VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2021 Bc. Jan Bachorec

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

WEBOVÁ APLIKACE PRO SYSTÉM DETEKCE RIZIKOVÝCH SITUACÍ NA ŽELEZNIČNÍM PŘEJEZDU

WEB APPLICATION FOR THE DETECTION SYSTEM OF RISK SITUATIONS AT THE RAILWAY CROSSING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Jan Bachorec

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Zdeněk Martinásek, Ph.D.

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Bc. Jan Bachorec ***ID:*** 195148
Ročník: 2 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Webová aplikace pro systém detekce rizikových situací na železničním přejezdu

POKYNY PRO VYPRACOVÁNÍ:

Cílem diplomové práce je návrh a implementace vhodného serverového řešení a grafického uživatelského rozhraní pro autonomní systém detekce rizikových situací v dopravě. Grafické uživatelské rozhraní bude mít formu interaktivní webové stránky a umožní vhodnou analýzu rizikových situací snímaných kamerovými moduly na železničním přejezdu. V teoretické části analyzujte vhodné šablony pro tvorbu rozhraní (.NET Core a C#) a také vhodné technologie pro tvorbu serverové části (př. Apache Kafka a Elassandra) včetně definice sledovaných parametrů. Výsledkem teoretické části bude kompletní návrh serverové aplikace a klientského uživatelského rozhraní. V rámci praktické části realizujte experimentální pracoviště a návrh webové aplikace implementujte. Experimentální pracoviště bude umožňovat simulaci několika kamerových modulů pro definované kategorie odesílaných zpráv. Funkčnost implementace ověřte.

DOPORUČENÁ LITERATURA:

[1] GASSTON, Peter. The modern Web: multi-device Web development with HTML5, CSS3, and JavaScript. San Francisco: No Starch Press, 2013. ISBN 15-932-7487-4.

[2] THEIN, Khin Me Me. Apache kafka: Next generation distributed messaging system. International Journal of Scientific Engineering and Technology Research, 2014, 3.47: 9478-9483.

Termín zadání: 1.2.2021 ***Termín odevzdání:*** 24.5.2021

Vedoucí práce: Ing. Zdeněk Martinásek, Ph.D.
Konzultant: Roman Kümmel (HACKER Consulting s.r.o.)

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ABSTRAKT

Diplomová práce se zabývá kompletním návrhem, vývojem a implementací serverového řešení a grafického uživatelského rozhraní pro autonomní systém detekce rizikových situací na železničním přejezdu. Serverové řešení se skládá ze serverové služby a databáze. K vývoji serverové služby byla použita platforma .NET Core. Jejím účelem je provádět sběr, zpracovaní a ukládání informací o rizikových dopravních situacích zaznamenaných kamerovými moduly. Pro bezpečné uložení dat tato služba používá clusterovou databázi Cassandra, přičemž schéma databáze bylo vytvořeno s ohledem na vysoký výkon. Vlastní uživatelské rozhraní je realizováno s pomocí webové aplikace, postavené na technologii Razor Pages. Tato aplikace prezentuje uživateli zaznamenané dopravní incidenty, přičemž jako zdroj dat jí slouží vlastní REST API služba. S její pomocí jsou implementovány veškeré logické operace s daty v databázi.

KLÍČOVÁ SLOVA

webová aplikace, vizualizace, databázové systémy, API služby, REST

ABSTRACT

The thesis deals with the complete design, development and implementation of the server solution and graphical user interface for an autonomous system for detecting risk situations at a railway crossing. The server solution consists of a server service and a database. The .NET Core platform was used to develop the server service and its purpose is to perform the collection, processing and storage of information about risky traffic situations recorded by the camera modules. The service uses a Cassandra cluster database for secure data storage, and the database schema was designed with a focus on high performance. The own user interface is implemented with a web application built on Razor Pages technology. This application presents recorded traffic incidents to the user, using its own REST API service as a data source. REST API implements all logical operations on the data in the database.

KEYWORDS

web application, visualisation, database systems, API services, REST

BACHOREC, Jan. *Webová aplikace pro systém detekce rizikových situací na železničním přejezdu*. Brno, 2020, 64 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Zdeněk Martinásek, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Webová aplikace pro systém detekce rizikových situací na železničním přejezdu“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Zdeňkovi Martináskovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. Ing. Leoši Jiříkovi pak za přínosnou zpětnou vazbu z praxe.

Obsah

Seznam obrázků

Úvod

V posledních letech je patrný nárůst intenzity dopravy. Statistika vývoje objemu provozu na českých silnicích, jejíž autorem je Ministerstvo dopravy [1], ukazuje mezi lety 2010 a 2016 růst intenzity dopravy na většině typů komunikací o 13 %. Stejný zdroj pak uvádí, na základě dat z automatizovaného sčítání vozidel, zvýšení jejich množství mezi lety 2016 a 2020 o 5 % až 16 %. S rostoucím počtem vozidel na pozemních komunikacích souvisí i zvyšující se množství dopravních nehod, jak zmiňuje statistika [2].

Podle zprávy BESIP [3], oddělení Ministerstva dopravy ČR, zabývajícího se oblastí bezpečnosti na pozemních komunikacích, množství nehod na železničních přejezdech netvoří významnou část z pohledu absolutního počtu nehod. Avšak dopravní nehody na železničních přejezdech jsou dlouhodobě jedny z nejzávažnějších. Od roku 2006, se dle [3], nedaří významně snižovat počty usmrcených osob při těchto nehodách. Současně od roku 2009 neklesá ani absolutní počet takových nehod. Na konci roku 2018 bylo v České republice evidováno 7 858 železničních přejezdů, z nichž byla téměř polovina zabezpečena pouze výstražným křížem. Na těchto technicky nezabezpečených přejezdech došlo v témže roce k více než polovině z celkového počtu dopravních nehod na železničních přejezdech.

Z výše zmíněných statistik jasně vyplývá potřeba vytvoření dalších bezpečnostních mechanismů, které by pomohly snížit množství nebezpečných situací na pozemních komunikacích. A přispěly tak k redukci počtu dopravních nehod, ztrát na životech, zdraví a majetku.

Tato diplomová práce se zabývá tvorbou serverového řešení a grafického uživatelského rozhraní pro autonomní systém detekce rizikových situací na železničních přejezdech za účelem zvýšení bezpečnosti provozu a prevence dopravních nehod. Hlavním cílem diplomové práce je vytvoření návrhu a následná implementace serverového řešení, které bude přijímat, zpracovávat a ukládat data z jednotlivých autonomních kamerových modulů. A dále pak vývoj interaktivní webové stránky, jenž umožní operátorovi systému zaznamenaná data vhodně analyzovat. V teoretické části se práce zabývá technologiemi k tvorbě webového uživatelského rozhraní a API (Aplication Programing Interface) služeb, jako je Razor Pages, .NET Core. Zhodnoceny jsou i bezpečnostní aspekty. Z pohledu tvorby serverového řešení jsou teoreticky rozebrány různé databázové systémy, jako například Cassandra, ale i technologie pro zpracování a streamování dat.

1 Teoretická část diplomové práce

1.1 Technologie pro tvorbu aplikačního rozhraní

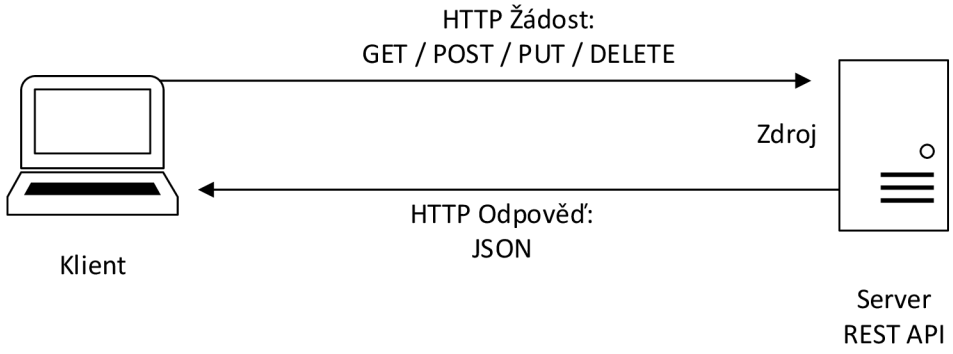
API (Application Programing Interface) definuje zdroj [4] jako specifikaci možností, jakými lze se softwarem komunikovat a využívat jeho funkcí. Dále API označuje i část softwarového komponentu, která komunikační rozhraní implementuje a s níž uživatel API přímo interaguje. Aplikační rozhraní slouží k vytvoření abstraktní úrovně, na které jsou složité úkoly řešeny jeho jednoduchým voláním. Z pohledu programátora, který chce nějakou z funkcionalit softwaru prostřednictvím API využít, není důležitá znalost konkrétní implementace dané funkcionality. Zajímá ho pouze, jakým způsobem ji použít a formát dat, která jsou funkcí přijímána. Případně, jestli jsou nějaké informace vraceny. A pokud ano, v jakém formátu.

Dle článku [4] je konceptu API využíváno delší dobu. Původně sloužil pro interakci programů pouze v rámci jednoho počítače. Avšak spolu s rozšířením počítačových sítí vyvstala potřeba vzájemného propojení aplikací mezi stanicemi a začalo docházet k propojení přes definované rozhraní s využitím komunikace po Internetu. S tím souvisí vznik veřejného API. Jedná se o komunikační rozhraní, které poskytuje funkcionalitu programu veřejně na Internetu. Tímto je umožněno vyvíjet programy, které v sobě díky využití konceptu API kombinují funkce více na sobě nezávislých poskytovatelů.

1.1.1 Architektura aplikačního rozhraní

REST (REpresentational State Transfer) je dle [5] styl architektury rozhraní pro komunikaci v rámci distribuovaných a modulárních systémů přes síť. Programy, které mají implementované API podle REST, se také nazývají RESTful systémy. V architektuře REST je implementace klienta a serveru na sobě vzájemně nezávislá. Je však nezbytné, aby obě komunikující strany podporovaly shodný formát zpráv. Jak uvádí [4], REST API pracuje se **zdroji** (resource). Každý zdroj na serveru je adresovatelný, má například přiřazenou URI (Uniform Resource Identifier). Schéma komunikace je pak následující. Klient pošle **žádost** (request) na **zdroj** na serveru, server přijme data v rámci žádosti, zpracuje je a vrátí klientovi **odpověď** (response). Další charakteristikou je bezstavovost. Server umožňuje pomocí standardních operací CRUD (Create, Read, Update, Delete) klientovi využívat zdroje, aniž by k tomu potřeboval znát vnitřní stav klienta. Veškerá potřebná data musí být obsažena v rámci každé operace se zdrojem.
V této práci bude podrobněji rozebrána implementace REST API pro webové aplikace. Takové RESTful aplikace používají pro výměnu dat obvykle protokoly HTTP

(Hypertext Transfer Protocol), HTTPS (Hypertext Transfer Protocol Secure) a jako formát dat JSON (JavaScript Object Notation). Obrázek 1.1 znázorňuje princip práce klienta se zdroji na serveru pomocí HTTP dotazů. Klient musí nejprve vy-



Obr. 1.1: Schéma REST API s použitím HTTP nebo HTTPS.

tvořit HTTP request. Obecně k tomu potřebuje vhodnou HTTP metodu, hlavičku, která obsahuje další informace o klientovi a URL (Uniform Resource Locator) zdroje. Základní HTTP metody, používané pro RESTful rozhraní jsou podle zdroje [5]:

- GET - získání a čtení dat ze specifikovaného zdroje
- POST - vytvoření nového zdroje určitého typu
- PUT - doplnění dat specifikovaného zdroje
- DELETE - smazání specifikovaného zdroje

HTTP hlavička umožňuje dle [6] klientovi a serveru vložit do HTTP žádostí a odpovědí dodatečné informace. Ty mohou sloužit k autentizaci, identifikaci uživatele nebo pro uložení informace o datových formátech, akceptovaných klientem. Skládá se z dvojic **název:hodnota**. URL je textový řetězec, který jednoznačně označuje umístění zdroje na serveru. Některé HTTP žádosti obsahují také data, obvykle ve formátu JSON.

Server žádost přijme, provede příslušné operace a odešle HTTP odpověď. Ta se skládá, podobně jako žádost, z hlavičky a případně dat. Pokud jsou data obsažena, musí hlavička disponovat polem content-type. To identifikuje formát dat v těle zprávy. Dále má HTTP response návratový status kód. Jedná se o tříčíselný kód, který vypovídá o výsledku práce se zdrojem. První číslice kódu určuje třídu, 2 značí úspěch, 4 a 5 poté chybu na straně serveru či klienta. Další číslice kódu poté upřesňují, k jaké situaci došlo. Pokud například klient použije úspěšně metodu HTTP GET, server mu odpoví kódem 200 (OK).

Webová stránka [7] definuje **JSON** jako formát pro výměnu dat, jehož charakteristickými vlastnostmi je snadná čitelnost člověkem, ale i jednoduché strojové zpracování a nízká režie. JSON vychází ze specifikací jazyka JavaScript. Avšak je kompletně nezávislý na programovacím jazyce, v rámci kterého je zpracováván. Tyto vlastnosti z něj činí ideální formát pro univerzální výměnu dat. Základní stavební kameny jsou dvojice název:hodnota. Ty mohou reprezentovat objekty a datové typy. A také uspořádaný seznam hodnot, jenž implementuje vektory, seznamy a další obdobné datové typy.

1.1.2 .NET Core

Jak uvádí Microsoft na stránce [8], jedná se o multiplatformní řešení pro vývoj mnoha druhů aplikací. Pro účely diplomové práce bude platforma využita k tvorbě webové aplikace a webového API. Jde o software open source, vydaný pod licencemi MIT a Apache 2. .NET Core podporuje 3 programovací jazyky, objektově orientovaný C#, funkcionální F# a Visual Basic. Vhodná vývojová prostředí jsou Visual Studio a Visual Studio Code od společnosti Microsoft.

Pro vývoj aplikací .NET Core musí být na stanici nainstalovaná sada vývojových nástrojů .NET Core SDK. Pro jejich spuštění pak ASP.NET Core Runtime. Aplikace je spouštěna následovně. ASP.NET Core Runtime obsahuje CLR (Common Language Runtime), virtuální stroj, sloužící jako běhové prostředí, které provádí kód aplikace a poskytuje jí potřebné služby. Aplikace napsaná například v jazyce C# je při kompilaci přeložena do nižšího programovacího jazyka, IL (Intermediate Language). Při jejím spuštění CLR vykonává JIT (Just In Time) překlad do strojového kódu. Ten je vykonáván procesorem. CLR také automaticky spravuje paměť. Obsahuje GC (Garbage Collector), starající se o alokování paměti z haldy v případě vytvoření nového objektu. Pokud halda neobsahuje dostatečné množství volných adres, GC smaže objekty, jenž v kódu nejsou dále použity a uvolní tak místo.

