API. Najdete tu i připojení Blazor WebAssembly aplikace, kterou webový server hostuje. Pro desktop verzi aplikace zde jsou i nutné nastavení pro Electron a vytvoření okna aplikace. Mimo tyto zmíněná nastavení hlavní soubor obsahuje všechny ostatní nutné konfigurace pro vytvoření webového serveru. V projektu dále najdeme třídy kontrolérů rozdělených podle domény aplikace a další konfigurační soubory.

- **Configurator.Server.Infrastucture** - V projektu najdeme hlavní infrastrukturu aplikace. V první řade kontejner aplikace vytvořený pomocí NuGet balíčku Castle.Windsor a v něm registraci všech servisových tříd a k nim příslušných rozhraní, kterou se používají v projektech spadajících pod webový server. V druhé řadě zde patří mapovací třídy pro převod z doménových datových objektů na ty transakční a naopak.

- **Configurator.Server.Shared** - Tento projekt slouží pro některé sdílené prvky pro ostatní projekty. Patří sem implementace logování aplikace včetně definice atributů, které můžeme používat pro metody. Například, když chceme zalogovat provolání dané metody nebo dobu jejího trvání.

- **Configurator.Server.Domain** - Projekt slouží pro definici tříd doménových objektů, které jsou použity mezi projekty webového serveru. Všechny doménové třídy objektů by měli být definovány právě na tomto společném místě.

- **Configurator.Server.Application** - Aplikační vrstva je tvoře převážně tímto projektem. Projekt obsahuje několik aplikačních servisek a jejich příslušná rozhraní. Jsou to servisky pro načtení a uložení konfigurace mikroskopu, dále pro vyhodnocování konfigurační pravidel nebo servisky, které slouží pro zajištění logiky XML editoru a některé další. Obecně byl v těchto

serviskách kladen důraz o dodržení minimální logiky, proto v některých případech slouží pouze jako předávače volání metod servisek z jiných projektů.

- **Configurator.Server.Store** - Projekt zahrnuje logickou část aplikace, která implementuje ukládání a načítání nakonfigurovaných hodnot z různých datových zdrojů. Tyto hodnoty jsou načítány a ukládány do datové struktury slovníku ve tvaru klíč-hodnota. Podporovanými datovými zdroji jsou registry a registrové soubory typu Regedit. Celá logika konfiguračního úložiště je udělána tak, aby ji v budoucnu bylo snadné rozšířit o další datové zdroje. Je zde implementována i podpora načítání a ukládání do více datových zdrojů. Tato logika se však zatím nepoužívá, je to příprava pro budoucí rozšíření projektu.

- **Configurator.Server.Rules** - Zde je implementována veškerá vyhodnocovací logika konfiguračních pravidel. Každému typu pravidla je přidělena vyhodnocovací funkce. Je zde zvolen sekvenční přístup, kdy vyhodnocovací funkce se volají jedna za druhou podle toho jak jsou pravidla za sebou zapsána v XML souboru. Každá vyhodnocovací metoda v podstatě vykonává následující body:

  1. Zkontroluje jestli v aktuální datové struktuře reprezentující FeatureList existují všechny komponenty, které jsou použity v pravidlu.
  2. Ověří se jestli pravidlo nemá být ignorováno.
  3. Proběhne vyhodnocení pravidla, které je pro každou vyhodnocovací metodu specifické.
  4. Pokud pravidlo bylo porušeno, uloží se popis daného pravidla pro následnou notifikaci uživatele v klientské části aplikace.

- **Configurator.Server.Utilities** - Do tohoto projektu jsou shromážděny všechny ostatní pomocné servisky, pro které není vytvořen speciální projekt. Najdeme tu hlavně servisky pro práci s XML soubory, které jsou zaměřeny na vyčítání dat a validaci nebo na získání originálního obsahu XML podoby souboru. V neposlední řadě tu je místo pro servisky, které obsahují různé pomocné funkce a nebylo je nikam jinam zařadit.

- **Configurator.Server.Data** - Pokud se podíváme do tohoto projektu na první pohled zjistíme, že projekt je místem pro statická data aplikace. Jsou tu uloženy XML soubory pro obě dvě produktové rodiny mikroskopu a to soubory pro vygenerování uživatelského rozhraní tak i soubory s konfiguračními pravidly. Ke každému XML souboru zde existuje i XSD soubor se šablonou pro validaci souboru. XML soubory jsou validovány jak při prvním načtení do aplikace, tak i pro provedení úprav v administrátorském módu. Všechny XML soubory s daty i jejich XSD soubory najdeme ve složce Resources.

Mimo statická data se v projektu nachází i servisky, které nějakým způsobem pracují přímo s daty. Patří k nim servisky pro načítání a ukládání hodnot do registrů nebo pro práci s registrovými soubory. Nesmíme zapomenout i na servisku, která poskytuje cesty ke statickým souborům s daty pro jejich budoucí načtení do aplikace.

6.3.2 Projekty webového klienta

- **Configurator.Client.WebApp** - Jak už název projektu napovídá, jde o hlavní projekt webové aplikace na straně klienta. Je to projekt vytvořený v Blazor WebAssembly. Najdeme tu všechny potřebné soubory pro vytvoření aplikace pro tento typ projektu. Dále bych popis tohoto projektu rozdělil zhruba na 3 části, které odpovídají i názvu složek v projektu.

  wwwroot - Tato složka představuje kořenový adresář, z kterého prohlížeč načte zdroje pro webovou aplikaci. Patří sem v prvé řadě soubor index.html, který je kořenovým souborem pro vytvoření webové aplikace. Obsahuje důležité nastavení pro vytvoření Blazor aplikace, importy na grafické styly a importy na javascriptové soubory, které jsou nutné pro některé grafické komponenty třetích stran použité v aplikaci. Pro grafické styly aplikace je použit **bootstrap** ve verzi 5.1.0. V kořenové složce tedy dále najdeme všechny potřebné soubory pro bootstrap, soubory s fonty a další soubory se skripty nebo s grafickými styly, které se nějak používají ve webové aplikaci.

  Components - Pod touto složkou objevíme systém Razor komponent, které tvoří uživatelské rozhraní. Jsou tu komponenty od těch nejobecnějších uživatelských prvků jako jsou tlačítka, formulářové prvky, komponenty pro typografii, tak i specifické komponenty, které už obsahují jistou logiku vykreslování a aktualizace dat. Jsou to primárně komponenty pro vygenerování uživatelského rozhraní a vytvoření konfiguračního průvodce.
  V druhé řade je to komponenta pro XML editor, která však pro vizualizaci editoru využívá komponentu třetí strany s podporou pro Blazor aplikace. Komponenty jsou od těch nejobecnějších po ty specifické do sebe navzájem vložené. V komponentách jsou využity jednak bootstrap styly tak i styly vlastně definované. Dále komponenty pro práci s daty a vnitřním stavem aplikace využívají logiku Fluxor Storu.

