VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Kvalita a měření softwarových procesů - BI tool

Software Process Quality and Measurement

Bc. Ivo Pešák

Diplomová práce

Vedoucí práce: Ing. Svatopluk Štolfa, Ph.D.

Ostrava, 2023

VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Č.j.VSB/22/078949
vsboee2202604B

Zadání diplomové práce

Student: **Bc. Ivo Pešák**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Kvalita a měření softwarových procesů - BI tool
Software Process Quality and Measurement

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem této diplomové práce je vytvořit Microsoft Power BI aplikaci pro metriky a měření kvality softwarového vývoje. Měření a s tím spojené zkoumání kvality prochází celým softwarovým vývojem od zadání projektu až po jeho ukončení a vyhodnocení. Framework pro metriky a měření kvality by měl postihnout celý tento proces. Práce navazuje na práci předchozích diplomantů.

Framework bude obsahovat zejména tyto oblasti:
1. Zaměření se na možnosti importu dat a jejich formáty, obecné template metrik atd.
2. Podporu a implementaci ukázkových vybraných metrik pro zajištění kvality vývoje, detekci chyb a vylepšování procesů.
3. BI aplikace pro implementaci všech navržených metrik, měření a vyhodnocení, metodiku pro zpracování dat.

Seznam doporučené odborné literatury:

[1] John F. Sowa, Knowledge Representation: Logical, Philosophical, and Computational Foundations, Brooks Cole Publishing Co., Pacific Grove, CA, ©2000
[2] Alec Sharp, Patrick McDermott: Workflow Modeling: Tools for Process Improvement and Application Development, Artech House; 2 edition (October 31, 2008)
[3] Pfleeger, Shari Lawrence, and Joanne M. Atlee. 2009. Software Engineering: Theory and Practice: Prentice Hall, ISBN 0136061699
[4] Pressman, Roger S. 2010. Software Engineering : A Practitioner's Approach. 7th ed. New York: McGraw-Hill Higher Education, ISBN 9780073375977
[5] Sommerville, Ian. 2010. Software Engineering. 9th ed, International Computer Science Series. Harlow: Addison-Wesley, ISBN 978-0137035151
[6]Norman Fenton, James Bieman. Software Metrics: A Rigorous and Practical Approach, (Chapman & Hall/CRC Innovations in Software Engineering and Software Development Series) 3rd Edition. CRC Press; 3rd edition (October 1, 2014) ISBN-10: 1439838224
[7] Capers Jones. A Guide to Selecting Software Measures and Metrics, Auerbach Publications; 1st edition (March 3, 2017).
Další literatura podle pokynů vedoucího diplomové práce.

17. listopadu 2172/15
708 00 Ostrava-Poruba
Česká republika

spojovatelka: +420 597 321 111
epodatelna: epodatelna@vsb.cz
ID datové schránky: d3kj88v

IČ: 61989100
DIČ: CZ61989100

email: studijni.fei@vsb.cz
www.fei.vsb.cz

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Svatopluk Štolfa, Ph.D.**

Datum zadání: 01.09.2022

Datum odevzdání: 30.04.2023

Garant studijního oboru: prof. RNDr. Václav Snášel, CSc.

V IS EDISON zadáno: 07.11.2022 11:59:22

Abstrakt

Cílem této práce je vytvořit aplikaci využívající Microsoft Power BI k zobrazování a analýze metrik pro měření kvality softwarového vývoje. Aplikace je navrhována tak, aby využila prvky automatizace pro efektivnější používání. Práce se věnuje návrhu takové aplikace na základě definovaných požadavků a procesů, které souvisí s aplikací tohoto typu. Návrh je tvořen s ohledem na moderní technologie a užitečné techniky z oblasti softwarového inženýrství. Dále práce popisuje implementaci navržené aplikace s využitím .NET technologie.

Klíčová slova

Microsoft Power BI; .NET; Metriky; Business Intelligence; Automatizace

Abstract

The goal of this thesis is to create an Microsoft Power BI application which is used to display and analyze software metrics. The application is designed with automatic features in mind for more efficient use. This thesis is focused on design of such application and takes into account the useful practises and methods in area of software engineering. Also modern technologies are kept in mind. The next thing this thesis contains is description of implementation of this application with .NET technology.

Keywords

Microsoft Power BI; .NET; Metric; Business Intelligence; Automatization

Poděkování

Rád bych poděkoval mému vedoucímu práce panu Ing. Štolfovi, Ph.D za jeho odborný přístup a rady při konzultacích. V neposlední řadě také své rodině a hlavně přítelkyni.

Obsah

Seznam použitých zkratek a symbolů

CIL	–	Common Intermediate Language
CLR	–	Common Language Runtime
REST	–	Representational State Transfer
API	–	Application Programming Interface
SPA	–	Single-Page Applications
CSV	–	Comma-separated values
SQL	–	Structured Query Language
IDE	–	Integrated Development Environment
DTO	–	Data transfer object
HTTP	–	Hypertext Transfer Protocol
JWT	–	JSON Web Token
URL	–	Uniform Resource Locator

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Při softwarovém vývoji je žádoucí nejenom vyvinout daný produkt, ale také se pokusit využívaný softwarový proces zefektivnit. Efektivnější a vyspělejší vývojový proces je výhodný z několika důvodů. První výhodou efektivního procesu je, že může ušetřit nemalé zdroje tomu, kdo jej používá. Další výhodou může být i menší chybovost při vývoji a lepší reprezentace a pochopení stavu vývoje a jeho požadavků. Je však otázkou, jakým způsobem je možné nějaký softwarový proces udělat efektivnější. Pokud bychom byli schopni jistá data o softwarovém procesu a vývoji sbírat a tato data dále pak vhodným způsobem analyzovat a aplikovat, dokázali bychom i upravit stávající proces k větší efektivitě, nebo by bylo i možné vytvořit ze starého procesu úplně nový. Je tedy možné si nadefinovat určité metriky a ty využívat pro vylepšení používaného procesu. K účelu zobrazování a analýzy metrik dnes existuje mnoho nástrojů. Jedním z nich je Microsoft Power BI.

Tato práce je zaměřena na vytvoření aplikace, která využívá Microsoft Power BI jako nástroj pro možné zlepšení vývoje. Zaměřena je také na aspekt automatizace v aplikaci a tak tedy i usnadnění používání aplikace jejím uživatelům.

Kapitola 2

State of the art

Každý proces při svém průběhu zpracovává a vytváří nějaká data, přičemž tento proces nemusí být nutně softwarový. Pokud bychom dokázali tato data správně interpretovat, mohli bychom je využít pro prospěch a zlepšení onoho procesu. Data můžeme nazvat metrikou, pokud pro ně máme nějaký kontext a při korektní analýze mohou velmi kladně přispět pro další správný vývoj projektu. Tedy například rychlejší a efektivnější psaní kódu, nebo rychlejší nasazování aplikace na koncový přístroj. Můžeme tím i rozumět lepší vývoj na trhu ve vztahu s konkurenceschopnosti produktu a tedy i lepší vývoj finanční části procesu a projektu, kterého se proces týká. Konkrétní účinky potom záleží na specifických metrikách, jež jsou sbírány a zkoumány. Business intelligence pak můžeme chápat jako zastřešující pojem pro koherentní množinu aktivit a nástrojů sloužících pro sbírání a analýzu těchto dat. [1]

Proces, který se v business intelligence využívá, by se dal obecně popsat následovně:

1. Shromažďování informací - nejdříve je nutné shromáždit metriky, se kterými můžeme později pracovat;

2. Analýza - metriky se analyzují. Je tak možné odhalit nesrovnalosti s očekávaným vývojem;

3. Zlepšení - v posledním kroku se provedou akce, které vznikly jako výstup z analýzy;

Tento proces je obecný a dá se říct, že každá firma ho má osvojený jinak a se svými specifiky, které jí vyhovují.

ID	Possible metrics (what can be measured)
SWE.1_M01	Number of system architecture requirements assigned to software and linked to software requirements / Number of all system architecture requirements assigned to software.
SWE.1_M02	Number of system requirements assigned to Software and linked to software requirements / Number of all system requirements assigned to software.
SWE.1_M03	Number of customer requirements assigned to software and linked to software requirements / Number of all customer requirements assigned to software.
SWE.1_M04	Number of system architecture requirements assigned to software and agreed by software / Number of all system architecture requirements assigned to software.
SWE.1_M05	Number of system requirements assigned to software and agreed by software / Number of all system architecture requirements assigned to software.
SWE.1_M06	Number of customer requirements assigned to software and agreed by software / Number of all system architecture requirements assigned to software.
SWE.1_M07	Number of software requirements linked to system requirements, system architecture, marked as internal requirement or customer requirements / Number of software requirements
SWE.1_M08	Number of software requirements with filled out attribute XY / Number of all software requirements
SWE.1_M09	Number of software requirements with the functional safety relevance set / Number of all software requirements
SWE.1_M10	Number of software requirements with verification criteria defined / Number of all software requirements
SWE.1_M11	Number of software requirements with release (phase) assignment / Number of all software requirements
SWE.1_M12	Number of software requirements with assigned test level / Number of all software requirements
SWE.1_M13	Number of software requirements with a status set to in work / Number of all software requirements
SWE.1_M14	Number of software requirements with a status set to changed / Number of all software requirements
SWE.1_M15	Number of software requirements with a status set to be reviewed / Number of all software requirements
SWE.1_M16	Number of software requirements with a status set to reviewed / Number of all software requirements
SWE.1_M17	Number of software requirements with a status set to implemented / Number of all software requirements
SWE.1_M18	Number of software requirements with a status set to (positively) tested / Number of all software requirements
SWE.1_M19	Number of software requirements aligned with all relevant parties / Number of all software requirements
SWE.1_M20	Number of software requirements with a status set to delivered / Number of all software requirements
SWE.1_M21	Number of functional/non-functional/process software requirements / Number of all software requirements

Obrázek 2.1: Příklad softwarových metrik [2]

Na obrázku 2.1 vidíme, že metriky si mohou být velmi podobné, co se týče jejich výpočtu. Jsou to například metriky *SWE.1_M08* a *SWE.1_M9*. Jejich význam je odlišný, ale v obou případech

se jedná o podíl dvou hodnot.

Nástrojů ke sledování metrik je k dispozici mnoho a jsou si většinou velmi podobné ve svých funkcích. Proto je zde uvedeno jen pár vybraných.

2.1 Tableau

Tableau je multiplatformní nástroj pro analýzu dat. Nabízí vizuální analýzu ve formě tvoření grafů a jiných vizuálních prvků. Nástroj uživateli umožňuje připojit se k různým datovým zdrojům a ty později zpracovávat, vytvářet z nich nástěnky a ty později sdílet s jinými uživateli v organizaci. Pro jeho používání není třeba mít žádné vývojářské znalosti, jelikož nabízí uživatelské rozhraní pro běžné uživatele a funkce jako je například Drag-Drop a jiné další, pomocí kterých je možné jednodušeji tvořit reporty. Používat Tableau je možné několika způsoby. Pracovat se dá v cloudové aplikaci - Tableau Cloud a nebo v aplikaci desktopové - Tableau Desktop. Co se týče umístění finálních dat, tak je možné využít i variantu lokálního Tableau serveru. Pro úpravu dat před tvořením grafů je nabízen nástroj Tableau Prep Builder.

Jedná se o placenou službu a její ceny jsou rozdílné pro firmu a individuální uživatele. Pro samostatné uživatele je cena 70 dolarů na měsíc. V případě firmy se ceny dělí podle nabízených nástrojů. Celkem jsou tyto licence tři a jejich ceny za měsíc jsou jednotlivě 70, 35 a 12 dolarů. [3]

2.2 SAP Business Objects

SAP Business Objects je reportovací nástroj nabízený ve formě lokálního řešení tzv. on-premise. Jelikož se jedná o nástroj od společnosti SAP, tak nabízí ulehčené propojení s ostatními produkty od této společnosti. Podobně jako Tableau nabízí připojení k různým datovým zdrojům a vytváření grafů a vizuálních prvků. Dále také umožňuje sledovat změny dat v reálném čase a exportovat reporty do formátu pro Microsoft PowerPoint. Obsah se dá sledovat i na mobilních zařízeních. Ceny jsou individuální a odvíjí se podle potřeb firem a zákazníků. [4]

2.3 Microsoft Power BI

Na tento nástroj je zaměřen obsah této práce a je tak více popsán v následujících kapitolách.

Kapitola 3

Řešený softwarový proces

Tato kapitola popisuje problém spjatý se zadáním této diplomové práce. Obsahuje popis problému, jeho možné řešení a poté podrobnější popis vybraného řešení, které je v této práci implementováno.

3.1 Popis problému

Dejme tomu, že máme firmu. Tato firma se zabývá vývojem softwaru a dalšími softwarovými zakázkami. Ve firmě pracují různé týmy vývojářů. Tyto týmy mají vždy nějakého vedoucího, ten na ně dohlíží a přiděluje jim další práci. Týmy pracují na zakázkách a sestavení týmu se určuje pro každou zakázku zvlášť a nezávisle na ostatních zakázkách.

Firma má nějaké nadefinované metriky pro různé zakázky. To může například být počet schválených softwarových požadavků, nebo počet schválených softwarových požadavků ku počtu celkových softwarových požadavků. Jde tedy obecně o nějaké nadefinované metriky. Tyto metriky by chtěla firma sledovat a vyhodnocovat. Metriky sleduje v rámci svého vlastního nadefinovaného firemního procesu. Je tedy otázkou, jak tyto metriky sledovat, vyhodnocovat a jak metriky rozdělit do jednotlivých zakázek, nebo je naopak mezi nimi sdílet.

3.2 Možné řešení

Tento problém se dá bezpochyby řešit mnoha způsoby, proto zde uvedu jen pár vybraných. Prvním naivním řešením by mohlo být sledování metrik manuálně bez pomocných nástrojů. Zaznamenané metriky tisknout a z tabulek se snažit vyhodnotit nějaký závěr. Tento postup by ale asi nedosahoval kýžených výsledků. Další řešení by bylo využít nějaké pomocné nástroje. Může se jednat právě o Power BI. Metriky by se zaznamenávaly a pak nahrávaly do Power BI. V něm by pak uživatelé vytvářeli reporty a z nich analyzovali výsledky práce. Metriky a uživatele by nyní bylo možné rozdělit do skupin a podle libosti jim omezit přístup. Tento postup je o něco sofistikovanější než první zmíněný, avšak stále se dá vylepšit. Rozhraní Power BI nemusí být nutně přístupné všem,

protože většinou obsah stejně nevytvářejí všichni uživatelé. Někteří ho jenom konzumují a analyzují. Power BI je také částečně placený produkt a vyžaduje vlastnictví licence, a to může být při více uživatelích značně nákladné. Navíc používání Power BI vyžaduje znalost jeho rozhraní, což může být pro někoho složité. Pokud by se komunikace s Power BI obalila vlastním systémem, tak by se částečně vyřešil problém se znalostí rozhraní, protože by bylo možné upravit rozhraní uživatelům na míru. Uživatelé, kteří by obsah jenom konzumovali, by nemuseli s rozhraním Power BI vůbec přijít do styku. Zároveň by nebylo potřeba vlastnit více licencí, ale stačila by jenom jedna, přes kterou by komunikoval zmíněný systém. Proces by se také dal částečně automatizovat. Vytváření, sledování a vyhodnocování reportů by se stále dělalo manuálně, ale rutinní úlohy, jako je aktualizace metrik, by mohly být automatizovány. Takový proces by bezpochyby ulehčil práci tomu, kdo by ho využíval a poskytl by mu i více času pro jiné úlohy.

3.3 Vybrané řešení

Vybrán byl poslední zmíněný proces obsahující prvky automatizace. Měla by tedy být použita aplikace obalující komunikaci s Power BI a automatizující některé úlohy na základě konfigurace. Takovou aplikaci by bylo vhodné používat odkudkoli bez nutnosti instalace, měla by podporovat proces na analýzu metrik a mít alespoň nějaké základní zabezpečení na základě rolí ale i na základě přihlašování. Kromě automatické aktualizace metrik by měla poskytovat i další automatické funkce, jako je například zálohování reportů. Následuje popis, jak by takový proces, který aplikaci využívá, mohl vypadat.

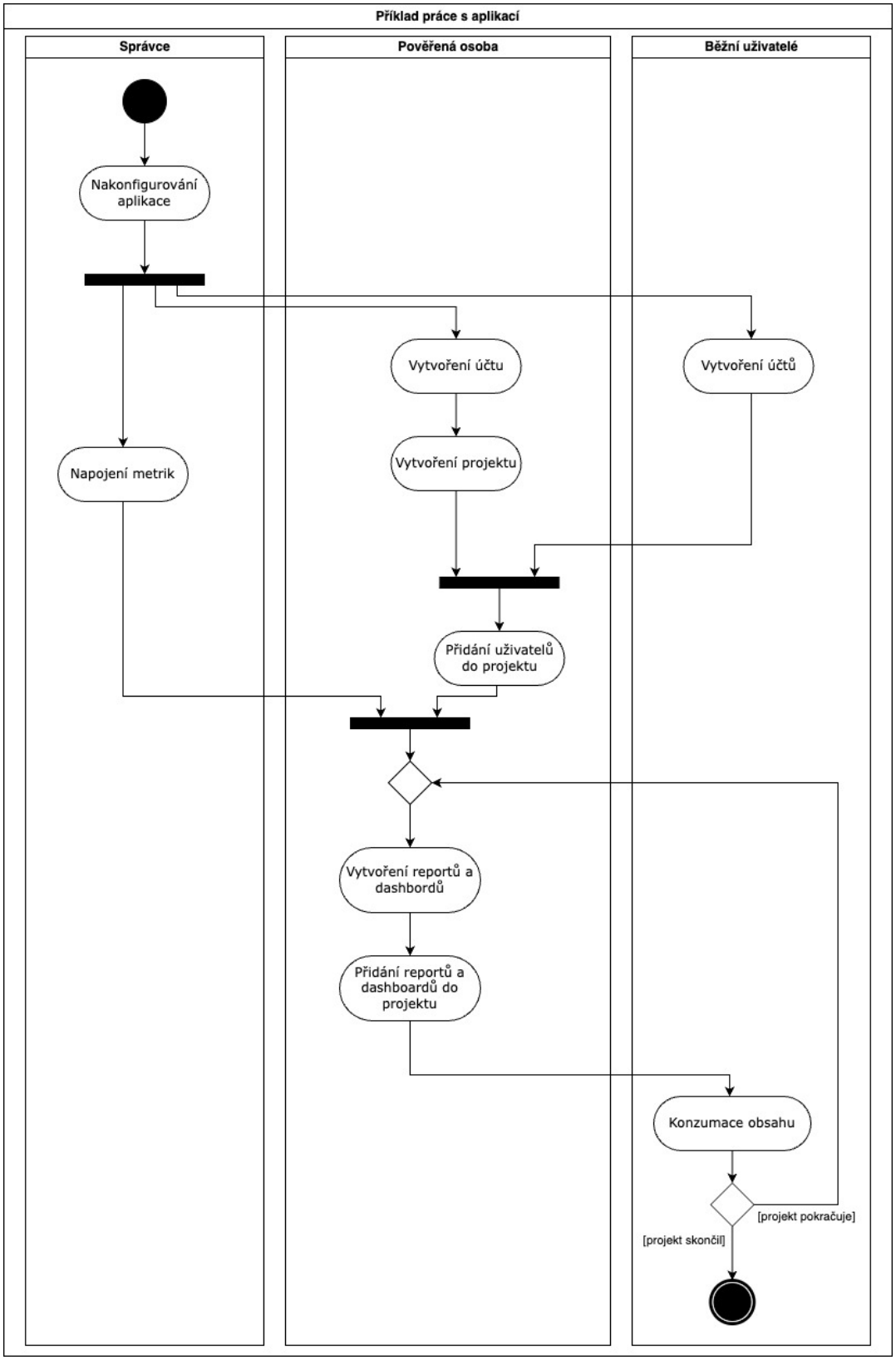
Správce aplikace nastaví komunikaci aplikace s Power BI. Firma si zaznamenává údaje o metrikách sama, nebo s využitím nějakého nástroje. Může jít například o docházkový systém pro sledování počtu odpracovaných hodin, nebo si vedoucí týmu sám zaznamenává data o jednotlivých žádaných metrikách. Tato data pak bude aplikace automaticky nahrávat do Power BI.

Pověřená osoba se zaregistruje v aplikaci a vytvoří v ní tzv. projekt. Tento projekt bude odpovídat projektu z reálného světa. Půjde tedy o nějakou softwarovou zakázku, kterou firma zrovna řeší. Následně si vytvoří uživatelské účty i ostatní na projektu podílející se vývojáři. Tatáž pověřená osoba, v tomto kontextu se jedná o vedoucího vývojového týmu, přidá do nově vytvořeného projektu v aplikaci další uživatele - další členy týmu. Dále může některým určitým členům týmu přiřadit role. Ty jim umožňují provádět další úkony v projektu.

Uživatel v roli správce aplikace napojí aplikaci na nové metriky, pro specifický projekt. Tímto je aplikace schopná metriky aktualizovat a nahrávat je do Power BI. Poté osoba odpovědná za projekt, v našem kontextu jde opět o vedoucího vývojového týmu, může vytvářet reporty. Pro toto má dvě možnosti. Reporty může vytvářet buď v Power BI Desktop, nebo v Power BI Service. Když jsou reporty a dashboardy vytvořeny, tak je uživatel v patřičné roli přidá do projektu. Uživatelé pak mohou obsah zobrazovat, pracovat s ním a vyhodnocovat ho. Když firma přijme další zakázku, tak

opakuje v procesu kroky od vytvoření zaznamenávání metrik po vytváření reportů, dashboardů a konzumování obsahu.

Ukázkový příklad výše popsaného procesu je znázorněn na diagramu níže.



Obrázek 3.1: Aktivitní diagram ukázkového příkladu softwarového procesu

Kapitola 4

Požadavky

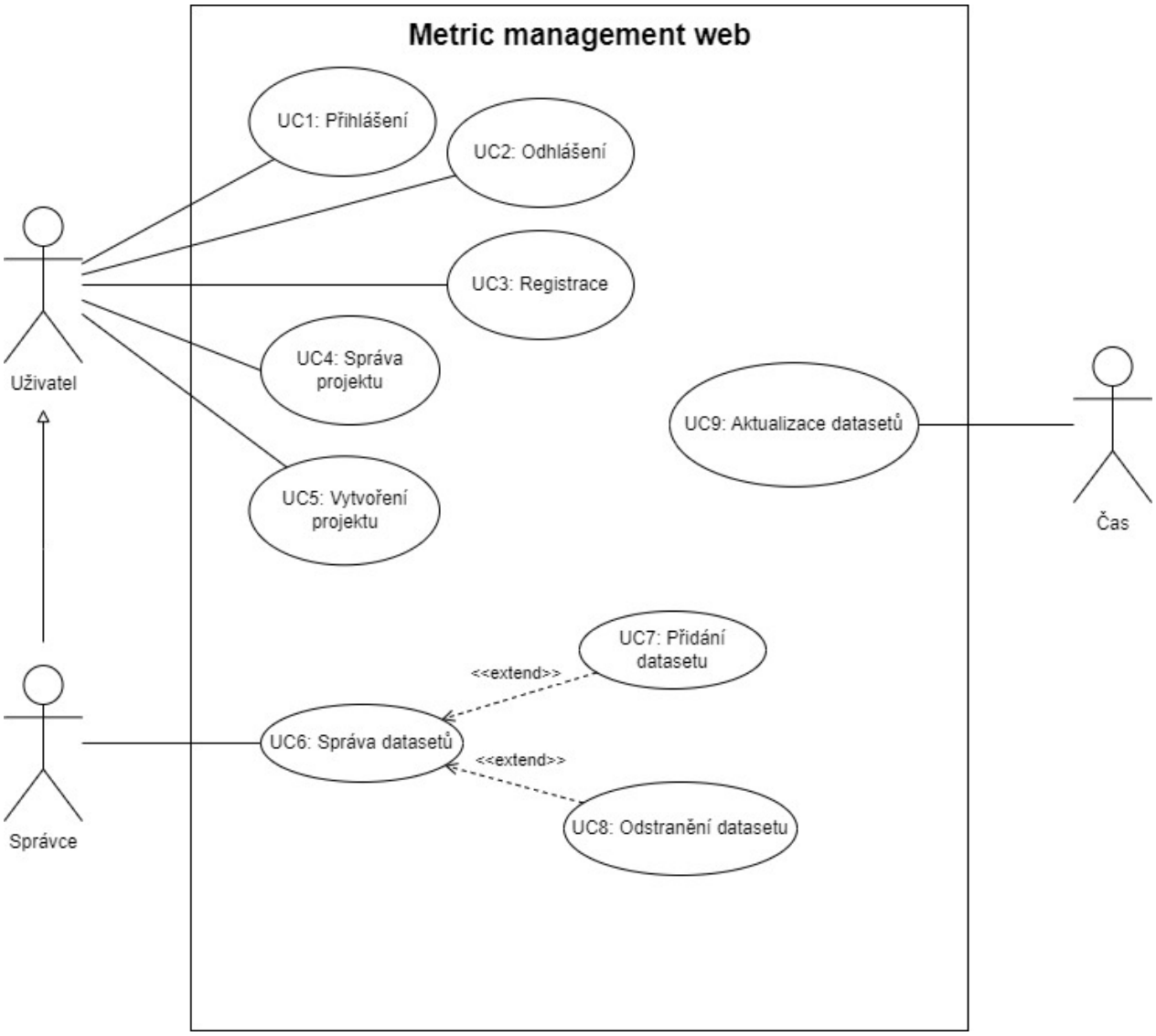
V průběhu vývoje byly posbírány různé požadavky na vznikající systém. Většina z nich vznikla ze samotného zadání práce a počátečních konzultací s vedoucím práce. Další požadavky vznikaly až v průběhu vývoje při konkrétnější představě o tom, jak bude systém vypadat vzhledem k funkčnosti. Tyto požadavky jsou poskládány do tabulky a rozděleny do skupin podle jejich typu. Dále z požadavků a konzultací vznikly případy užití. Ty jsou pak zobrazeny ve dvou diagramech, přičemž každý diagram odráží jiný kontext. Požadavky také vycházejí z výše popsaného procesu.