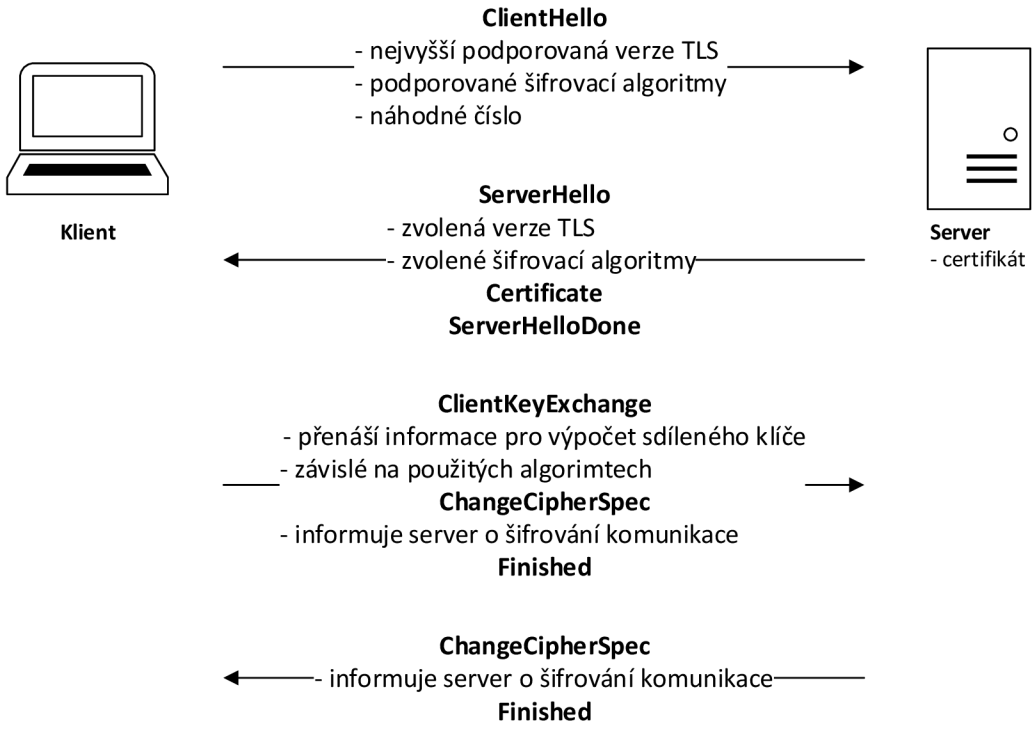
Pro snadné přidávání knihoven do .NET Core projektů slouží **NuGet**. Jedná se o open source balíčkovací systém. Každý balíček je archiv .zip, s příponou .nupkg, obsahující zkompilovaný kód příslušné knihovny a její popis, například informace o verzi.

Platforma .NET Core umožňuje tvořit RESTful služby za pomoci jazyka C#. Pro obsluhu jednotlivých žádostí (request) jsou použity kontrollery. Knihovna ASP.NET Core obsahuje namespace **Microsoft.AspNetCore.Mvc**. Ten zajišťuje potřebné atributy a třídy, pomocí nichž lze web API implementovat.

1.1.3 Bezpečnost API

Nezisková organizace OWASP, zabývající se zlepšováním bezpečnosti softwaru, ve svých dokumentech uvádí následující techniky a postupy pro zajištění bezpečnosti v rámci REST API [9].

Za účelem zachování důvěrnosti, integrity a autentičnosti komunikace mezi serverem a klientem je nezbytné používat bezpečný komunikační protokol, například **HTTPS**. Jedná se o šifrovanou variantu protokolu HTTP, rozšířenou o **TLS** (Transport Layer Security). Jak uvádí zdroj [10], jde o bezpečnostní protokol aplikační vrstvy komunikačního modelu TCP/IP. Komunikace s použitím TLS má dvě fáze. První fází je TLS handshake, proces výměny autentizačních dat, dohodnutí použitých kryptografických algoritmů a tvorby sdílených klíčů pro symetrické šifrování samotné komunikace. Jeho schéma je na obrázku 1.2.



Obr. 1.2: Proces handshake v rámci protokolu TLS.

Pro autentizaci komunikujících stran tento protokol pracuje s certifikáty, tedy elektronickými dokumenty, obsahujícími informace o identitě a veřejném šifrovacím klíči svého vlastníka. Certifikáty navíc obsahují i informace o certifikační autoritě, která je vydala. V rámci TLS obvykle dochází pouze k autentizaci serveru.

Samotný certifikát musí být vydaný důvěryhodnou certifikační autoritou. Ve speciálních případech může být server konfigurován tak, aby vyžadoval i autentizaci klienta. V druhé fázi, po úspěšném handshake, následuje samotná šifrovaná komunikace. HTTPS využívá obvykle port 443 protokolu TCP a aktuální specifikace z roku 2018 popisuje TLS ve verzi 1.3.

Dalším důležitým aspektem bezpečnosti REST API je proces řízení přístupu ke zdrojům serveru. Zajištění autentizace a autorizace je samozřejmě nezbytné zavést ve chvíli, kdy není žádoucí, aby se zdroji serveru pracoval libovolný neoprávněný klient. Zde nacházejí uplatnění bezpečnostní tokeny. Dle zdroje [9] se konkrétně standard **JWT** (JSON Web Token), definovaný v RFC 7519, stává velmi rozšířeným v této oblasti. Takový token obsahuje množinu takzvaných claims (tvrzení). Jde o soubor informací o entitě, která tokenem disponuje. Součástí tokenu je i jeho časová platnost. Server na základě těchto informací provádí řízení přístupu ke svým zdrojům. Aby nebylo možné token podvrhnout, je použito různých kryptografických algoritmů. S pomocí nich je zajištěna integrita dat tokenu. Server vydávající token má tajný klíč, kterým vydaný token podepisuje či vytváří hash. Když se klient snaží přistoupit ke zdroji serveru, zašle mu svůj token, server ověří tvrzení přítomná v tokenu a jeho časovou platnost, validuje integritu a v případě, že je popsaný proces úspěšný a klient je oprávněný nakládat s požadovaným zdrojem, je mu přístup ke zdroji umožněn. Každý server, který disponuje tajným klíčem, použitým pro podpis tokenu, může token validovat a také může vydávat nové tokeny. Pro zajištění integrity tokenu je používán například algoritmus SHA 512.

Pak je zde ale případ veřejného API. I zde je vhodné nějakým způsobem kontrolovat využití zdrojů, a to z důvodů prevence přetížení serveru. Text [9] pro tyto případy doporučuje implementovat podporu API klíčů. Takové klíče jsou identifikátory klienta, přistupujícího ke zdroji. Server je může vydávat a současně limitovat například maximální využití svých zdrojů pro každý konkrétní unikátní klíč.

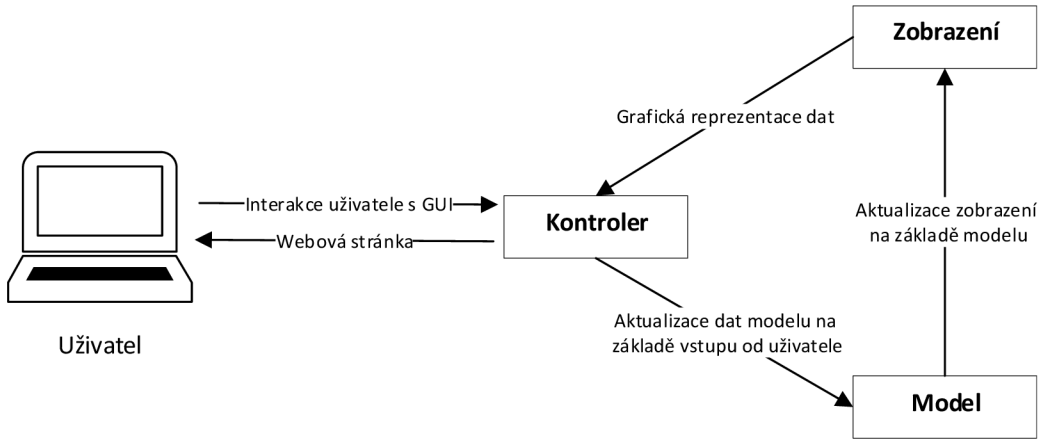
Text [9] uvádí jako další techniky pro zajištění bezpečnosti například validaci vstupů či obsahu zpráv vyměňovaných mezi serverem a klientem. Je třeba kontrolovat datové typy i velikosti zasílaných dat. Hesla nesmí být přenášena v čitelné podobě. Preferuje se využití otisků hesel, generovaných s pomocí bezpečných hashovacích funkcí, jako například SHA-512. Ošetření chyb v rámci API musí být řešeno tak, aby nedocházelo k únikům citlivých informací. API by v případě chyby v rámci práce se zdrojem mělo vracet takové chybové hlášky, které neobsahují přílišné podrobnosti, odhalující vnitřní implementaci zdrojů serveru.

1.2 Technologie pro tvorbu interaktivní webové stránky

Interaktivní webová stránka tvoří grafické uživatelské rozhraní pro systém detekce rizikových situací na železničním přejezdu. Základním požadavkem na tuto stránku je, aby umožnila vhodnou analýzu rizikových situací, zaznamenaných kamerovými moduly. K vývoji této webové aplikace tak musí být použity technologie, které umožní nejen prezentaci textových dat o jednotlivých událostech, ale i multimediálních záznamů a různých grafů pro lepší vyhodnocení incidentů a stavů kamer uživatelem.

I zde se, podobně jako pro serverovou část aplikace, počítá s využitím open source frameworku .NET Core. A to z důvodu snazší údržby díky sjednocení technologií v rámci vytvářených aplikací. Zdroj [11] rozlišuje tradiční a jednostránkové webové aplikace, nazývané také zkratkou SPA (Single Page Application). **Tradiční webové aplikace** jsou charakteristické zpracováním interakcí s uživatelem na straně serveru. Jejich nasazení je vhodné v případech, kdy stránka slouží zejména k prezentaci dat a nebo pokud není možné v klientském prohlížeči spouštět kód v jazyce JavaScript. Mezi technologie pro tvorbu tradičních webových aplikací patří například PHP.

Jako zástupce, založeného na .NET Core, lze zmínit například ASP.NET Core MVC. Tato technologie implementuje návrhový vzor MVC (Model View Controller), tedy model, zobrazení a kontroler. Jeho schéma je na Obr. 1.3. Hlavní myšlenkou popisovaného přístupu je rozdělení aplikací do tří základních komponent. Model reprezentuje data v aplikaci a zajišťuje jejich validaci. Zobrazení je odpovědné za grafickou reprezentaci dat v rámci modelu. Jedná se dynamicky generované HTML stránky. Kontroler je pak prvek, který obsluhuje žádosti od klientů předávané pomocí protokolu HTTP/HTTPS. Dalším zástupcem je i technologie Razor Pages.



Obr. 1.3: Schéma návrhového vzoru MVC.

Hlavní charakteristikou **jednostránkových aplikací SPA** je zpracování uživatelských interakcí v rámci prohlížeče klienta. Tyto aplikace, jejichž zdrojový kód je většinou psaný v jazyce JavaScript, jsou přímo spouštěny v internetovém prohlížeči. Pokud například dochází ke změnám grafického rozhraní webové stránky na základě interakce ze strany uživatele, je stránka renderována přímo v jeho prohlížeči. Pro práci s daty či jiné akce, které není možné v prohlížeči vykonat, tyto aplikace komunikují na pozadí prostřednictvím asynchronního volání API serveru, implementujícího požadované operace. Mezi SPA technologie patří třeba **Angular**. Dle zdroje [11] se jedná o jeden z nejpopulárnějších JavaScript frameworků. Nevýhodou takových aplikací je v případě, kdy je backend implementován například s pomocí .NET Core, nutnost využití jiného programovacího jazyka a jiné technologie pro tvorbu uživatelského rozhraní. V současnosti vzniká v rámci frameworku .NET Core technologie Blazor, která má umožnit s pomocí jazyka C# vytvářet SPA. Avšak pro účely této práce je preferováno využití v praxi více rozšířené technologie.

Jak uvádí zdroj [12], **Razor Pages** jsou technologií pro tvorbu webových aplikací, generovaných na straně serveru. Jedná se o součást .NET Core, popsaného v předchozí kapitole. Razor Pages implementují návrhový vzor MVC, avšak oproti výše zmíněnému ASP.NET Core, zjednodušují a zpřehledňují vývoj webové aplikace. Díky těmto vlastnostem byly Razor Pages vybrány pro účely této práce. Technologie je označována jako page-focused [1], v rámci vývoje jsou společně soubory definující jak grafický vzhled, tak chování stránky, drženy pohromadě. V rámci této technologie se pracuje se stránkami (page). Jde o soubory sloužící jako šablony výsledných HTML stránek. S pomocí Razor symbolů a jazyka C# tyto šablony generují HTML kód dynamicky na základě vstupů. Dále zde existují modely stránek (page model). V těchto souborech je implementována logika zpracování dat. Razor Pages obsahují různé formuláře a další prvky uživatelského rozhraní. Podporovány jsou možnosti navigace mezi stránkami a použití parametrů v rámci navigace. Dále je s nimi možné implementovat například streamování multimediálního obsahu. Pro tvorbu grafů z dat je vhodné použít spíše JavaScript knihovnu.

Bootstrap je dle zdroje [13] open source CSS (Cascading Style Sheets) framework, použitý v rámci Razor Pages pro tvorbu konzistentní podoby grafického uživatelského rozhraní. Software byl vyvinut v roce 2012 autory Mark Otto a Jacob Thornton. Bootstrap jednak obsahuje předdefinované styly pro jednotlivé prvky uživatelského rozhraní, s pomocí nichž lze snadněji dosáhnout jednotného designu webové aplikace. Dále implementuje pokročilé možnosti responzivního rozložení HTML stránky za použití kontejnerů, řad a sloupců. Díky tomu se vytvořené webové stránky automaticky přizpůsobují různým velikostem a rozlišením displejů zařízení.

[1] Zaměřena na webovou stránku.