  Store - Podle pojmenování složky bude snad zřejmé, že jde o místo pro definici vnitřního stavu aplikace. Kromě souboru, který definuje akce pro Fluxor Store, jsou zde umístěny i čtyři podsekce vnitřního stavu aplikace rozdělené podle příbuznosti ukládaných dat. V nich jsou definované základní Fluxor soubory reprezentující jednotlivé stavy Storu. Máme stav pro obecné data aplikace jakou jsou její aplikační módy, stav pro uložení dat týkajících se samotné konfigurace mikroskopu. Potom stav pro práci s konfiguračním průvodcem a také stav pro data, které potřebuje XML editor.

- **Configurator.Client.DataAccess** - Tento projekt má na starost přístup k datům z webového serveru. Ten je zaručen implementací několika servisek rozdělených podle názvů přístupových bodů k API. V nich jsou obsaženy funkce, které zajistí vytvoření HTTP volání na server a předání vstupních dat. Následně funkce asynchronně čekají na odpověď ze serveru. Jakmile přijde, tak v případě vrácených dat je přeloží do interních objektů a vrátí jako výsledek volání dané funkce.

- **Configurator.Client.Utilities** - V projektu jsou umístěny různé pomocné servisky, které jsou volány přímo z webového klienta. Jde většinou jen o minimální logiku, která se může vykonat přímo na klientovi.

- **Configurator.Client.Types** - Jde o projekt, ve kterém jsou definovány specifické datové typy, které využívá jen webový klient.

6.3.3 Sdílené projekty

- **Configurator.Shared.Types** - Toto je projekt pro všechny sdílené typy, datové struktury, enumerace a rozhraní napříč všemi projekty aplikace. Tedy jsou určeny jak pro serverovou tak i pro klientskou část aplikace.

- **Configurator.Shared.Dto** - Zde patří definice všech datově transakčních objektů. Ty slouží pro přenos dat mezi klientem a serverem. Na straně webového serveru jsou tyto datově transakční objekty vždy překládány do interních doménových objektů a jsou využity pouze při odeslání dat směrem na webového klienta. Oproti tomu webový klient tyto data už nijak nepřevádí do interních objektů, ale rovnou s nimi v aplikaci pracuje. Kromě datově transakčních objektů v projektu najdeme i datové struktury, které sdružují transakční objekty do jednoho společného, kvůli snadnějšímu přenosu mezi serverem a klientem.

6.3.4 Testovací projekty

V průběhu implementace vznikly pro některé projekty webového serveru i jejich testovací projekty. Pro testování je použit testovací framework Xunit a další NuGet balíčky nutné pro otestování daných projektů. Chtěl bych z nich zmínit balíček, který nese název Moq a je určen pro nasimulování závislostí testované servisky.

Z napsaných unit testů bych chtěl nejvíce zdůraznit testy pro konfigurační pravidla, které mi pomáhaly ověřovat správnou implementaci pravidel a tím předejít i chybám v budoucnu.

6.4 Detaily implementace

Mezi detailnější popis implementace můžeme zařadit některá specifika kódu, která nemusejí být na první pohled vidět nebo na ně nebereme moc velký zřetel, když aplikaci používáme. Pro samotný vývoj a budoucí ladění aplikace při provozu jsou však nesmírně důležité. Konkrétně mám na mysli například zpracování a ošetření chyb nebo tvorbu logů v čase běhu aplikace.

6.4.1 Zpracování a ošetření chyb

Zpracování a následné ošetření chyb je v aplikaci ve většině případů vyřešeno podobným přístupem. Tam kde může dojít k neočekávané chybě nebo naopak známé výjimce způsobené špatným scénářem, je kód obalen do try-catch bloku. Neočekávaná nebo neznámá chyba je zpracována a následně převedena pod známý typ chyby s tím, že původní popis chyby zůstane ve většině případů zachován. Následně je takto převedená chyba předána volající funkci. Pokud se objeví již očekávaná chyba, je předána rovnou volající funkci. Jsou případy, kde se vyvolaná chyba ani nezachycuje a propluje rovnou do volající funkce. Takto je chyba předávána směrem nahoru až směrem ke kořenové volající funkci.
V případě že chyba propluje až do kontroléru API, je zachycena pomocí tzv. filtru výjimek, kde je následně zpracována. Popis chyby je zapsán do logů aplikace a chyba je předána webovému klientovi jako chybný výsledek HTTP dotazu.

Webový klient následně zkontroluje výsledek HTTP dotazu a v případě chybného návratového kódu vyvolá výjimku s chybou do volanou funkce. Volanou funkcí je Effect metoda Fluxor Storu, která vyvolanou výjimku zachytí a vyvolá metodu pro ošetření chyby. Tato metoda ve většině případů vytvoří dialogové okno s chybou, kterým informuje uživatele. Metoda pro ošetření chyby musí být Effect metodě předána v parametrech vyvolané Fluxor akce.

Chyby v aplikaci mohou být různorodé. Může jít o špatnou práci s registry, problémy se zápisem nebo čtením z nich. Dále například chyby způsobené s nenalezením souboru se statickými daty, případně chyby spojené s nepovoleným přístupem k souboru. Obecně by tedy mělo jít především o chyby, které se mohou vyskytnout při práci s daty.

6.4.2 Logy z běhu aplikace

Aplikace zapisuje logy výhradně do souboru, který najdeme ve složce pod cestou:

- **C:/ProgramData/Thermo Scientific/Log/Installer/Configurator**

Je tu přítomen jak textový soubor s logy pro poslední spuštění aplikace, tak i několik dalších souborů s logy pro předchozí běhy aplikace. Logy do serveru jsou zapisovaný pomocí aplikačního serveru, kde je pro to využit NuGet balíček log4net. V projektech aplikačního serveru můžeme do logů zapisovat informaci o volané funkci i s obsahem jejich argumentů nebo můžeme do logů zapsat vlastní

zprávu či výpis dat. V souboru s logy jednotlivé řádky vždy začínají nejprve časovou značkou vytvoření logu. Dále následuje typ zprávy, která informuje i o její závažnosti. Jako poslední je název souboru včetně jeho názvu jmenného prostoru. Ten udává odkud v projektu byla zpráva do logů zapsána. Jako poslední už je samotný obsah zprávy.