ID	Popis požadavku	Typ	Typ požadavku	Zdroj požadavku
1	**Specifikace sw. požadavků**	H		
2	**Uživatelské rozhraní**	H		
3	Část pojednává o systémových požadavcích v souvislosti s uživatelským rozhraním	I		
4	Systém bude v levé části obsahovat navigační menu	R	Funkční	CR
5	Systém bude obsahovat v navigačním menu tlačítko pro odhlášení	R	Funkční	CR
6	Systém bude obsahovat v horní části název přihlášeného uživatele	R	Funkční	CR
7	Systém bude na stránce s detailem reportu obsahovat tlačítko pro navigaci zpět	R	Funkční	CR
8	Systém bude na stránce s detailem dashboardu obsahovat tlačítko pro navigaci zpět	R	Funkční	CR
9	**Uživatelé**	H		
10	**Část pojednává o systémových požadavcích v souvislosti se správou a autentizací uživatelů**	I		

11	Systém odkáže nepřihlášeného uživatele na přihlašovací formulář	R	Funkční	CR
12	Systém bude umožňovat nepřihlášenému uživateli se zaregistrovat	R	Funkční	CR
13	Systém bude pro přihlášení požadovat uživatelské jméno a heslo	R	Funkční	CR
14	Systém zpracuje přihlášení uživatele do 10 vteřin	R	Nefunkční	CR
15	Systém zpracuje registraci uživatele do 10 vteřin	R	Nefunkční	CR
16	**Persistence dat**	H		
17	**Část pojednává o systémových požadavcích v souvislosti s persistencí dat metrik a uživatelů**	I		
18	Systém bude pro uložení metrik využívat Power BI	R	Funkční	CR
19	Systém bude poskytovat metriky ke čtení přes HTTP protokol	R	Funkční	CR
20	Systém bude pro uložení uživatelských dat využívat vlastní databázi	R	Funkční	CR
21	**Metriky a projekty**	H		
22	**Část pojednává o systémových požadavcích v souvislosti s metrikami a projekty**	I		
23	Systém bude zobrazovat uživateli pouze projekty, ke kterým má přístup	R	Funkční	CR
24	Systém bude pro zobrazování reportů metrik používat Power BI Embedded	R	Funkční	CR
25	Systém bude pro zobrazování dashboardů metrik používat Power BI Embedded	R	Funkční	CR
26	Systém bude umožňovat uživateli vytvořit projekt pro správu metrik	R	Funkční	CR
27	Systém bude umožňovat přiřadit k projektu další uživatele a přiřadit jim roli	R	Funkční	CR
28	Systém bude uživatelům v projektu poskytovat funkce podle uživatelské role v konkrétním projektu	R	Funkční	CR

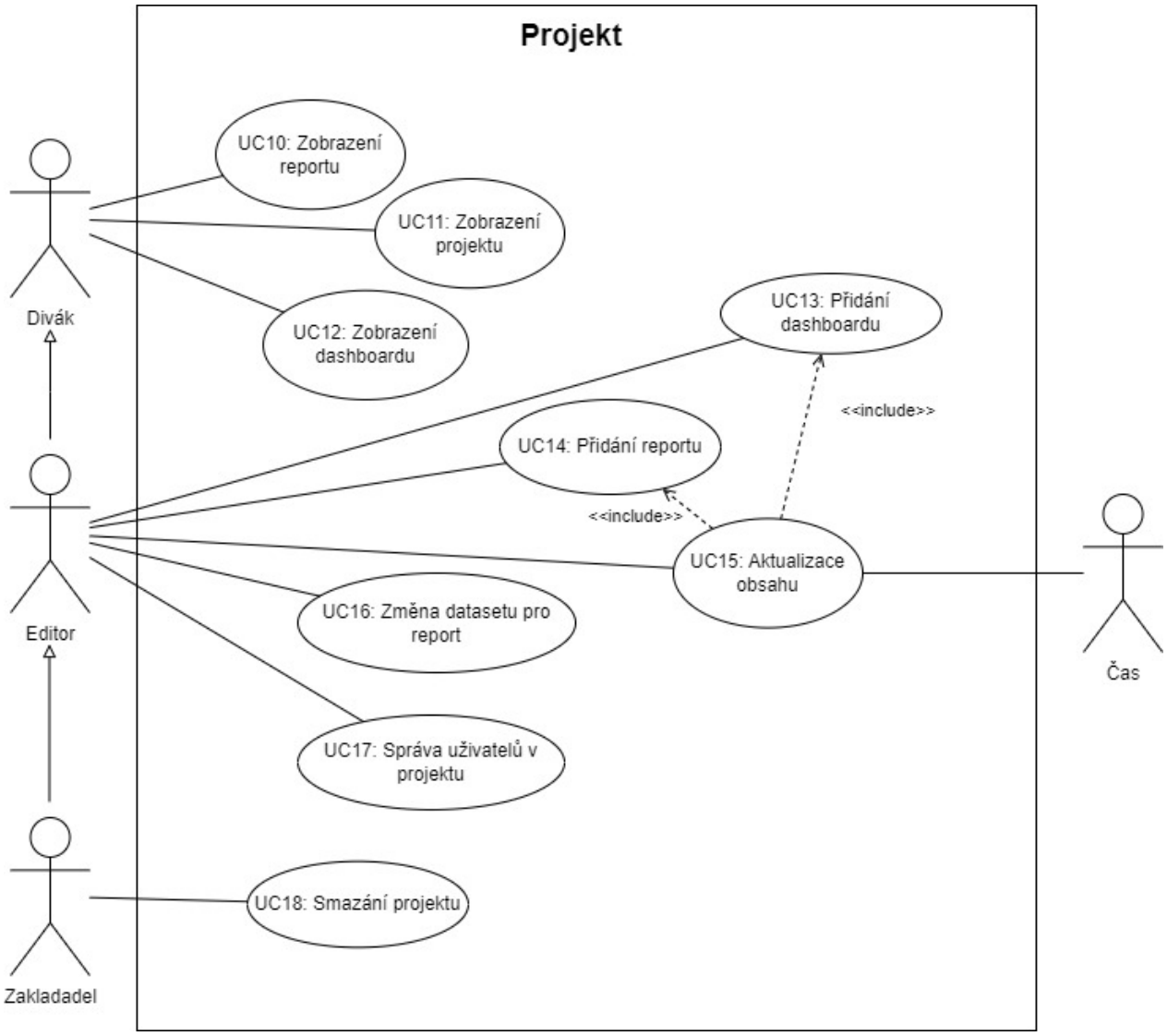
29	Systém bude umožňovat uživatelům v projektu odstranit ostatní uživatele v projektu	R	Funkční	CR
30	**Automatizace**	H		
31	**Část pojednává o systémových požadavcích v souvislosti s automatickými funkcemi**	I		
32	Systém bude aktualizovat metriky v Power BI podle konfigurace	R	Funkční	CR
33	Systém bude stahovat reporty z Power BI podle konfigurace	R	Funkční	CR
34	Systém bude aktualizovat obsah v projektech podle konfigurace	R	Funkční	CR

Tabulka 4.1: Tabulka požadavků a jejich rozdělení do skupin



Obrázek 4.1: Diagram případů užití v kontextu webové aplikace jako celku

Z pohledu aplikace mají uživatelé dvě role - uživatel a správce. Funkce těchto dvou rolí jsou znázorněny v diagramu 4.1. Jelikož se v aplikaci uživatelé dále rozdělují do tzv. projektů, tak v rámci těchto projektů jim je přiřazena role s vlastními funkcemi. Ty jsou pak zobrazeny v diagramu 4.2. Uživatelé musí být pro používání aplikace přihlášení, uživatelům bez uživatelského účtu je pak umožněna registrace.



Obrázek 4.2: Diagram případů užití v kontextu projektu

4.1 Podrobnější popis

Některé případy užití, které jsou pro práci zajímavější, jsou popsány detailněji níže. Jde o slovní popis a v některých případech i o diagramy pro lepší pochopení.

4.1.1 UC7: Přidání datasetu

Popis: Případ užití umožní aktérovi přidat nový dataset

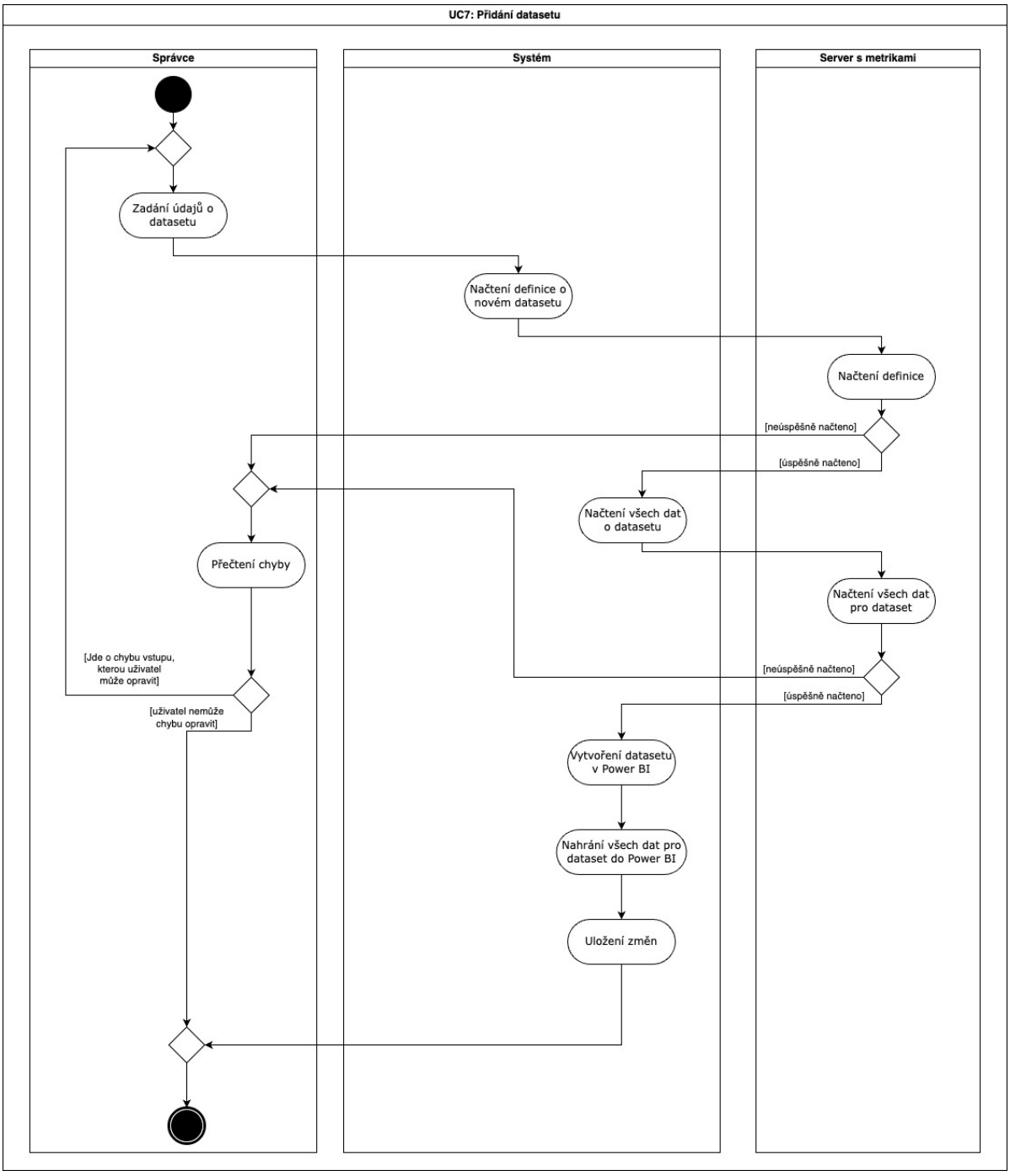
Aktéři: Správce, Systém

Základní tok:

1. Správce zadá údaje o datasetu
2. Systém stáhne definici o novém datasetu
3. Systém stáhne všechny data o datasetu
4. Systém vytvoří dataset v Power BI
5. Systém nahraje všechna data o datasetu do Power BI

Alternativní tok:

2.1 Systém zjistí, že není možné definici stáhnout
2.2 Systém zobrazí chybovou hlášku uživateli
2.3 Správce je schopný opravit chybu na vstupu
2.3.1 Správce opraví chybné údaje o datasetu
2.3.2 Pokračuje se krokem 2 podle hlavního scénáře
2.4 Správce není schopný opravit chybu na vstupu
2.4.1 Případ užití končí
3.1 Systém zjistí, že není možné data stáhnout
3.2 Systém zobrazí chybovou hlášku uživateli
3.3 Správce je schopný opravit chybu na vstupu
3.3.1 Správce opraví chybné údaje o datasetu
3.3.2 Pokračuje se krokem 2 podle hlavního scénáře
3.4 Správce není schopný opravit chybu na vstupu
3.4.1 Případ užití končí



Obrázek 4.3: Aktivitní diagram případu užití 7

Na obrázku 4.3 a 4.4 označuje „Systém“ aplikaci, se kterou pracuje uživatel a „Server s metrikami“ na obrázku 4.3 označuje server, na kterém jsou uloženy metriky pro čtení.

4.1.2 UC9: Aktualizace datasetů

Popis: Případ užití umožní systému automaticky aktualizovat datasety po uplynulém časovém intervalu

Aktéři: Čas, Systém

Základní tok:

1. Čas indikuje systému, že má aktualizovat datasety
2. Systém načte data o datasetech
3. Systém stáhne inkrementy pro každý dataset
4. Systém ověří, že inkrementy jsou pro dataset nové
5. Systém pošle inkrementy do Power BI Service

Alternativní tok:

3.1 Systém zjistí, že inkrement pro dataset není dostupný

3.2 Systém přeskočí aktualizaci pro aktuální dataset

3.3 Pokračuje se podle hlavního scénáře

4.1 Systém zjistí, že dataset je již z tohoto inkrementu aktualizován

4.2 Systém přeskočí aktualizaci pro aktuální dataset

5.1 Systém zjistí, že služba Power BI Service není dostupná pro poslání inkrementu

5.2 Systém přeskočí aktualizaci pro aktuální dataset

4.1.3 UC15: Aktualizace obsahu

Popis: Případ užití umožní uživateli systému aktualizovat obsah pro projekt

Aktéři: Editor, Čas, Systém

Základní tok:

1. Editor započne aktualizaci obsahu pro projekt
2. Systém načte data o projektech
3. Systém stáhne data o reportech z Power BI Service
4. Systém vyfiltruje pro každý projekt reporty na základě shody názvu projektu s prefixem názvu stáhnutého obsahu
5. Systém aktualizuje informace o projektech a jejich reportech
6. Systém stáhne data o dashboardech z Power BI Service
7. Systém vyfiltruje pro každý projekt dashboardy na základě shody názvu projektu s prefixem názvu stáhnutého obsahu
8. Systém aktualizuje informace o projektech a jejich dashboardech

Alternativní tok:

1.1 Aktualizaci započne čas, až uplyne časový interval

1.2 Pokračuje se podle hlavního scénáře

3.1 Systém zjistí, že služba Power BI Service není dostupná pro stáhnutí dashboardů

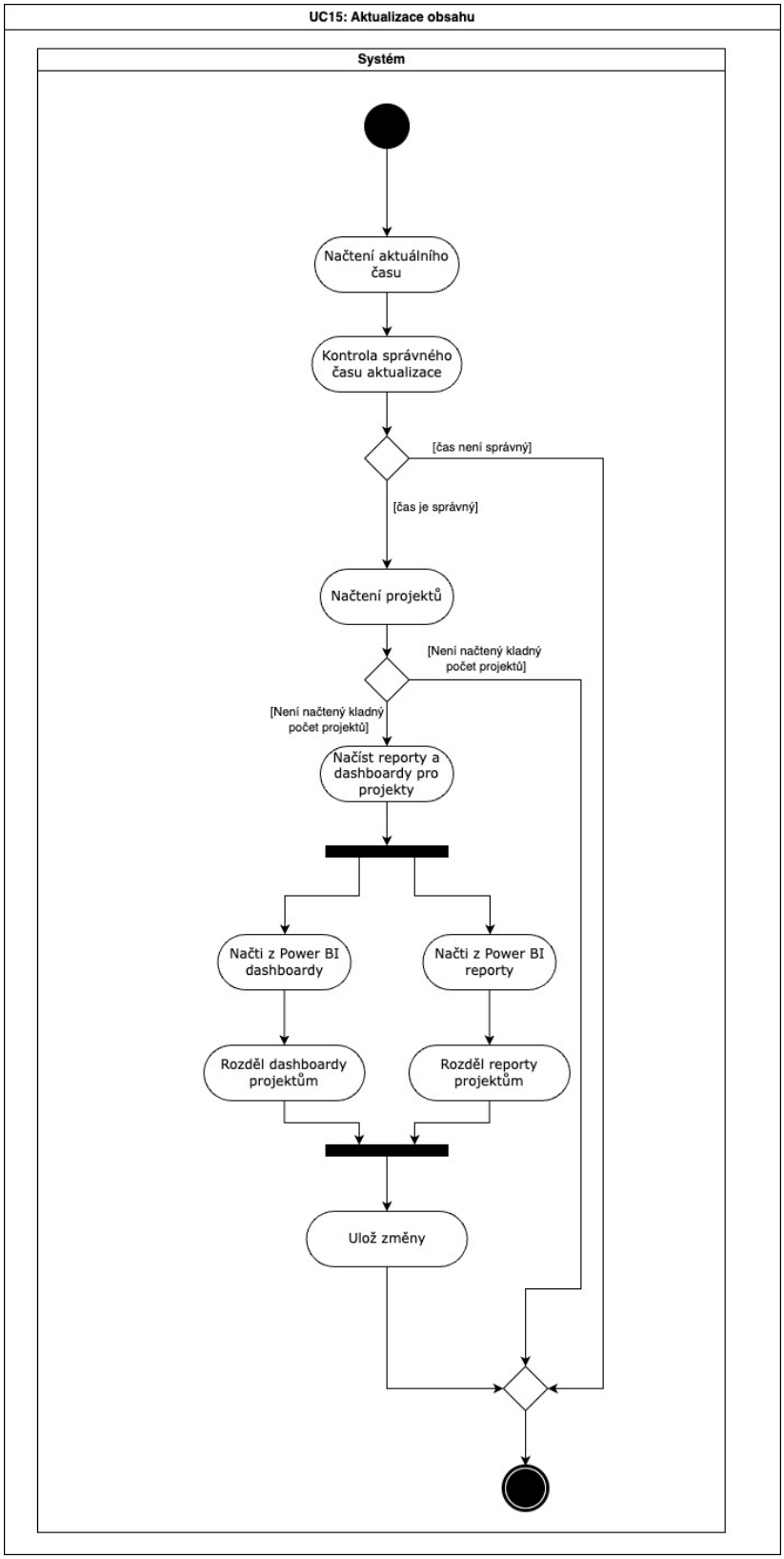
3.2 Systém přeskočí aktualizaci reportů pro aktuální projekt

3.3 Pokračuje se podle hlavního scénáře

6.1 Systém zjistí, že služba Power BI Service není dostupná pro stáhnutí reportů

6.2 Systém přeskočí aktualizaci dashboardů pro aktuální projekt

6.3 Pokračuje se podle hlavního scénáře



Obrázek 4.4: Aktivitní diagram pro případ kdy aktualizaci iniciuje časovač systému

Kapitola 5

Použité technologie

V této kapitole jsou popsány technologie použité pro vývoj aplikace. Vývojář má v dnešní době na výběr ze široké škály technologií, mnohé z nich se však mohou stát v krátkém časovém horizontu technologií zastaralou. Z těchto důvodů byla vynaložena snaha o zvolení technologií, které jsou aktivně vyvíjeny a jsou aktivně stále používány.

5.1 .NET

Jde o pojem zastřešující celou skupinu technologií. Jedná se o open-source vývojářská platforma vyvíjená společností Microsoft a je tedy dostupná pro použití zcela zdarma. Historicky byla technologie rozdělena na .NET Framework a .NET Core. .NET Framework byl použitelný pouze pro operační systém Windows, zatímco .NET Core byl multiplatformní. Později byly však tyto technologie spojeny do jedné multiplatformní technologie nazvané pouze .NET.

Pro aplikaci byl použit programovací jazyk C# a technologie ASP.NET Core. Technologie .NET obsahuje i další programovací jazyky. Konkrétně jsou to F# a Visual Basic. Kromě ASP.NET Core frameworku obsahuje například framework Xamarin sloužící pro vývoj nativních aplikací pro mobilní zařízení nebo .NET MAUI, což je framework pro vývoj nativních desktopových aplikací. [5]

5.1.1 C#

C# je objektově orientovaný jazyk vycházející z jazyků C a C++, ale má také inspiraci v programovacím jazyku Java. Kód napsaný v C# se kompiluje do tzv. CIL kódu. Můžeme si ho představit jako jakýsi mezikód, který se později pomocí Just-In-Time kompilace kompiluje a spouští na virtuálním stroji v prostředí .NET nazývající se CLR. Tento virtuální stroj není nepodobný Java Virtual Machine. Prostředí, ve kterém kód běží, ulehčuje vývojáři práci, protože obsahuje mnoho pomocných prvků. Jde například o správu paměti pomocí Garbage Collectoru a zpracování výjimek.

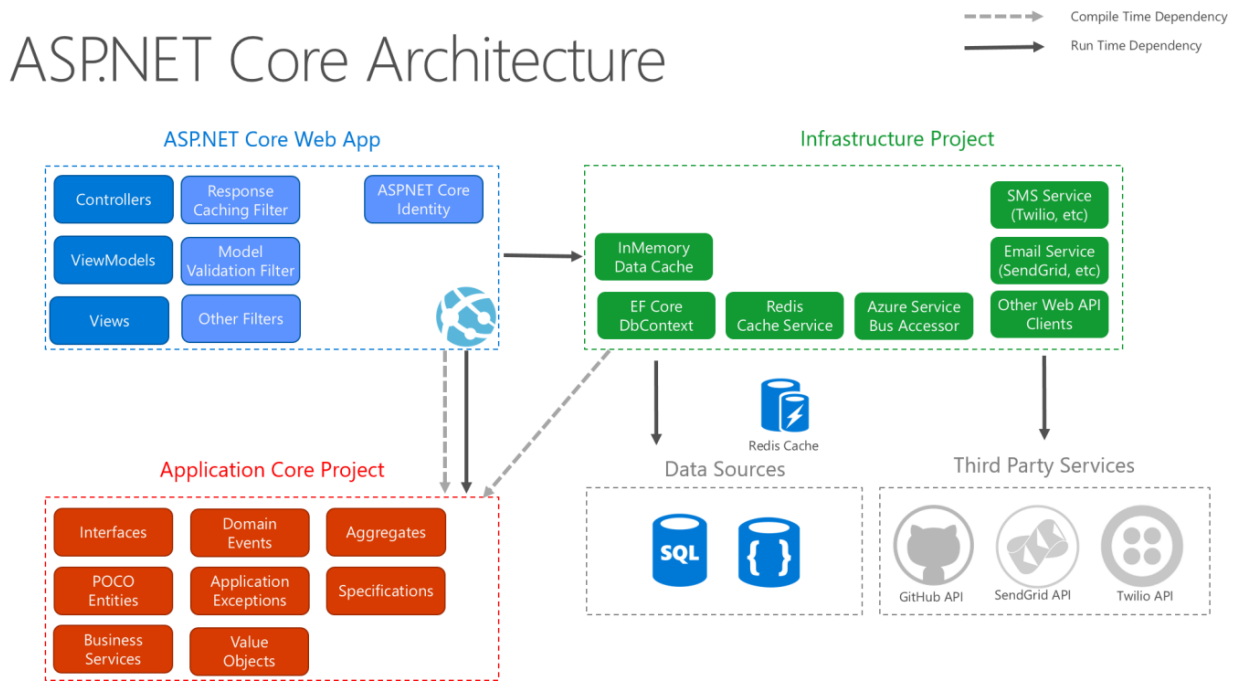
Význam přeložení nejprve do CIL kódu je ale důležitější z pohledu použitelnosti kódu v ekosystému .NET. Všechny programovací jazyky v .NET se překládají nejprve do CIL a až poté jsou

spouštěny. Díky tomuto mezikroku jsou jednotlivé jazyky vzájemně kompatibilní. Není tedy problém napsat část aplikace v C# a jinou část ve Visual Basic. Tímto krokem si také vývojáři z Microsoft ulehčili práci se psaním kompilátorů pro virtuální stroje. Stačí jim kompilátory z jednotlivých jazyků do CIL a ze CIL do virtuálních strojů jednotlivých platforem. Není tedy nutné mít kompilátor pro každou dvojici jazyk-platforma. [6]

5.1.2 ASP.NET

Existují dva druhy tohoto frameworku. Podobně jak to bylo u samotného .NET, kdy existovala současně verze pouze pro windows a multiplatformní verze, tak i u ASP.NET existuje multiplatformní verze ASP.NET Core. Pro aplikace vznikající v rámci této diplomové práce byla použita právě tato multiplatformní verze.