Bezpečnost v Razor Pages lze rozdělit na bezpečnost komunikace mezi serverem a klientem, a na zajištění řízení přístupu. Pro zajištění autentičnosti, integrity a důvěrnosti zpráv zasílaných mezi serverem a klientem je nezbytné využít šifrovaný protokol HTTPS, popsaný v předchozí kapitole. V rámci řízení přístupu popisovaná technologie podporuje autentizaci i autorizaci. K autentizaci lze použít ASP.NET Identity. Jde o komplexní modul v rámci .NET Core, umožňující implementaci přihlašování uživatelů s pomocí účtů služeb třetích stran, jako je Microsoft či Google účet. Avšak pro menší aplikace, které takto komplexní řešení nevyžadují, lze použít autentizaci na základě HTTP Cookies. Jedná se o malé soubory, které může server uložit do prohlížeče klienta. Do těchto Cookies server ukládá informace o uživateli, době platnosti a přiřazených zásadách. Právě s pomocí těchto zásad je i definována autorizace. Pro přístup na jednotlivé stránky aplikace se nastavují vyžadované zásady.

1.3 Technologie pro streamování a zpracování dat

Streamování událostí je dokumentací projektu Apache Kafka [14] definováno jako zachytávání dat z různých zdrojů v reálném čase a jejich spolehlivé ukládání pro pozdější použití. Směrování streamů událostí mezi jednotlivými aplikacemi, zajišťujícími zpracování dat. Zdrojem události může být senzor, služba či nějaká aplikace. Streamovací aplikace zajišťuje správný a souvislý tok událostí mezi jejich producenty a konzumenty.

Praktické uplatnění nachází streamování událostí v oblastech, kde je kladen důraz na průběžné zpracování dat v reálném čase. Jde například o systémy pro monitoring logů, dohled nad dopravou, průmyslem. Pro účely diplomové práce bylo zvažováno použití nástroje Apache Spark a Apache Kafka ke zpracování přijímaných dat z dopravních kamer v reálném čase. Nicméně se během tvorby návrhu serverového řešení ukázalo, že veškeré výpočetně náročné operace zpracování dat realizuje přímo kamerový modul, a proto není využití těchto nástrojů nutné.

1.3.1 Apache Kafka

Jedná se o kompletní platformu pro streamování událostí. Projekt je open source a dostupný pod licencí Apache License 2.0. Kafka implementuje tři klíčové vlastnosti, které umožňují realizovat steamování mezi jednotlivými prvky systému.

- Čtení a zápis do streamů událostí.
- Spolehlivé uložení dat streamů po neomezeně dlouhou dobu - limitací je pouze dostupné místo na disku.
- Umožnění zpracování dat ve streamech v reálném čase nebo zpětně.

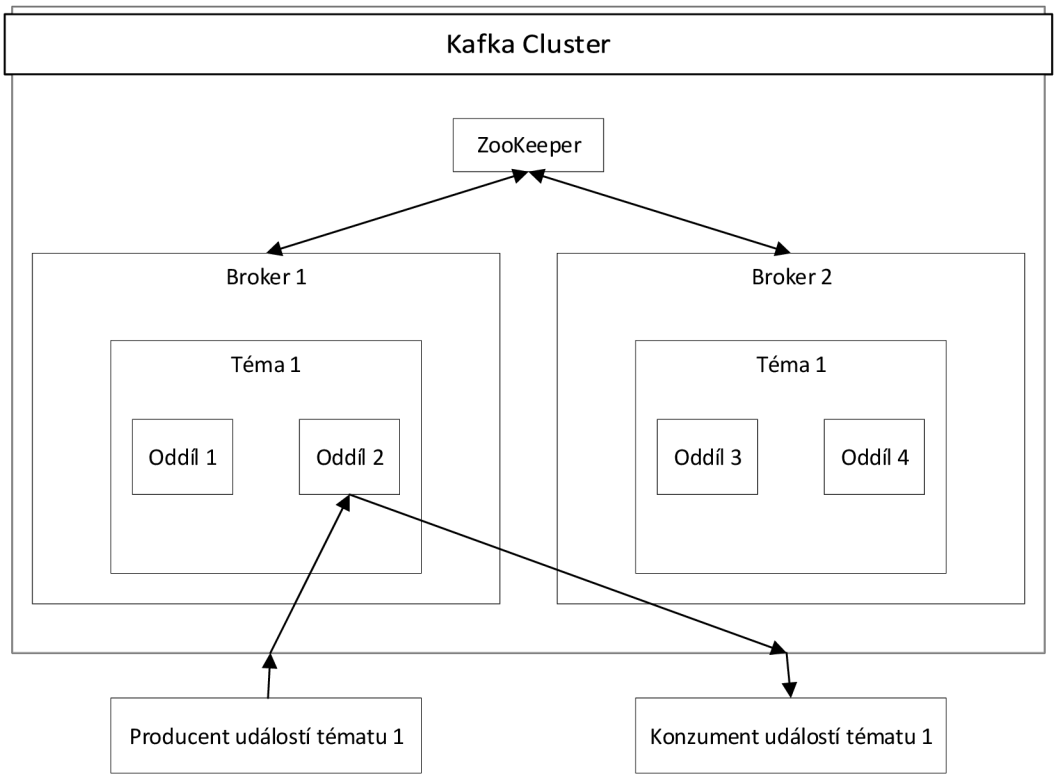
Systém Apache Kafka je podle [15] distribuovaný. Celý cluster se skládá z několika komponent. **Kafka Brokers** [2] jsou servery pro ukládání dat streamů. Mezi ně je rozložena zátěž, každý z Brokers je zodpovědný za určitou část dat. Kafka může fungovat i s pouze jedním Brokerem, avšak v takovém případě nelze zajistit redundanci v případě výpadku. Pro zajištění spolehlivosti je nezbytné provozovat alespoň 3 Kafka Brokers. **ZooKeeper** je server v rámci Kafka clusteru, který slouží jako koordinátor celého clusteru. Jeho úkolem je informovat Brokers v případě změny topologie clusteru a koordinovat uzly clusteru.

Základní datovou jednotkou systému je **event** (událost). Jedná se o záznam o nějaké skutečnosti s přiřazeným časovým razítkem. Událost se skládá z klíče události, neboli jednoznačného identifikátoru události. Hodnoty, informace nesené událostí a časové značky. Spolu související události jsou v Kafka Clusteru seskupovány do **topics** (téma). Každé téma může mít libovolný počet producentů i konzumentů. To znamená, že do něj může současně zapisovat více entit. Pro čtení platí totéž. Současně neexistuje žádné omezení počtu čtení událostí, po přečtení události konzumentem tato událost v systému zůstává. Nedochází k jejímu smazání. Kafka umožňuje nastavit, jak dlouho budou události uchovávány. Po naplnění této podmínky dochází k odstranění starých událostí.

Jednotlivá témata jsou rozdělena na menší celky, **partitions** (oddíly). Takto je zajištěna distribuce rozložení událostí v tématech mezi uzly clusteru. Každý broker má na svém úložišti jeden či více oddílů, patřících pod různá témata. Oddíl je dále nedělitelný, na disku brokeru se nachází vždy celý. Současně může být jeden oddíl ve více kopiích fyzicky umístěn na více uzlů clusteru, a to za účelem zajištění spolehlivosti prostřednictvím redundance. Pro reálné systémy se doporučuje replikační faktor 3, tedy uchovávání tří kopií oddílu na třech různých uzlech clusteru.

Pro rozhodování, do jakého oddílu Kafka událost uloží, slouží algoritmus, který na základě klíče události vytvoří hash s pomocí speciální hashovací funkce. Ta slouží k namapování klíče na konkrétní oddíl. Události s identickým klíčem v rámci stejného tématu tak budou uloženy v jednom oddílu. A to v pořadí, odpovídajícím posloupnosti, v níž docházelo k jejich zápisu. Události v oddílu jsou identifikované pomocí pořadového čísla, offset, a jsou přidávány vždy na poslední pozici oddílu. Základní principy fungování Kafka clusteru jsou na obrázku 1.4.

[2] Kafka zprostředkovatelské servery, v textu je dále použit originální název Kafka Brokers.



Obr. 1.4: Princip směrování událostí mezi jejich konzumenty a producenty pomocí Kafka Clusteru.

1.3.2 Confluent Platform Community

Confluent Platform je kompletní řešení pro streamování událostí, postavené na Apache Kafka. Podle oficiální dokumentace [16] se jedná o balík aplikací, pomocí nichž lze pohodlněji integrovat systém Kafka do konkrétních streamingových projektů. Každý balíček Confluent obsahuje nejnovější dostupnou verzi Apache Kafka a množinu nástrojů pro její lepší intergaci a správu.

Existují dvě edice, přičemž Confluent Commercial je placená a oproti Conflent Community zahrnuje navíc nástroje jako Confluent Control Center, grafické uživatelské rozhraní pro správu Kafka clusteru. Nebo Confluent Auto Data Balancer. Nástroj pro pokročilé automatické rozložení zátěže v clusteru. Komerční verze Confluent mimo jiné nabízí i bezpečnostní rozšíření.

Avšak i zdarma dostupná komunitní verze balíčku Confluent obsahuje mnohé užitečné nástroje pro práci s Apache Kafka. **ksqlDB** je rozšíření, s jehož pomocí lze zpracovávat streamy událostí prostřednictvím skriptů jazyka SQL. Lze tak bez nutnosti implementace dalších programů data filtrovat, provádět jejich agregaci či vytvářet nové vazby mezi událostmi. Confluent Community dále obsahuje sadu soft-

warových knihoven, určených k práci s událostmi a jejich streamy v Kafce. Podporovanými jazyky, v nichž lze s použitím těchto knihoven vytvářet klienty, jsou C/C++, Python, Go a .NET. Knihovny obsahují metody pro posílání zpráv do Kafky, jejich konzumování. Dále je dostupná podpora serializace dat událostí, například do formátu AVRO. Správa AVRO schémat je realizována pomocí Confluent Schema Registry. Tento modul slouží jako úložiště schémat událostí jednotlivých témat a umožňuje verzování dostupných schémat.

Apache AVRO je systém pro serializaci dat **AVRO** [17]. Slouží k převedení datového objektu do binární podoby. A to za účelem uložení dat do paměti nebo pro jejich přenos po síti. AVRO využívá k definici struktury zpráv schémata. Ta jsou definována ve formátu JSON a musí být přítomna při zápisu i čtení AVRO dat. Samotná data pak neobsahují žádné hlavičky, definující datové typy hodnot. Výsledkem je minimalizace přenosu režijních dat a tím vyšší rychlost.

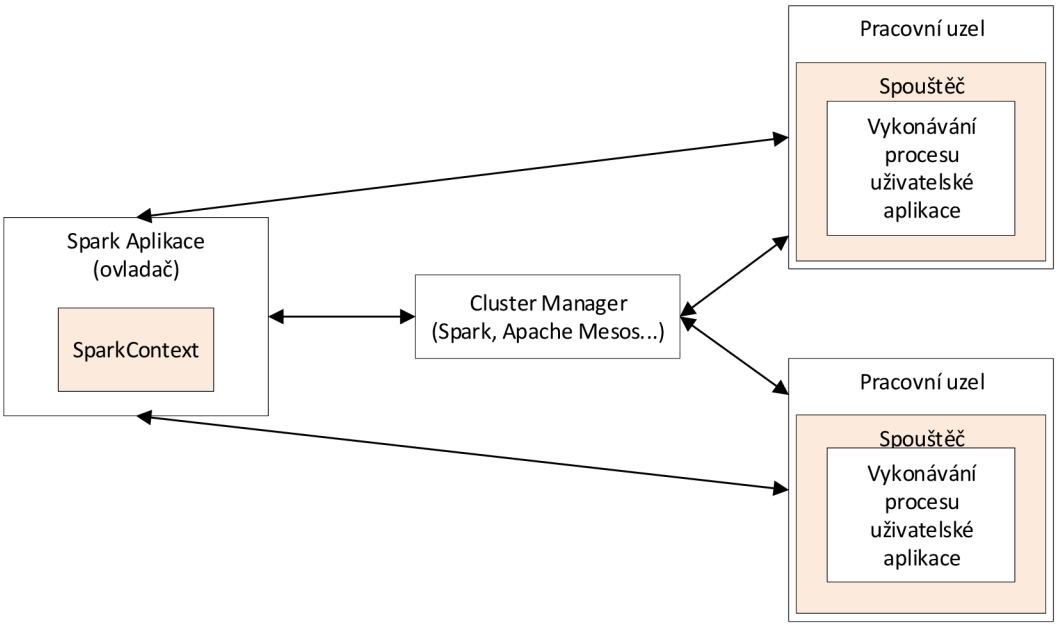
1.3.3 Apache Spark

Apache Spark je nástroj pro distribuované zpracování velkých objemů dat v clusteru [18]. Jde o open source software, licencovaný pod Apache License 2.0 a vytvořený v jazyce Scala. Původním autorem projektu je Matei Zaharia.

Projekt Apache Spark se skládá z několika částí. Základní komponentou je jádro systému, clusterový výpočetní engine. Jedná se o množinu uzlů tvořených stroji s nainstalovaným Apache Spark. Úkolem tohoto softwaru je spouštění distribuovaných výpočtů. Jeho architektura je master-slave. Roli mastera plní **cluster manager**[3]. Dle oficiální dokumentace [18] je možné cluster manager v současnosti realizovat pomocí uživatelsky přívětivého integrovaného řešení v rámci Spark, případně pomocí externího cluster manageru. Například Apache Mesos či Hadoop YARN. Úkolem cluster manageru je alokování zdrojů na slave uzlech zvaných **worker nodes** (pracovní uzly).

Uživatelská aplikace psaná pro Apache Spark obsahuje vždy ovladač a v rámci něj objekt **SparkContext**. Ten má za úkol se připojit na kompatibilní cluster manager. Poté, co se tak úspěšně stane, cluster manager alokuje jednotlivé výpočetní kapacity na pracovních uzlech pro běh kódu uživatelského programu. Dokumentace je označuje jako executors (spouštěč). Dalším krokem je odeslání zdrojových kódů aplikace všem zapojeným executors. Každý executor má vlastní JAVA virtuální stroj, tudíž jsou Spark aplikace od sebe kompletně oddělené. V případě potřeby sdílení dat mezi různými Spark aplikacemi tak musí dojít k výměně dat přes nějaké externí úložiště. Popisovaný princip fungování Spark Aplikace je na obr. 1.5.

[3]Správce clusteru, dále bude v textu uváděn originálním anglickým názvem.



Obr. 1.5: Princip fungování Apache Spark clusteru.

Spark Streaming je rozšíření jádra platformy Spark, umožňující zpracování streamů událostí. Charakteristickými vlastnostmi [18] jsou odolnost vůči chybám a vysoká propustnost. Zdrojem dat událostí může být například streamovací platforma Apache Kafka. Zpracovaná data lze ukládat do databáze. Uplatnění nachází Spark Streaming také v obblasti strojového učení.