Webový klient logy nezapisuje přímo do souboru, ale posílá je pomocí HTTP požadavku přímo na server. Ten pak přijaté zprávy zapíše do souboru s logy. V něm by mělo být označeno, že jde o zprávy z webového klienta.

Pokud si otevřeme soubor s logy, potom co provedeme nějaké akce ve spuštěné aplikaci, měli bychom být schopni z něho přečíst v jakém pořadí se vykonávaly příslušné funkce na serveru a s jakými pracovaly argumenty. Najdeme v něm i výpis konfiguračních hodnot, které se na začátku načetly z registrů nebo z registrového souboru. Neměl by v něm chybět ani výpis nově nakonfigurovaných hodnot při opětovném uložení do registrů, potom co dokončíme naši konfiguraci mikroskopu.

V případě spuštění aplikace v administrátorské módu některé metody, které zajišťují práci s XML editorem, nezapisují do logů informaci o tom, že byly provolány. Je to z toho důvodu, že se mezi nimi přenášejí příliš rozsáhlá data, která by znepřehlednila soubor s logy.

6.5 Grafické komponenty třetích stran

Jeden z dalších kroků implementace byl výběr a použití grafických komponent třetích stran pro webového klienta. Použití grafický komponent třetích stran jsem zvolil pro jisté ušetření implementace některých složitějších komponent na straně klienta. Snažil jsem se hledat převážně mezi komponentami zveřejněnými jako NuGet balíčky. Největší výzva bylo najít grafickou komponentu pro XML editor, nakonec se to však podařilo.

6.5.1 BlazorMonaco

Tato komponenta nebo v tomto případě spíše knihovna třetí strany představuje použitelný textový editor pro nejrozšířenější programovací jazyky tak i pro datové struktury jako je JSON, XAML nebo právě XML. Pro některé programovací jazyky dokonce komponenta nabízí i validaci zapsaného kódu a grafické znázornění chyb. Pro XML bohužel tato podpora ještě chybí a tak je nutné ji řešit jiným způsobem. To byl jeden z hlavních důvodů proč je validace XML v aplikaci řešena až na straně serveru.

Samotná komponenta je uvnitř pouze Blazor obalem pro jinou javascriptovou komponentu, která má název Monaco. Jedná se o komponentu od Microsoftu volně dostupnou i na NPM, která graficky odpovídá podobě vývojového prostředí VS Code.

Během implementace aplikace jsem jako první použil ještě trochu jinou Blazor knihovnu, která stejně jako tato byla pouze obalem pro Monaco editor. Bohužel pro ni skončila podpora ze strany autora, a proto jsem ji musel nahradit

právě knihovnou BlazorMonaco. Ta v podstatě nabízí všechny potřebné funkce a nastavení jako ta předchozí, proto nebylo těžké ji za ni nahradit.

V komponentě editoru z knihovny BlazorMonaco máme na výběr z několika možných nastavení. Můžeme nastavit podporovaný jazyk editoru v našem případě to bude XML. Dále počáteční textový obsah editoru, povolení či zakázání minimapy a bezpočet dalších nastavení, z kterých bych ještě zmínil nastavení fontu editoru i použití vlastně definovaných stylů.

6.5.2 Blazored.Modal

Tento NuGet balíček poskytuje grafickou komponentu pro správu dialogových oken v aplikaci. Zajišťuje všechnu potřebnou logiku pro vytvoření dialogu. Nabízí i výchozí grafický styl dialogu, můžeme si však vytvořit vlastní Razor komponentu, kde si podobu dialogu přizpůsobíme našim potřebám.

6.5.3 Blazored.Toast

Notifikace pomocí tzv. toustů jsou v aplikaci použity jako podpůrná informace pro uživatele. Najdeme jej v aplikaci na místech, kde chceme uživatele informovat o tom, že nějaká akce byla úspěšně vykonána, ale on si to z nějakého důvodu nemusel v uživatelském rozhraní všimnout. Pro tvorbu a správu těchto toustů nám slouží právě tento NuGet balíček.

6.6 Vývojářské nástroje

Během vývoje aplikace jsem se setkal s některými vývojářskými nástroji, které mi usnadnili a zpřehlednili práci. Uvádím některé z nich, s kterými jsem nejvíce pracoval.

6.6.1 Visual Studio 2022

Visual Studio bylo jistě nejvíce používaným nástrojem při vývoji. Pro vývoj v .NET je myslím nejvíce používaným vývojovým prostředím vůbec. Existují však i jiné alternativy vývojových prostředí, ve kterých můžeme v této technologii vyvíjet.

Z hlediska podpory webových aplikací pro Blazor nabízí Visul Studio plnou podporu. Při vytvoření nového projektu vygeneruje ukázkovou aplikaci pro Blazor, ve které si můžeme vyzkoušet jak co funguje. I z hlediska ladění aplikace nabízí Visual Studio pro Blazor dobré nástroje. Pokud například měníme jen HTML šablonu v Razor komponentě nebo upravujeme grafické styly aplikace, můžeme použít tzv. hot reload a nové změny se nám bez restartu běhu aplikace okamžitě projeví v uživatelském rozhraní. To samozřejmě platí pouze při běhu aplikace v prohlížeči.

6.6.2 ReSharper

ReSharper je rozšiřující plugin do Visual Studia. Nabízí nejrůznější nástroje pro vývoj, umožňuje snadné generování kódu, hromadné editace a další podpůrné funkce. Je skvělým rozšířením do základního Visual Studia. Při vývoji aplikace jsem tento nástroj hojně využíval. Bohužel nabízí jen třiceti denní bezplatnou verzi, pak je nutné si licenci dokoupit.

6.6.3 Swagger

Swagger je nástroj pro dokumentaci API. Při vytvoření projektu v ASP.NET Core je již Swagger automaticky v projektu nastavený, takže se nemusíme o nic více starat. Technicky jde o NuGet balíček. Pokud je aplikace spuštěná ve webové verzi v prohlížeči stačí zadat URL adresu **https://localhost:7061/swagger** nebo **http://localhost:5061/swagger** a zobrazí se nám dokumentace k API.