Jedná se o open-source framework pro vývoj webových aplikací. Nabízí několik šablon pro druhy webových aplikací pro ulehčení vytváření nových projektů.



Obrázek 5.1: Architektura ASP.NET [7]

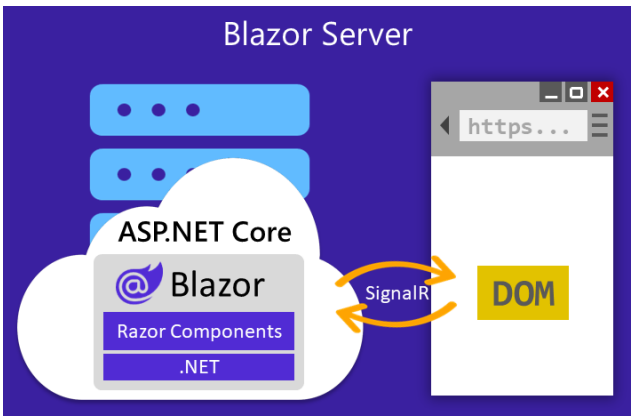
5.1.3 Blazor

Blazor je framework pro vývoj interaktivních webových aplikací aplikací pomocí .NET ekosystému. Tento open-source webový framework je stejně jako ASP.NET Core zcela zdarma. Pro vývoj aplikací používá jazyk C# místo jazyka JavaScript, je však schopný JavaScript využívat a je možné ho například používat pro přístup k HTML prvkům.

Základní stavební kámen Blazor aplikace je komponenta. Jde o nějakou část uživatelského rozhraní, například formulář nebo navigační lištu, zabalenou do samostatného souboru s příponou „.razor“ . Tato samostatná komponenta využívá Razor formát, což je speciální syntaxe, která kombinuje klasické HTML a CSS spolu s kódem psaným v programovacím jazyce C#. Tyto „Razor komponenty“ v sobě obsahují jak vzhled dané komponenty, tak i její logiku. Tato logika je velmi podobná třídě napsané v jazyce C#. [8]

Aplikace vytvořena s Blazor může být spuštěna na serveru, nebo v prohlížeči uživatele. Záleží na tom, o jaký typ Blazor aplikace se jedná, přičemž Blazor nabízí následující typy aplikací:

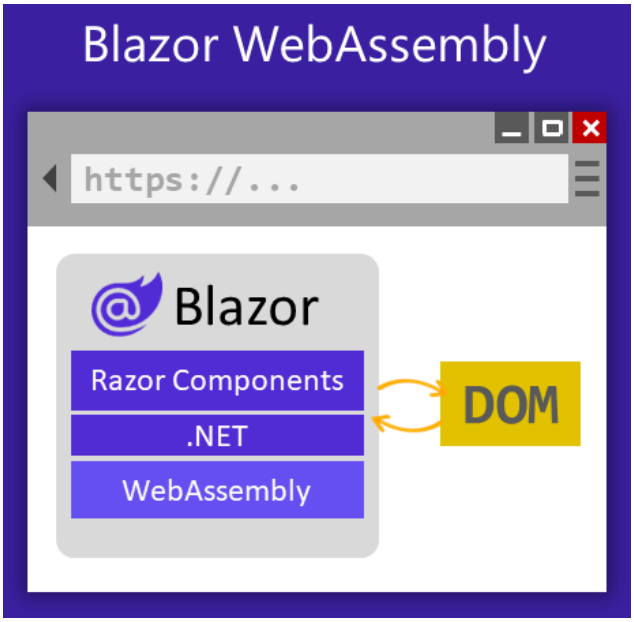
- Blazor Server - aplikace je spuštěna na serveru v rámci ASP.NET Core aplikace. Změny v uživatelském rozhraní jsou prováděny pomocí technologie SignalR. Klient si nestahuje stránku do prohlížeče celou, pouze její malou část a reaguje až na změny přicházející ze serveru. Díky tomu je rychlost načtení stránky vyšší. Kvůli umístění na serveru může být přístupná větší sada knihoven pro vývoj aplikace, která by při umístění u klienta nemusela fungovat tak, jak má. Také umožňuje plné ladění aplikace při vývoji. Ovšem můžou vznikat prodlevy mezi požadavky, jelikož pro každou změnu stránky musí proběhnout spojení se serverem a výměna informací s klientem. Klient také musí být neustále připojený k serveru, jinak aplikace přestane fungovat.



Obrázek 5.2: Blazor Server [7]

- Blazor WebAssembly - aplikace je spuštěna u klienta v prohlížeči. Celá aplikace je stažena do prohlížeče. Jedná se o aplikaci typu SPA. Jsou dva druhy WebAssembly aplikace. První se nazývá „samostatný“ a jde o případ kdy aplikace je samostatná a bez backendu. Druhý je „hostovaný“ a v tomto případě je aplikace připojená s backendem, který ji poskytuje klientovi. V druhém případě je vlastně aplikace rozdělena na klasický frontend a backend, kde frontendem je Blazor aplikace a backendem ASP.NET Core aplikace. Aplikace tedy funguje i v případě, že nemá spojení se serverem, za předpokladu, že s ním nepotřebuje zrovna komunikovat. Dále také je na server kladena menší zátěž, protože aplikace běží přímo na stroji

klienta. Tato výhoda se ale může stát i nevýhodou, pokud nemá klient dostatečně výkonný stroj. Protože je aplikace stažena celá, tak doba načtení může být delší.



Obrázek 5.3: Blazor WebAssembly [7]

- Blazor Hybrid - tento typ slouží k vytváření nativních desktopových a mobilních aplikací. Razor komponenty tedy běží přímo v nativní aplikaci. Díky tomu má aplikace oproti prvním dvěma typům přístup k funkcím zařízení, na kterém je spuštěna. Komponenty se dají znovu použít mezi aplikacemi, ale aplikace musí být pro samostatně udržovány pro různé druhy zařízení.

Pro tuto aplikaci byl zvolen typ Blazor hostované WebAssembly. Následující tabulka obsahuje srovnání zmíněných přístupů. [9]

Funkce	Server	WebAssembly	Hybrid
Plná kompatibilita s .NET API	Ano	Ne	Ano
Přímý přístup ke zdrojům serveru	Ano	Ne	Ne
Rychlé načítací časy a menší zátěž sítě	Ano	Ne	Ne
Kód aplikace je zabezpečený a nepřístupný	Ano	Ne	Ne
Aplikace běží i v offline režimu bez připojení k serveru	Ne	Ano	Ano
Aplikace ve formě statických souborů	Ne	Ano	Ne
Menší zatížení serveru	Ne	Ano	Ano
Přístup k funkcím zařízení	Ne	Ne	Ano
Aplikace je nasazena na webové stránce	Ano	Ano	Ne

Tabulka 5.1: Srovnání typů Blazor aplikací [9]

5.2 Microsoft Power BI

Jedná se o kolekci softwarových služeb a aplikací sloužících k přetvoření datových zdrojů na jejich grafické znázornění a jejich lepší reprezentaci. Jde o technologii od společnosti Microsoft zaměřenou na business intelligence. Umožňuje vytvářet tzv. visual, což jsou grafy nebo ovládací prvky sloužící k ovládání právě zmíněných grafů. Tyto grafy lze kombinovat do reportů a to vše v grafickém rozhraní, tudíž tuto technologii může využít i uživatel bez velkého technického pozadí. Grafy a reporty jsou navíc interaktivní a dokáží se přizpůsobit podle uživatelského nastavení. Power BI si data, která si uživatel přeje zpracovat, uloží do datasetů. Před importem je možné data nějakým způsobem upravit. Z těchto datasetů se později vytváří grafy a reporty. Zdroje dat, ze kterých dokáže Power BI vytvořit datasety, nabízí aplikace hodně. Základem můžou být Excel a CSV soubory, dále ale může jít například o databázi, webovou stránku, XML soubor nebo Microsoft Azure.

Power BI se skládá z několika elementů, které dokáží spolupracovat pro maximální produktivitu:

- Power BI Desktop - je to nativní desktopová aplikace pro operační systém Windows. Je k dispozici zdarma. Aplikace se dá nainstalovat přes Windows Store a její vývojáři se snaží jí měsíčně aktualizovat. Uživatel si také může stáhnout instalační soubor zvlášť přímo z webových stránek firmy Microsoft.

- Power BI Service - jde o webovou aplikaci. Funguje podobně jako aplikace desktopová, ale její funkce jsou oproti druhé zmíněné v některých směrech omezenější a v jiných má zase funkčnost širší. Navíc například nabízí svoje API pro použití v aplikacích třetích stran. Existují tři základní druhy licencí pro použití: zdarma, Power BI Pro a Power BI Premium. Power BI Premium se ještě dále dělí na Premium Per user a Premium Per capacity.

- Power BI mobile apps - aplikace na mobilní zařízení. Slouží hlavně k prohlížení vytvořených reportů a dashboardů.

Power BI aplikace se skládájí z několika základních stavebních kamenů. Jsou to dataset, visual, report a DAX. Power BI Service tyto kameny rozšiřuje o workspace a dashboard. [10]

DAX je zkratka pro Data Analysis Expressions. Jde o jazyk vyvinutý společností Microsoft a používaný právě v Power BI. Používá se k upravování a přidávání nových prvků do datasetu. Tyto prvky jsou počítaná tabulka, počítaný sloupec a míra. Počítaný sloupec je speciální sloupec, který se nevyskytuje v původních datech a obsahuje nějakou pomocnou hodnotu například pro vytvoření grafů. Jaká hodnota se vyskytuje v jednotlivých řádcích sloupce, se definuje ve vzorci tvořeném právě v jazyku DAX. Může jít například o součet dvou hodnot sloupce v aktuálním řádku.

```
NovyPocitanySloupec = Tabulka1[hodnota1] + Tabulka1[hodnota2]
```

Listing 5.1: Ukázka definice počítaného slouce v DAX

Míra je podobná počítanému sloupci, ale obsahuje pár podstatných rozdílů. Zatímco počítaný sloupec počítá hodnotu pro každý řádek sloupce, tak míra obsahuje pouze jednu hodnotu a to pro celý sloupec. Může jít například o průměr hodnot v sloupci. Počítaná tabulka je tabulka vytvořená pomoci DAX vzorce.

V DAX funkci je možné použít kromě aritmetických funkcí i funkce pro filtrování dat. Tímto způsobem se dají některé řádky z datasetu vyfiltrovat pryč a nejsou tak obsaženy ve finální spočtené hodnotě. [11]

Dataset se zakládá na nějakém zdroji dat. Můžeme si ho představit jako nějakou tabulku přímo v Power BI. Tato tabulka se dá dále v aplikaci doplňovat o další prvky zmíněné výše. Power BI Service datasety rozšiřuje ještě o tzv. push datasety, streaming datasety a PubNub streaming dataset. Push dataset je takový dataset, který je vytvořen přes API poskytované Power BI Service. Dále se s ním pracuje stejným způsobem, jako s klasickým datasetem, ale s tím rozdílem, že jeho struktura se po vytvoření již nedá dále upravovat. Nelze tak například přidat míru. Pokud chce uživatel něco takového provést, musí tak učinit při vytváření datasetu. Další druhy se používají pro zobrazení dat v reálném čase v dashboardu a nebudou detailněji popsány, protože se v této práci nevyužívají. Jejich obecné srovnání je potom v tabulce níže. V práci byly používány výhradně push datasety kvůli jejich možnosti trvale ukládat data. [12]

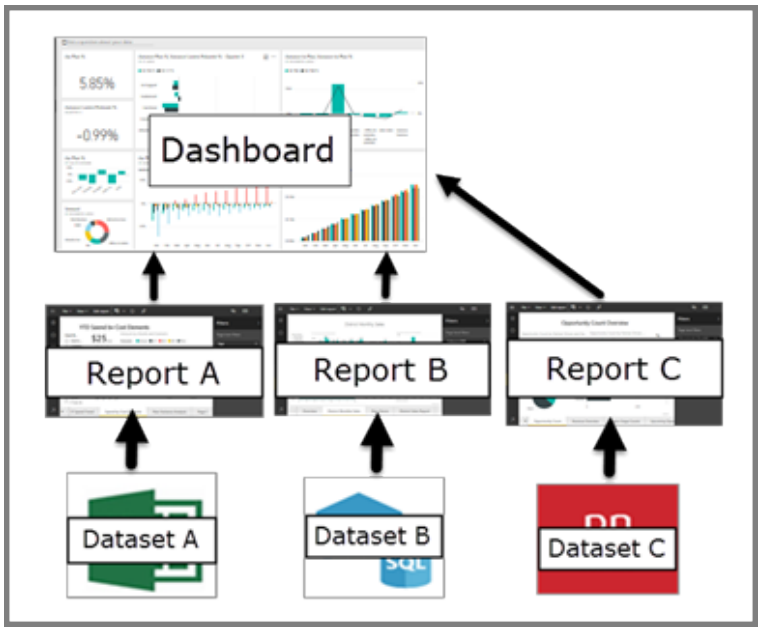
Funkce	Push	Streaming	PubNub
Dashboard se aktualizuje v reálném čase s aktualizací dat	Ano	Ano	Ano
Dashboard se dokáže aktualizovat animacemi	Ne	Ano	Ano
Data jsou trvale ukládány v Power BI	Ano	Ne	Ne
Lze na jejich základě vytvářet reporty	Ano	Ne	Ne

Tabulka 5.2: Srovnání datasetů [12]

Visual je nějaký vizuální prvek reprezentující data z datasetu. Může jít přímo o graf obsahující hodnoty, nebo o ovládací prvek, který slouží k nastavení grafů například k zobrazení omezených hodnot.

Pokud několik vizuálních prvků seskupíme dohromady, vytvoříme report. Tímto způsobem lze reprezentovat data z datasetu do souvislého zobrazení pro lepší pochopení dat. Report může obsahovat i více stránek s vizuálními prvky.

V Power BI Service se vyskytuje tzv. dashboard. Jde o jednu stránku obsahující prvky pro rychlý náhled do dat. Detailnější informace jsou potom obsaženy v jednotlivých reportech, ze kterých je dashboard složen.



Obrázek 5.4: Struktura prvků v Power BI [13]

Všechny zmíněné prvky jsou v Power BI Service uloženy v nějakém workspace. Workspace si lze představit jako jakousi složku nebo kontejner. Uživatel může nastavit i přístup k workspace jiným uživatelům pro společnou kolaboraci na reportech a dashboardech. Každý uživatel má automaticky vytvořený osobní workspace nazvaný MyWorkspace. Do tohoto kontejneru má přístup pouze jeho

majitel. Do ostatních vytvořených kontejnerů může poskytnout přístup i jiným uživatelům. Oprávněným uživatelům lze přiřadit role, určující jejich pravomoci a činnosti, které mohou v konkrétním workspace provádět. Typickým příkladem je pak oprávnění pouze pro čtení obsahu. [14]

5.2.1 Power BI embedded analytics

Jde o technologii, jenž umožňuje zaintegrovat Power BI prvky do vlastní webové aplikace. Výhoda tohoto přístupu je, že uživatelé nemusí pracovat přímo v Power BI Service, ale ve vlastní aplikaci, která může nabízet jiné funkce, hodící se pro konkrétní klientské požadavky. Existují dva druhy, jak Power BI embedded analytics používat: [15]

- Vložit pro vaše zákazníky (embed for your customers) - jde o způsob jak vložit obsah do aplikace tak, aby po uživateli nevyžadoval interaktivní přihlašování. Uživatel tak nemusí mít Power BI licenci nebo účet a pracuje pouze se svojí webovou aplikací. Přihlašovací proces spravuje samotná aplikace, nezávisle na uživateli, který ji používá. Obsah, se kterým uživatel pracuje, tedy nemusí patřit přímo jemu, ale spíše samotné aplikaci řídící autentizaci.
- Vložit pro vaši organizaci (embed for your organization) - dalo by se říci, že jde o druhou stranu mince integrace obsahu. Autentizaci provádí přímo uživatel a musí tak vlastnit účet a licenci pro Power BI. Vytvářený obsah tak může patřit přímo jemu.

Srovnání způsobů integrace obsahu:

Vložit pro vaše zákazníky	Vložit pro vaši organizaci
Data patří aplikaci	Data patří uživateli
Vhodné pro externí uživatele	Vhodné pro interní uživatele
Aplikace autentizuje uživatele vlastní metodou	Uživatele se autentizují proti Azure AD
Aplikace se proti Power Bi autentizuje sama	Proti Power Bi se autentizují uživatelé
Uživatelé nepotřebuji Power BI licenci	Uživatelé potřebuji Power BI licenci
Přistup service principal nebo master user	Proti Power Bi se autentizují uživatelé

Tabulka 5.3: Srovnání přístupů Power BI embedded analytics [15]

Pro tuto práci byl zvolen první přístup, tedy způsob vložení pro zákazníky. Bylo tak zvoleno s ohledem na požadavky a konkrétně na možnost využití aplikace uživateli, kteří nemají přístup do Power BI Service. Aplikace používá pro vlastní autentizaci přístup service principal nebo master user.

S přístupem master user je vytvořen jeden hlavní účet, skrze který se aplikace autentizuje. Tomuto účtu poté patří všechen vytvořený obsah. Účet potřebuje mít Power BI Pro nebo Power BI

Premium licenci. Toto znamená, že na serveru musí být uloženy přihlašovací údaje pro Power BI autentizaci. Z tohoto důvodu je tento přístup potenciálně méně bezpečný než přístup druhý.

Přístup service principal využívá tzv. service principal object. Jde o entitu vlastnící oprávnění pro přístup k nějakým zdrojům, mezi které patří například i Power BI. Uživatelé se potom sami autentizují proti této entitě a získají přístup ke zdrojům této entity. [16]

5.3 Microsoft SQL Server

Je to relačně databázový systém od společnosti Microsoft. Kromě klasického SQL používá také Transact-SQL, což je jazyk, který doplňuje jazyk SQL o další možnosti a funkce. Technologie byla zvolena pro vysokou kompatibilitu s ostatními technologiemi, zejména pak technologií .NET. Microsoft SQL Server nabízí několik edicí. V této práci byla použita základní edice Express, protože nejsou vyžadovány žádné další funkce z rozšířenějších. Na výběr jsou i další edice: Standard, Developer a Enterprise. Edice Express a Developer jsou zdarma, ostatní vyžadují placenou licenci. [17]

Pro vytvoření databáze byl použit Entity Framework Core, což je open-source a multiplatformní technologie pro práci s databází. Umožňuje vyvíjet takovým způsobem, kdy vývojář pouze definuje entity v aplikaci a jejich vzájemné vztahy a framework za něho sám vygeneruje SQL příkazy pro vytvoření databáze a tabulek. Tyto změny uchovává v migračních souborech. Součástí databáze je pak i navíc tabulka, obsahující informace o tom, která migrace je právě aplikována. Vývojář se pak může i vrátit k předešlé verzi databáze. [18]

Z důvodu vývoje na Windows a MacOS byl pro MacOS použit Microsoft SQL Server v Docker kontejneru a na Windows klasická instalace. Docker kontejner byl použit, protože na čipech Apple Silicon není klasická instalace stabilní.

Docker bude popsán spíše stručně, jelikož byl používán okrajově. Jde o open-source platformu, která je využívána pro spouštění aplikací nezávisle na prostředí. Toto je umožněno pomocí tzv. Docker kontejnerů. Jde o uzavřenou a izolovanou komponentu obsahující aplikaci a její závislosti. Kontejnery jsou spouštěny podobně jako virtuální stroje, ale jsou výkonnější, protože jsou odlehčenější než klasické virtuální stroje. [19]

5.4 IDE

Pro vývojáře používající .NET technologie se nabízí primárně Visual Studio od společnosti Microsoft a pro operační systém Windows. Toto IDE je s vývojem v technologiích .NET velmi úzce spojené, jelikož aktualizace pro .NET jsou velmi brzo, ne-li hned promítnuty i do Visual Studia. Dále může být využito například pro vývoj s C++ technologiemi. Visual Studio obsahuje všechny možné funkce potřebné běžným vývojářem. Jako méně náročná alternativa se nabízí Visual Studio Code, což je vlastně pouze textový editor s nějakými funkcemi navíc, jako je třeba integrovaný terminál přímo v programu. Jeho síla ale spočívá v celé plejádě rozšíření, která jsou k dispozici ke stažení. Dá se tak

nakonfigurovat i pro vývoj v jiných technologiích. Je taky méně náročný na systém a na slabších strojích může být preferovanější oproti klasickému Visual Studiu. Toto může být ještě umocněno v případě, že vývojář nevyužívá více funkcí nabízené Visual Studiem. Visual Studio je nabízeno v několika edicích, konkrétně to jsou: Community, Professional a Enterprise. Community edice je k dispozici zdarma, ostatní jsou placené a poskytují navíc dodatečnou funkcionalitu. Pro operační systém Windows a dostatečně silný hardware je Visual Studio téměř jasná volba, jak bylo zmíněno výše. Problém nastává pokud vývoj probíhá na jiném operačním systému. Pro Linux se nabízí Visual Studio Code, je totiž multiplatformní. Problém může nastat v případě MacOS. Microsoft sice nabízí verzi Visual Studia nazvanou Visual Studio for Mac, která je, jak už název napovídá, určena speciálně pro tento operační systém, ale funkce tohoto IDE nejsou tak obsáhlé jako funkce jeho protějšku pro Windows a v mnoha případech jsou znatelně limitovány. Některým uživatelům uvyklým na Windows verzi také nemusí vyhovovat jeho vzhled. Tito uživatelé tak mohou zvolit JetBrains Rider IDE. [20]

JetBrains Rider je vývojové prostředí od společnosti JetBrains. Je multiplatformní. Na rozdíl od Visual Studia nenabízí vývojářům základní neplacenou edici. Uživatel jej může vyzkoušet na měsíc zdarma. Studenti mohou využít neplacenou studentskou verzi. Rider nabízí standardní funkce Visual Studia a i mnohé navíc. Za zmínku stojí ReSharper, což je placené rozšíření do Visual Studia od společnosti JetBrains, které ulehčuje vývojáři práci rozšířenou analýzou kódu a dalšími užitečnými funkcemi jako je například rozšířené generování kódu. Toto rozšíření je v Rideru automaticky zdarma obsaženo. Další výhodou je větší výkon a téměř totožný vzhled programu na všech operačních systémech. Uživatel tak může pracovat stejně efektivně při jeho změně. Nevýhodou může být, kromě již zmíněné absence neplacené edice, absence možnosti vývoje v některých technologiích nedostupných mimo Windows platformu. [21]

Jelikož pro práci byl využíván Windows a MacOS, tak z výše uvedených důvodů bylo pro Windows použito IDE Visual Studio a pro MacOS JetBrains Rider.

5.5 Verzovací systém

Pro vývoj byl použit Git, což je open-source nástroj pro verzování. Standardně se ovládá přes terminál, ale pro ulehčení práce jde vybírat z několika programů nabízejících pro používání Gitu grafické rozhraní. Jako tento program byl zvolen GitKraken. Jedná se o placený program, ale studenti mohou využít jeho neplacenou studentskou verzi. GitKraken nabízí grafické znázornění větví a historie vývoje a oproti jeho neplaceným alternativám, má rozšířenou funkcionalitu. Integrovaný terminál obsahuje i příkazy navíc, které nejsou obsaženy ve standardní verzi Gitu. [22]

Jako cloudové úložiště byl použit server GitHub. Neposkytuje sice takové funkce jako jeho konkurence GitLab, ale jako základní úložiště naprosto stačí. GitHub a GitKraken byly využity pro přenos změn i kvůli využití více zařízení pro vývoj.