1.4 Databázové systémy

Zdroj [19] definuje databázi jako organizovanou sbírku dat, typicky v elektronické podobě. Správu databáze zajišťuje systém pro správu databáze DBMS (Database Management System). Pojem databázový systém pak označuje DBMS spolu s uloženými daty. Jak uvádí [19], existuje několik základních typů databázových systémů. Nelze říci, že by nějaký z nich byl horší, nebo lepší. Vždy záleží, jaká data budou do databáze ukládána a jakým způsobem s nimi bude nakládáno.

1.4.1 Přehled typů databázových systémů

Relační databáze začaly být masivně nasazovány od 80. let minulého století. Záznamy v nich jsou organizovány do množiny tabulek. Každá z tabulek pak obsahuje sloupce hodnot. Základní strukturu relační databáze tvoří relační model. Ten vyjadřuje vzájemné vztahy mezi daty v tabulkách. Funguje na principu primárních

a cizích klíčů. Jednoznačný identifikátor řádku tabulky je primární klíč. Je tvořen unikátními hodnotami jednoho či více sloupců dané tabulky. Cizí klíč pak reprezentuje nějaký sloupec tabulky, v němž obsažené hodnoty současně odkazují na primární klíč jiné tabulky. Tím udržují vzájemný vztah těchto dvou tabulek. Databázový systém zajišťuje integritu dat v databázi na základě systému relačního modelu a klíčů. Pro práci s daty v rámci téměř všech relačních databázích je používán programovací jazyk SQL (Structured Query Language). Byl vyvinut v 70. letech společnostmi IBM a Oracle. V současnosti implementují jednotlivé systémy rozšířené verze, oproti SQL ANSI. Například Microsoft SQL Server využívá Transact-SQL a Oracle Database zase PL/SQL. Mezi nejznámější relační databázové systémy patří Microsoft SQL Server, open source řešení MySQL a Oracle Database.

Objektově orientované databáze reprezentují data a vazby mezi nimi prostřednictvím objektů. Používají podobné principy jako objektově orientované programování. Mezi objekty mohou být vztahy. V takovém případě se zde uplatňují některé z principů relačních databází.

Distribuované databáze data ukládají do dvou či více souborů. Tyto soubory mohou být fyzicky uloženy na nezávislých úložištích. Toto řešení slouží k prevenci selhání prostřednictvím redundance. Nebo k rozložení zátěže.

Datové sklady slouží jako centralizovaná úložiště pro velká množství dat. Podle webové stránky [20] mohou obsahovat agregované informace z více zdrojů. Jsou určeny k dotazování se na data za účelem jejich analýzy. Jejich základní části jsou relační databáze pro uložení záznamů. Dále pak aplikace, tvořící nástavbu, sloužící ke čtení a zápisu dat do skladu. Nástroje pro statistickou analýzu a vizualizaci.

NoSQL (Not Only SQL) definuje zdroj [21] jako označení pro skupinu databázových systémů, pro něž je charakteristická velká variabilita datových modelů. Jak uvádí [22], NoSQL systémy jsou nasazovány v případech, kdy dochází ke zpracování velkých objemů dat s požadavkem na nízké zpoždění a flexibilitu datového modelu. Těchto vlastností je dosaženo snížením některých požadavků na konzistenci a integritu dat, typických například pro relační databáze. Základní typy NoSQL systémů jsou podle [21] následující:

- **Klíč-hodnota** je NoSQL systém tvořený jednoduchým datovým modelem, založeným na párování unikátních klíčů a hodnot. Důsledkem jednoduchosti popisovaného modelu je vysoký výkon a škálovatelnost. Databáze tohoto typu bývají používány pro caching dat webových aplikací. Některé z nich jsou implementované s důrazem na využití paměti RAM (Random Access Memory), jiné pak zase na operace s pevnými disky. Dle [22] je databáze tohoto typu, konkrétně DynamoDB společnosti Amazon, používaná sociální sítí Snapchat pro ukládání dat v rámci funkcionality Snapchat Stories. Dalšími příklady NoSQL systémů jsou Berkeley DB a MemchacheDB.

- **Dokumentové databáze** uchovávají strukturovaná data ve formě dokumentů. Nejčastěji jsou tyto dokumenty ve formátu JSON. Dále se používá i strukturovaný formát XML (eXtensible Markup Language). Výhodou takových databází je možnost provádění změn ve struktuře dat bez nutnosti zásahu do hlavního datového modelu, čímž je usnadněn vývoj nových verzí aplikací. Dále systém umožňuje vývojářům v rámci databáze pracovat s datovým modelem odpovídajícím objektům v programu. Popisovaný přístup je hojně používám v aplikacích, ve kterých jsou dokumenty v databázi měněny v čase. Vhodný je například pro reprezentaci uživatelských profilů. Jako zástupce dokumentové databáze lze uvést MongoDB, multiplatformní open source řešení.
- **Wide-column databáze** je dle [23] jednou z kategorií NoSQL databáze. Mezi charakteristické vlastnosti patří skvělý výkon v případě ukládání velkého množství dat a datový model založený na tabulkách a řádcích. Avšak oproti relačním databázím nemusí mít každý řádek stejné sloupce. Vhodným scénářem pro nasazení wide-column systému je ukládání logů a reportů. Mezi zástupce wide-column databáze patří Apache Cassandra.
- **Grafové databáze** reprezentují data jako uzly grafu. Hrany znázorňují souvislosti mezi daty. Oproti relačním systémům je však možné v průběhu času měnit datový model prostřednictvím modifikace propojení jednotlivých datových uzlů. Grafové databáze nachází uplatnění v aplikacích, které pracují se vztahy mezi daty. Jako příklad lze uvést rezervační systém, který přiřazuje volná místa klientům. Mezi zástupce řadí [21] systém Graph od společnosti IBM nebo AllegroGraph.

1.4.2 Výběr databáze pro diplomovou práci

V rámci této práce budou do databáze v reálném čase ukládána data z autonomních kamerových systémů. Pro testování se počítá se zapojením pouze několika modulů. Avšak v reálném provozu může být množství sledovaných kamer daleko vyšší. Proto je důležité vybrat vhodný databázový systém, který umožní dostatečnou rozšiřitelnost serverové aplikace. Současně je nezbytné při výběru databázového systému zohlednit i potřebu statistického zpracování dat uložených na serveru. Jak bylo popsáno výše, pro zápis velkého množství dat v reálném čase je vhodný NoSQL databázový systém typu wide-column. Jako zástupce této kategorie, který nejlépe vyhovuje potřebám diplomové práce, byl zvolen systém Apache Cassandra. A to z důvodu, že se jedná spolehlivou a velice výkonou distribuovanou databázi. Navíc jde o podporovaný open source projekt. Aby bylo nad daty z kamer možné provádět i složitější

analytické operace, bude v projektu použita distribuce Cassandra v rámci balíčku Elassandra, který kombinuje právě databázi Cassandra spolu s vyhledávacím nástrojem Elasticsearch. Opět se jedná o open source projekt, který je dostupný pod licencí Apache verze 2.0.

1.4.3 Databáze Cassandra

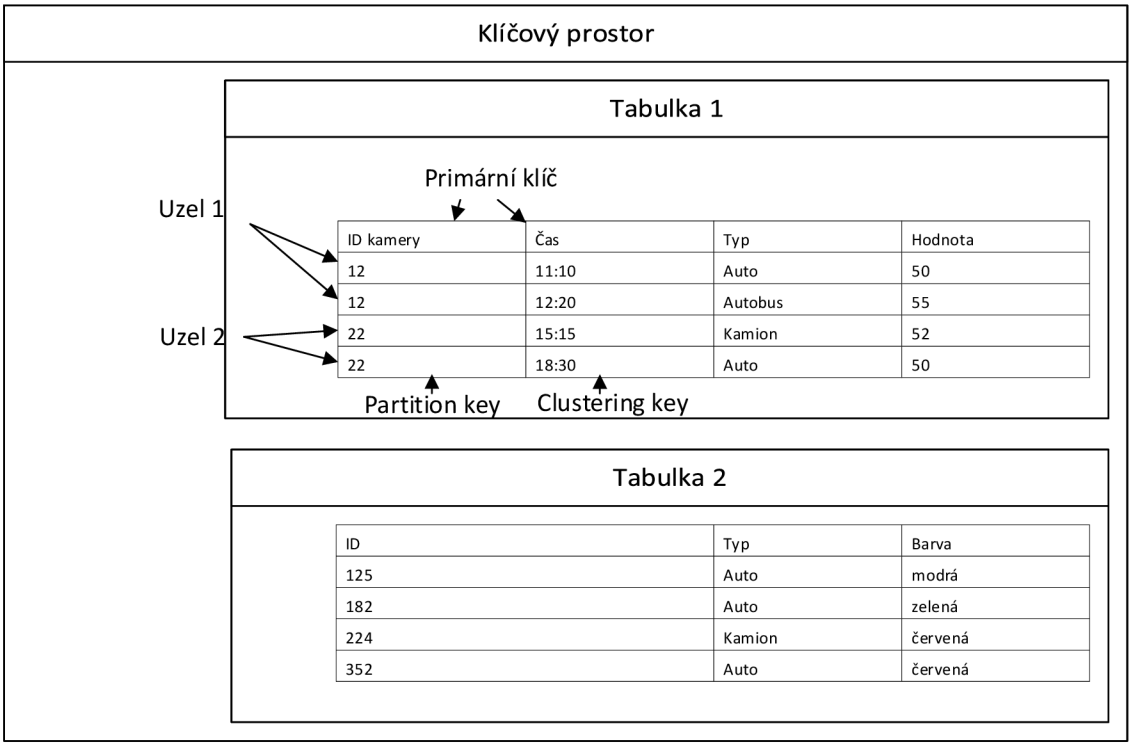
Apache Cassandra je dle zdroje [24] open source projekt distribuovaného NoSQL databázového systému. Původně vyvinutý společností Facebook, avšak v roce 2008 uvolněný veřejnosti a v současnosti dostupný pod licencí Apache License 2.0. Cassandra byla navržena tak, aby kombinovala vlastnosti databáze Amazon Dynamo s vlastnostmi datového modelu systému Google Bigtable. Splňuje požadavky na vysokou výkonnost a spolehlivost při velkém objemu uložených dat i dotazů na data.

Distribuovaný databázový systém Cassandra obsahuje dle [25] následující komponenty. Cassandra cluster, datacentrum a uzel neboli node. Uzel je nejzákladnější jednotka systému, která slouží k ukládání dat. Lze si ho představit jako fyzický server s běžící službou Apache Cassandra. Ve skutečnosti na Cassandra serverech běží více virtuálních uzlů, které se označují jako vnode. Defaultní počet vnode na jednu instalaci Cassandra je 256. Všechny uzly jsou vzájemně rovnocenné a na každý z nich se lze připojit a přistupovat přes něj k datům celého DB (databáze) systému. Datacentrum je množina, například geograficky oddělených, uzlů. Cluster se skládá z veškerých uzlů daného databázového systému.

Základní prvky datového modelu Cassandra jsou keyspace (klíčový prostor), table (tabulka), partition (oddíl), row (řada) a column (sloupec). Obrázek 1.6 znázorňuje princip datového modelu popisovaného systému. **Klíčový prostor** je v hierarchii modelu nejvýše, plní roli databáze ve smyslu nezávislé organizované množiny dat. Jednotlivé klíčové prostory jsou v rámci Cassandra clusteru kompletně oddělené. Nad každým klíčovým prostorem je definováno několik vlastností. Jedná se například o replikační faktor, číslo vyjadřující, na kolika uzlech jsou data zálohovaná. Klíčový prostor se skládá z jednotlivých **tabulek**. Ty jsou definovány schématem **sloupců**, a datovými typy. Tabulky obsahují **řádky**, ve kterých jsou uložena samotná data, a to ve sloupcích.

Každý řádek dat je jednoznačně identifikován prostřednictvím **primary key**[4]. Ten může být tvořen pouze partition key[5] nebo složen z partition key a clustering key[6]. V druhém případě se nazývá compound primary key. Roli partition key může plnit jeden či více sloupců tabulky. Compound partition key, partition key tvořený

[4]Primární klíč, v textu je dále použit originální název.
[5]Klíč oddílu, v textu je dále použit originální název.
[6]Shlukovací klíč, v textu je dále použit originální název.

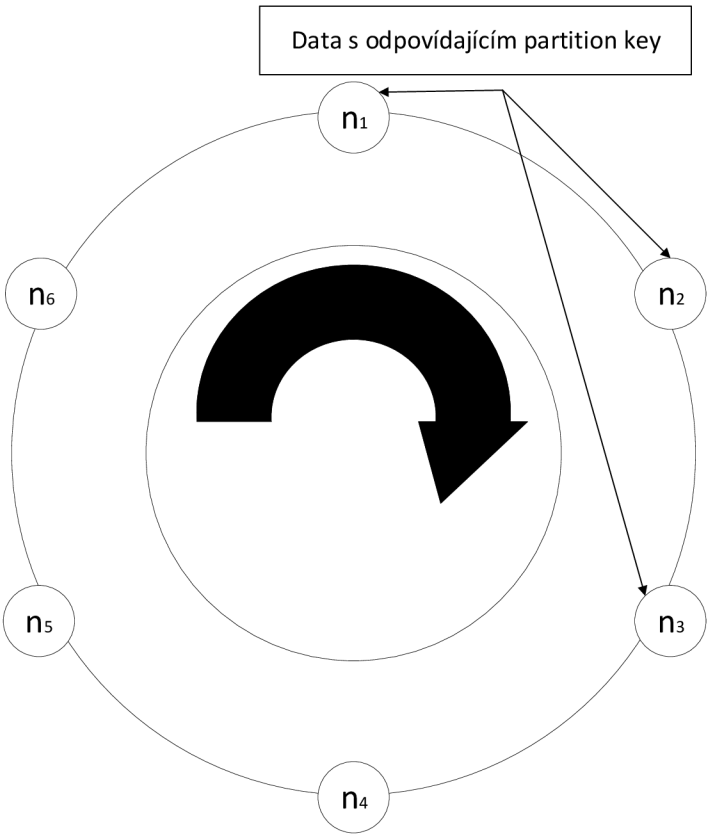


Obr. 1.6: Základní schéma datového modelu Apache Cassandra.

více sloupci tabulky, se zavádí zejména v případech, kdy by s jednoduchým partition key nešlo data efektivně rozdělit mezi jednotlivé oddíly. Data jsou na uzlech clusteru uložena v oddílech. Jednotlivé oddíly bývají rozloženy ve více kopiích na různých uzlech clusteru. Jejich identifikátorem je partition key. Algoritmus Cassandry rozhoduje o tom, do jakého oddílu řádek dat přidá právě na základě něj. Princip distribuce dat mezi uzly a oddíly je podrobněji popsán v následujícím odstavci. Každý oddíl obsahuje vždy řádky s odpovídajícím partition key. Clustering key je reprezentován hodnotami jednoho či více sloupců tabulky. Na základě jeho hodnoty dochází k řazení řádků v jednotlivých oddílech.