Při vývoji byl pro mě tento nástroj k nezaplacení. V okamžiku, když jsem chtěl otestovat nově přidané funkce v API, použil jsem právě tento nástroj. Osvědčil se mi i při ladění aplikace a hledání chyb. Dokonce funguje i to, že pokud si ve Visual Studiu vytvořím při ladění aplikace breakpoint mohu se k němu dostat přes API metodu provolanou z dokumentace Swaggeru. Díky němu jsem nemusel vše zkoušet až na klientské části aplikace, pokud jsem věděl, že chyba bude někde na webovém serveru.

6.6.4 Webové nástroje pro vývoj

Pro vývoj front endu existuje jistě mnoho dobrých nástrojů, bohatě jsem si však vystačil pouze s nativními nástroji v prohlížeči dostupné pod klávesovou zkratkou F12. Najdeme tam webovou konzoly, nástroj pro zobrazení HTML obsahu jednotlivých elementů v DOMu a taky například monitorování HTTP komunikace.

6.6.5 Redux DevTools

Pokud používáme správu vnitřního stavu front end aplikace pomocí technologie Fluxor, můžeme využít i nástroj Redux Devtools v prohlížeči. Do Blazor aplikace je nutné doinstalovat NuGet balíček pro tento nástroj a správně ho nastavit. K tomu nám jistě poslouží dokumentace k technologii Fluxor. Do prohlížeče je pak nutné přidat rozšíření pro Redux Devtools. Bez problému by mělo jít přidat pro prohlížeče Chrome i Edge.

Ve vývojářských nástrojích prohlížeče najdeme pak toto rozšíření pod záložkou Redux. Když se do ní přepneme, uvidíme přehled vnitřního stavu aplikace. Ten nabízí jednak aktuální data vnitřního stavu aplikace, tak i seznam vyvolaných akcí, jak šli v čase za sebou. U nich si rovněž můžeme zobrazit vnitřní stav aplikace a porovnat jak se při vyvolané akci změnil.

6.7 Mock data

Protože data aplikace, které zahrnují produktové názvy mikroskopů a názvy jejich komponent, představují pro společnost Thermo Fisher Scientific příliš citlivé informace, které nechce zveřejňovat, bylo nutné v aplikaci nahradit tyto data za smyšlená. Tyto data označujeme za tzv. mock data.

Musel jsem nahradit jednak všechny data pro generování uživatelského rozhraní a to pro obě produktové rodiny, tak i data pro konfigurační pravidla. Naštěstí jsem mohl zachovat původní strukturu konfiguračních stránek i včetně definovaných pravidel. Zbývalo mi tak jen pro všechny texty použité v datech, vymyslet náhradní názvy a na všech místech, kde se vyskytovaly, je za tyto náhradní názvy nahradit. To sebou jistě neslo určitou práci navíc, hlavně když jsem musel společně s daty určenými pro generování uživatelského rozhraní kontrolovat a editovat i texty pro konfigurační pravidla.

7 Uživatelská dokumentace

Z hlediska uživatele a uživatelských rolích existuje několik návodů jak z aplikací pracovat. Aplikaci taky můžeme spustit buď přímo v prohlížeči (jsou podporované prohlížeče Chrome, Edge) nebo pomocí EXE souboru, pokud chceme pracovat s její desktop verzí. Rozdíl potom je ve způsobu přepínání módů aplikace.

7.1 Přepínání módu aplikace

Pro aplikaci spuštěnou v prohlížeči můžeme módy aplikace přepnout pomocí volitelných parametrů v URL. V případě spuštění aplikace jako localhost a použití jednoho z výchozích portů je můžeme zadat tímto způsobem:

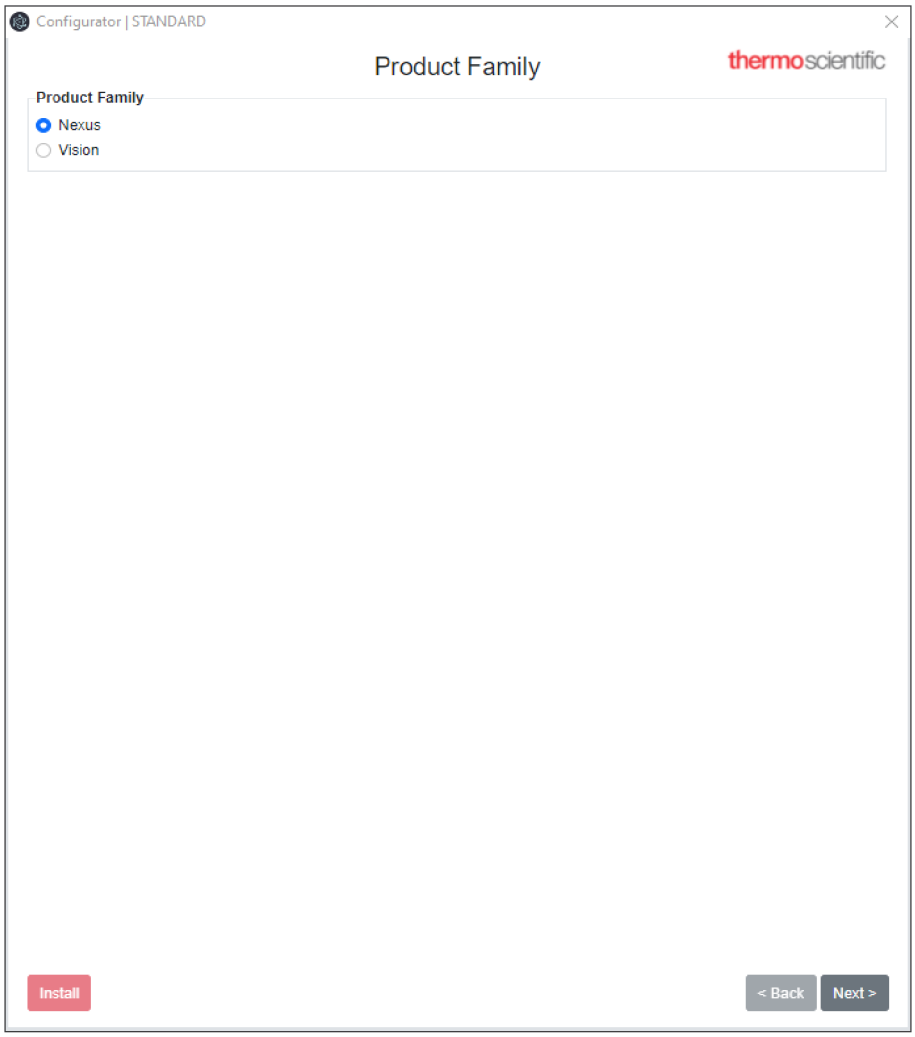
- **https://localhost:7061/?mode=standard** - Spustí aplikaci ve standardním módu. Pokud tento URL parametr nezadáme aplikace použije tento mód jako výchozí.
- **https://localhost:7061/?mode=developer** - Spustí aplikaci v developer módu.
- **https://localhost:7061/?mode=admin** - Spustí aplikaci v administrátorském módu. Je podporována i zápis parametru jako: *mode=administrator*.
- **https://localhost:7061/?edit=true** - Spustí aplikaci v rozšiřujícím edit neboli standalone režimu. Je podporována i kombinace s developer módem, použije se tento zápis: *https://localhost:7061/?mode=developer&&edit=true*.