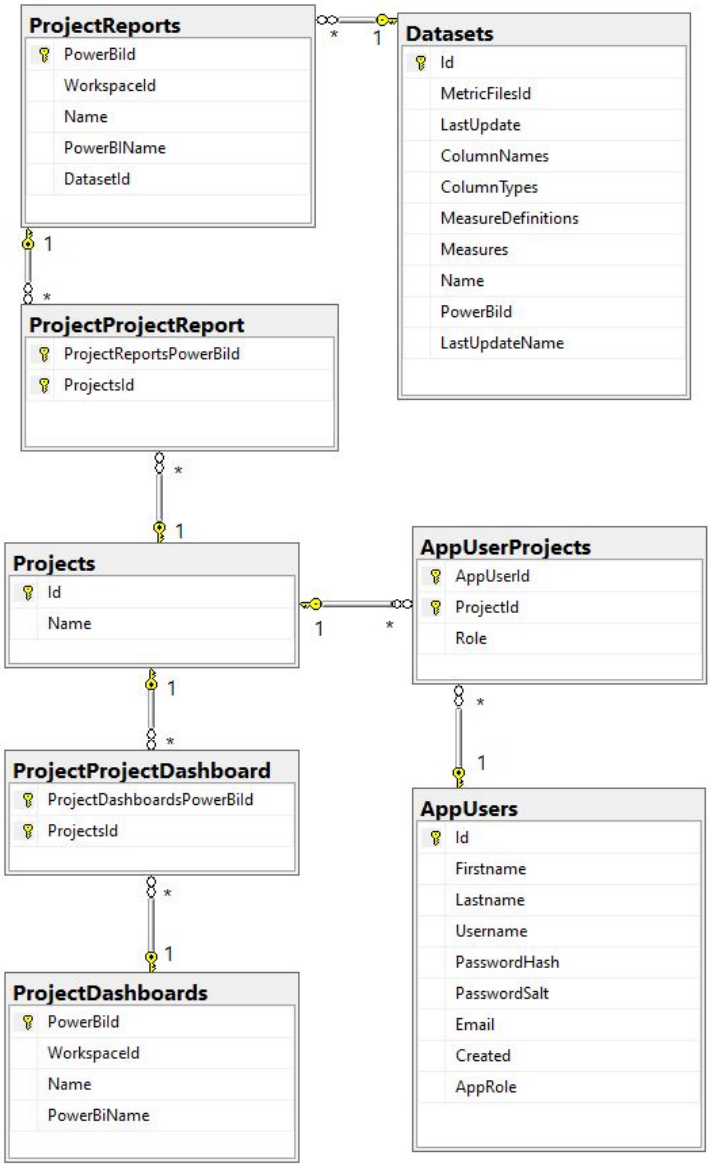
Kapitola 6

Architektura a návrh

Při návrhu architektury aplikace je třeba brát ohled na požadavky a koncept pro systém. Koncept pro tuto práci je aplikace, která ulehčuje uživatelům správu metrik automatizací. Dále je žádoucí, aby byla aplikace přístupná i uživatelům, kteří nemají přístup do Power BI Service a aby bylo možné tuto aplikaci využívat online. Z důvodů níže popsaných vzniknou dvě aplikace, jenž budou realizovány ve formě webové aplikace.

6.1 Webová aplikace

Webová aplikace bude sloužit uživatelům pro správu metrik a kolaboraci s dalšími uživateli. Aplikace je rozdělená na dvě části. Pro první část je použita technologie ASP.NET Core a jedná se o backend aplikace. Druhá část aplikace je frontend a je pro ní použita technologie Blazor WebAssembly. Pro přístup k Power BI obsahu využívá službu Power BI Embedded. Dále metriky musí být někde k dispozici pro systém k získání. S metrikami pracuje hlavně systém a ne uživatel, jelikož nám jde o to, aby byl systém v tomto ohledu automatický. Metriky tedy nebude uživatel nahrávat sám manuálně ve formě nějakých souborů ve formátu csv, nýbrž pouze řekne systému, jaké metriky má spravovat. To systém bude sám metriky aktualizovat a získávat nejnovější data. Informace jak často toto bude systém dělat, budou konfigurovatelné správcem aplikace. Uživatel tedy pouze použije aplikaci pro napojení metrik do systému a systém už potom automaticky aktualizuje data. Tato aktualizace se promítne i do Power BI, jelikož metriky jsou reprezentovány nějakým Power BI datasetem. Data se tedy přidají i do něj. Webová aplikace dále pracuje s SQL databází. Níže je E-R model, který popisuje strukturu databáze, entity které jsou v ní uloženy a jejich vlastnosti, které je třeba uchovat.



Obrázek 6.1: E-R model databáze

Některé entity mají prefix „Project“ z důvodu kolize. Při využívání ASP.NET Core frameworku se importují i třídy a objekty v něm obsažené, které programátor nemusí využívat, ale můžou se jmenovat stejně jako programátorem vlastní definované. Jde o prostou shodu názvů. Programátor potom musí explicitně označovat o jaký objekt se jedná a zdrojový kód může tak být zbytečně dlouhý a nepřehledný. Pokud se ale zabrání kolizi pojmenování, tak tento problém odpadá. Z tohoto stejného důvodu je u jiných entit prefix „App“. V databázi jsou uloženy informace o uživatelských účtech, aby bylo uživateli umožněno se přihlašovat. Aplikace totiž nepoužívá přihlašovací metody třetích stran, ale uživatelé si v ní vytváří vlastní uživatelské účty. Přes tyto účty pak s aplikací pracují. Bylo tak rozhodnuto i z důvodu, že aplikace by měla umožňovat práci s metrikami a

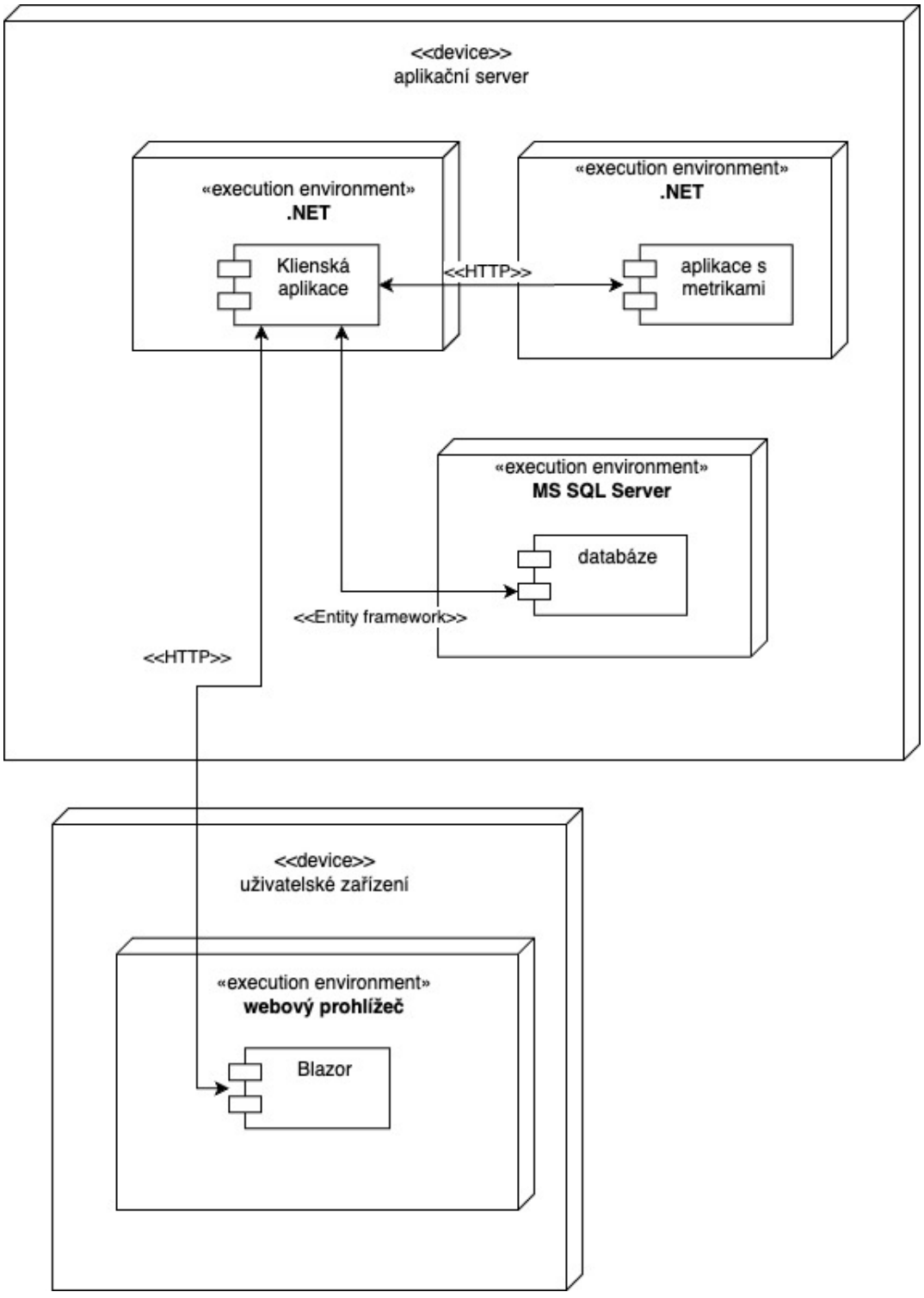
přístup i uživatelům, kteří nedisponují Power BI licencí, bez které nejde používat Power BI Service. Uživatelé tedy pracují s aplikací pod vlastním aplikačním účtem a aplikace komunikuje s Power BI Service přes jeden společný účet s jednou licencí.

Jak již bylo zmíněno, tak data metrik jsou uložena v Power BI. Aplikace tedy ukládá v databázi identifikátory od datasetů v Power BI, které reprezentují jednotlivé metriky a jejich data. Databáze obsahuje kromě identifikátorů i další vlastnosti metrik, jako je například jméno, ale hlavně definici datasetu uloženou na serveru s metrikami. Jedná se o informaci o počtu sloupců a jejich datovém typu a dále o počtu a definici měr, ze kterých se finální dataset skládá. Definice měr je psaná v jazyce DAX, jelikož míry se vyskytují pouze v konečném datasetu a skládají se z dat ve sloupcích, nebo z jiných měr. Prakticky jde tedy o nějakou dopočtenou hodnotu a nevyskytuje se explicitně v původních datech. Tato definice je potřeba pro správnou interpretaci dat a správné nahrání do Power BI. Identifikátory jsou uloženy i v případě dashboardu a reportu. Datasety obsahují navíc informaci o tom, z jakého souboru byly naposledy aktualizovány, aby nedošlo omylem k nahrání stejných dat vícekrát.

Uživatelé se rozdělují do projektů. Uživatel může patřit i do více projektů, nebo naopak do žádného. Proto se mezi tabulkami AppUsers a Projects vyskytuje vazba M:N. Dále je v průnikové tabulce ještě informace o roli uživatele. Jde o role znázorněné na obrázku 4.2 a reprezentovány jsou celočíselnou hodnotou. Nejsou rozděleny do zvláštní tabulky, jelikož aplikace nepotřebuje uchovávat o rolích žádné další informace a oddělená tabulka by byla nadbytečná. K projektům jsou dále přiřazeny dashboardy. Dashboardy mohou být přiřazeny k libovolnému počtu projektů. Z tohoto důvodu je mezi tabulkami Projects a ProjectDashboards vazba M:N. Toto platí i v případe reportů. Datasety jsou v datovém modelu navázány na reporty. Zde je vazba 1:N, jelikož report může využívat v jeden čas pouze jeden dataset. Dataset ale může poskytovat svá data více reportům současně. V databázi je uložena informace o vztahu reportů s datasety pro možnost přepnutí reportu na jiný dataset. Tato funkce umožňuje z jednoho reportu vytvořit pomyslnou šablonu a tu používat pro více datasetů stejného typu.

6.2 Server s metrikami

Tento server obsahuje zdrojová data o metrikách, které si webová aplikace získává a aktualizuje. Jedná se o samostatný server, jelikož je potom jednodušší webovou aplikaci napojit na jiný datový zdroj, což je umožněno díky tomu, že tento server není přímou rigidní součástí webové aplikace a komunikují spolu přes HTTP. Metriky na serveru jsou rozděleny do vlastních složek a jejich data jsou aktualizována podle konfigurace správcem aplikace.



Obrázek 6.2: Diagram nasazení aplikací

Kapitola 7

Implementace

Pro využití Power BI Service API je třeba nejdříve zaregistrovat aplikaci v portálu Azure. Aplikace byla vytvořena pomocí školního účtu a v rámci Azure VŠB skupiny. Microsoft má pro tento proces pomocnou stránku, která aplikaci sama vytvoří a nastaví ji potřebná oprávnění. Proces je detailně popsaný v jejich dokumentaci a při využití pomocné stránky je téměř automatický. Při vytváření aplikace je nutné se předem rozhodnout, jaký typ autentizace chceme použít. Jednotlivé kroky při vytváření aplikace se totiž liší. Po vytvoření aplikace je nutné povolit Azure aplikaci přístup k Power BI účtu uživatele, jehož údaje budou použity pro Master User přihlášení. [23]

V následujících kapitolách je popsána implementační část této diplomové práce. Popis je rozdělen na dvě části. První je popis implementace webové aplikace se kterou pracují uživatelé a druhý je popis implementace serveru s metrikami.

7.1 Webová aplikace

Webová aplikace je implementována s pomocí technologie ASP.NET Core. Implementace vychází ze šablony Blazor Hosted WebAssembly. Aplikace je vnitřně rozdělena na klienta a server. Zdrojové kódy jsou rozděleny do projektů a ty jsou součástí jednoho tzv. *solution*. *Solution* je označení pro soubor jednoho nebo více projektů sdružených spolu. Toto ulehčuje vývojáři práci při vývoji, protože Visual Studio otevře přes jedno *solution* všechny přidružené projekty najednou v jednom okně a programátor tedy může pracovat efektivněji mezi projekty. V této práci je *solution* první aplikace složena ze tří projektů a sice Shared, Client a Server. Popis jednotlivých projektů v *solution* pro webovou aplikaci je níže.

- Shared - jak název napovídá, jde o projekt sdílený jak projektem klienta, tak projektem serveru. Obsahuje primárně entity pro přenos dat tzv. DTO. Není ale omezen pouze na to. Může obsahovat různé pomocné třídy s jinými užitečnými funkcemi. Tato práce je ale nevyužívá a pracuje pouze s DTO;

- Client - jde o frontend aplikace. Právě zde je použita technologie Blazor WebAssembly. Klient komunikuje v backendem a získává z něho data;
- Server - jde o backend aplikace vytvořený na technologii ASP.NET Core Web API s kontroléry;

Následující sekce podrobněji popisují jednotlivé projekty, strukturu a funkcionalitu. V sekci 7.1.2 je i základně popsána funkcionalita, kterou aplikace nabízí uživateli a tato funkcionalita je poté podrobněji implementačně popsána v sekci 7.1.3

7.1.1 Sdílená část

Projekt obsahuje tři složky a to Datasets, Projects a Users. V každé složce jsou obsaženy DTO související s názvem složky. DTO jsou využívané pro přenos dat mezi klientem a serverem. Server má potom vlastní entity, se kterými pracuje. Do těchto entit si také převede data z DTO. Klient na druhou stranu s vlastními entitami nepracuje. Je tak učiněno, protože klient nevyžaduje nějakou speciální funkcionalitu, ale potřebuje pouze zobrazovat data z těchto DTO.

7.1.2 Klientská část

Projekt má strukturu popsanou níže.

- Properties - Adresář obsahující konfigurační soubor *launchSettings*. Tento soubor slouží ke konfiguraci chování při spouštění aplikace.
- wwwroot - Tento adresář obsahuje soubory, které jsou kopírovány přímo do kořenového adresáře na webovém serveru, kde je aplikace spuštěna. Jedná se hlavně o JavaScript soubory a CSS soubory.
- Pages - Adresář obsahující komponenty aplikace.
- Shared - Zde jsou umístěny komponenty, používané napříč celou aplikací. Například jde o hlavní navigační menu.
- Utilities - Adresář obsahující pomocné třídy, funkce a služby. Převážně jde o funkce pro komunikaci se serverem a pro získání dat ze serveru.

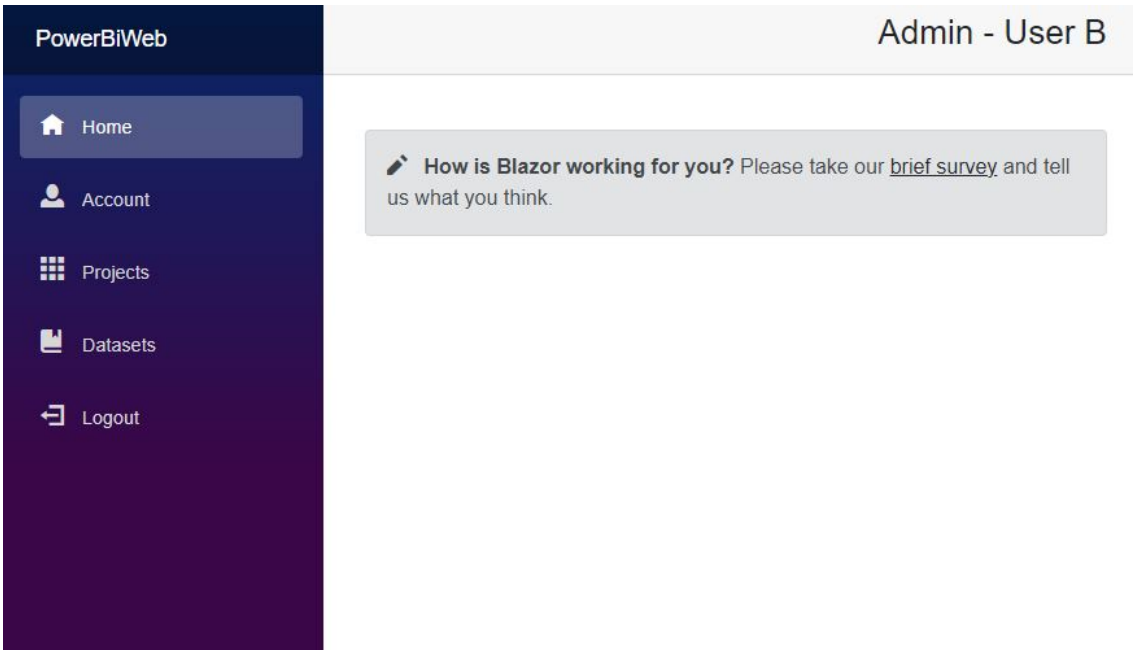
Soubor *program.cs* je základní složka klientské části aplikace. Registrují se v ní základní komponenty Blazor aplikace. Dále je nakonfigurován HttpClient. Jde o třídu provádějící požadavky přes HTTP. To, že ho nakonfigurujeme již při startu aplikace, ulehčuje případné modifikace. Při používání této komponenty pro komunikaci stačí pouze doplnit konec cesty, na kterou má HTTP dotaz směřovat. Pokud by v budoucnu nastala změna serveru, se kterým klient komunikuje, tak stačí zde při konfiguraci změnit základní adresu. Za předpokladu, že nový server bude mít rozhraní totožné se serverem starým, nebude aplikace vyžadovat rozsáhlé změny. V souboru jsou registrovány služby

pro komunikaci se serverem a jejich registrace zde je umožňuje využívat v rámci vkládání závislostí. Funkce služeb jsou definované rozhraním. Toho rozhraní je použito při registraci a je možné tedy vyměnit konkrétní implementaci nějaké služby za jinou, a to bez aplikování velkých změn v aplikaci. Třídy, které služby využívají, totiž pracují s jejím rozhraním a ne s její konkretní implementací. Takto je využit návrhový vzor vkládání závislostí a princip obrácení závislosti. Kromě těchto služeb jsou zaregistrovány i služby balíčků třetích stran a některé funkce samotného frameworku ASP.NET Core. Jedná se například o funkce autorizační a nebo služba pro obsluhu lokálního úložiště v prohlížeči uživatele.

Pro používání aplikace musí být uživatel přihlášen. Toto obstarává komponenta Login. Jde o přihlašovací a registrační formulář. Při přihlášení je odeslán HTTP požadavek na server a v případě úspěšného přihlášení je vrácen JWT. Aplikace využívá právě JWT pro autentizaci a autorizaci uživatele. Token v sobě nese informaci o identitě uživatele a jeho roli v rámci celé aplikace, tedy jestli se jedná o běžného uživatele, nebo je to uživatel s většími právy, tedy se jedná o správce. Na základě tohoto rozdělení nejsou některé funkce uživateli se základními právy zpřístupněny. Toto rozdělení přístupu podle práv je aplikováno i na serveru. Při registraci je uživatel rovnou i přihlášen pod nově vytvořeným účtem.

Za správu tokenu je odpovědná třída *JwtAuthStateProvider*. Implementuje třídu *AuthenticationStateProvider*, což je třída z frameworku ASP.NET Core a slouží právě k autorizaci uživatele a k získání informací o něm v dalších komponentách. Při správné implementaci je pak možné využívat i jiných komponent obsažených ve frameworku. Je to například *AuthorizeView*, který skryje obsah stránky podle role. *JwtAuthStateProvider* ukládá token do lokálního úložiště pro pozdější čtení a hlavně ho přečte a získá z něho relevantní data. Jedná se právě o dříve zmíněnou roli a dále přezdívku uživatele. Jestliže je token neplatný, nebo token neexistuje, tak vrací komponentám informaci, že se jedná o anonymního uživatele. V takovém případě komponenty přesměrují uživatele na přihlašovací stránku. V případě kdy se uživatel odhlásí, se smaže z lokálního úložiště token a uživatel je poté opět přesměrován na přihlašovací stránku.

Základní navigaci obsluhuje komponenta *NavMenu* obsahující navigační menu, které umožňuje uživateli dostat se k hlavním stránkám aplikace. Jedna záložka je přístupná pouze uživatelům s většími právy. Jde o záložku Datasets. Ta slouží, jak název napovídá, ke správě datasetů. Jelikož jde o důležitou funkcionalitu, na které může být závislých mnoho uživatelských projektů a reportů v rámci aplikace, a proto je zpřístupněna pouze pracovníkům aplikace s větším oprávněním.



Obrázek 7.1: Hlavní navigační menu - vlevo

Při načítání záložky s datasety se pomocí služby načtou data ze serveru. Tyto data reprezentují všechny datasety aplikace. Ty zobrazuje do tabulky.

```
private async Task LoadData()
{
    HttpResponse<List<DatasetDTO>> response = await _datasetService.GetAllAsync();
    if (response.IsSuccess)
        _datasets = response.Value;
    else
        toastService.ShowError(response.ErrorMessage);
}
```

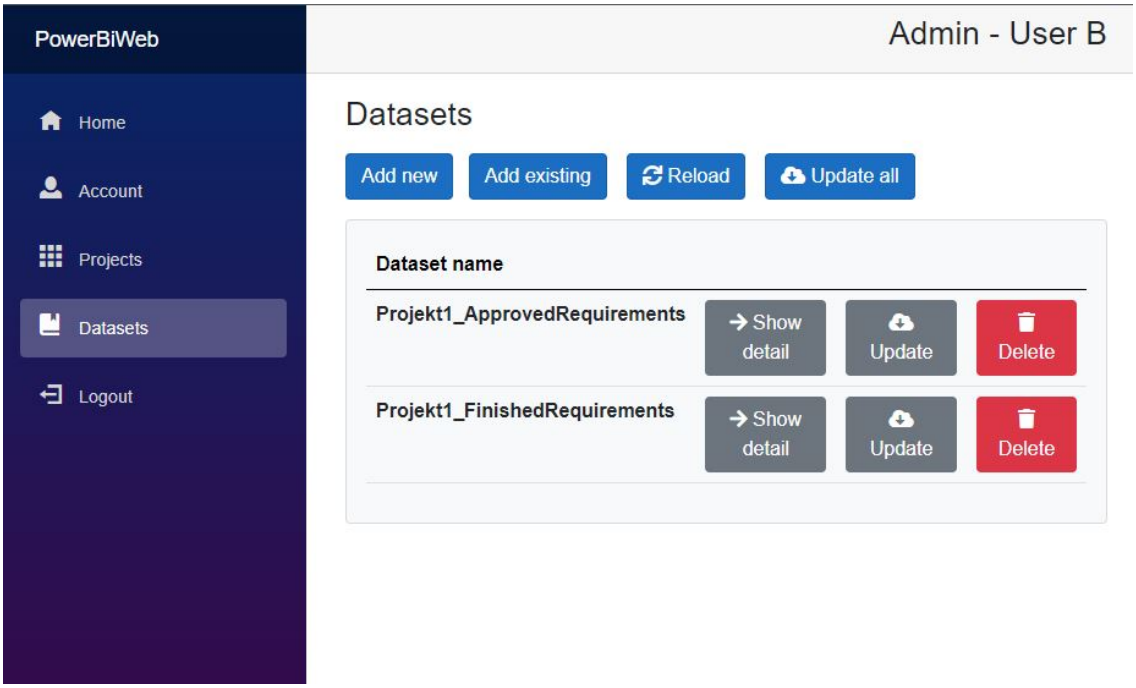
Listing 7.1: Zjednodušený zdrojový kód načtení datasetů

Služby komunikující se serverem obalují výsledek HTTP požadavku do vlastní třídy *HttpResponse*. Jde o třídu obsahující navíc informaci o tom, jestli byl výsledek požadavku úspěšný. Server také může chtít vrátit klientovi nějakou chybovou hlášku, na kterou může uživatel adekvátně reagovat a případně osobně napravit chyby. Proto je ve třídě ještě informace o chybové hlášce. V případě, kdy HTTP požadavek nedopadl úspěšně, tak komponenty zobrazí chybovou hlášku uživateli. V opačném případě korektně zpracuje výsledek požadavku.

```
public class HttpResponse<T>
{
    public bool IsSuccess { get; set; }
    public T? Value { get; set; } = default;
    public string ErrorMessage { get; set; } = string.Empty;
}
```

Listing 7.2: První varianta HttpResponse

Tato třída je implementována dvěma způsoby. První varianta je určena pro požadavky, které očekávají nějaká data ze serveru a je zobrazena na obrázku 7.2. Využívá generického parametru T. Díky tomu je možné využívat tuto třídu pro různé datové typy bez nutnosti ji předem definovat. Druhá varianta je pro požadavky, pro které se neočekává žádná hodnota k vrácení. Implementačně je totožná s variantou první, ale neobsahuje generický T parametr.