Dokumentace Apache Cassandra [24] uvádí následující principy, díky nimž systém provádí rovnoměrné rozložení zátěže v rámci clusteru, zajišťuje horizontální škálovatelnost a předchází ztrátě dat v případě výpadku nějakého uzlu. Data jsou rozdělena mezi uzly na základě partition key. Všechny uzly vytváří kruhovou topologii. Současně existuje hashovací tabulka, jejíž hashe odpovídají uzlům systému. Algoritmus přiřazování dat do oddílů vytváří každému záznamu, vloženému do databáze, hash s pomocí konzistentní hashovací funkce. Tato hashovací funkce zajišťuje, že se v případě rozšíření tabulky všech možných hashů v důsledku změny počtu uzlů, změní mapování hodnoty na hash jen pro malou část hodnot. Na základě hashe je

nalezen odpovídající uzel a konkrétní oddíl, do kterého jsou data vložena. Současně dojde k uložení dat i na další uzly, které jsou v topologii po směru hodinových ručiček. Princip je zobrazen na obr. 1.7. Tím je zajištěna redundance pro případ výpadku uzlu. Pro lepší rozložení zátěže v systémech s menším množstvím fyzických uzlů byl vytvořen koncept vnodes. Jak bylo popsáno výše, data se rozdělují mezi uzly na základě hashe. Aby bylo rozdělení rovnoměrnější, má každý uzel několik vnodes a v kruhové distribuční topologii tak vystupuje jako několik uzlů.



Obr. 1.7: Princip distribuce dat mezi oddíly na jednotlivých uzlech clusteru.

Cassandra implementuje dvě různé konfigurace replikace a zálohování dat za účelem zajištění jejich dostupnosti za každých okolností. Vždy platí, že jsou oddíly s daty v několika kopiích rozprostřeny mezi uzly. Replikační faktor definuje počet kopií každého oddílu. Network Topology Strategy umožňuje individuálně konfigurovat replikační faktor pro jednotlivá datacentra. Jedná se o konfiguraci vhodnou pro rozsáhlejší systémy nebo pro malý systém, u nějž se předpokládá budoucí rozšíření. Simple Strategy definuje pro celý keyspace pouze jeden globální replikační faktor. Cassandra umožňuje kontrolu konzistence dat v replikovaných oddílech prostřednictvím položky nastavení zvané Consistency Levels. Operace zápisu je prováděna

vždy do všech replikovaných oddílů. Při čtení musí být stejná odpověď vrácena minimálně takovým počtem uzlů, aby byla splněna podmínka v rámci Consistency Levels. V rámci clusteru mezi sebou uzly komunikují přes Gossip protokol. Každá změna dat oddílu je nejprve zapisována do **Commit Log** uzlu, na němž se oddíl nachází. Jedná se o log, umožňující pouze přidávání dalších záznamů. Je uložen na disku uzlu. Tento prvek také zajišťuje zotavení uzlu v případě výpadku, a to zapsáním změn dat z Commit Log do Memtables ihned po obnovení činnosti uzlu. **Memtables** je struktura v paměti uzlu a slouží jako zásobník. Po jeho naplnění dojde k zápisu dat na disk, a to konkrétně do **SSTables**.

Při čtení dat z databáze plní uzel, prostřednictvím kterého klient do databáze přistupuje, roli koordinátora. Nejprve na základě DB dotazu vygeneruje hash partition key a lokalizuje uzly, na kterých se nachází kopie oddílů s daty. Následně mu uzel, jehož odpověď přišla jako první, předá data dotazu. Z těch generuje koordinátor hash a pomocí něj zkontroluje, zda se i na ostatních uzlech, obsahujících kopii příslušného oddílu, nachází stejná data. Pokud je vše v pořádku, koordinátor vrátí odpověď na dotaz klientovi.

Cassandra automaticky vytváří **index** pro sloupce primary key. Pro vyhledávání záznamu v databázi na základě podmínky, odkazující se na jiný sloupec než primary key, je nezbytné vytvořit sekundární index. Ten umožní rychlé a efektivní prohledání tabulky. Pokud je index vytvořen, je automaticky rozšiřován na každou novou řadu tabulky.

Pro práci s daty se v rámci Apache Cassandra používá CQL (Cassandra Query Language). Jedná se o programovací jazyk, podobný SQL, pomocí nějž se dá vytvářet schéma databáze a také pracovat s uloženými daty. Dle zdroje [24] Cassandra přímo nepodporuje cizí klíče, tedy relace a zachování integrity mezi tabulkami. Dále není podporována množinová operace JOIN[7], s níž lze provázat data mezi tabulkami kartézským součinem. Avšak dokumentace uvádí, že operace čtení a zápisu dat jsou z hlediska výpočetních zdrojů velmi nenáročné. Proto je třeba již při návrhu datového modelu zohlednit, jakým způsobem bude s daty nakládáno a dle toho definovat potřebné tabulky. Při návrhu datového modelu se podoba tabulek řídí podobou předpokládaných databázových dotazů na data.

Databázový systém Cassandra podporuje základní datové typy. Například int pro číslo a text pro textové řetězce. Implementovány jsou i kolekce. List zastupuje klasický uspořádaný seznam prvků, map slouží k uložení párů klíč - hodnota a set je množinou unikátních prvků. Cassandra dále umožňuje definovat uživatelské datové typy. S nimi jde modelovat objekty a přizpůsobit tak databázi potřebám projektu.

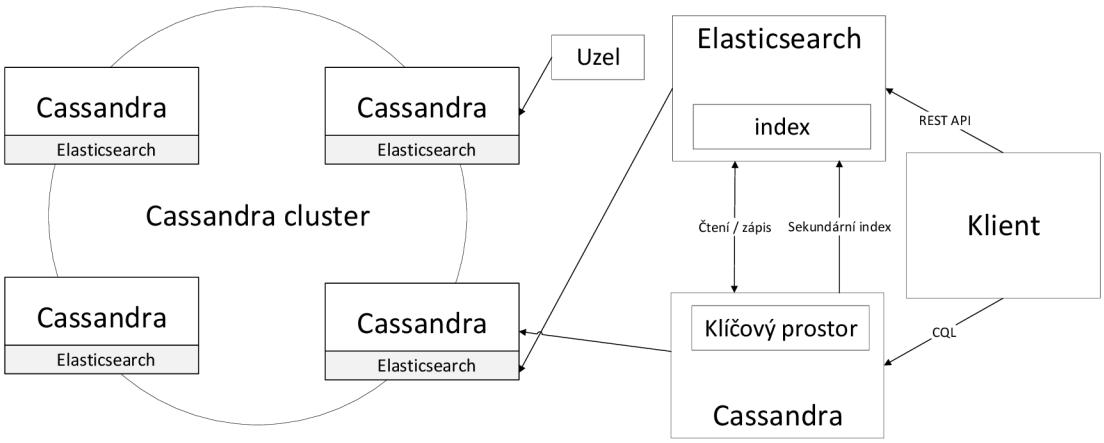
[7]Databázová operace, která na základě uživatelsky stanovené podmínky kombinuje sloupce stávajících databázových tabulek do tabulky nové.

1.4.4 Elasticsearch

Elasticsearch je open source distribuovaný systém [26] pro rychlé a efektivní vyhledávání v datech různých formátů. Jádro softwaru tvoří vyhledávací knihovna Apache Lucene. **Elassandra** je podle zdroje [27] open source distribuce Apache Cassandra, obohacená o Elasticsearch. Elasticsearch Cassandře poskytuje sekundární index, který umožňuje vyhledávání téměř v reálném čase. Současně je možné data Cassandry prohledávat fulltextově a přistupovat k nim i prostřednictvím REST API.

Cluster Elasticsearch se skládá z uzlů. Těmito uzly jsou fyzické či virtuální stroje s běžícím Elasticsearch. Každý uzel je schopen obsluhovat klientské požadavky na cluster, jako například požadavky na vyhledání dat, přes REST API. V případě Elasandra je fyzický uzel clusteru jak uzlem pro Cassandru, tak i pro Elasticsearch. Schéma systému je na obr. 1.8.

Základní datovou jednotkou systému je dokument. Tento prvek obsahuje vazbu klíčů, na základě nichž je vyhledáván, na požadovaná data. Dokumenty mohou být různých typů. Index je kolekcí všech dokumentů různých typů, které však mají nějakou společnou vlastnost. Pokud bychom přirovnali model Elasticsearch ke klasické relační databázi, index by byl databází, typ dokumentu tabulkou a dokument samotným řádkem tabulky s daty. U balíčku Elassandra při vkládání dat do DB dochází současně k plnění indexu Elasticsearch.



Obr. 1.8: Princip propojení Cassandra a Elasticsearch v rámci balíčku Elassandra.

2 Praktická část diplomové práce

Tato kapitola se věnuje návrhu, implementaci a testování vlastního serverového řešení a grafického uživatelského rozhraní pro systém detekce rizikových situací na železničním přejezdu. Systém se skládá z několika komponent. Jako zdroj dat pro serverové řešení slouží kamery s podporou autonomního rozpoznávání různých nebezpečných dopravních situací. Účelem serverového řešení, které vzniká v rámci této diplomové práce, je získaná data z kamer agregovat, zpracovat a uchovat. Interaktivní webová stránka, která je také vyvíjena v rámci této práce, má s pomocí dat, poskytnutých serverovým řešením, umožnit uživateli systému vhodně analyzovat zaznamenané rizikové dopravní situace.

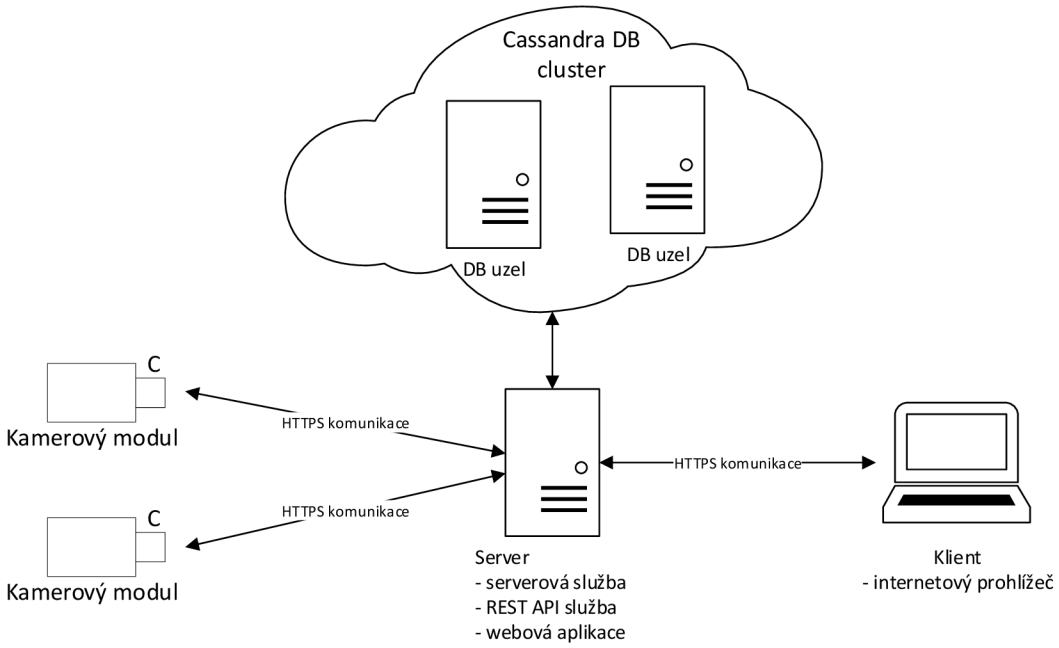
2.1 Návrh základních komponent systému

Podkapitola se zabývá návrhem jednotlivých komponent systému detekce rizikových situací na železničním přejezdu. Část textu je věnována popisu kamerových modulů, jejichž realizace není předmětem této práce, avšak z pohledu návrhu serverového řešení je nezbytné specifikovat jejich komunikační rozhraní. Obrázek 2.1 znázorňuje architekturu celého systému.

Kamerové moduly jsou komplexní zařízení poskytující zdroj dat pro vytvářené serverové řešení. Skládají se z obrazového snímače a hardwarových a softwarových komponent pro autonomní detekci rizikové situace na železničním přejezdu. Vyhodnocování událostí funguje na principu strojového rozpoznávání obrazu. Každá kamera je schopná komunikovat po síti a poskytovat tak serveru informace o svém stavu a zaznamenaných událostech. Z hlediska návrhu serverového řešení je důležitý popis komunikačního rozhraní kamery. To je tvořeno třemi částmi.

- K přenosu zpráv **o stavu kamery a o zaznamenaných událostech** je využit protokol HTTPS. Kamerový modul implementuje odesílání HTTPS žádostí na REST API serverového řešení. To zajišťuje jejich zpracování a ukládání. Přenos informací probíhá v těle HTTPS žádostí. Data jsou ve formátu JSON objektu. Existují dva typy zpráv. **Status message** (zpráva stavu) obsahuje informace o stavu kamery, zde lze najít například časové razítko zprávy, vytížení hardwaru kamery či identifikaci kamery. Event message (zpráva události) pak informuje o událostech zjištěných kamerou. Jde o zprávu kompletně popisující rizikovou událost na železničním přejezdu. Aby byla zajištěna bezpečnost a data žádosti na serverové řešení nemohl odesílat jiný subjekt, než oprávněný kamerový modul, používá se zde HTTPS s autentizací jak serveru, tak i klienta.

- K přenosu multimediálních dat o zjištěných rizikových událostech z kamery na serverové řešení se využívá protokol SCP (SSH file transfer protocol). Jedná se o šifrovaný protokol pro přenos dat. Jeho základem je níže zmíněný protokol SSH (Secure Shell).
- SSH server je softwarový nástroj, který implementuje komunikaci pomocí protokolu SSH. Jde o šifrovaný komunikační protokol, jenž se v praxi používá pro vzdálený přístup ke stanici nebo přenos souborů. Díky tomuto protokolu lze na vzdálené stanici přistupovat k příkazovému řádku a spouštět takové příkazy, které je její příkazový řádek schopný interpretovat. S pomocí SSH protokolu je možné provádět vzdálenou konfiguraci kamery.



Obr. 2.1: Architektura systému pro detekci rizikových dopravních situací.