U desktop verze je přepínání parametrů obdobné, jen se použije jiný tvar zápisu. Při spuštění Electron aplikace z příkazové řádky můžeme přidat volitelné argumenty. Zápis je následující:

- **Configurator.Server.Api.exe /args –mode=standard** - Spustí aplikaci ve standardním módu. Pokud tento argument nezadáme aplikace použije tento mód jako výchozí.
- **Configurator.Server.Api.exe /args –mode=developer** - Spustí aplikaci v developer módu.
- **Configurator.Server.Api.exe /args –mode=admin** - Spustí aplikaci v administrátorském módu.
- **Configurator.Server.Api.exe /args –edit=true** - Spustí aplikaci v rozšiřujícím edit neboli standalone režimu. Je podporována i kombinace s developer módem, použije se tento zápis:
  Configurator.exe /args –mode=developer –edit=true.

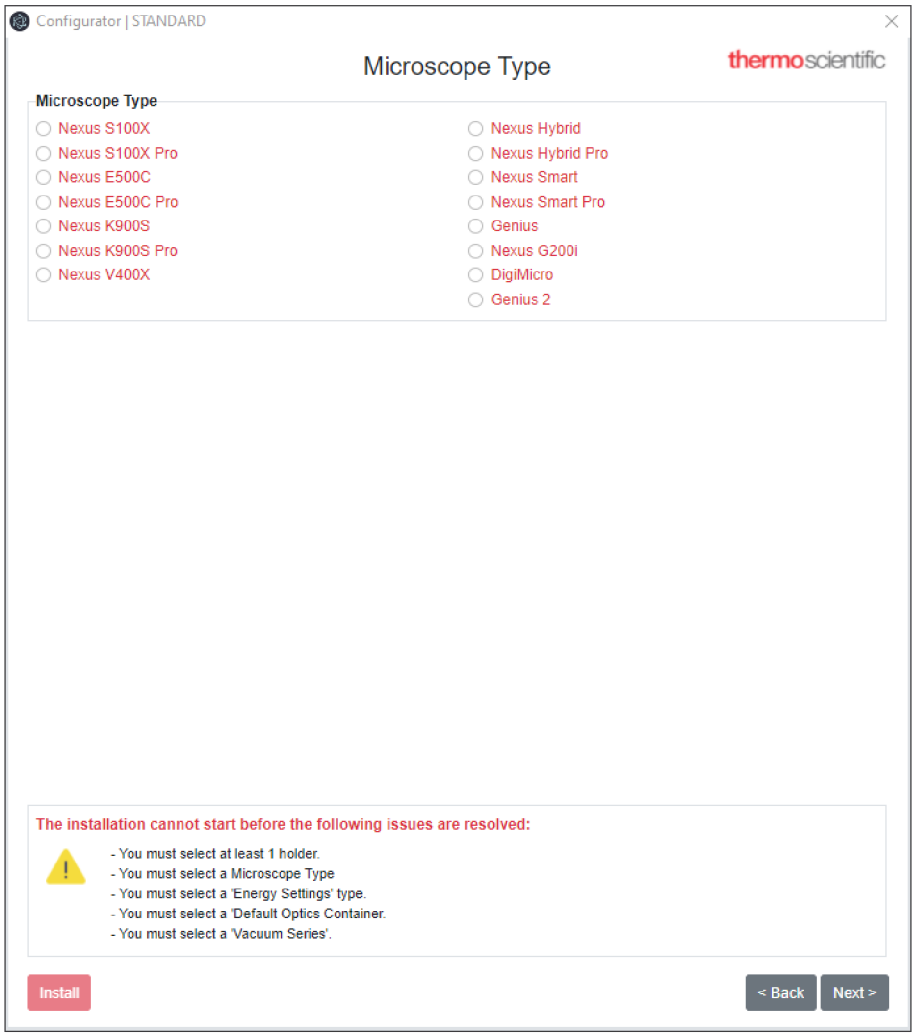
7.2 Standardní mód

Ve výchozím módu aplikace se jako první dostaneme na úvodní stránku, kde máme na výběr z dvou možností výběru produktové rodiny mikroskopu. Jednu z nich zvolíme a můžeme pokračovat na další stránku.