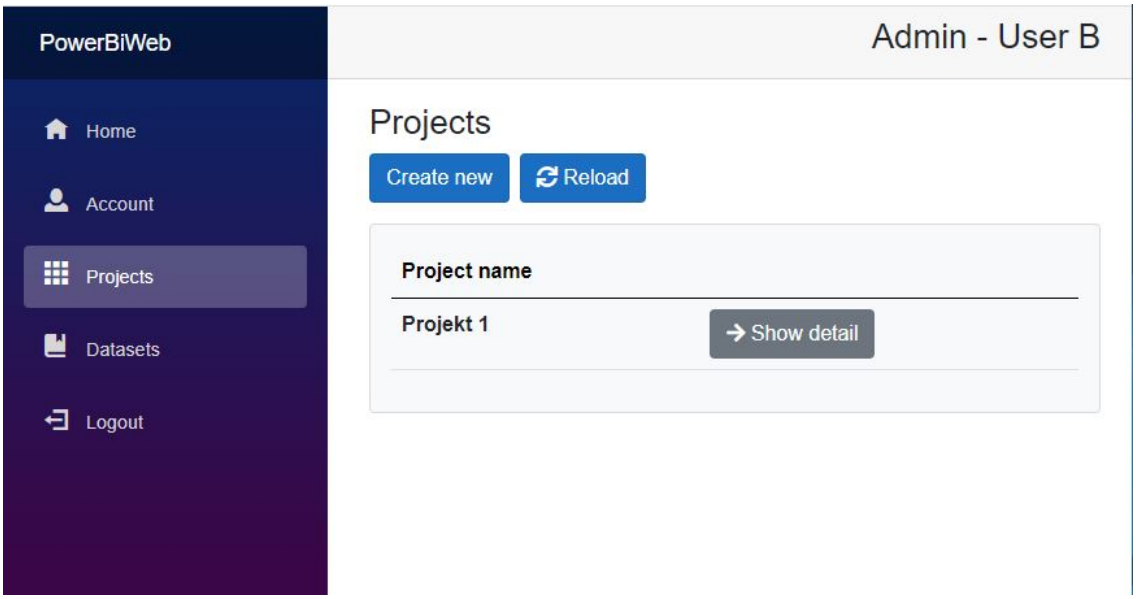
Obrázek 7.2: Záložka s datasety

V záložce s datasety může uživatel přidat nové datasety. Datasety se přidávají podle jejich identifikátorů. V případě, že uživatel chce přidat dataset úplně nový, tak stačí jeho identifikátor, pod kterým je uložen na serveru s metrikami. Při přidání nového datasetu se automaticky stáhne i inkrement. Může ale nastat i situace, kdy bude chtít uživatel přidat v Power BI již existující dataset, ale informace o něm není ještě uložena v aplikaci a ta s ním tedy nemůže pracovat. Toto se může stát v případě, kdy byl například dřívěji vytvořený dataset smazán v aplikaci, ale v Power BI smazán nebyl. Pokud taková situace nastane, tak nestačí zadat pouze identifikátor datasetu, ale je nutné

k němu přidat i identifikátor konkrétního datasetu v Power BI. Power BI Service ale nepodporuje zpracovávat tímto způsobem datasety nevytvořené přes Power BI API. Datasety takto přidané musí být vytvořeny přes jejich API. Při přidání existujícího datasetu ale nastává problém, kdy aplikace neví kdy a z jakého souboru byl dataset naposledy aktualizován. Při spuštění aktualizace se pak aktualizuje z prvního možného zdroje (inkrementu na serveru s metrikami). Proto je potřeba postupovat opatrně a přidávat takto pouze datasety, které ještě aktualizovány nejsou a nehrozí u nich přidání duplicitních dat. Pokud si uživatel není jistý, jestli to je tento případ, tak je lepší přidat dataset jako úplně nový.

Záložka taky disponuje možností znovu manuálně načíst data. Dále u každého datasetu v seznamu si může uživatel zobrazit jeho podrobnosti. Ty obsahují informace o jeho identifikátorech a definici, tedy to jsou informace o tom, z jakých sloupců se skládá a jaké míry jsou z nich spočteny. Dataset je možné i vymazat. Není však možné datasety přejmenovávat. Toto rozhodnutí plyne z toho, že všechny informace o datasetu by měly být obsaženy v definici na serveru s metrikami. Tímto zajistíme, že definice se bude chovat jako jediný zdroj pravdy a nemůže pak dojít ke špatné interpretaci významu datasetu. Aktualizovat datasety je možné zvlášť, nebo může správce spustit aktualizaci pro všechny datasety najednou.

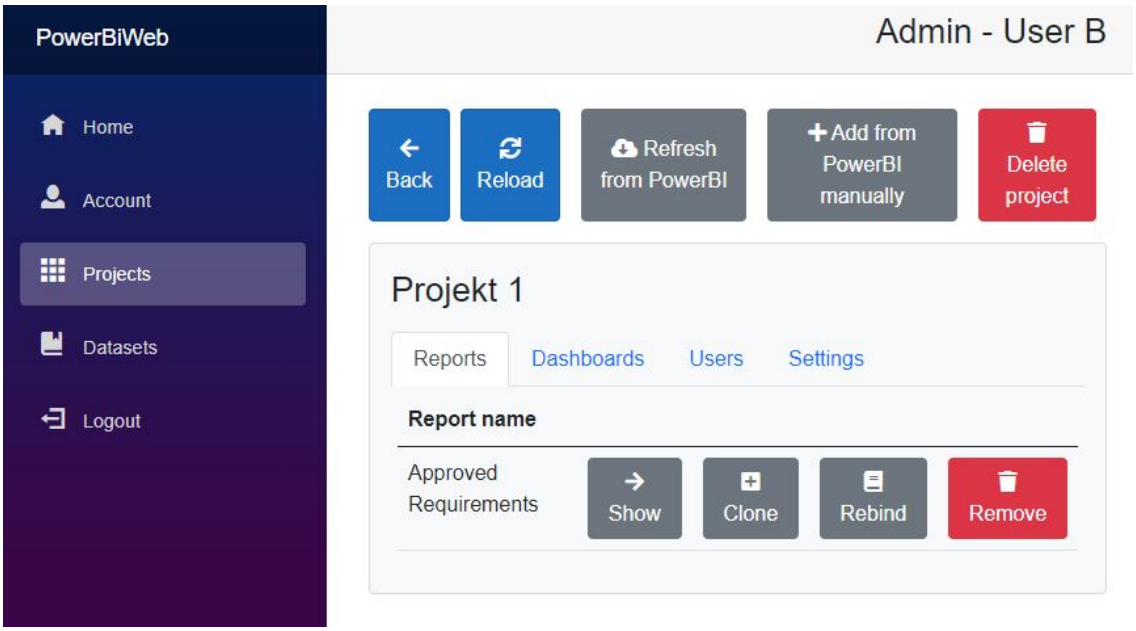
Další záložkou, kterou disponuje webová aplikace, je záložka s projekty. Ta je na rozdíl od záložky s datasety přístupná všem běžným uživatelům, tedy i uživatelům se základním přístupovým oprávněním.



Obrázek 7.3: Záložka s projekty

Princip načtení dat je stejný jako v případě předchozí záložky. Načtená data se taky dají manuálně znovu načíst. Další nabízená funkce je založení nového projektu. Nový projekt může založit

jakýkoli uživatel. Při zakládání projektu je vyžadován pouze jeho název. Po založení může uživatel přistoupit k jeho podrobnému zobrazení s více informacemi a detaily. V této záložce jsou zpřístupněny další funkce vztahující se k projektu. Kromě toho je možné i spravovat reporty a dashboardy, které jsou přidruženy k tomuto projektu.



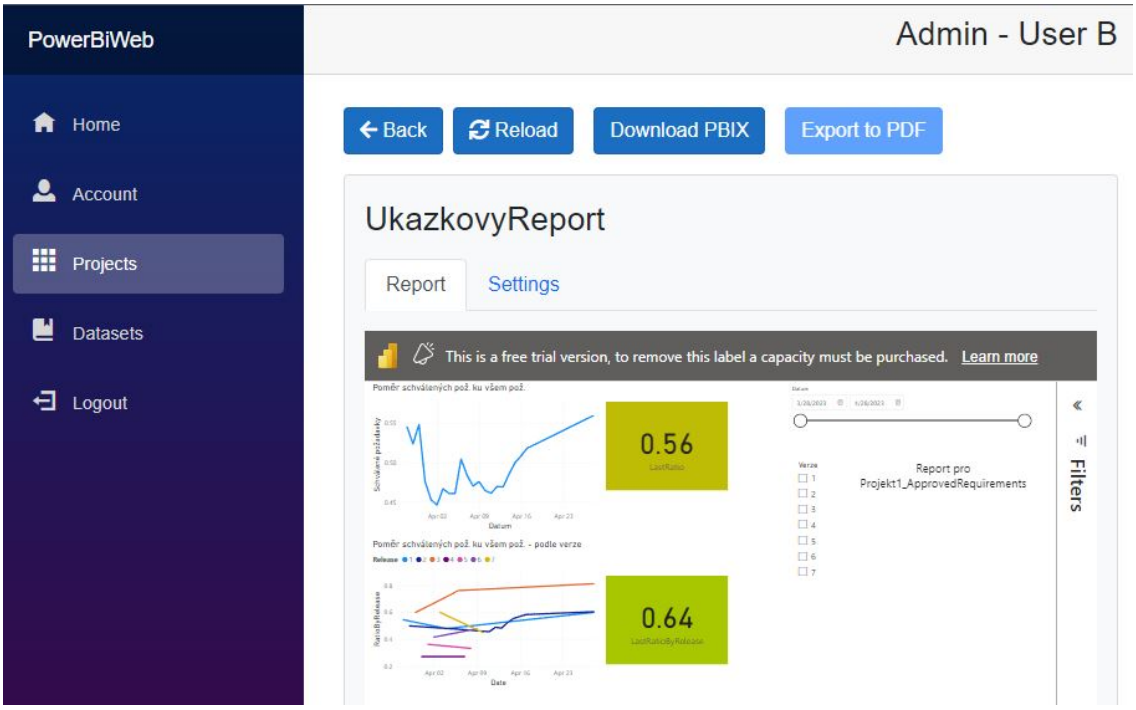
Obrázek 7.4: Záložka s detailem projektu

Uživatelé jsou v rámci projektu rozděleni do rolí. Jejich pravomoci podle rolí jsou zobrazeny v diagramu případu užití 4.2. Uživateli, jenž projekt vytvořil, je automaticky nastavena role zakladatele. Přidat další uživatele je pak možné manuálně. Při přidávání se zadává email, přes který se uživatel do aplikace registroval. Dále se při přidávání rovnou nastavuje role, kterou má nově přidaný uživatel obdržet. Přidat nového uživatele může i uživatel s rolí editora. Nemůže ale nastavit roli nově přidaného uživatele na roli větší, než má on sám. Toto omezení platí i při úpravě rolí, nebo mazání. Uživatelé v roli diváka nemohou přidávat a odebírat žádné uživatele. Nemohou ale ani sami projekt opustit. Možnost opustit projekt jim není zpřístupněna z důvodu, že v roli diváka je uživatel pouhý konzument obsahu v projektu. Nevytváří žádný nový obsah. Jeho rolí je pouze si zobrazovat obsah již vytvořený někým jiným. To může být například jeho nadřízený v reálném světě, který má v aplikaci ve stejném projektu roli s větším oprávněním. Divák by poté mohl nedopatřením opustit projekt a vznikly by zbytečně kroky potřebné k napravení této chyby. Proto je odebírání uživatelů a opouštění projektu umožněno pouze uživatelům ve vyšších rolích, u kterých se předpokládá, že vědí jaký obsah mají vytvářet a jaké uživatele chtějí v projektech mít. V případě, že je uživatel s vyšší rolí v projektu sám, nemůže ani on projekt opustit. V této situaci ho aplikace upozorní, že má projekt rovnou smazat, jelikož nedává smysl mít projekt bez uživatelů. Nikdo by do něj nemohl přidat nové uživatele, jelikož není nikdo, kdo by měl v daném projektu jakoukoli roli. Pokud uživatel

zůstane v projektu sám, protože všichni ostatní projekt opustili, tak je mu automaticky nastavena role zakladatele. Jinak by byl v projektu uvězněn, nemohl by ho smazat ani opustit. V případě kdy je projekt smazán, tak ho opustí i všichni uživatelé k němu přiřazení.

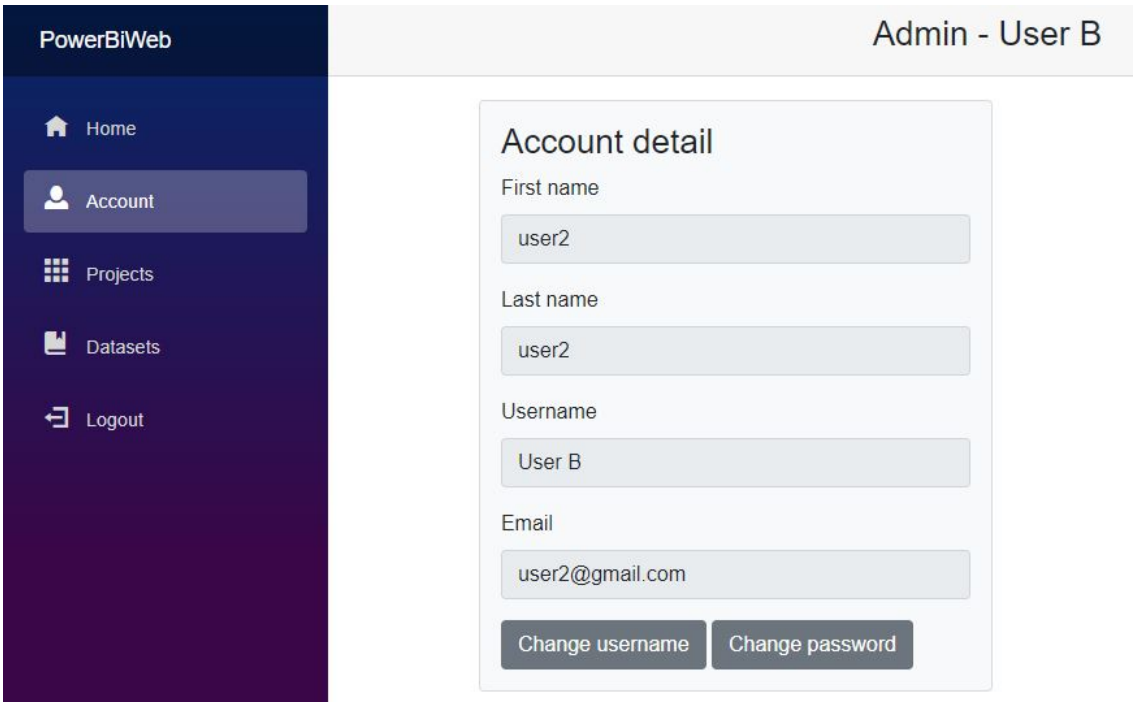
Obsah pro projekt je možné přidat pomocí spuštění aktualizace. Tento proces automaticky přidá příslušné reporty a dashboardy z Power BI Service do projektu. Podrobněji je proces popsán v následující kapitole, kde je popsána serverová část aplikace. Uživatel také může přidat konkrétní report nebo dashboard manuálně pomocí jeho identifikátoru v Power BI Service.

Reporty je možné naklonovat a přepojit na jiný dataset. Tímto se vytvoří v Power BI Service kopie naklonovaného reportu. Poté se dá využít druhá zmíněná funkce a to přepojení reportu na jiný dataset. Struktura a prvky reportu zůstanou stejné, ale nahradí se data, která report zobrazuje. Přepnutí jde použít korektně pouze v případě, kdy mají oba dva datasety stejnou strukturu a report tedy slouží jako šablona pro jejich zobrazení. Reporty je možné i stáhnout jako soubor s příponou pbix. Ten je možné poté otevřít v Power BI Desktop. Report využívá přímé spojení s datasetem v Power BI Service a má tak aktuální data. Problém je exportovat report do souboru pdf. Power BI Service sice tuto funkcionalitu přes své API nabízí, ale vyžaduje pro něj licencí větší než Power BI Pro. Logika je naimplementována, ale nebylo možné ji otestovat, protože není možné pro tento účel použít Power BI Service API. Místo toho je nabídnuta funkcionalita stáhnutí pbix souboru. Aby bylo možné report stáhnout, tak je nutné ho vytvořit v Power BI Desktop a poté nahrát do Power BI Service. Takto vytvořený report je pak dále možné naklonovat a klon bude také možné stáhnout.



Obrázek 7.5: Záložka s detailem reportu

V sekci uživatelského účtu vidí uživatel informace o svém uživatelském účtu. Uživateli je zde umožněno změnit heslo ke svému účtu a změnit přezdívku, přes kterou se uživatel přihlašuje. V případě kdy si uživatel mění svou přezdívku ke svému učtu, je uživatel vyzván k opětovnému přihlášení.



Obrázek 7.6: Záložka s detailem účtu

V navigačním menu je tlačítko pro odhlášení uživatele. Při odhlášení dojde k odstranění tokenu, který byl uložen při přihlášení. Poté je uživatel přesměrován na přihlašovací stránku.

7.1.3 Serverová část

Projekt se serverovou částí aplikace má následující strukturu:

- Properties - adresář obsahující konfigurační soubor *launchSettings*. Tento soubor slouží ke konfiguraci chování při spouštění aplikace;
- Controllers - složka obsahuje kontroléry obsluhující HTTP požadavky mířené na server;
- Interfaces - složka obsahuje soubory s rozhraními pro třídy;
- Middlewares - v tomto adresáři jsou třídy, které slouží jako middleware pro zpracování HTTP požadavků;
- Migrations - složka se soubory obsahujícími migrace databáze. Tyto soubory využívá Entity Framework Core;

- Models - složka obsahující třídy reprezentující entity v databázi;
- Repositories - složka obsahující třídy komunikující s úložištěm dat. Jedná se hlavně o databázi a Power BI Service;
- Services - v této složce jsou třídy, které obsahují hlavně aplikační logiku aplikace;
- Utilities - v této složce se nachází pomocné třídy a metody;

V souboru *appsettings.json* a *appsettings.Development.json* jsou definované konstanty, které aplikace využívá. Je zde uložen přihlašovací řetězec pro připojení k databázi. Taky jsou zde uloženy konstanty pro přihlášení a využívání Power BI Service. Dále obsahuje konfiguraci automatických funkcí aplikace.

Protože serverová část aplikace slouží jako backend aplikace a frontend s ní má komunikovat pomocí HTTP požadavků, tak server musí být schopný tyto požadavky přijímat a odpovídat na ně. Příchozí HTTP požadavky jsou zpracovávány pomocí kontrolérů. Kontrolér je označení pro třídu, která má *Controller* nebo *ApiController* atribut. Rozdíl mezi nimi je v tom, že *ApiController* automaticky přidává funkcionalitu pro REST odpovědi. Tedy automaticky nastavuje některé hlavičky do odpovědi a usnadňuje programátorovi nastavování HTTP kódu do odpovědi. Kontroléry v této práci dědí ze třídy *ControllerBase*. Třída *ControllerBase* je součástí ASP.NET frameworku a obsahuje zmíněný atribut *Controller*. Pro REST API ale spíše potřebujeme atribut druhý, proto je ještě manuálně přidaný i atribut *ApiController*. Dědění z třídy *ControllerBase* je hlavně pro zdědění logiky přístupu k HTTP požadavku. Framework ještě nabízí i třídu *Controller*. Tato třída ale obsahuje i logiku pro podporu pohledů. Toto není v této práci vyžadováno, protože serverová část aplikace slouží jako REST API. Kontroléry jsou registrovány v souboru *Program.cs*. Každý kontrolér má svou adresu, pomocí které se dají volat jeho metody přes HTTP požadavky. Tato adresa je definována atributem *Route* v kontroléru. Komunikace s frontendem aplikace pomocí HTTP požadavků také usnadňuje práci pro případ, kdy je požadováno nahradit frontend jinou technologií. Backend pouze poskytuje HTTP rozhraní, se kterým frontend komunikuje.

Autentizace probíhá podobně jako v klientské části aplikace a to pomocí JWT. Avšak některé kontroléry musí být přístupné i požadavkům bez JWT. Dobrým příkladem je kontrolér obsluhující registraci nebo kontrolér s přihlašováním uživatelů. Kontrolér, který má kontrolovat token, je označen atributem *Authorize*. Protože v tokenu je možné uložit více informaci, tak je nutné definovat, jaké informace má aplikace kontrolovat a oproti jakému šifrovacímu klíči. Toto je nastaveno v *Program.cs* souboru. Pokud jsou kontroléry zabezpečeny pomocí *Authorize* atributu, tak očekávají JWT v HTTP *Authorization* hlavičce. Její obsah by měl mít formát „Bearer <token>“ kde <token> nahradí vygenerovaný token, který uživatel obdrží po přihlášení do aplikace. Pokud se token v hlavičce nenachází, má špatný formát, nebo neobsahuje potřebné informace, tak je uživateli vrácen kód 401 Unauthorized. V případě, že jsou poslány nevalidní vstupní data, tak je vrácen kód 400 Bad Request. Jestli slouží požadavek k získání nějakých dat a server požadované data nenalezne,

tak je vracen kód 404 Not Found. Pokud je token platný, ale uživatel nemá oprávnění k danému požadavku, tak je uživateli vrácen kód 403 Forbidden. Poslední varianta může nastat například v případě, kdy uživatel chce přistoupit k projektu, kterého není součástí.

```
[Route("api/[controller]")]
[ApiController]
[Authorize]
public class ReportControler : ControllerBase
{ }
```

Listing 7.3: Ukázka kontroléru s jeho atributy

Aplikace obsahuje 6 kontroléru. Jejich popis a rozhraní jsou popsány níže.

AppUsersController obsluhuje požadavky pro přidávání a úpravu údajů o uživatelích. HTTP požadavkům poskytuje následující rozhraní:

- GetByIdAsync - metoda vrací informace o uživateli podle zadaného id;
- PostAsync - metoda zaregistruje nového uživatele do aplikace;
- ChangeUsernameAsync - metoda slouží ke změně přihlašovacího jména uživatele. Nové přihlašovací jméno se posílá v URL adrese;
- ChangePasswordAsync - metoda pro změnu hesla uživatele. Metoda požaduje staré i nové heslo v těle HTTP požadavku;

AuthController obsluhuje požadavky pro přihlašování. HTTP požadavkům poskytuje metodu *Login*. Tato metoda zkontroluje poslané informace o uživateli a pokud jsou správné, tak vygeneruje a vrátí JWT pro ověřování uživatele. Pokud poslané údaje nejsou správné, tak uživateli vrátí kód 401 Unauthorized.