Vytvořené **Serverové řešení** pro sběr, zpracování a uchovávání dat z kamerových modulů se skládá ze dvou základních komponent. Serverové služby a databáze. Účelem **serverové služby** je poskytnout aplikační rozhraní pro příjem zpráv z kamery. A dále provést zpracování a uložení zpráv do databáze. Jako technologie pro implementaci serverové služby byla vybrána platforma .NET Core a programovací jazyk C#. S pomocí těchto nástrojů lze realizovat REST API, nutné pro příjem zpráv událostí i zpráv stavů z kamer. Také existuje softwarový open source balíček

ovladače databáze Cassandra od společnosti Datastax pro C#. Ten usnadňuje práci s databází ve zmíněném programovacím jazyce. Jako technologie webového serveru pro běh serverové .NET Core služby byl vybrán Microsoft IIS (Internet Information Services). Jak uvádí dokumentace [28], jde o server dodávaný spolu s operačním systémem Windows a Windows Server. Konkrétně aktuální verze IIS 10.0 je dodávaná spolu s operačním systémem Windows 10 a instalovaným balíkem aktualizací Fall Creators Update a také se serverovým operačním systémem Window Server 2016 v. 1709.

K ukládání dat v rámci serverového řešení slouží **databázový cluster Cassandra**. Pro účely vyvíjeného serverového řešení je tato databáze instalována v rámci balíčku Elassandra. Jde o software kombinující databázový systém Cassandra s vyhledávacím nástrojem Elasticsearch. V rámci této práce je Cassandra použita jako úložiště pro zprávy událostí a stavů z kamer. Díky Elasticsearch je možné v budoucnu rozšířit funkce serverového řešení. S pomocí Elasticsearch lze totiž data v databázi vyhledávat na základě komplexnějších podmínek.

Elassandra je multiplatformní software, který ke svému běhu, jak udává dokumentace [27], potřebuje Java Virtual Machine. Doporučená je verze Java Runtime Environment 1.8. Zde je možné použít i open source implementaci Java platformy OpenJDK. Jako operační systém pro stanice, které budou v rámci serverového řešení plnit roli Elassandra uzlů, byla zvolena Linux distribuce Ubuntu ve verzi 18.04 LTS s prodlouženou dobou podpory. Volba operačního systému vychází z následujících faktů. Ubuntu je široce podporovaný open source operační systém. Tudíž odpadá nutnost nákupu licence a s tím spojených dodatečných nákladů. Oficiální stránka tohoto operačního systému zmiňuje několik jeho verzí [29]. Pro účely vývoje je vhodný Ubuntu Desktop s grafickým uživatelským rozhraním pro pohodlnější testování. V produkčním prostředí se počítá s nasazením Ubuntu Server.

Nároky na hardware strojů s Elassandra uzly vychází z dokumentace projektu Elassandra [27] a také z dokumentace databáze Cassandra [30]. Elassandra je díky kombinaci databáze a vyhledávacího nástroje více náročná na operační paměť. Naprosté minimum pro její běh je 4 GB RAM (Random Access Memory). Ovšem pro produkční prostředí je doporučená minimální hodnota 8 GB. Co se týká nároků na procesor stanice, databázový systém hojně uplatňuje paralelní vykonávání procesů, a proto je třeba volit procesory s vyšším počtem jader. Cassandra mnoho operací s daty vykovává v operační paměti a jejich zápis na disk provádí hromadně. Proto není nezbytně nutné v rámci Elassandra uzlů používat SSD (Solid-State drive). Systém dobře funguje i s klasickými plotnovými disky.

Interaktivní webová stránka tvoří grafické uživatelské rozhraní systému pro detekci rizikových situací na železničním přejezdu. Jejím hlavním účelem je umožnit uživateli přehledně analyzovat zaznamenané kritické situace. Mezi další nároky

na ni kladené, patří bezpečnost a podpora uživatelských rolí pro účely řízení přístupu. Z hlediska implementace je interaktivní web realizován prostřednictvím dvou komponent. A to webové aplikace a REST API služby.

Webová Aplikace implementuje samotné grafické rozhraní, které je zobrazováno uživateli v jeho webovém prohlížeči. Pro její tvorbu byla vybrána technologie Razor Pages. Ta je součástí .NET Core a v rámci tohoto frameworku se jedná o spolehlivou a hojně používanou technologii k tvorbě webových stránek na straně serveru. Webová aplikace obsahuje různé moduly, s pomocí nichž lze jak kontrolovat události a výstrahy zaznamenané kamerami, tak i sledovat stav připojených kamerových modulů. Za účelem zvýšení přehlednosti a umožnění budoucí rozšiřitelnosti systému je použito modulární řešení, kdy samotná webová aplikace přímo neimplementuje operace s databází. K tomuto účelu používá REST API službu. K zajištění bezpečnosti komunikace mezi serverem a klientem je použit protokol HTTPS. K autentizaci klienta ve webové aplikaci slouží HTTP cookies.

REST API služba implementuje veškeré logické operace a práci s databází. Webové aplikaci poskytuje své zdroje přes bezpečný protokol HTTPS. Vyměňovaná data jsou ve formátu JSON. K jejímu vývoji je použit, podobně jako v případě webové aplikace, .NET Core. V případě, že je v rámci webové aplikace potřeba například zobrazit data o nějaké zaznamenané události, webová aplikace zavolá na URL (Uniform Resource Locator) adresu příslušného zdroje v rámci REST API služby. V případě, že daný zdroj vyžaduje nějaký vstupní parametr, je tento parametr webovou aplikací ve formátu JSON službě předán. Následně REST API služba požadavek zpracuje a opět ve formátu JSON vrátí webové aplikaci odpověď. Ta poté s pomocí šablon a získaných dat vygeneruje HTML kód výsledné webové stránky a ten pošle do prohlížeče klienta. REST API služba webové aplikaci také poskytuje prostředky pro autentizaci a autorizaci uživatelů. Tyto informace jsou přenášeny ve formě JWT tokenů. Jako webový server pro běh obou těchto komponent uživatelského rozhraní je použit, podobně jako v rámci serverového řešení, server IIS od společnosti Microsoft.

2.2 Vlastní implementace serverového řešení

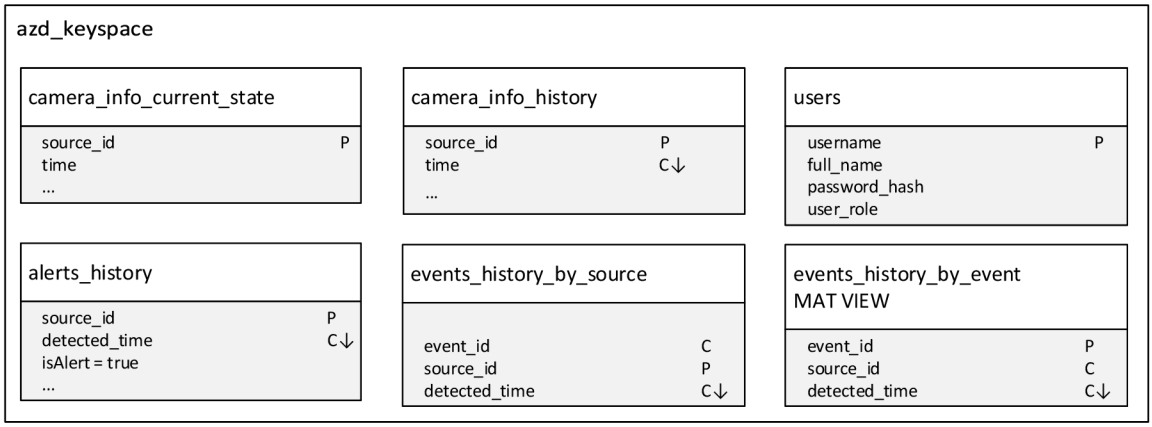
Tato podkapitola se věnuje podrobnějšímu popisu implementace a testování vytvořeného serverového řešení. To se skládá ze serverové služby a databázového clusteru postaveného na nástroji Elassandra, kombinujícím databázi Cassandra s vyhledávacím softwarem Elasticsearch. Text se zabývá i manuálním a automatizovaným testováním vzniklého serverového řešení s pomocí nástroje Postman.

2.2.1 Databázový cluster

V rámci realizace databázového clusteru bylo třeba vhodně navrhnout schéma tabulek databáze Cassandra. Data v těchto tabulkách slouží jako zdroj informací pro webové GUI. Správný návrh Cassandra databáze se řídí několika základními pravidly.

Data musí být v rámci clusteru rozložena rovnoměrně. Design tabulek by měl zajistit, aby operace čtení dat pracovaly co nejvíce s daty v rámci jednoho oddílu. Dále je nutné sloupce tabulek definovat s ohledem na předpokládané databázové dotazy. Dokumentace [24] tento přístup k tvorbě datového modelu DB označuje jako Query-driven modeling, tedy modelování řízené dotazy.

Nejprve bylo tedy nutné definovat předpokládané databázové dotazy. Poté na základě nich sestavit konkrétní tabulky a vytvořit tak kompletní model databáze. Jeho podobu reprezentuje obr. 2.2. Veškeré tabulky jsou definované v rámci keyspace `azd_keyspace`. Pro vyšší přehlednost grafická reprezentace modelu obsahuje pouze ty sloupce tabulek, které slouží jako databázové klíče. Výjimku tvoří tabulka `users`, uchovávající uživatelské údaje. To proto, že oproti jiným tabulkám má relativně málo sloupců. Veškeré tabulky byly navrženy s důrazem na rovnoměrné rozložení dat v clusteru, které je odvozeno od hodnot ve sloupcích klíčů clustering key a partition key. Podoba dotazů a s nimi spojených tabulek vychází z funkcionalit dostupných v rámci webového grafického uživatelského rozhraní. Nyní budou popsány databázové dotazy a odpovídající tabulky v kontextu s funkcemi v rámci jednotlivých modulů GUI.



Obr. 2.2: Schéma tabulek databáze.

Přehled kamer je modul, pomocí nějž lze sledovat aktuální stav kamer. V rámci tohoto modulu se očekávají dotazy na aktuální stav kamery s určitým identifikátorem a také na souhrnný stav všech kamer systému. V databázi je toto realizováno

tabulkou `camera_info_current_state`. Jedná se o tabulku s podrobnými aktuálními záznamy o kameře, v rámci níž lze vyhledávat na základě identifikátoru kamery, který slouží jako partition key.

Detail kamery má za úkol zprostředkovávat informace o stavu kamery v průběhu času. Implementace v databázi je řešena tabulkou `camera_info_history`, jejímž obsahem jsou podrobné informace o kamerách, sbírané průběžně po celou dobu existence kamery v systému. Záznamy každé konkrétní kamery se ukládají do jednoho oddílu databáze, protože identifikátor kamery zde tvoří partition key. Současně jsou v rámci oddílu řazeny podle časového razítka sestupně, tedy od nejnovějšího záznamu po nejstarší. Díky tomu lze z databáze dotazem snadno získat konkrétní počet nejnovějších záznamů k relevantní kameře. A současně už není třeba na aplikační úrovni záznamy dále řadit. K tomuto modulu se váže i nastavování uživatelsky definovaného názvu kamery. Pro tento účel slouží jeden ze sloupců tabulky `camera_info_current_state`, udržující aktuální hodnotu názvu. V případě, že uživatel žádný název nenastaví, je k identifikaci kamery ve webovém rozhraní použit její identifikátor.

Modul výstrahy slouží jako přehled zaznamenaných rizikových situací na železničním přejezdu. V rámci databáze je toto řešeno tabulkou `alerts_history`. Očekávány jsou dotazy na nejnovější výstrahu z konkrétní kamery. Proto je partition klíč tvořen identifikátorem zdroje události s výstrahou a clustering key jejím časovým razítkem, řazeným sestupně.

Události je modul, který by měl umožnit dotazy na veškeré zaznamenané události, včetně těch bez příznaku výstrahy. Databázová tabulka, která jej reprezentuje, se jmenuje `events_history_by_source`. Podporováno je vyhledávání na základě zdroje události a řazení v oddílech probíhá na základě času detekce.

Detail události umožní vyhledat událost na základě jejího zdroje či identifikátoru události. Tabulka `events_history_by_source` odpovídá databázovému dotazu na událost dle zdroje. Jejím obsahem jsou podrobná data o konkrétních událostech, ukládaná do oddílů databáze podle kamery, jež událost zaznamenala, a současně řazená na základě času. Aby databáze podporovala i dotaz, hledající událost podle identifikátoru události, byl vytvořen materializovaný pohled `events_history_by_event`, obsahující pouze klíče tabulky, `events_history_by_source`, od níž byl odvozen. Materializovaný pohled otáčí pořadí klíčů pro vyhledávání a popisovaný dotaz umožňuje realizovat.

Řízení přístupu je realizováno serverovou službou. Data uživatelů systému jsou uložena v tabulce `users`. Hesla jsou samozřejmě v nečitelné formě. V databázi se nachází pouze jejich otisk, generovaný s pomocí hashovací funkce SHA-512. V této tabulce se očekávají dotazy, které v ní vyhledávají na základě uživatelského jména. Proto tento údaj tvoří partition key.