Obrázek 1: Úvodní obrazovka aplikace

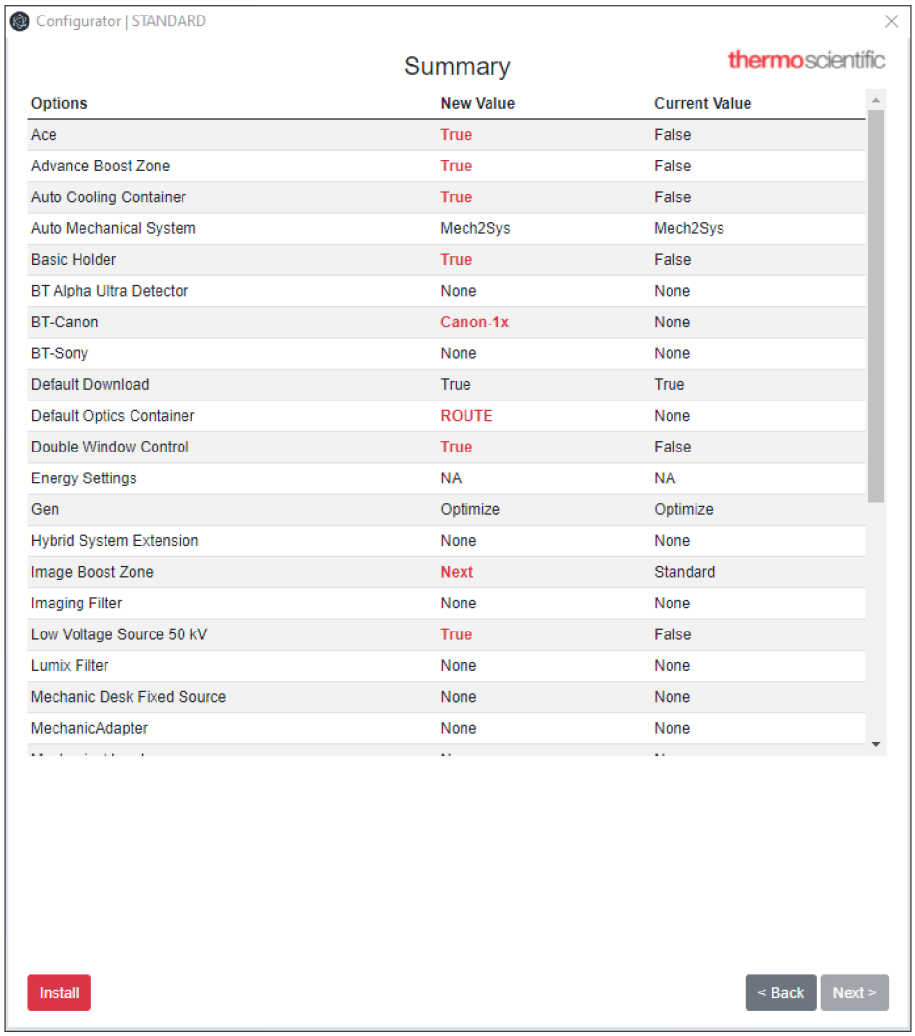
Následuje stránka nás vybízí k výběru typu mikroskopu, který chceme nainstalovat. Zároveň ve spodní části okna aplikace vidíme upozornění na konfigurační pravidla, které nebyly splněny. Dokud je nevyřešíme, nemůžeme přejít k instalaci mikroskopu. Zvolíme jeden z nabízených typů mikroskopů a postupně projdeme následující stránky konfigurace až k poslední shrnující obrazovce. Může se nám stát, že některé položky konfigurace budou kvůli pravidlům zakázány jiné naopak bude vyžadované zvolit. Pokud najedeme myší na zakázanou položku, zobrazí se nám popisek s vysvětlením použitého pravidla pro danou položku.



Obrázek 2: Výběr typu mikroskopu

Na shrnující obrazovce najdeme seznam všech položek, které jsme nakonfigurovali. Je na ní barevně rozlišeno, pokud některá položka konfigurace změnila svoji hodnotu vůči výchozí hodnotě nebo hodnotě načtené z registrů při spuštění aplikace.

Jakmile splníme všechna nevyřešená pravidla a dostaneme se až na shrnující obrazovku, můžeme pokračovat v instalaci mikroskopu. Nakonfigurované hodnoty se uloží do registrů, při příštím otevření aplikace se již načtou uložené hodnoty v registrech. Po úspěšném uložení hodnot do registrů se zobrazí toust s notifikací.



Options	New Value	Current Value
Ace	True	False
Advance Boost Zone	True	False
Auto Cooling Container	True	False
Auto Mechanical System	Mech2Sys	Mech2Sys
Basic Holder	True	False
BT Alpha Ultra Detector	None	None
BT-Canon	Canon-1x	None
BT-Sony	None	None
Default Download	True	True
Default Optics Container	ROUTE	None
Double Window Control	True	False
Energy Settings	NA	NA
Gen	Optimize	Optimize
Hybrid System Extension	None	None
Image Boost Zone	Next	Standard
Imaging Filter	None	None
Low Voltage Source 50 kV	True	False
Lumix Filter	None	None
Mechanic Desk Fixed Source	None	None
MechanicAdapter	None	None

Obrázek 3: Shrnující obrazovka konfigurace

7.3 Developer mód

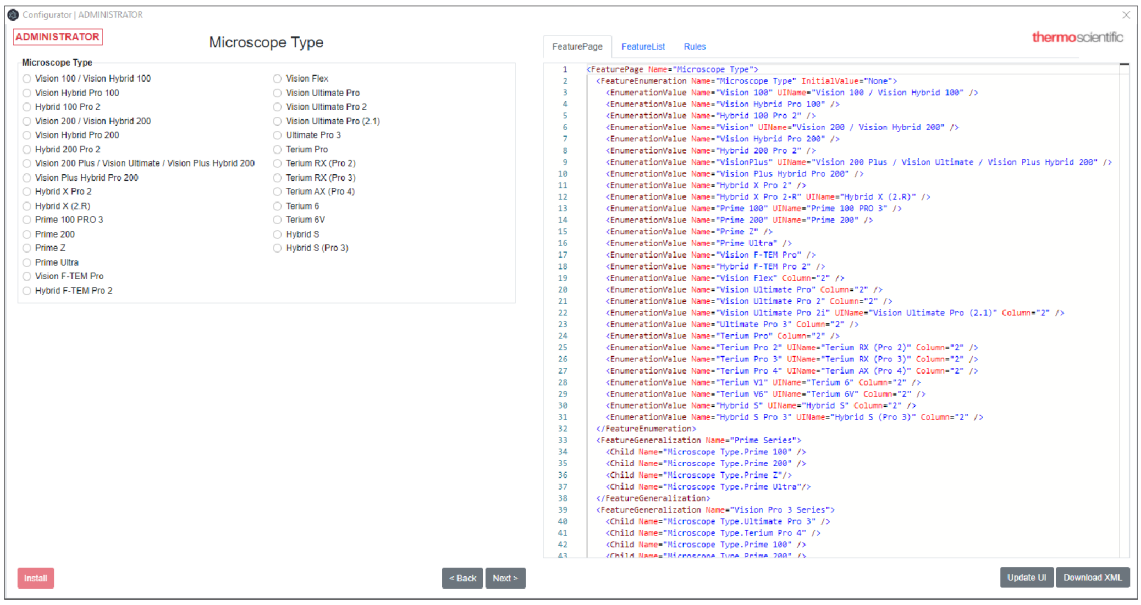
Developer mód z hlediska uživatelského rozhraní nepředstavuje mnoho rozdílů. Pouze zobrazí navíc některé konfigurační položky a bude ignorovat některá pravidla. Rozdíl mezi obsahem konfiguračních položek nejlépe uvidíme při přepínání mezi tímto a standardním módem aplikace. Upozornění na to, že jsme v tomto módu můžeme vidět v levém horním rohu okna aplikace.

7.4 Administrátorský mód

Administrátorský mód nabízí celou škálu možností jak měnit uživatelské rozhraní. Konkrétně jak editovat jednotlivé stránky konfiguračního průvodce a jak měnit pravidla. Po přepnutí do tohoto módu se nám celé okno aplikace alespoň o polovinu rozšíří a napravo se zobrazí XML editor.