DashboardsController obsluhuje požadavky pro aktualizaci dashboardů a získaní jejich informací. HTTP požadavkům poskytuje následující rozhraní:

- GetByIdAsync - metoda vrací informace o dashboardu a token pro jeho zobrazení. Metoda potřebuje id dashboardu a id projektu, do kterého dashboard patří;
- UpdateDashboardsAsync - metoda aktualizuje dashboardy v daném projektu oproti Power BI Service. Je podrobněji popsána níže;
- UpdateDashboardSettingsAsync - metoda aktualizuje jméno dashboardu. Očekává údaje o dashboardu;

DatasetsController obsluhuje požadavky pro přidání a odstranění datasetů a získaní jejich informací. Téměř všechny jeho funkce jsou přístupné pouze uživatelům v roli správce. HTTP požadavkům poskytuje následující rozhraní:

- GetAll - metoda vrací informace o všech datasetech;
- GetById - metoda vrací informace o datasetu podle zadaného id;
- DeleteById - metoda odstraní dataset podle zadaného id;
- AddDatasetById - metoda přidá nový dataset podle zadaného identifikátoru na serveru s metrikami, stáhne jeho data ze serveru s metrikami a přidá je do Power BI Service;
- AddExistingDatasetById - metoda přidá dataset podle zadaných identifikátorů, které už existují v Power BI Service. Očekává identifikátor datasetu na serveru s metrikami a identifikátor v Power BI Service;
- UpdateAll - metoda aktualizuje všechny datasety inkrementem na serveru s metrikami;
- UpdateById - metoda aktualizuje dataset se zadaných id inkrementem na serveru s metrikami;

ProjectsController obsluhuje požadavky pro přidání a odstranění projektů, získaní jejich informací a přidávání obsahu do nich. HTTP požadavkům poskytuje následující rozhraní:

- GetAll - metoda vrátí informace o všech projektech, do kterých patří uživatel posílající požadavek;
- GetById - metoda vrátí informace o projektu podle specifikovaného id;
- Post - metoda vytvoří nový projekt a přidá do něj uživatele, který ho vytváří. Po vytvoření do něj přidá obsah z Power BI. Očekává informace a novém projektu;
- AddReport - metoda přidá report do projektu podle specifikovaného id reportu a id projektu;
- RemoveReport - metoda odstraní report z projektu podle specifikovaného id reportu a id projektu;
- AddDashboard - metoda přidá dashboard do projektu podle specifikovaného id dashboardu a id projektu;
- RemoveDashboard - metoda odstraní dashboard z projektu podle specifikovaného id dashboardu a id projektu;
- AddToUser - metoda přidá uživatele do projektu a nastaví mu roli. Očekává informace o projektu a roli pro přiřazení;
- EditUser - metoda aktualizuje roli uživatele v daném projektu. Očekává novou roli uživatele a jeho id;
- EditProject - metoda aktualizuje název projektu. Očekává nový název a id projektu;

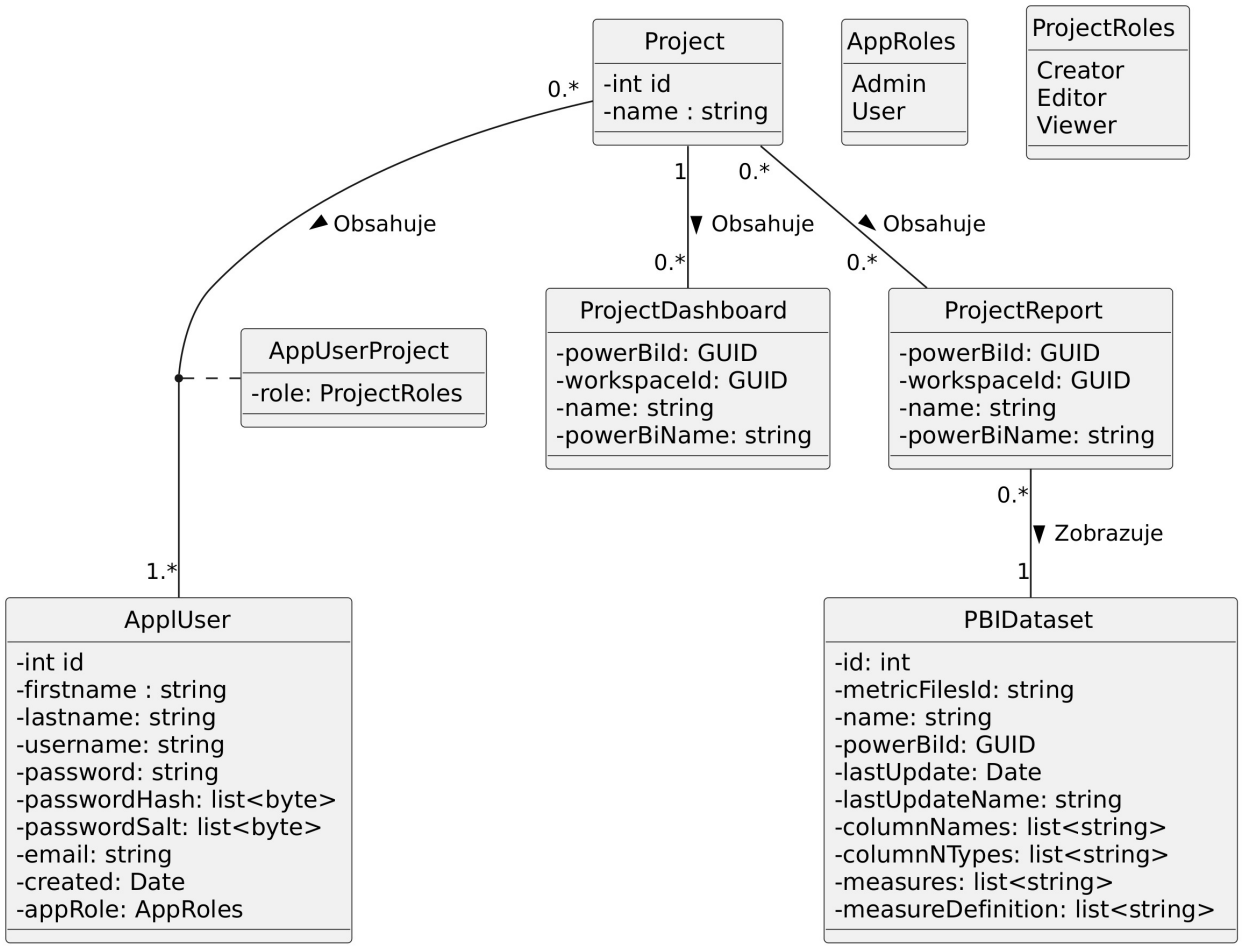
- DeleteUser - metoda odstraní uživatele z projektu podle zadaného id projektu a id uživatele;
- DeleteProject - metoda odstraní projekt podle zadaného id;

ReportsController obsluhuje požadavky pro aktualizaci reportů, získaní jejich informací, export a stahování. HTTP požadavkům poskytuje následující rozhraní:

- GetByIdAsync - metoda vrátí informace o reportu a token pro jeho zobrazení. Očekává id reportu a id projektu, do kterého report patří;
- UpdateReportsAsync - metoda aktualizuje reporty v daném projektu oproti Power BI Service. Je podrobněji popsána níže;
- CloneReportAsync - metoda naklonuje report v Power BI Service podle zadaného id reportu;
- RebindReportAsync - metoda přepne report na jiný dataset. Očekává id reportu a id datasetu, na který ho má přepnout;
- UpdateReportSettingsAsync - metoda aktualizuje jméno reportu. Očekává informace o reportu;
- ExportReportAsync - metoda vrátí pdf soubor s reportem podle specifikovaného id reportu. Vyžaduje účet s licencí větší než Power BI Pro, takže je v době psaní této práce neaktivní;
- DownloadReportAsync - metoda vrátí pbix soubor s reportem podle specifikovaného id reportu;

Kontroléry neobsahují aplikační logiku aplikace. Tuto logiku obsahují třídy se suffixem Service tzv. služby. Každý kontrolér má vytvořenou vlastní takovou třídu pro svoje rozhraní, které poskytuje HTTP požadavkům. Služby fungují na stejném principu jako v klientské části aplikace. Takže implementují nějaké rozhraní a pokud nějaká třída chce využívat její funkcionalitu, tak využívá nadefinované rozhraní. Konkrétní implementace se pak registruje v souboru *Program.cs*. Kontroléry tak upravují vstup a volají metody služeb. Služby pro kontroléry zde nebudou podrobněji popsány, protože obsahují metody se stejným účelem, jako obsahují metody kontrolérů. Jejich hlavním účelem je oddělení logiky od kontrolérů a jednoduchá výměna aplikační logiky s využitím rozhraní.

Podobně jak služby obsahují aplikační logiku, tak repozitáře obsahují logiku pro přístup k úložišti. Jde o třídy se suffixem Repository. Tyto třídy využívají Entity Framework Core pro přístup k databázi a Power BI klienta pro přístup k Power BI Service. Jak každý kontrolér má vlastní službu, tak má každá služba vlastní repozitář pro přístup k datům, který ukládá změny na datech, nebo získává a vrací požadovaná data. Stejně jako služby tak i repozitáře fungují na principu definování rozhraní a registrace konkrétní implementace. Metody, které repozitáře využívané službami obsahují, korespondují s metodami služeb a kontrolérů.



Obrázek 7.7: Třídní diagram entit na serveru

Serverová část obsahuje tři služby schopné běhu na pozadí. Spouští se na základě konfigurace v *appsettings* souboru. Jsou popsány níže.

BackgroundDownloadMetricsService je služba sloužící k automatickému zálohování reportů. Při aktivaci své funkce postupně projde všechny reporty a stáhne jejich pbix soubor. Adresář pro stáhnutí se nastavuje v konfiguraci, stejně tak jako se v konfiguraci nastavuje interval spouštění.

BackgroundUpdateContentService je služba, která slouží k automatické aktualizaci obsahu v projektech. Při spuštění postupně projde všechny projekty a aktualizuje jejich obsah oproti Power BI Service. Obsah rozpoznává podle shody prefixu názvů reportů a dashboardů vůči názvu projektu. Shoda je citlivá na malá a velká písmena.

BackgroundUpdateMetricsApiService je služba sloužící k automatické aktualizaci datasetů. Při spuštění postupně projde všechny datasety a stáhne jejich nový inkrement ze serveru s metrikami a přidá je do Power BI Service. Kontroluje však název souboru obdrženého ze serveru se jménem souboru poslední aktualizace a v případě shody tak aktualizaci konkrétního datasetu přeskočí.

K aktualizaci obsahu pro projekty se používají metody *UpdateReportSettingsAsync* a *UpdateDashboardsAsync*. Pokud aktualizaci iniciuje automatická služba, tak se tyto metody volají postupně v příslušných službách. V případě, že aktualizaci započne uživatel, tak jsou volány stejnojmenné metody v kontrolérech. Ty vlastně obalují metody od služeb pro přístup přes HTTP požadavky. Při aktualizaci reportů se načtou všechny reporty z Power BI Service a vyberou se ty, které mají patřit do aktuálního projektu. Reporty se vybírají podle shody názvu projektu s prefixem názvu reportu. Shoda je citlivá na malé a velké znaky. Prefix je brán od začátku názvu do prvního podtržítka. Například pro projekt s názvem „Foo25“ bude vybrán report s názvem „Foo25_OdpracovaneHodiny“. Pak se vybrané reporty do projektu přidají. Kromě automatické aktualizace je možné přidat report nebo dashboard i manuálně. Tato funkce je vhodná v případě, kdy chceme sdílet obsah ve více projektech najednou a nechceme soubory pojmenovávat podle názvu jednoho konkrétního projektu. Toto řeší funkce *AddReport* a *AddDashboard*. Proces automatické aktualizace je níže popsán pseudokódem. Tento proces je poté obsažen v sekvenčním diagramu 7.8.

Funkce *AktualizujReportInfo* upraví informace o názvu reportu v Power BI. V pseudokódu je to zobrazeno funkcí, protože je možné, že v budoucnu bude potřeba upravit i další informace o reportu a ne jenom název. Funkce *PridejDatasetInfo* přidá k reportu informaci o tom, na jaký dataset je připojen. Toto se ovšem děje pouze v případě, že report existuje. Pseudokód pro aktualizace dashboardů zde není zobrazen, protože používá podobný proces. Rozdíl je v tom, že na konci se nezjišťují informace o datasetu, jelikož dashboard žádný dataset připojený nemá. V implementaci je funkce doplněna o návratovou hodnotu řetězce. V případě chyby pak funkce vrací chybovou hlášku pro uživatele.

Algoritmus 1: Aktualizace reportů pro projekt

```
   Input: projektId : Id projektu
 1 projektVdb ← NajdiProjektVDatabazi(projektId);
 2 if projektVdb is null then return;
 3 pbiKlient ← VytvorKleinta();
 4 reporty ← pbiKlient.NactiReporty();
 5 foreach report in reporty do
 6 |   entitaReportu ← projektVdb.NajdiReport(report.Id);
 7 |   if entitaReportu is not null then
 8 |   |   AktualizujReportInfo(entitaReportu, report)
 9 |   else
10 |   |   if report má správný prefix then
11 |   |   |   entitaReportu ← NajdiReportVdb(report.Id);
12 |   |   |   if entitaReportu is not null then
13 |   |   |   |   projekt.PridejReport(entitaReportu);
14 |   |   |   |   AktualizujReportInfo(entitaReportu, report);
15 |   |   |   else
16 |   |   |   |   entitaReportu ← VytvorNovyReport(report);
17 |   |   |   |   projekt.PridejReport(entitaReportu);
18 |   |   |   |   PridejReportDoDatabaze(entitaReportu);
19 |   |   |   end
20 |   |   end
21 |   end
22 |   if entitaReportu is not null then
23 |   |   dataset ← NajdiDatasetVdb(report.datasetId);
24 |   |   if dataset is not null then
25 |   |   |   PridejDatasetInfo(entitaReportu, dataset);
26 |   |   end
27 |   end
28 end
```

Automatické služby se konfigurují každá zvlášť. Čtení konfigurace uvnitř služby je možné díky použitého rozhraní *IOptions*, které je součástí .NET technologie. Toto rozhraní umožňuje napojit informace o konfiguraci z konfiguračního souboru na model reprezentující tuto konfiguraci. Napojení konfigurace na model se provádí v *Program.cs* souboru. Existují i další typy tohoto rozhraní lišících se ve způsobu čtení konfigurace a podpory změny konfigurace za běhu aplikace. Toto není vyžadováno a tak je použito základní rozhraní pouze pro napojení na model. Modely pro konfiguraci automatických služeb mají stejnou strukturu až na ten, který je pro službu automatického záloho-

vání, ten obsahuje navíc informaci o adresáři pro uložení souborů. Proto všechny modely dědí ze společného předka, který obsahuje společné vlastnosti a zmíněný model navíc obsahuje informaci o adresáři uložení. Pro konfiguraci je možné vybrat interval týdne, dne, nebo hodiny. Automatickou službu je taky možné úplně deaktivovat. Podrobnosti jsou popsány níže, stejně tak jako diagramy některých automatických procesů.

- Týden (Week) - služba je spuštěna každý týden v nastavený den, hodinu a minutu;
- Den (Day) - služba je spuštěna každý den v nastavenou hodinu a minutu. Hodnota zvoleného dne se nepoužívá;
- Hodina (Hour) - služba je spuštěna každou hodinu v nastavenou minutu. Hodnota zvoleného dne a hodiny se nepoužívá;

```
"DatasetsUpdate": {
  "enabled": false,
  "UpdateFrequency": "Week",
  "DayOfWeek": "Saturday",
  "Hour": 4,
  "Minute": 10
},
"ContentUpdate": {
  "enabled": false,
  "UpdateFrequency": "Week",
  "DayOfWeek": "Sunday",
  "Hour": 23,
  "Minute": 10
},
"DownloadPbix": {
  "enabled": false,
  "SavePath": "./export",
  "UpdateFrequency": "Week",
  "DayOfWeek": "Sunday",
  "Hour": 23,
  "Minute": 10
}
```

Listing 7.4: Konfigurace automatických služeb

```
builder.Services.Configure<DatasetUpdateOptions>(
    builder.Configuration.GetSection("DatasetsUpdate"));
```

Listing 7.5: Napojení konfigurace na model

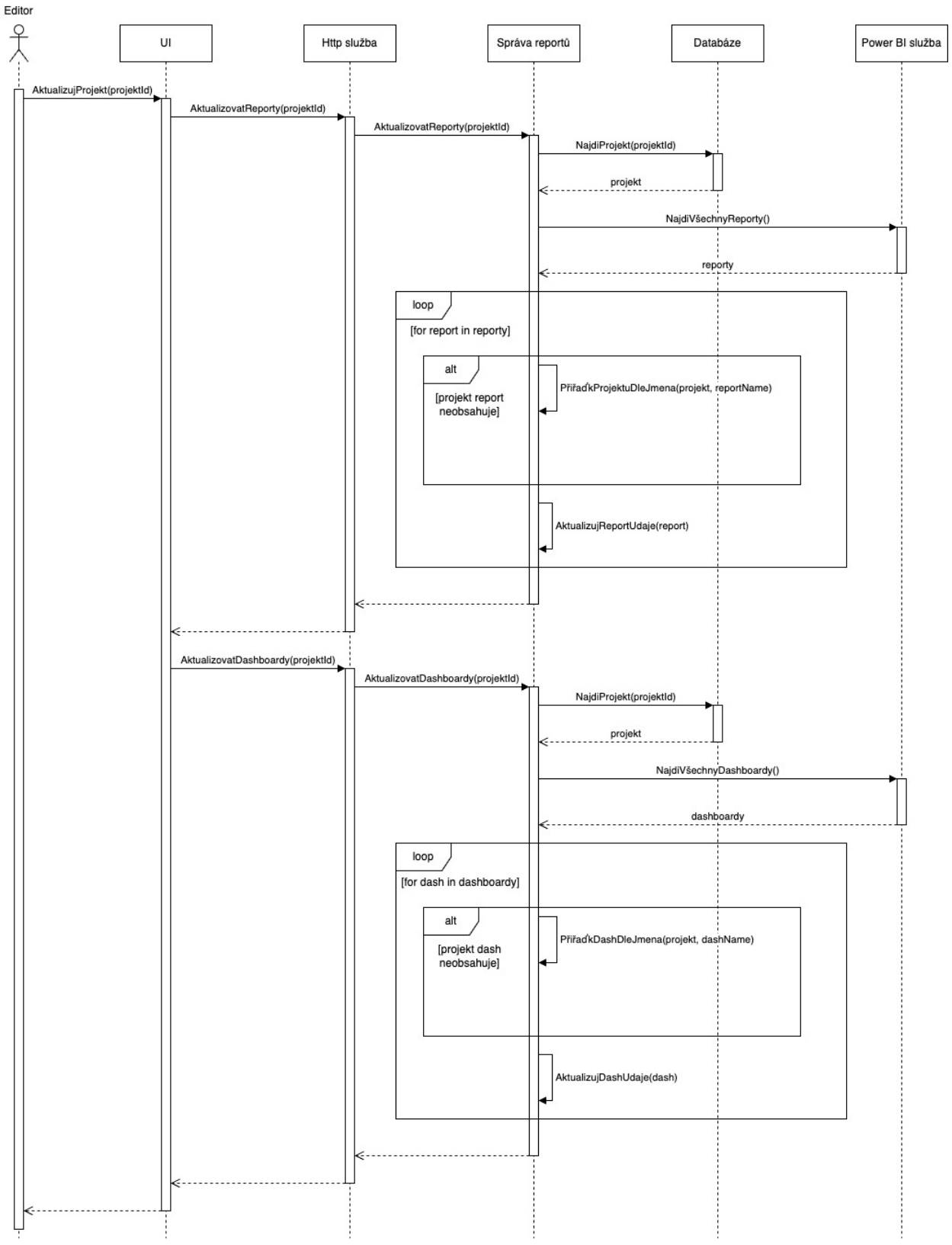
Při konfiguraci doby spuštění je nutné vzít do úvahy pořadí, ve kterém se budou služby spouštět. Například pokud chceme službou *BackgroundDownloadMetricsService* stáhnout nové reporty, tak se tato služba musí spustit až po aktualizaci datasetů, jinak by data v nich nebyla aktuální. Naopak pokud chceme stáhnutím zálohovat staré soubory, pak nastavíme dobu spuštění dříve, než se spustí aktualizace.

Jako časovač, který spouští hlavní funkci automatické služby, je použit *PeriodicTimer*. Na rozdíl od ostatních časovačů nebere v úvahu dobu trvání spouštěné funkce. O trochu delší doba trvání tedy nemůže ovlivnit spuštění v budoucnu. Časovač se ale může zpozdit v případě, kdy doba trvání funkce překročí interval časovače.

V aplikaci je používán jeden vlastní middleware. Je to třída *ExceptionCatcher*. Každý požadavek zpracovávaný aplikací prochází každým middlewarem na cestě do aplikace a na cestě zpět. *ExceptionCatcher* slouží k lepšímu odchytávání výjimek v aplikaci. Uživatel tak nevidí popis vnitřních chyb, které mohou na serveru nastat. Aplikace také používá vlastní typ výjimky a to *MessageException*. Ta je použita v případě, kdy chceme uživateli přímo vrátit nějakou konkrétní chybovou hlášku. Middleware jí odchytí a zapíše její informace do odpovědi.

Pro převod mezi DTO objekty a entitními objekty na serveru slouží pomocná třída *DTOExtensions*. Jde o statickou třídu a metody v ní jsou implementovány jako metody rozšíření, takže mohou být volány přímo na DTO objektech v případě převodu na server, nebo na entitních objektech v případě vracení objektů klientovi.

Pro uložení hesel v databázi je použit hašovací algoritmus SHA256. Hesla ale nejsou hašována samotná, ale je k nim i přidána i tzv. kryptografická sůl. Jde o náhodně vygenerovaný řetězec přidaný k heslu. V případě získání údajů z databáze tak nejsou nutně ohrožena hesla uživatelů. Tento proces přidávání kryptografické soli a hašování je použit i v případě přihlašování. Porovnává se hašované heslo se solí v databázi oproti hašovanému heslu, které uživatel vyplní do přihlašovacího formuláře.



Obrázek 7.8: Sekvenční diagram automatické aktualizace obsahu

Pro spojení s Power BI Service slouží pomocné třídy *AadService* a *PowerBiUtility*. Obě třídy se nachází uvnitř složky *Utilities*. První zmíněná třída, tedy *AadService*, slouží k získání přístupového tokenu pro komunikaci s Power BI. K tomuto získání využívá přihlašovací údaje definované v *appsettings* souboru. Třída získává token ze služby Azure. Druhá třída vytváří komunikačního klienta obsahujícího metody přímo pro komunikaci s Power BI Service. K vytvoření tohoto klienta potřebuje přístupový token, který je vytvořen třídou *AadService*. *PowerBiUtility* není předán samotný token, ale je jí předána instance třídy, ze které si token získá sama. *AadService* si token uchovává v paměti pro pozdější použití. Power BI totiž vrací chybovou hlášku, pokud je překročen limit požadavků na vytvoření tokenu za nějaký časový interval.

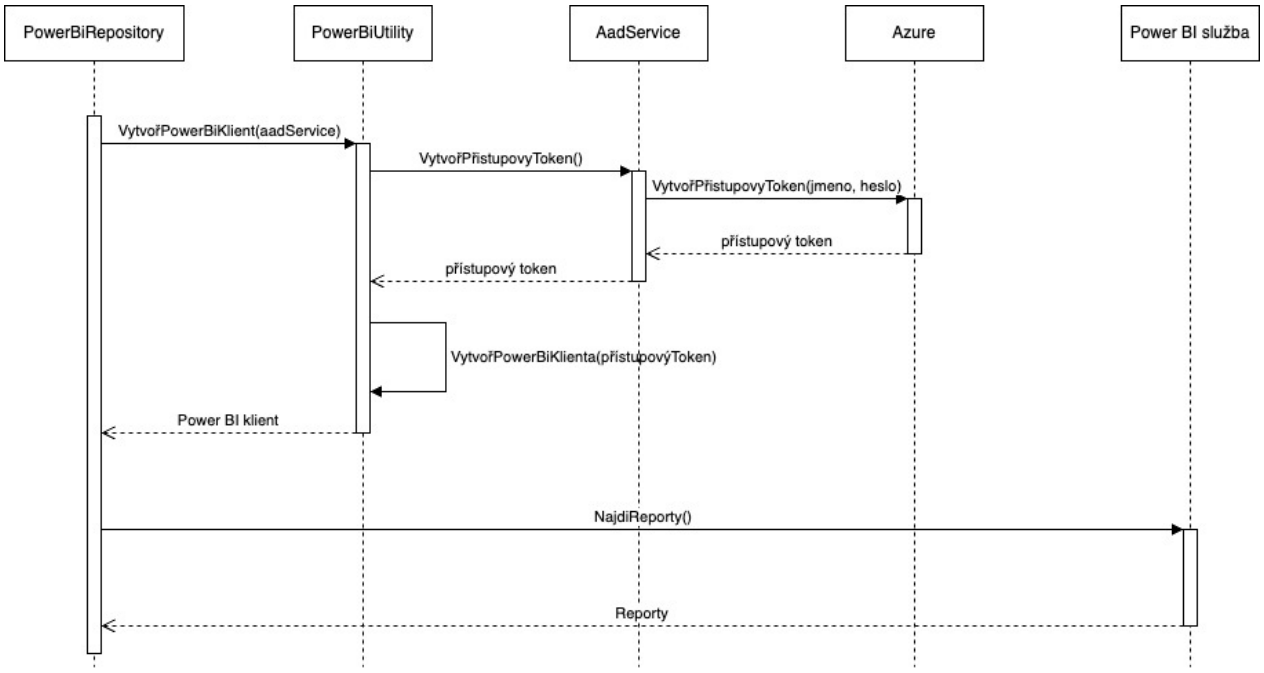
```
public class PowerBiOptions
{
    public string WorkspaceId { get; set; } = string.Empty;
    public string ReportId { get; set; } = string.Empty;
    public string AuthenticationMode { get; set; } = string.Empty;
    public string AuthorityUrl { get; set; } = string.Empty;
    public string ClientId { get; set; } = string.Empty;
    public string TenantId { get; set; } = string.Empty;
    public string[] ScopeBase { get; set; } = Array.Empty<string>();
    public string PbiUsername { get; set; } = string.Empty;
    public string PbiPassword { get; set; } = string.Empty;
    public string ClientSecret { get; set; } = string.Empty;
    public string PowerBiApiUrl { get; set; } = string.Empty;
}
```

Listing 7.6: Model reprezentující konfiguraci Power BI připojení

```
"PowerBiOptions": {
    "AuthenticationMode": "masteruser",
    "AuthorityUrl": "https://login.microsoftonline.com/organizations/",
    "ClientId": "clientId",
    "TenantId": "tenantId",
    "ScopeBase": [ "https://analysis.windows.net/powerbi/api/.default" ],
    "PbiUsername": "username",
    "PbiPassword": "password",
    "ClientSecret": "clientSecret",
    "WorkspaceId": "workspaceId",
    "PowerBiApiUrl": "https://api.powerbi.com"
}
```

Listing 7.7: Konfigurace Power BI připojení z appsettings souboru

Pro spojení s Power BI slouží repozitář *PowerBiRepository*. Ke komunikaci používá výše zmíněný Power BI klient. Pro jeho vytvoření používá pomocnou třídu *PowerBiUtility*. Poté volá metody vytvořeného klienta, který poskytuje stejnou funkcionalitu jako samotné ruční posílání HTTP požadavků. V jednom případě je ale nutné ručně takový požadavek vytvořit. Je to případ přidávání dat do datasetů. Metoda *PostRows*, kterou disponuje klient, nekorektně serializuje objekt reprezentující data pro přidání. Z tohoto důvodu je HTTP požadavek vytvořen ručně, stejně tak je ručně spuštěna serializace objektu. Požadavek je poté zpracováván jako klasický HTTP požadavek pomocí HTTP klienta.