2.2.2 Serverová služba

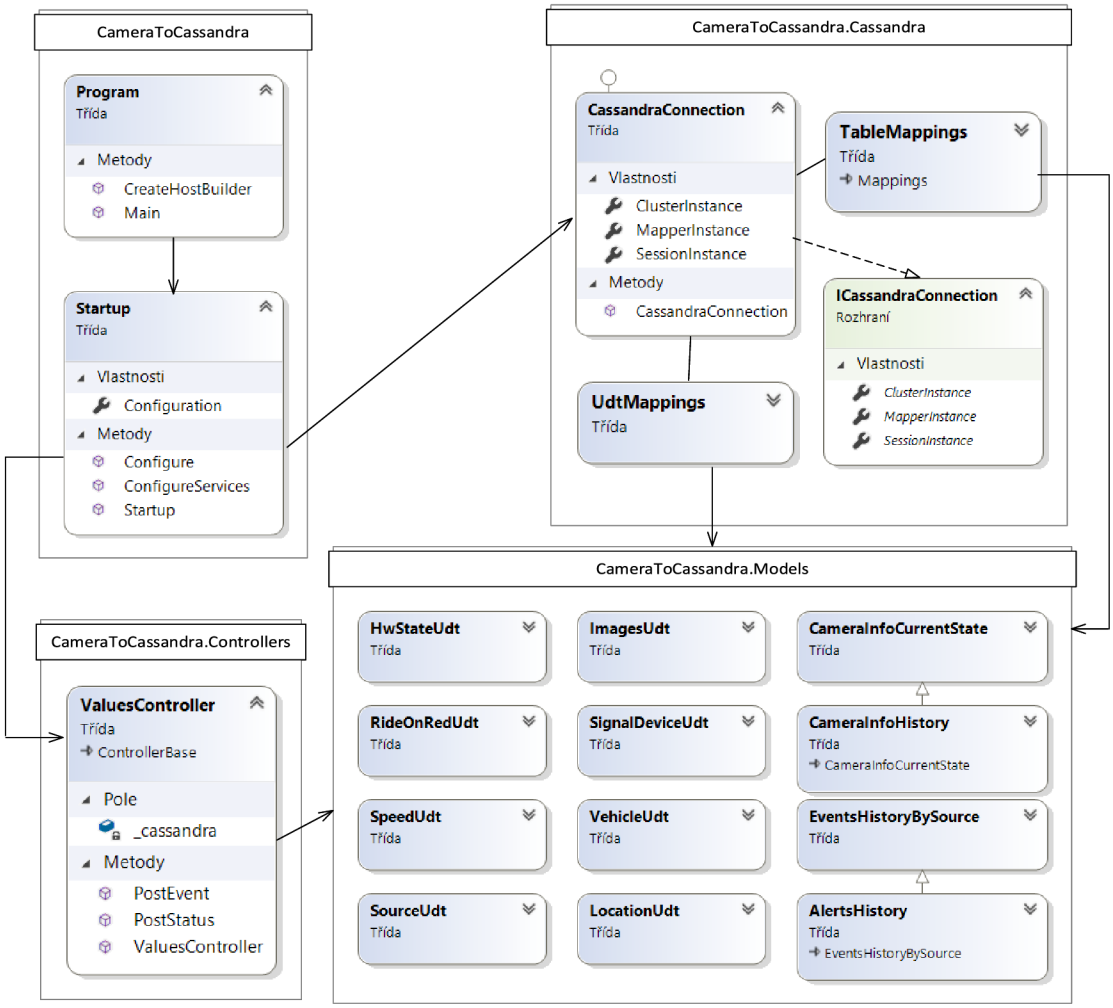
Jde o .NET Core aplikaci implementující REST API pro příjem zpráv z kamer a také pro jejich zpracování a uložení do databáze. K její realizaci byla použita vývojová platforma .NET Core 3.1 a programovací jazyk C#. Architekturu aplikace znázorňuje diagram tříd na obr. 2.3. Třídy jsou pro větší přehlednost seskupovány do jmenných prostorů na základě svého účelu. Názvy jmenných prostorů jsou v rámci diagramu uvedeny vždy na horní straně grafického prvku kontejneru, který jmenný prostor symbolizuje.

Jmenný prostor `CameraToCassandra.Models` obsahuje třídy, které reprezentují model přijímaných zpráv o stavu kamery a o zaznamenané události. Jejich podoba vychází ze struktury Json objektů. Zpráva události i zpráva stavu přichází na REST API serverové služby z kamery ve formě HTTP žádosti. Data zprávy se nachází v rámci těla žádosti v podobě Json objektu. Jeho struktura je neměnná. Avšak definice Json objektu obsahuje jak položky povinné, tak i ty nepovinné. Povinná položka se nachází ve zprávě vždy, nepovinná nemusí být obsažena. Tento fakt je třeba ošetřit z pohledu ukládání hodnot ze zpráv do tabulek databáze. Veškeré C# vlastnosti implementující nepovinné atributy zpráv jsou definovány nulovatelným datovým typem a jsou přednastaveny na hodnotu null. Takto je zajištěno, aby se do databáze nezanesla nesmyslná, defaultní, hodnota při práci s nepovinným atributem.

`CameraToCassandra.Cassandra` je jmenný prostor implementující relaci spojení serverové služby s databází Cassandra. Pro práci s databázovým serverem je použit ovladač **CassandraCSharpDriver** verze 3.16.1. Třída `CassandraConnection` obsahuje objekty spojení a relace databázového clusteru. Dále se zde nachází objekt typu `Mapper`. Použitý databázový ovladač umožňuje s databází pracovat přímo pomocí databázových příkazů, zadávaných ve formě textových řetězců. Avšak dostupný je i daleko elegantnější přístup, který je v případě této aplikace uplatněn. Objekt `Mapper` ve svém konstruktoru umožňuje zadat definice mapování vlastností C# na sloupce tabulek databáze. Tyto definice se nachází ve třídě `TableMappings`.

Cassandra pracuje i s uživatelsky definovanými datovými typy. Pomocí takových sloupců databáze jsou reprezentovány některé záznamy. Proto je při tvorbě objektu databázové relace vložena do konstruktoru definice těchto uživatelských datových typů, obsažená ve třídě `UdtMappings`. Rozhraní `ICassandraConnection` slouží potřebám injektáže závislostí do objektu ovladače REST API.

`CameraToCassandra.Controllers` obsahuje definici REST API. V třídě `ValuesController` se nachází URI REST API a metody implementující navázané akce. V rámci těchto metod dochází ke zpracování zpráv z kamer a k jejich ukládání do databáze. V případě, že vše probíhá v pořádku, vrací server kameře návratový kód HTTP 204, značící úspěch operace. V opačném případě je vrácen chybový ná-



Obr. 2.3: Zjednodušený diagram tříd serverové služby.

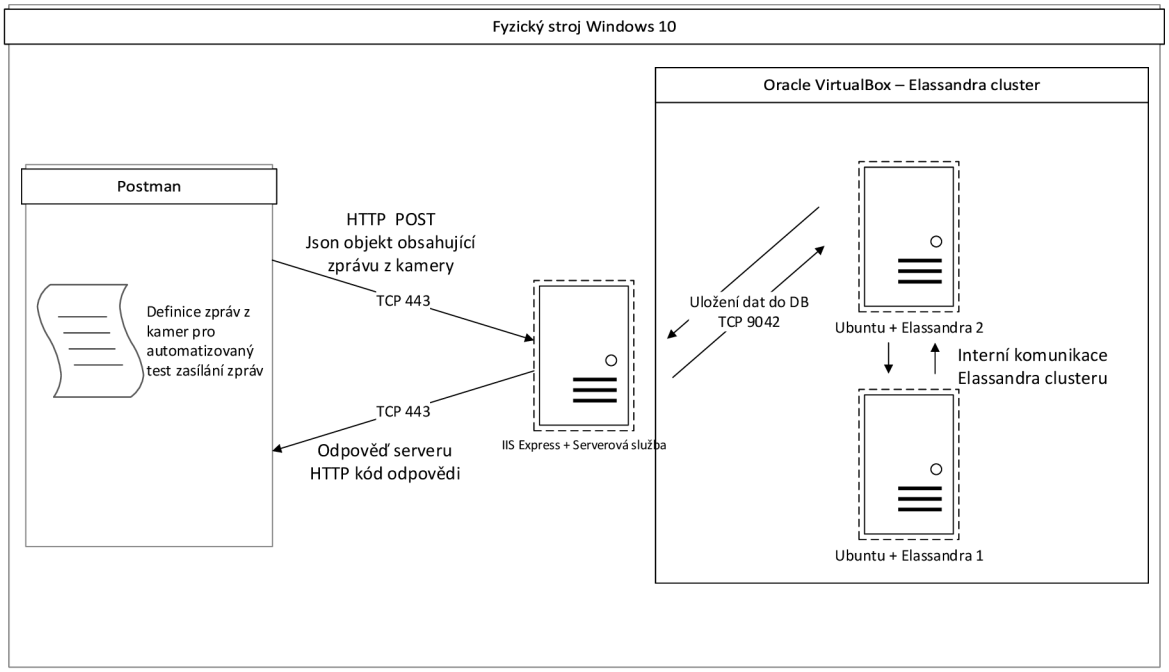
vratový kód a text chyby. Metoda `Main()` v rámci třídy `Program` spouští celou aplikaci serverové služby. V této metodě je provolána metoda `Run()` nad instancí objektu `IHostBuilder`, čímž je spuštěno REST API. Tento objekt má na starosti kompletní konfiguraci základních komponent .NET Core aplikace. V tomto případě přebírá třídu `Startup`. Ta mimo jiné do aplikace přidává kontrolery a provádí injektáž závislostí. Do kontroleru je objekt spojení databáze v souladu s dokumentací ovladače vkládán tak, aby se v rámci celého běhu služby pracovalo s pouze jednou jeho instancí a tím se zajistil optimální výkon.

2.2.3 Vývojové prostředí pro serverové řešení

Pro vývoj a testování serverové služby pro systém detekce nebezpečných situací na železničním přejezdu bylo vytvořeno virtualizované vývojové prostředí. Jeho archi-

tektura je na obr. 2.4. Základ tvoří fyzická stanice s operačním systémem Windows 10. Pro vytvoření Elassandra uzlů je použita virtualizace. Konkrétně prostřednictvím programu VirtualBox 6.1.12 od společnosti Oracle. Jak uvádí oficiální webová stránka softwaru [31], jde o zdarma dostupný open source software pod licencí GNU GPL v.2.

Simulace zasílání zpráv z kamer je realizována programem **Postman** [32]. Tento nástroj umožňuje komplexní testování webového API. S jeho pomocí lze definovat zprávy posílané na server, v rámci těchto zpráv také nastavit proměnné a ty pomocí jazyka JavaScript v průběhu testování modifikovat. Postman umožňuje i automatizované testy REST API a automatizovanou kontrolu odpovědí generovaných serverem.



Obr. 2.4: Architektura prostředí pro vývoj serverového řešení.

Pro realizaci Elassandra clusteru byly vytvořeny dvě virtuální stanice s operačním systémem Ubuntu 18.04 LTS ve verzi s GUI. Na ně byla nasazena Elassandra 6.2.3.31. Každá z těchto virtuálních stanic má přiřazeno 8 GB operační paměti a čtyři jádra procesoru. Serverová služba pro příjem, zpracování a ukládání dat do databáze běží na fyzickém stroji v rámci IIS Express. Jde o odlehčenou verzi Microsoft IIS pro účely vývoje. Výhodou je zde snadné nasazení vytvářené aplikace do IIS Express i s pomocí vývojového prostředí Microsoft Visual Studio, čímž se usnadní testování. Veškerá síťová komunikace probíhá v rámci fyzického stroje a virtuální sítě programu VirtualBox. Komunikace mezi serverovou službou a kamerou probíhá

přes HTTPS na TCP portu 443. Serverová služba si data s databází vyměňuje na TCP portu 9042. Elasticsearch běží pak na TCP 9200. Porty TCP 7000, 7001 a 9300 využívají uzly clusteru pro vzájemnou komunikaci.

2.2.4 Testování

Základní testování vytvořeného serverového řešení probíhalo s pomocí nástroje Postman. Na REST API serverové služby byly zasílány generované HTTP žádosti se zprávami stavu kamery a zprávami událostí. Byly provedeny jak manuální, tak i automatizované testy.

Manuální testování bylo využito zejména pro ověření, zda serverová služba správně zpracovává a ukládá přijímaná data do tabulek Cassandra clusteru. Došlo tedy k sestavení JSON objektů pro zprávu stavu a zprávu události. Podoba JSON objektu vycházela ze šablony zpráv z kamer. Následně byly zasílány HTTP žádosti s těmito JSON objekty na API služby. Poté byla provedena kontrola uložených dat v databázi. Ta kladla důraz jednak na to, zda obsah tabulek odpovídá zasílaným zprávám, jednak zda dochází ke správnému řazení řádků záznamů podle databázového clustering klíče. Na obrázku 2.5 je vidět výpis testovacích záznamů databázové tabulky `events_history_by_event`.

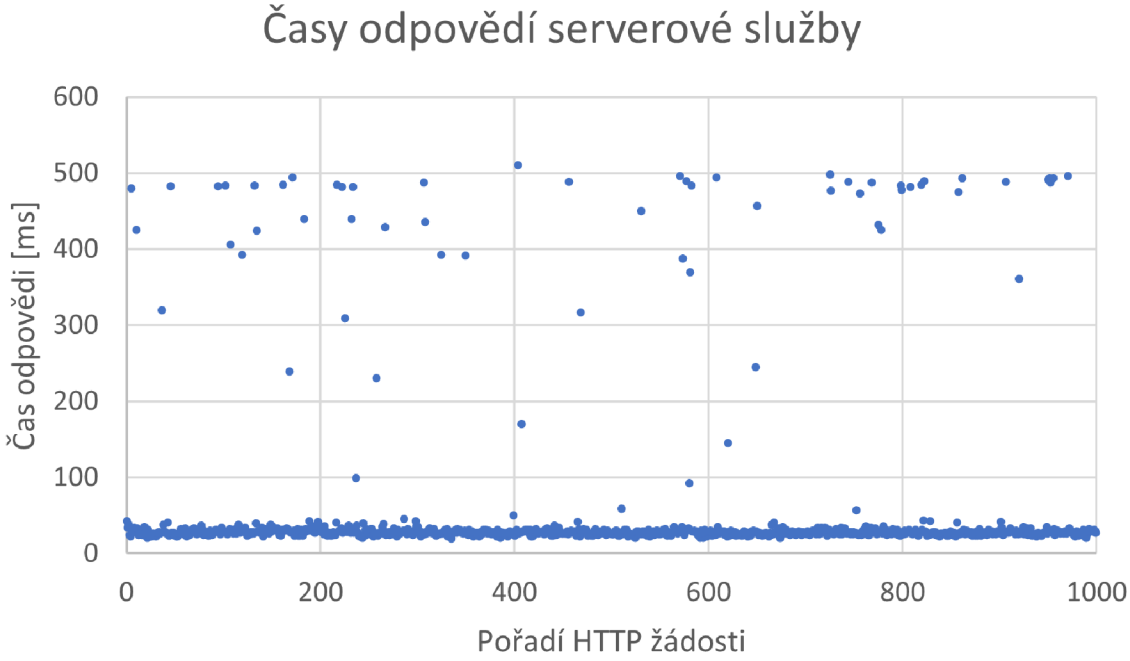
```
cqlsh> select * FROM azd_cameras.events_history_by_event  ;

 event_id | source_id | detected_time                 | event_seq_number | event_type     | hash
                      | is_alert | location
_event_id               | ride_on_red                                  | severity | signal_device
                | speed                                                        | vehicle
----------+-----------+-------------------------------+------------------+----------------+---------
--------------------------+----------+-----------------------------------------------------------------
-----------------------+-----------------------------------------------+----------+----------------------
------------------+-------------------------------------------------------+----------------------
-------------------------------------------------------------------------------------------------------
    eid1 |   source1 | 2020-03-17 09:30:00.000000+0000 |                1 | vehiclePassage | 017f9db1
9ddb53', image_type: null}] |    False | {city: 'Brno', street: 'Uvoz', direction: 'centrum', info: 'c
-54b6-4243-8077-9cff05c6 | {red_delay: 200, tolerated_red_delay: 80} |      low | {signal_type: 'dddd'
sw_name: 'lpr01'} | {speed_value: 200, rounded_speed: 80, limit_speed: 300} | {registration: 'registra
bhu', color: 'dcbh', number_of_axles: 3, lights_enabled: True, manufacturer: 'Hkdk', model: 'Hkdk', pr
```

Obr. 2.5: Výpis obsahu tabulky databáze.