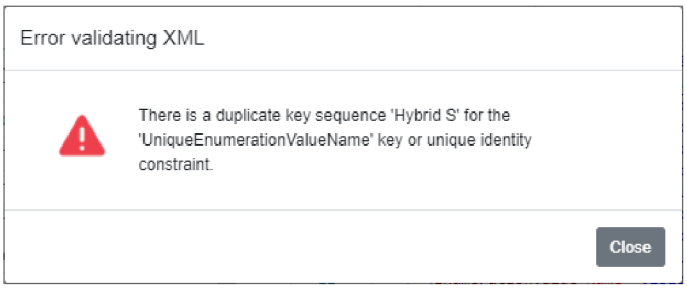
XML editor nabízí 3 režimy editace. Můžeme editovat jen aktuálně zobra-

zenou stránku v průvodci konfigurace, všechny konfigurační stránky zároveň v sekci FeatureList nebo i konfigurační pravidla na kartě Rules. Pokud jsme na kartě FeaturePage nebo FeatureList, tak nejsou na konfigurační položky aplikována žádná pravidla. Pokud naopak jsme na kartě Rules, tak jsou konfigurační pravidla aplikována. V tomto režimu XML editoru můžeme jednotlivá pravidla konfigurace editovat nebo přidávat nové. Pokud jsme v administrátorském módu nejsou do aplikace načteny konfigurační data z registrů.



Obrázek 4: Administrátorský mód aplikace

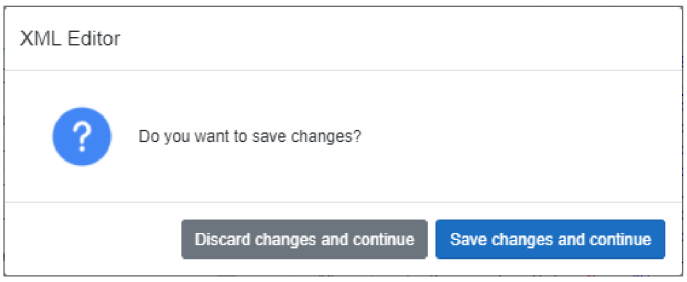
Jakmile provedeme všechny potřebné změny v rámci aktuální karty XML editoru, můžeme aktualizovat uživatelské rozhraní. Pokud jsme nedodrželi předepsaná pravidla pro úpravu XML obsahu aplikace, zobrazí se nám dialog s popisem chyby. Jakmile tyto problémy vyřešíme můžeme, se znovu pokusit aktualizovat uživatelské rozhraní. Pokud byla aktualizace úspěšná budeme o tom informováni pomocí notifikace.



Obrázek 5: Chybový dialog s popisem chyby v XML

Když provedeme nějaké změny v jedné kartě editoru a chceme přejít na jinou kartu, nebo když jsme na kartě FeaturePage a chceme přejít na další stránku,

musíme provedené změny nejprve uložit. Uložíme je tím, že aktualizujeme uživatelské rozhraní. Pokud na to zapomeneme, zobrazí se nám dialog o neuložených změnách, kde buď můžeme změny uložit nebo pokračovat bez nich. Toto platí i při neuložených změnách, když chceme na první stránce změnit produktovou rodinu. Aktuální podobu XML dokumentu si pro každou sekci může uživatel stáhnout lokálně jako XML soubor.



Obrázek 6: Informační dialog o neuložených změnách v XML editoru

7.5 Edit/Standalone mód

Editovací mód je jedním z dalším rozšířeních aplikace. Na úvodní obrazovce si uživatel musí nejprve zvolit, jestli chce počáteční hodnoty načíst z registrů nebo z registrového souboru. Fyzicky jde o textový soubor, ve kterém je dodržena struktura pro Regedit formát. Tento soubor na disku nalezneme, pokud jsme si ho nestáhli z dřívějšího běhu aplikace. Je pojmenovaný jako tzv. DAR z aplikace. Pokud samozřejmě dodržíme strukturu tohoto souboru, můžeme si ho i vytvořit a zkusit načíst do aplikace,

Po splnění prvního kroku se uživateli jako obvykle vygeneruje průvodce konfigurací, kde si může navolit položky. Po projití tohoto průvodce však nemůže pokračovat v instalaci, ale může si vybranou konfiguraci stáhnout jako textový soubor ve formátu Regedit. Konfigurační položky se uloží ve formátu klíč-hodnota.

Pokud máme aktivní tento mód, můžeme to chápat tak, že aplikace je spuštěna jen v režimu pro čtení. Případ užití tohoto módu aplikace je pro uložení nebo přenesení konkrétní navolené konfigurace a možnost jejího budoucího načtení do aplikace.

Závěr

Cílem práce bylo vytvořit plnohodnotnou aplikaci pro konfiguraci mikroskopu před jeho instalací. Aplikace měla být vytvořena tak, aby dokázala vygeneroval uživatelské rozhraní na základě vstupních souborů s daty. Byla vyžadována webová i desktop verze. Hlavním rozšířením aplikace byla podpora administrátorského módu, ve kterém může uživatel editovat vstupní soubory s daty a následně i aktualizovat uživatelské rozhraní.

Myslím, že všechny vytyčené cíle se v technické části práce podařilo naplnit. Mimo to textová část se zabývá uvedením do dané problematiky a vysvětlím některých pojmů. Dále popisuje způsob generování uživatelského rozhraní a vyhodnocování konfiguračních pravidel. Nesmíme zapomenout i na představení použitých technologiích a popis navržené architektury včetně struktury aplikace. Poslední kapitoly se věnují technické a uživatelské dokumentaci.

V závěru práce bych ještě rád něco málo zmínil o budoucím nasazení vytvořené aplikace a jejím dalším plánovaném rozvoji. Prvním krokem bude začlenění všech projektů aplikace a jejich závislostí do společného repositáře, kde jsou již přítomny všechny ostatní projekty a další aplikace spojené s instalačním procesem mikroskopu. Se začleněním projektu bude spojené i vytvoření podpory nové aplikace pro interní infrastrukturu, která provádí kompilaci projektů v repositáři a zveřejnění binárních souborů na instalační ISO.