Obrázek 7.9: Sekvenční diagram ukázky vytvoření Power BI klienta a komunikace s Power BI Service

7.2 Server s metrikami

Účel této aplikace je shromažďovat data o metrikách a dále poskytovat rozhraní pro HTTP požadavky ke čtení dat od těchto metrik. Jako implementace byla zvolena webová aplikace s ASP.NET technologií, stejně jako v případě uživatelské aplikace. Server s metrikami ale neobsahuje žádné uživatelské rozhraní.

Rozhraní nabízené serverem není rozsáhlé a tak aplikace nepoužívá kontroléry ale tzv. minimal API. Jde o odlehčenou verzi kontrolérů. Výkon aplikace se tak může zlepšit, jelikož aplikace nemusí

registrovat žádné kontroléry a při zpracovávání požadavku jej jednodušeji navede do metody, která ho zpracovává.

```
app.MapGet(
    "/metrics/definition/{metricName}",
    (string metricName, IMetricLoaderService loader) =>
        {
            return loader.LoadMetricDefinition(metricName);
        }
);
app.MapGet(
    "/metrics/total/{metricName}",
    (string metricName, IMetricLoaderService loader) =>
        {
            return loader.LoadMetricData(metricName, false);
        }
);
app.MapGet(
    "/metrics/inc/{metricName}",
    (string metricName, IMetricLoaderService loader) =>
        {
            return loader.LoadMetricData(metricName, true);
        }
);
```

Listing 7.8: Poskytované rozhraní od serveru s metrikami

Projekt má strukturu popsanou níže.

- Properties - adresář obsahující konfigurační soubor *lauchSettings*. Tento soubor slouží ke konfiguraci chování při spouštění aplikace.
- Interfaces - složka obsahuje soubory s rozhraními pro třídy;
- Metrics - složka obsahuje data s metrikami;
- Models - složka obsahující třídy, které reprezentují entity v projektu;
- Services - v této složce jsou třídy obsahující hlavně aplikační logiku aplikace;

Aplikace nabízí tři metody. Umožňuje čtení definice, čtení všech dat od určité metriky a čtení inkrementu od určité metriky. Data o metrikách jsou rozdělena do složek. Každá složka od metriky obsahuje tři podložky. První je složka s názvem *Definition* a obsahuje definici reprezentace metriky

jako Power BI dataset. Druhá je složka s názvem *Increment* a obsahuje inkrement pro metriku. Poslední je složka s názvem *Total* a obsahuje všechna data pro metriku. Inkrementem je myšlen přírůstek dat v nějakém definovaném intervalu. Aplikace podporuje dva formáty pro metrikové soubory. Je to formát csv a formát json.

7.2.1 Aktualizace inkrementu

Aplikace počítá při aktualizaci inkrementu pro metriku s následujícím procesem. Data ze souboru z inkrementu se zkopírují a vloží na konec souboru s celkovými daty. Dále se obsah souboru s inkrementem nahradí novými daty a upraví se název souboru na aktuálnější datum. Obsah souboru s definicí zůstává stejný. Při vytváření nové metriky je potřeba vytvořit novou složku se třemi výše zmíněnými podsložkami a naplnit je příslušnými soubory.

Pro správný chod aplikace je potřeba nakonfigurovat interval přidávání nových inkrementů. Protože aplikace čte data ze souborů, tak vybírá soubory podle názvu. Který soubor má přečíst, se určuje podle zmíněné konfigurace. Je tak rozhodnuto proto, aby server nedal uživateli omylem starý inkrement. Systém konfigurace služby je podobný jako v případě automatických služeb u serveru klientské aplikace, ale obsahuje ještě navíc možnost nastavit automatické generování inkrementů pro účely testování.

```
"MetricsUpdate": {
  "UpdateFrequency": "Hour",
  "DayOfWeek": "Saturday",
  "Hour": 5,
  "MockUpdate": {
    "Enabled": false,
    "Minute": 15
  }
}
```

Listing 7.9: Konfigurace aktualizace metrik s testováním

Po nakonfigurování intervalu aktualizace aplikace počítá s tím, že v dané době už bude existovat nový soubor s odpovídajícím jménem a ten bude v případě požadavku hledat. Logika pojmenování souboru se mění podle konfigurace intervalu a je popsána následovně:

- Týden (Week) - aplikace hledá soubor ve formátu „nazevMetriky_00_dd_MM_yyyy“, kde „dd“ označuje den inkrementu, „MM“ označuje jeho měsíc a „yyyy“ jeho rok. Den a měsíc v názvu jsou ve formátu dvou číslic a rok ve formátu čtyř číslic. Očekává se nejbližší uplynulé datum odpovídající konfiguraci. Takže název souboru „nazevMetriky_00_15_04_2023“ odpovídá konfiguraci pro *DayOfWeek* hodnotu nastavenou na *Saturday* a bude platit od 15.04. až do následující soboty 22.04, kdy ji nahradí soubor s tímto datem;

- Den (Day) - aplikace hledá soubor ve formátu stejném jako v případě týdnu, ale jako datum očekává den, který inkrement reprezentuje;
- Hodina (Hour) - aplikace hledá soubor ve formátu „nazevMetriky_HH_dd_MM_yyyy“, kde „HH“ označuje hodinu inkrementu, „dd“ označuje jeho den, „MM“ označuje jeho měsíc a „yyyy“ jeho rok. Hodina, den a měsíc v názvu jsou ve formátu dvou číslic a rok ve formátu čtyř číslic. Jako datum očekává hodinu, kterou inkrement reprezentuje;

Logika pojmenování je hlavně pro přečtení správného souboru a obsah souboru teoreticky nemusí odpovídat datu v pojmenování. Dává ale smysl inkrementy plnit relevantními daty, které byly posbírány právě v uplynulém intervalu. Při dodržení tohoto postupu je také menší prostor pro chyby při vytváření inkrementu. Je nutné podotknout, že všechny časy se zadávají v koordinovaném světovém čase (UTC). Formát datumu v metrikách musí být ve tvaru „dd.MM.yyyy“, takže například 23.03.2023.

Aplikace sama soubory s metrikami neaktualizuje. Výjimkou je zmíněné generování inkrementu pro testování. Po konfiguraci aktualizace aplikace počítá s tím, že soubory budou už aktualizované správcem aplikace. Jako řešení se nabízí například automatický skript, který bude spouštěn službou Cron na systému Linux. Nebo je také možné aplikaci doplnit o sofistikovanou automatickou službu zápisu inkrementů podle reálných dat.

7.2.2 Testovací generování inkrementu

Aplikace nabízí automatickou službu pro ukázku aktualizace souborů zvanou *MockBackgroundMetricUpdater*. Ta aktualizuje inkrement výše zmíněným procesem. Do souboru generuje náhodné hodnoty. Interval spouštění je shodný s intervalem, kdy aplikace očekává nový inkrement. Jinými slovy, tato služba sdílí konfiguraci s konfigurací aplikace. Dále ale obsahuje ještě informaci o tom, ve které minutě se má spustit. Toto je hlavně pro účely testování. Vývojář si může nastavit minutu spuštění na blízkou dobu a vygenerovat tak nový inkrement. Nemusí tak čekat do celé hodiny. Největší smysl má testovat tuto službu v konfiguraci hodinové aktualizace. Vývojář tak nemusí nastavovat hodnoty dne a hodiny, protože služba se spustí každou hodinu v nakonfigurovanou minutu. Pokud bychom však chtěli službu využít v produkčním prostředí pro generování nových inkrementů, tak nejlépe by se měla spouštět na začátku nastaveného intervalu. Aplikace totiž počítá s tím, že v této době už bude existovat soubor inkrementu s příslušným názvem. Přesnou dobu přidání inkrementu je pak možné vyladit, pokud dopředu víme jaké aplikace budou číst metriky a v jakou dobu.

Kapitola 8

Závěr

Na základě zadaných požadavků bylo nutné vytvořit dvě webové aplikace. První aplikace využívající Microsoft Power BI pro vizuální reprezentaci metrik a jejich analýzu by byla určena pro používání vývojovými týmy, podobnými firmami nebo těmi, kteří chtějí analyzovat své firemní procesy na základě metrik. Druhá aplikace by sloužila jako poskytovatel dat o metrikách. První zmíněná aplikace by komunikovala s druhou a získávala z ní data, která by posílala do Microsoft Power BI. A protože je také žádoucí, aby byl proces částečně automatizován, tak z tohoto důvodu by aplikace měla nabízet automatickou funkcionalitu na základě konfigurace. Požadavky a procesy, se kterými uživatelé a aplikace pracují, bylo potřeba zaznamenat pro lepší pochopení těchto procesů a spolehlivější vývoj.

Obě aplikace byly navrhnuty a implementovány. Požadavky a procesy s nimi spojené byly zachyceny pomocí metod a technologií používaných v oblasti softwarového inženýrství, jako jsou například různé druhy diagramů. Implementace poté vychází z těchto artefaktů. První aplikace určená pro práci s Power BI obsahem umožňuje také uživatele rozdělit do projektů a nastavit jim v rámci projektů role, podle nichž se pak odvíjí jejich možné funkce. V kontextu celé aplikace se uživatelé dělí na roli správce a běžného uživatele. Do projektů je pak možné přidávat obsah v podobě reportů a dashboardů, jenž jsou součástí Power BI Service. V rámci automatizace nabízí celkem tři automatické služby, které jsou spouštěny na základě konfigurace aplikace. Druhá aplikace poskytuje HTTP rozhraní pro čtení metrik a obsahuje automatickou službu k aktualizaci metrik pro lepší testování. Tyto výsledky splnily očekávané cíle funkčnosti aplikací a podporu procesu pro analýzu metrik.

V průběhu vývoje se bylo nutné vypořádat s obtížemi v používání Power BI Service kvůli jejich systému licencí a způsobu povolení pro využívání jejího API. Obě aplikace jsou implementovány takovým způsobem, aby mohly být v budoucnu doplněny o širší funkcionalitu, nebo aby ta stávající mohla být pozměněna. V další iteraci vývoje by se například mohlo jednat o vylepšení rozhraní serveru s metrikami, aby bylo možné vybírat i starší data v jiném časovém rozmezí. U druhé aplikace by rozvoj mohl jít směrem k širší kontrole nad datasety a jejich daty, protože tyto dva aspekty aplikací vzájemně spolupracují.

Literatura

1. *What is business intelligence?* [online]. [cit. 2023-03-09]. Dostupné z: https://www.ibm.com/topics/business-intelligence.
2. STEGER, Bernhardt; EKERT, Damjan; MESSNARZ, Richard; STOLFA, Jakub; STOLFA, Svatopluk; VELART, Zdenek. Metrics and Dashboard for Level 2 – Experience. In: YILMAZ, Murat; NIEMANN, Jörg; CLARKE, Paul; MESSNARZ, Richard (ed.). *Systems, Software and Services Process Improvement*. Cham: Springer International Publishing, 2020, s. 652–672. ISBN 978-3-030-56441-4.
3. *The world's leading analytics platform* [online]. [cit. 2023-03-13]. Dostupné z: https://www.tableau.com/.
4. *SAP BusinessObjects Business Intelligence* [online]. [cit. 2023-03-13]. Dostupné z: https://www.sap.com/products/technology-platform/bi-platform.html.
5. *What is .NET?* [online]. [cit. 2021-02-25]. Dostupné z: https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet.
6. WAGNER, Bill; NEWCOMB, Ben; WARREN, Genevieve; PINE, David; SCHONNING, Nick; ADDIE, Scott; KULIKOV, Petr; WENZEL, Maira; VICTOR, Youssef; LATHAM, Luke; ONDERKA, Petr. *A tour of the C# language* [online]. 2021. [cit. 2021-02-25]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/.
7. *Common web application architectures* [online]. [cit. 2022-04-17]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures.
8. *ASP.NET Core Blazor* [online]. Microsoft Corporation, 2022 [cit. 2023-01-13]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-7.0.
9. *ASP.NET Core Blazor hosting models* [online]. Microsoft Corporation, 2022 [cit. 2023-01-13]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/blazor/hosting-models?view=aspnetcore-7.0%5C#blazor-hybrid.
10. *What is Power BI?* [online]. Microsoft Corporation, 2023 [cit. 2023-01-13]. Dostupné z: https://learn.microsoft.com/cs-cz/power-bi/fundamentals/power-bi-overview.

11. *DAX overview* [online]. Microsoft Corporation, 2022 [cit. 2023-01-14]. Dostupné z: https://learn.microsoft.com/en-us/dax/dax-overview.

12. *Real-time streaming in Power BI* [online]. Microsoft Corporation, 2023 [cit. 2023-01-15]. Dostupné z: https://learn.microsoft.com/en-us/power-bi/connect-data/service-real-time-streaming.

13. *What is a dashboard?* [online]. Microsoft Corporation, 2021 [cit. 2023-01-14]. Dostupné z: https://learn.microsoft.com/en-us/power-bi/consumer/end-user-dashboards.

14. *Workspaces in Power BI* [online]. Microsoft Corporation, 2023 [cit. 2023-01-15]. Dostupné z: https://learn.microsoft.com/en-us/power-bi/collaborate-share/service-new-workspaces.

15. *What is Power BI embedded analytics?* [online]. Microsoft Corporation, 2023 [cit. 2023-01-15]. Dostupné z: https://learn.microsoft.com/en-us/power-bi/developer/embedded/embedded-analytics-power-bi.

16. *Tutorial: Embed Power BI content using a sample embed for your customers' application* [online]. Microsoft Corporation, 2022 [cit. 2023-01-15]. Dostupné z: https://learn.microsoft.com/en-us/power-bi/developer/embedded/embed-sample-for-customers?tabs=net-core.

17. PÉREZ, Sergio Darias. *What is microsoft sql server and what is it for?* [online]. 2021. [cit. 2023-01-15]. Dostupné z: https://intelequia.com/en/blog/post/2948/what-is-microsoft-sql-server-and-what-is-it-for.

18. *Entity Framework Core* [online]. Microsoft Corporation, 2021 [cit. 2023-01-15]. Dostupné z: https://learn.microsoft.com/en-us/ef/core/.

19. *Use containers to Build, Share and Run your applications* [online]. [cit. 2023-01-15]. Dostupné z: https://www.docker.com/resources/what-container/.

20. *It's how you make software* [online]. Microsoft Corporation [cit. 2023-01-15]. Dostupné z: https://visualstudio.microsoft.com/.

21. *Fast & powerful cross-platform .NET IDE* [online]. [cit. 2023-01-15]. Dostupné z: https://www.jetbrains.com/rider/.

22. *GitKraken Client Features* [online]. [cit. 2023-01-15]. Dostupné z: https://www.gitkraken.com/git-client/features.

23. *Set up Power BI Embedded* [online]. Microsoft Corporation, 2022 [cit. 2023-01-15]. Dostupné z: https://learn.microsoft.com/en-us/power-bi/developer/embedded/register-app?tabs=customers.

24. FOWLER, Martin; RICE, David; FOEMMEL, Matthew; HIEATT, Edward; MEE, Robert; STAFFORD, Randy. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.

Příloha A

Struktura archivu

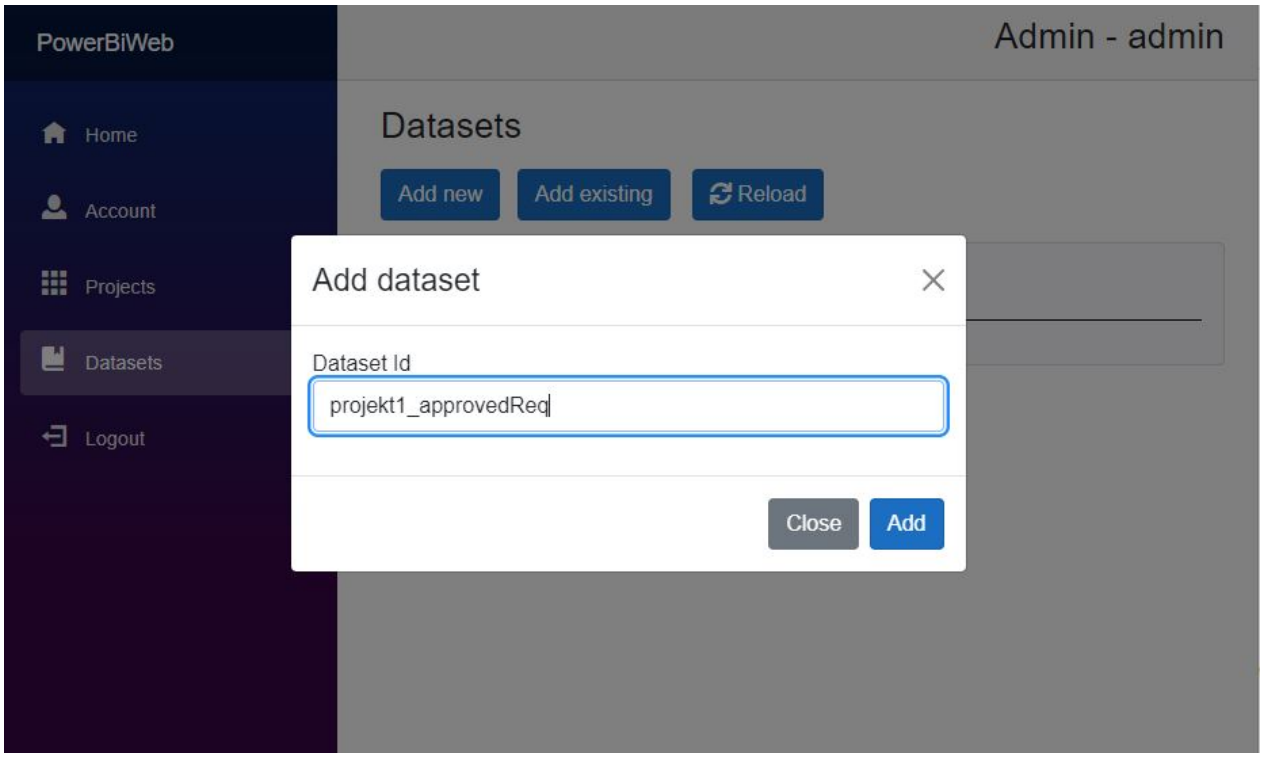
/
- `README.txt` ..... Soubor obsahující návod ke spuštění aplikací
- `README.md` ..... Návod v upraveném Markdown formátu
- `PowerBiWeb` ..... Adresář obsahující aplikaci pro uživatele
- `MetricsApi` ..... Adresář obsahující aplikaci pro poskytování metrik
- `Data` ..... Adresář obsahující ukázkové metriky a reporty

Příloha B

Ukázka aplikace

Zde je popsána základní ukázka aplikace a popis některých funkcí pomocí textu a obrázků. Počítá se s tím, že aplikace jsou už nakonfigurované pomocí přiloženého návodu a spuštěné. Klientská aplikace obsahuje dva uživatelské účty a jejich společný projekt. Server s metrikami zase tři ukázkové metriky.

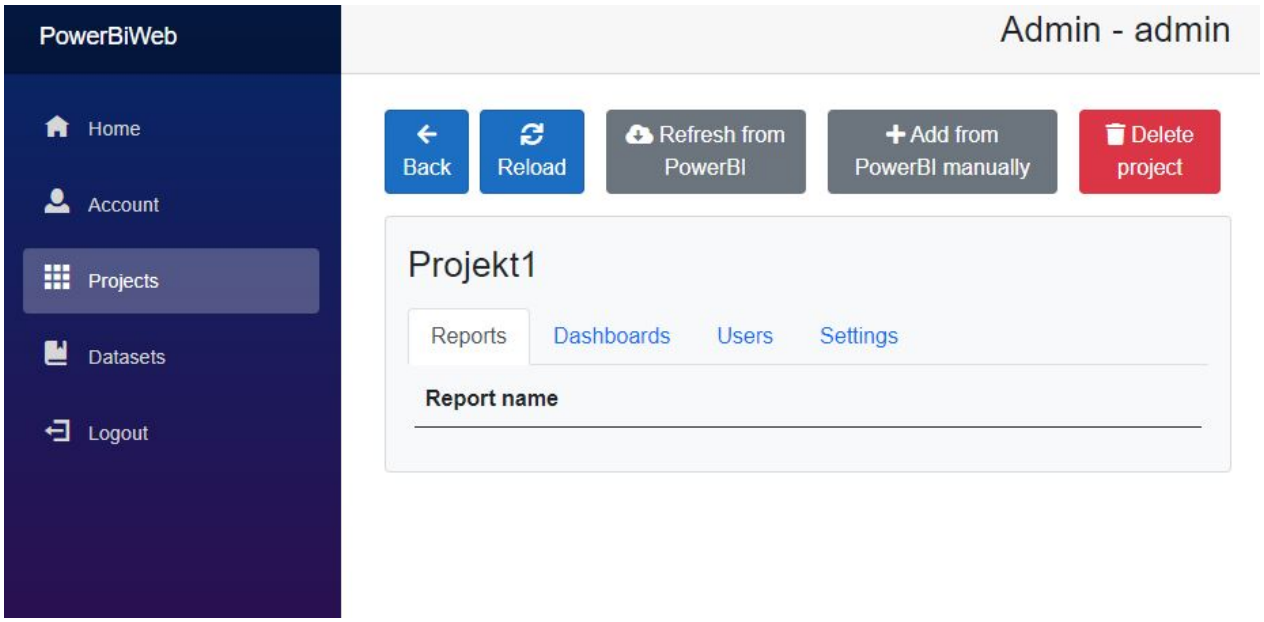
Uživatel v roli správce se přihlásí a v záložce s datasety přidá nový dataset *projekt1_approvedReq* a *projekt1_finishedReq* pomocí tlačítka *Add new*.



Obrázek B.1: Přidání nového datasetu *projekt1_approvedReq*

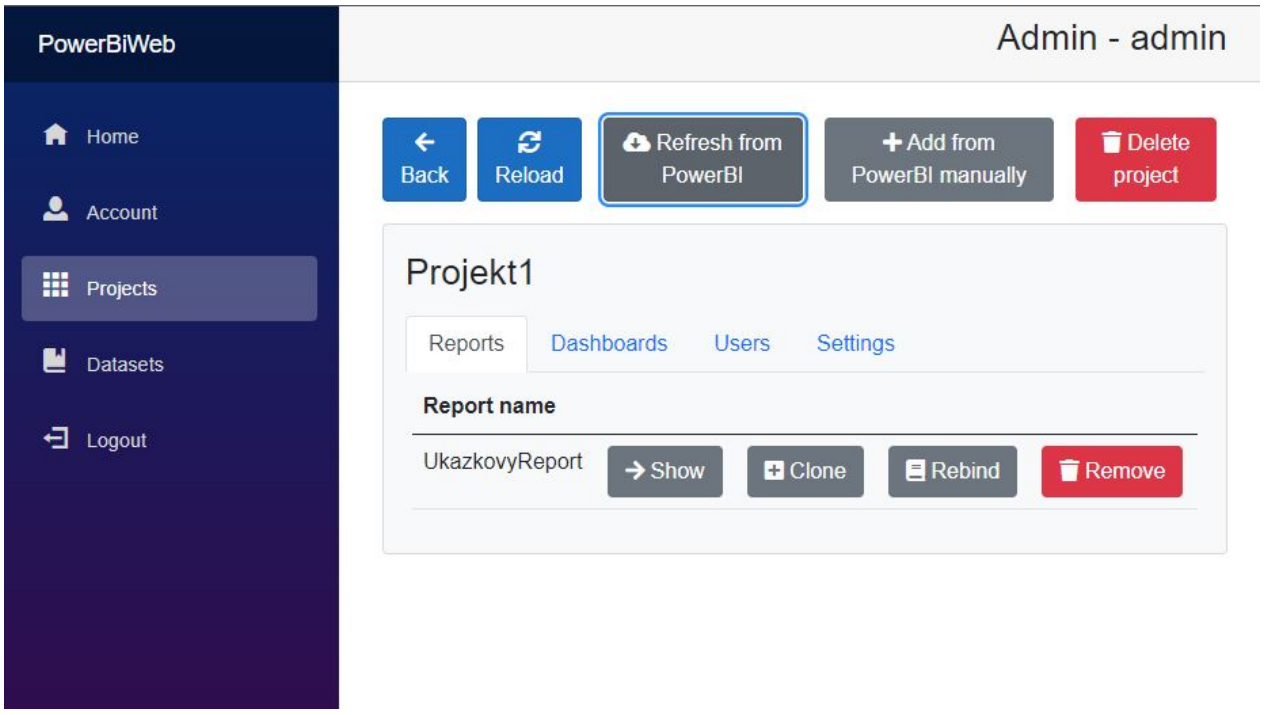
V Power BI Desktop se přihlásí přes Power BI účet, který má přístup do nastaveného workspace

a vytvoří report z přidaných datasetů a nahraje je do Power BI Service. V rámci této práce lze otevřít přiložený report *Projekt1_UkazkovyReport* ve složce Data, ten přepnout na nově přidaný *projekt1_approvedReq* dataset. Protože je report při otevření napojený na dataset v jiném workspace, tak Power BI sám vybídne k nastavení jiného datasetu. Po přepnutí je report možné nahrát do nastaveného Power BI workspace.



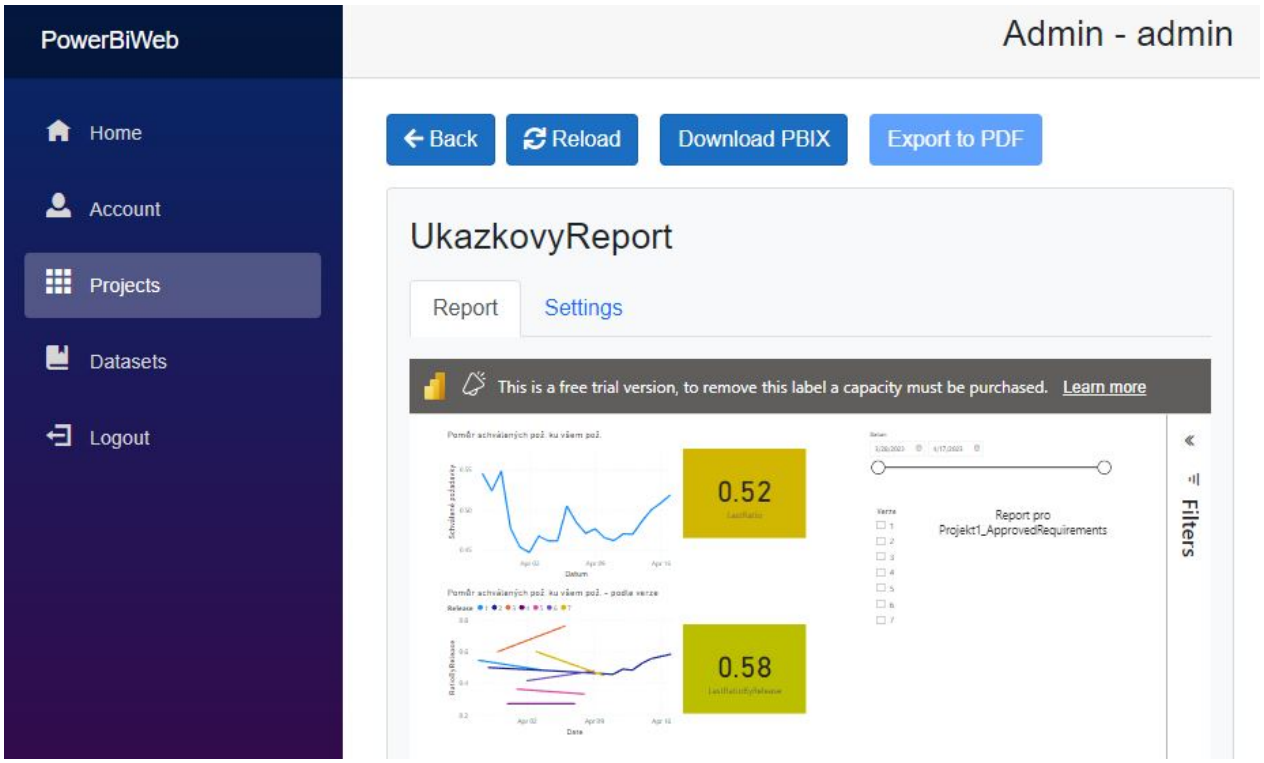
Obrázek B.2: Projekt před aktualizací

Poté lze aktualizovat obsah projektu pomocí tlačítka *Refresh from PowerBI*, nebo pomocí tlačítka *Add from PowerBI manually* a vyplnění formuláře.



Obrázek B.3: Projekt po aktualizaci

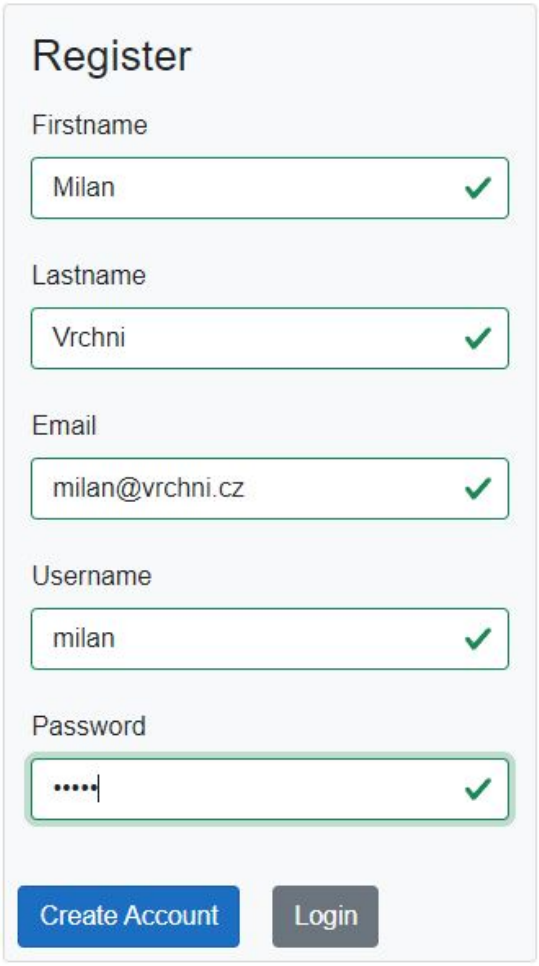
Když je obsah aktualizovaný, tak je možné si report zobrazit. Report je také možné v záložce s nastavením přejmenovat.



Obrázek B.4: Zobrazený report

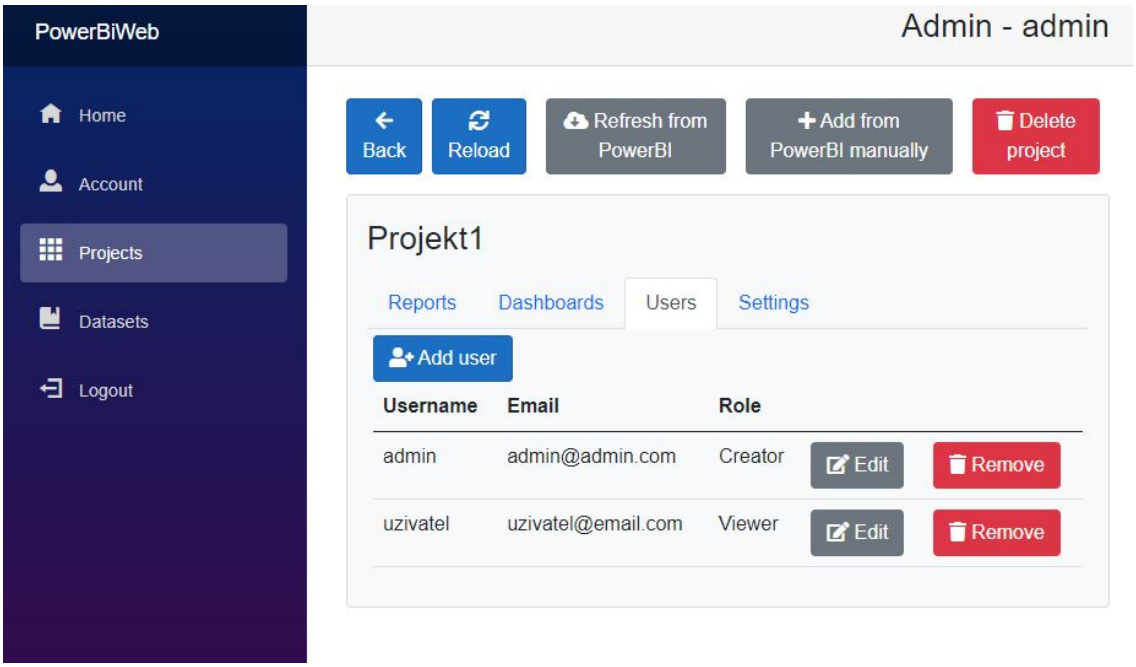
Obrázek B.5: Detail report

V případě potřeby přidání nového uživatelského účtu se uživatel zaregistruje přes přihlašovací stránku, ke které má uživatel přístup, pokud není přihlášen.

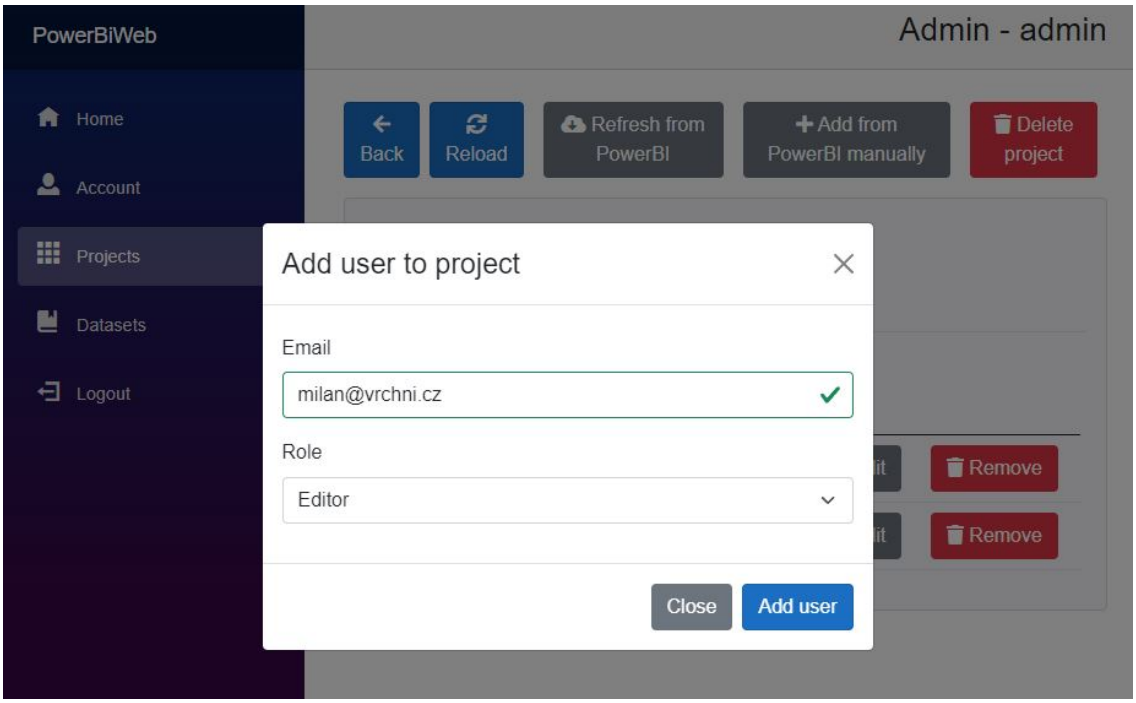


Obrázek B.6: Registrační formulář

Správce pak může uživatele přidat do projektu pomocí jeho emailu v záložce s uživateli přes tlačítko *Add user*. V tomto případě bude mít nově přidaný uživatel v projektu roli editora.

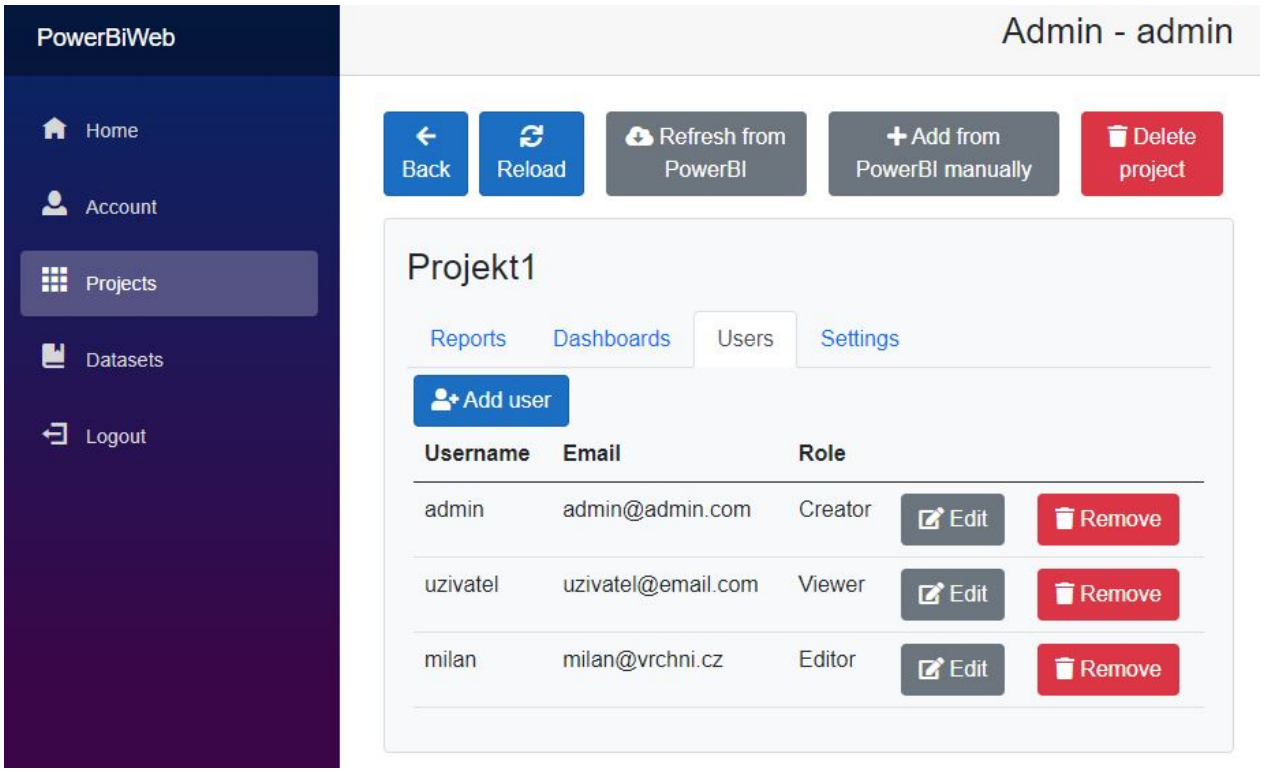


Obrázek B.7: Sekce uživatelů v projektu



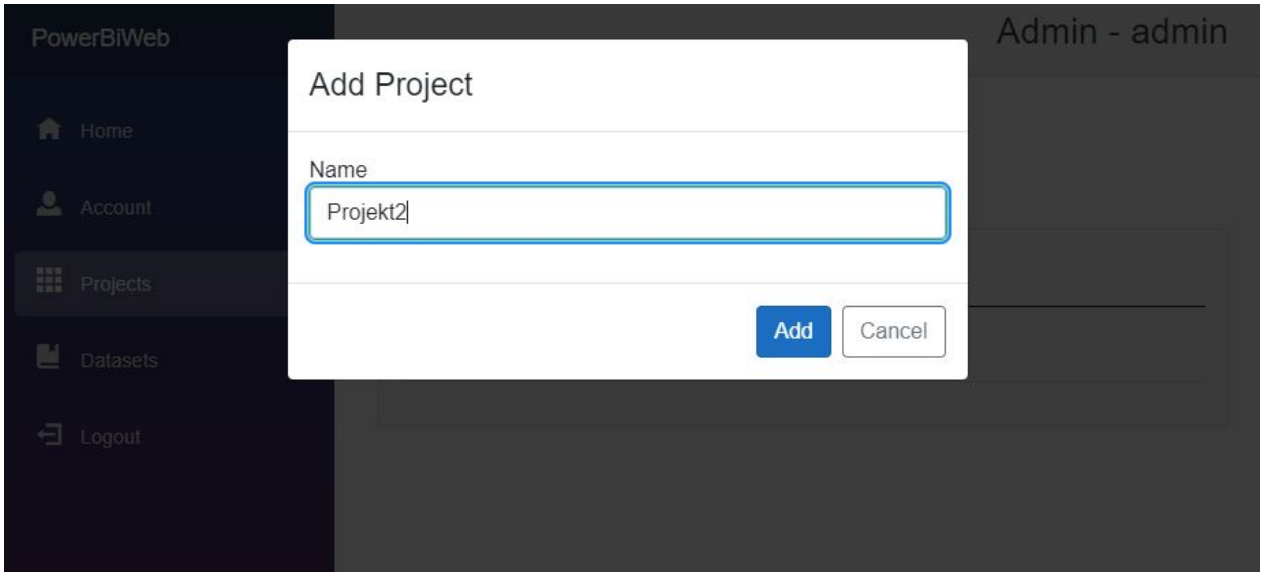
Obrázek B.8: Dialog pro přidání uživatele

Uživatel se pak zobrazí v seznamu uživatelů v projektu.



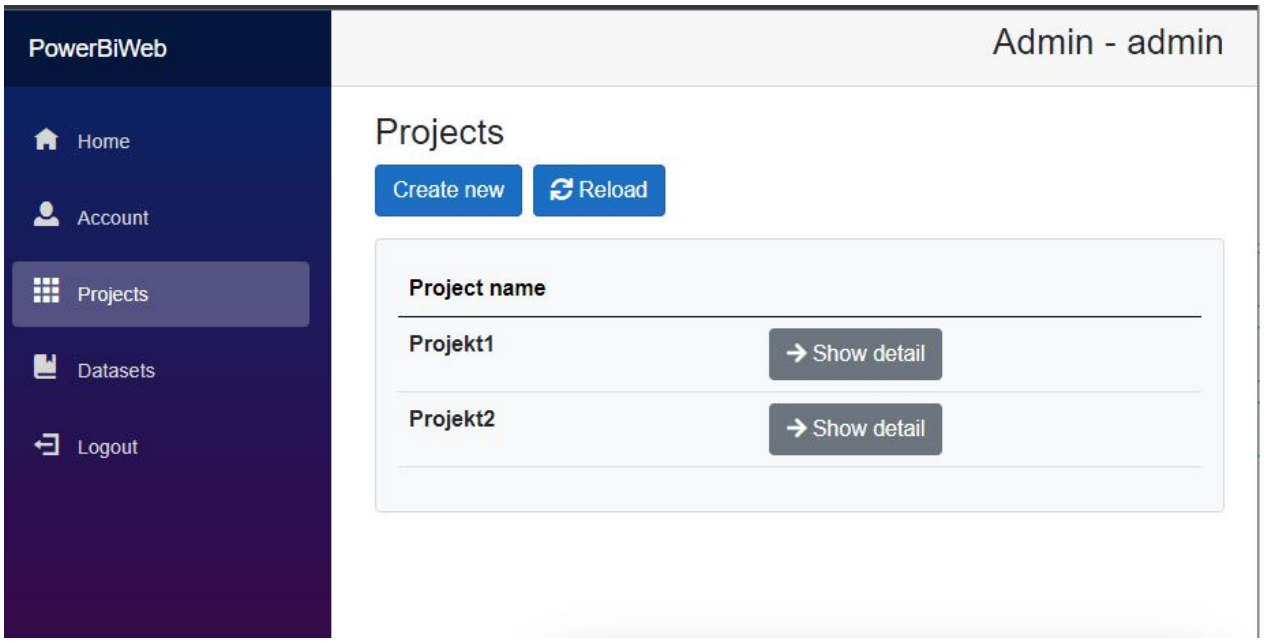
Obrázek B.9: Seznam uživatelů v projektu

V ukázkových datech je již jeden projekt vytvořen. Pokud uživatel chce vytvořit další projekt, tak lze tak učinit skrze tlačítko *Create new* v záložce s projekty.



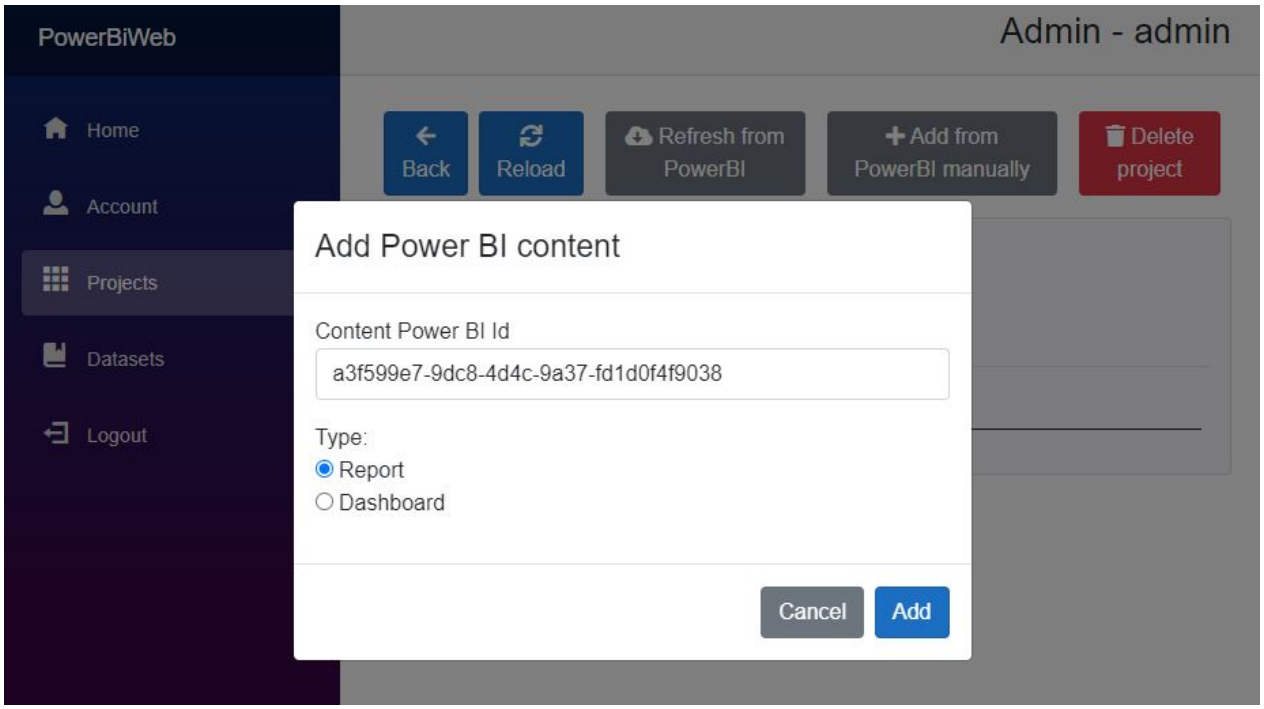
Obrázek B.10: Dialog pro vytvoření nového projektu

Nový projekt se zobrazí v seznamu.



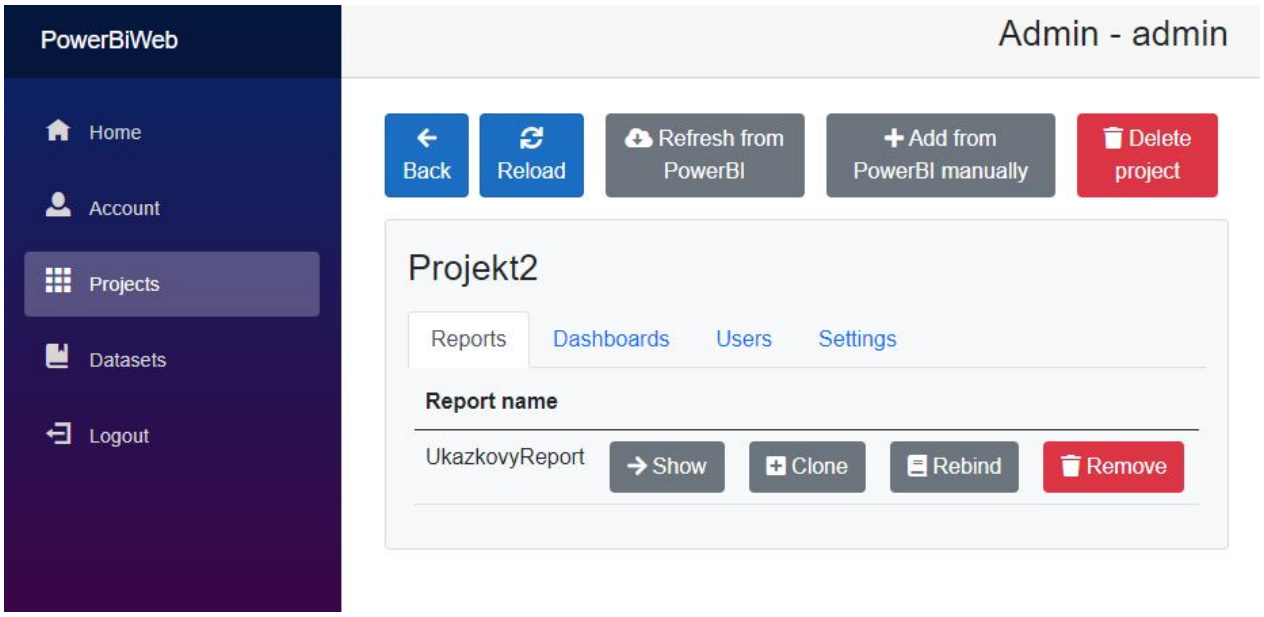
Obrázek B.11: Seznam projektů

Po zobrazení detailu projektu je možné přidat další uživatele, stejně jako v případě projektu předešlého. Tento projekt ale nemá žádné reporty, které odpovídají jeho jménu pro automatickou aktualizaci. Proto uživatel může přidat report pomocí tlačítka *Add from PowerBI manually*.



Obrázek B.12: Manuální přidání reportu

Report, který uživatel vytvořil, se teď nachází v obou projektech.



Obrázek B.13: Druhý projekt s přidaným reportem

Dashboardy jdou vytvářet pouze v Power BI Service, protože se skládají z jiných reportů. Po vytvoření dashboardu je ho možné přidat do projektu stejně jako report. Buď ho lze přidat pomocí

jeho identifikátoru, nebo v případě vhodného pojmenování i pomocí automatického obnovení.