Druhým krokem potom bude testovací pilotní provoz aplikace, před kterým je nutné doplnit do aplikace všechny chybějící data. Je to z toho důvodu, že během vývoje nové aplikace konfigurátoru nedocházelo k pravidelné aktualizaci dat. Během tohoto období přibyly nové požadavky na změnu uživatelského rozhraní, které se však musely přidat ještě do staré aplikace. Tento pilotní provoz bude sloužit převážně pro testovací účely aplikace a pro další interní použití. Nepůjde ještě o plné nahrazení aplikace v prostředí u zákazníka. Bude však možné tuto aplikaci v testovacím režimu vyzkoušet a pomocí ní nakonfigurovat mikroskop a spustit jeho instalaci.

Po tomto období, kdy bude aplikace plně připravena na nasazení do provozu, dojde k plnému nahrazení za starou aplikaci. Všechny nově příchozí požadavky na přidání nových komponent do aplikace se již budou provádět pomocí nové aplikace a jejího administrátorského módu. To rozhodně uspoří mnoho pracovních kapacit a nová aplikace tak bude obrovským přínosem pro celý vývojářský tým okolo instalačního procesu. Nemluvme o tom, že budoucí rozšiřování nové aplikace přinese týmu nové znalosti v oblasti webového vývoje.

Vytvořená aplikace byla po jejím dokončení prezentována členům vývojářského týmu. Každý z nich si ji tak mohl u sebe nejprve zprovoznit a následně i vyzkoušet její funkce. Nejvíce byli nadšeni z toho, jak jednoduše mohou změnit uživatelské rozhraní přímo v aplikaci. Mohl jsem od nich přijmout jejich zpětnou vazbu a nové nápady na rozšíření aplikace.

Conclusions

The goal of the work was to create a full-fledged application for configuring the microscope before its installation. The application had to be created in such a way that it could generate a user interface based on input data files. Both web and desktop versions were required. The main extension of the application was the support of the administrator mode, in which the user can edit input files with data and subsequently update the user interface.

I think that all the set goals were met in the technical part of the work. In addition, the text part deals with an introduction to the given issue and an explanation of some terms. It also describes how to generate the user interface and evaluate the configuration rules. We must not forget the presentation of the technologies used and the description of the proposed architecture, including the structure of the application. The last chapters are devoted to technical and user documentation.

At the end of the work, I would like to mention a little about the future deployment of the created application and its further planned development. The first step will be to integrate all application projects and their dependencies into a common repository where all other projects and other applications associated with the microscope installation process are already present. The inclusion of the project will also involve the creation of support for a new application for the internal infrastructure, which compiles projects in the repository and publishes binary files to installation ISOs.

The second step will then be the test pilot operation of the application, before which all missing data must be added to the application. This is because the data was not regularly updated during the development of the new configurator application. During this period, there were new requests to change the user interface, but they still had to be added to the old application. This pilot operation will serve mainly for testing purposes of the application and for further internal use. It will not yet be a full replacement for the application in the customer's environment. However, it will be possible to try this application in test mode and use it to configure the microscope and start its installation.

After this period, when the application is fully ready for deployment, it will be fully replaced with the old application. All newly arriving requests to add new components to the application will already be done using the new application and its administrator mode. This will definitely save a lot of work capacity and the new application will be a huge benefit for the entire development team around the installation process. Not to mention that the future expansion of the new application will bring new knowledge of web development to the team.

The created application was presented to the members of the development team after its completion. Each of them could thus first put it into operation at home and then try out its functions. They were most excited about how easily they could change the user interface right in the app. I was able to receive their feedback and new ideas to expand the app from them.

A Obsah přiložených souborů

web_configurator/

bin/

Debug/
Sestavená webová verze aplikace v případě cílové platformy AnyCPU nebo x86. Spuštění webové aplkace je možné pouze z vývojového prostředí.

Desktop/
Sestavená desktop verze aplikace. Obsahuje všechny potřebné soubory Electronu a závislosti projektů aplikace. Najdete zde EXE soubor pro spuštění desktop aplikace.

packages/
NuGet balíčky nutné pro kompilaci a spuštění aplikace. Po spuštění ve vývojovém prostředí by se měly balíčky automaticky synchronizovat.

WEB_CONFIGURATOR/
Kompletní adresářová struktura a zdrojové texty aplikace se všemi potřebnými (příp. převzatými) zdrojovými texty, knihovnami a dalšími soubory potřebnými pro bezproblémové spuštění a provoz aplikace.

doc/
Text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu, tj. zdrojový text textu, vložené obrázky, apod.

install/
Instalátory aplikací, runtime knihoven a jiných souborů potřebných pro provoz aplikace, které nejsou standardní součástí operačního.

README.md
Instrukce pro spuštění aplikace, včetně všech požadavků pro její bezproblémový provoz.

Literatura

[1] PETR OTYCH, *Blazor - část 1. - úvod, Gurot Blog* [online]. 2020. [cit. 2023-04-30]. Dostupné z: https://www.gurot.cz/blog/blazor-cast1-uvod

[2] MIREK MAREK, *Dependency Injection ještě jednou, Root.cz* [online]. 2023. [cit. 2023-04-30]. Dostupné z: https://blog.root.cz/mirek-marek/dependency-injection-jeste-jednou/

[3] EDWARD CHARBENEAU, *Blazor Hybrid Web Apps with .NET MAUI, Code Magazine* [online]. 2021. [cit. 2023-04-30].
Dostupné z: https://www.codemag.com/Article/2111092/Blazor-Hybrid-Web-Apps-with-.NET-MAUI

[4] ASP.NET CORE, *ASP.NET Core 6 Documentation* [online]. [cit. 2023-04-30]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0

[5] BLAZOR, *Blazor Documentation* [online]. [cit. 2023-04-30]. Dostupné z: https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor

[6] BLAZOR MONACO, *BlazorMonaco XML Editor* [online]. [cit. 2023-04-30]. Dostupné z: https://github.com/serdarciplak/BlazorMonaco

[7] BLAZORED, *Blazored Components* [online]. [cit. 2023-04-30]. Dostupné z: https://github.com/Blazored

[8] FLUXOR, *Fluxor State Management* [online]. [cit. 2023-04-30]. Dostupné z: https://github.com/mrpmorris/Fluxor

[9] BOOTSTRAP, *Bootstrap 5.1 Documentation* [online]. [cit. 2023-04-30]. Dostupné z: https://getbootstrap.com/docs/5.1/getting-started/introduction/

[10] ELECTRON.NET, *Electron.NET Documentation* [online]. [cit. 2023-04-30]. Dostupné z: https://github.com/ElectronNET/Electron.NET

[11] ELECTRON BUILDER, *Electron Builder Configuration* [online]. [cit. 2023-04-30]. Dostupné z: https://www.electron.build/configuration/configuration.html