Manuální testování ověřilo, že vzniklé serverové řešení funguje správně a nedochází k žádným logickým chybám při zpracování přijatých dat. Současně bylo ověřeno, že serverová služba vhodně ošetřuje příjem chybných zpráv z kamery pomocích chybových HTTP kódů.

Automatizované testy slouží k zjištění, zda se serverová služba i při vyšší zátěži chová korektně a zda nedochází například ke ztrátám dat. Pro jejich realizaci byl opět vybrán nástroj Postman, konkrétněji jeho funkcionalita Runner (spouštěč). V rámci Postman lze definovat HTTP žádosti a ty rozšířit o skripty v jazyce JavaScript.

V tomto případě byly tedy definovány skripty spouštěné před odesláním HTTP žádosti na server a skripty pro testování návratových kódů. Současně byly některé položky JSON objektů zpráv nahrazeny pomocí dynamicky se měnících proměnných pro zajištění jejich unikátnosti během automatizovaného testu. Tímto bylo připraveno testovací prostředí. Následoval automatizovaný test, spouštějící předdefinované HTTP žádosti opakovaně v iteraci. Na server bylo odesláno 1000 HTTP žádostí se zprávou události. Celý test proběhl úspěšně. Po každém testu byla navíc zkontrolována data v databázi. Ani zde nenastal problém. Během veškerých testů nebyl zjištěn jediný případ ztráty dat. Graf na obr. 2.6 zobrazuje časy odpovědí serveru během automatizovaného testu. Průměrná doba odpovědi činila 51,32 ms.



Obr. 2.6: Časy odpovědí serveru na HTTP žádosti.

2.3 Vlastní implementace interaktivní webové stránky

Podkapitola je věnována vývoji, implementaci a testování interaktivní webové stránky. V jejím návrhu byl uplatněn modulární přístup, kdy je uživatelské rozhraní implementováno jako samostatná webová aplikace, která data z databáze systému autonomní detekce rizikových situací na železničním přejezdu sama nezpracovává, avšak k tomuto účelu využívá volání příslušných zdrojů REST API služby.

2.3.1 REST API služba

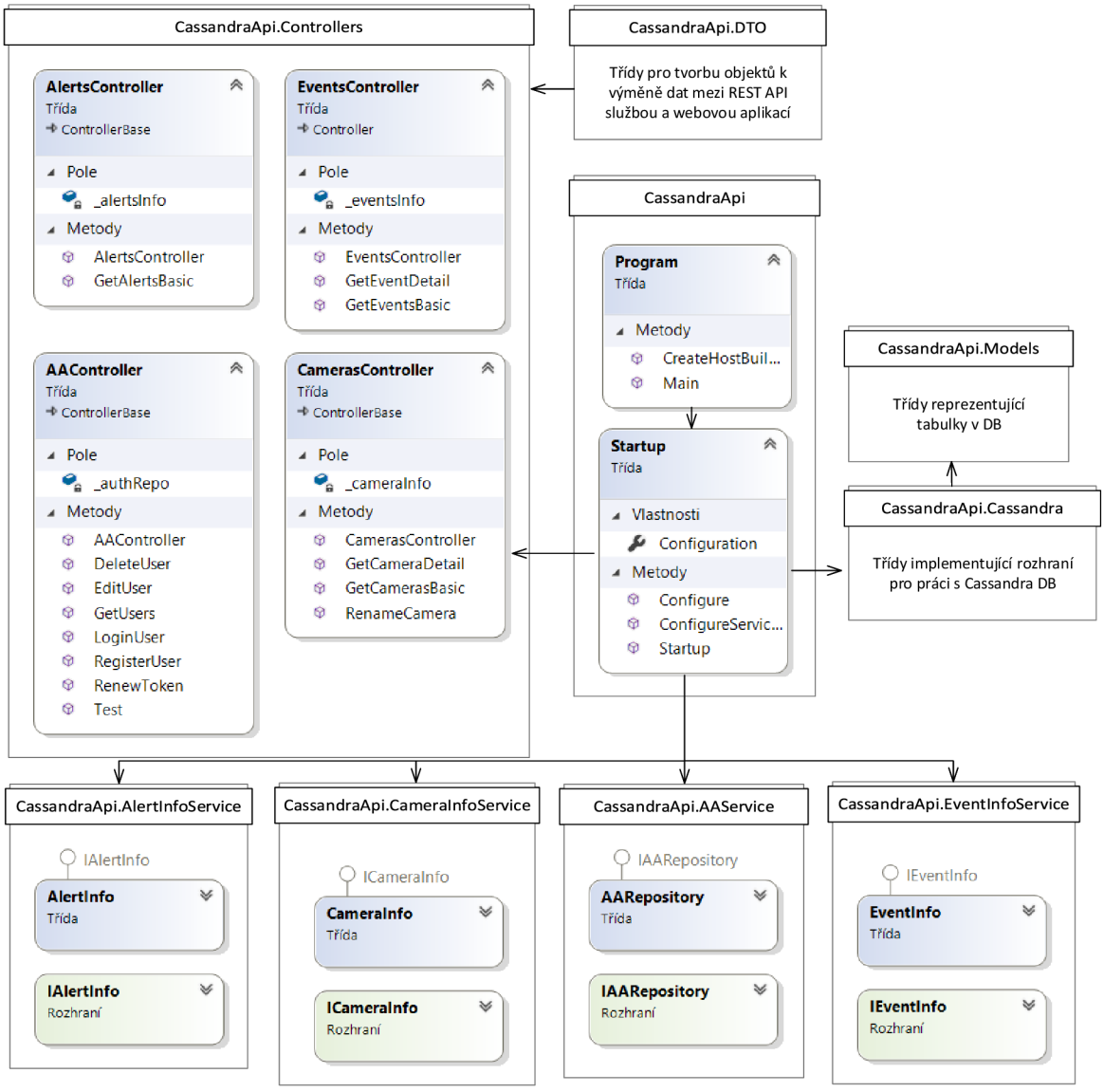
Tato služba implementuje veškeré logické operace s daty a databází, a výsledky těchto operací prostřednictvím REST API poskytuje webové aplikaci. K jejímu vývoji byl použit .NET Core ve verzi 3.1 a programovací jazyk C#. Její architekturu znázorňuje zjednodušený diagram tříd na obrázku 2.7. Grafické kontejnery reprezentují jednotlivé jmenné prostory. Z důvodu zachování přehlednosti diagramu některé jmenné prostory nemají své třídy zobrazené, pouze je obecně naznačena problematika, řešená třídami v rámci těchto jmenných prostorů.

REST API služba je spouštěna statickou metodou `Main()` v rámci třídy `Program`. V ní dochází k vytvoření instance objektu typu `IHostBuilder`. Pro jeho tvorbu je využito konfigurace ve třídě `Startup`. A nad tímto objektem jsou nakonec provolány metody, díky nimž dojde k spuštění REST API kontrolerů.

Třída `Startup` slouží ke konfiguraci služeb, dostupných v rámci .NET Core aplikace. Definuje se zde přidání API kontrolerů nebo JSON souboru, nesoucího některá nastavení aplikace, například IP adresu databázového serveru. Ve `Startup` je také prováděna DI (Dependency Injection), neboli vkládání závislostí. Jde o techniku, kdy jeden objekt může pro své potřeby využívat jiný objekt, aniž by na něj měl v době sestavování programu referenci. S pomocí popsané techniky jsou třídám obsahujícím API kontrolery předávány objekty implementující služby, poskytované těmito kontrolery.

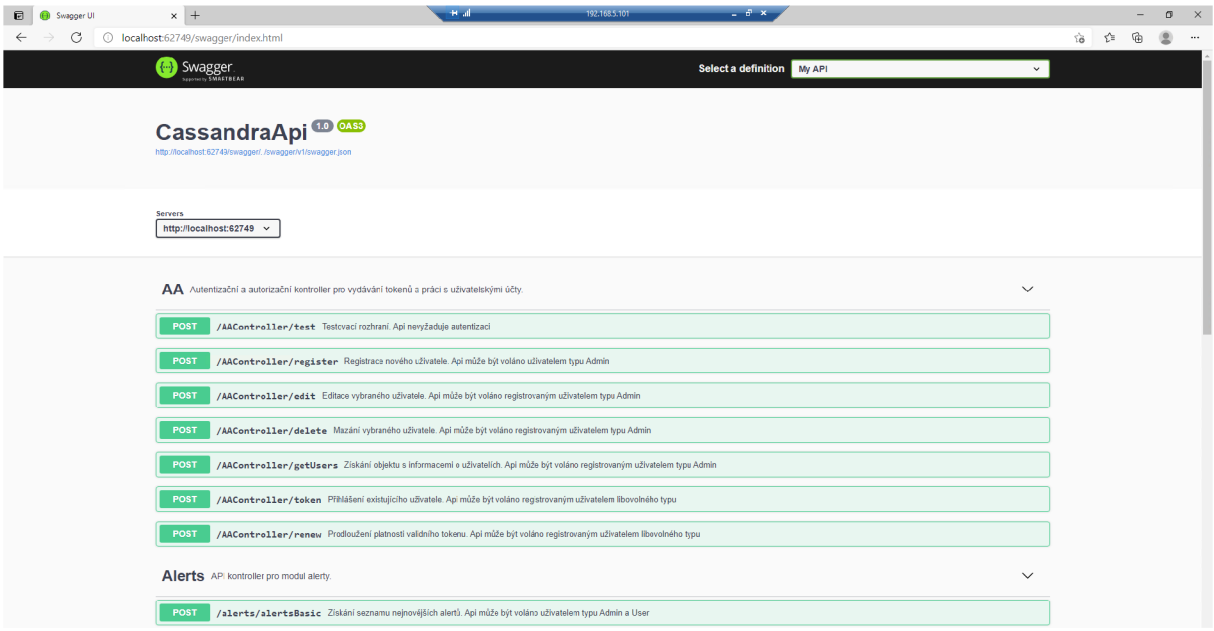
V neposlední řadě je zde konfigurována autentizační služba pro ověřování platnosti JWT tokenů. Aby byl token považován za platný, musí být ověřen tajný klíč, použitý k zajištění jeho integrity s pomocí hashovací funkce. V tomto případě je použit **HMAC** (Keyed-hash Message Authentication Code), tedy autentizační kód zprávy. Jde o kód, který se tvoří následujícím způsobem. Ke chráněným datům je přidán tajný klíč subjektu, zajišťující integritu zprávy. Následně jsou data a klíč použity jako vstup hashovací funkce, zde konkrétně **SHA-512**. Její výstup pak tvoří onen HMAC. Tajný klíč je nastaven v konfiguračních souborech této REST API služby. Ke konfiguračnímu souboru na serveru musí být omezen přístup pouze pro oprávněné uživatele. V případě vyzrazení klíče by totiž mohl platné JWT tokeny

vydávat kdokoliv a s jejich pomocí libovolně přistupovat ke zdrojům služby. Dalším ověřovaným parametrem je i časová platnost tokenu. Jakmile dojde k jeho expiraci, je považován za neplatný.



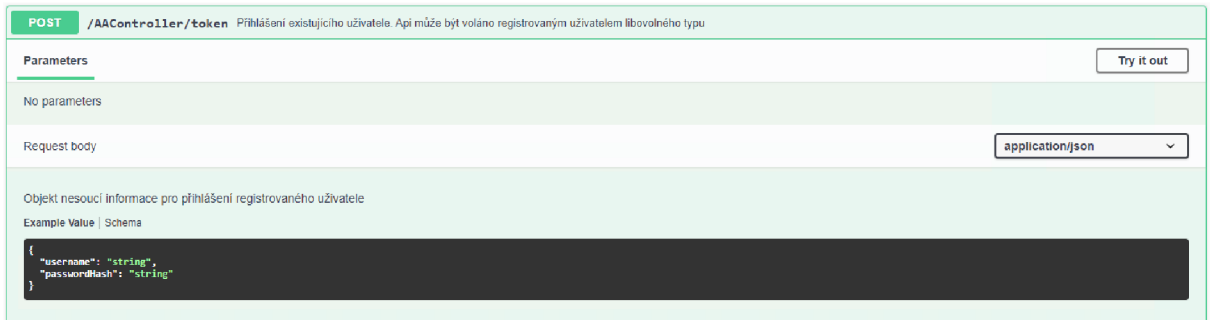
Obr. 2.7: Zjednodušený diagram tříd REST API služby.

Každý zdroj, poskytovaný REST API službou, má svou URL adresu, pod kterou je dostupný. Z důvodu, že tato služba implementuje poměrně velké množství zdrojů, je použit nástroj **Swagger** k jejich přehledné dokumentaci. Jde o open source framework, určený k návrhu a popisu REST API rozhraní. Jak uvádí dokumentace [33], pro .NET Core existuje NuGet balíček, přidávající podporu Swaggeru. V rámci třídy `Startup` je provedeno volání potřebných metod tohoto balíčku. Definována je zde i URL adresa webové stránky, jejíž rozhraní je na obrázku 2.8. Popisovaný



Obr. 2.8: Vytvořená webová stránka pro dokumentaci a testování REST API služby.

web je vygenerovaný s pomocí zmíněného nástroje. Obsahuje veškeré zdroje služby, a to včetně upřesňujících popisků. Tyto upřesňující popisky jsou definovány v XML (Extensible Markup Language) komentářích přímo ve zdrojových kódech API kontrolerů. Prostřednictvím vygenerované webové stránky lze funkčnost jednotlivých zdrojů i vyzkoušet. Detail grafického rozhraní pro testování API zdrojů v rámci Swagger dokumentační webové stránky je na obrázku 2.9.



Obr. 2.9: Grafické rozhraní pro testování zdroje REST API služby.

Práci s databází Cassandra implementují metody v rámci tříd jmenného prostoru `CassandraApi.Cassandra`. Ty využívají databázový ovladač CassandraCSharpDriver ve verzi 3.16.1. Data získaná z databáze je třeba vhodně reprezentovat. K tomuto účelu slouží třídy ve jmenném prostoru `CassandraApi.Models`. Na jejich atributy je prováděno automatické mapování dat z Cassandry. Podrobněji je problematika práce s databází v rámci C# aplikace popsána v podkapitole 2.2.2.

Třídy jmenného prostoru `CassandraApi.DTO` slouží k vytváření **DTO** (Data Transfer Object). DTO, neboli objekty pro přenos dat, se využívají k výměně informací mezi různými počítačovými programy. V případě této popisované REST API služby tedy slouží pro výměnu dat mezi ní a webovou aplikací. Ve zdrojových kódech se nachází DTO dvou typů.

DTO žádostí webové aplikace. Ty nachází uplatnění, pokud webová aplikace potřebuje využít nějaký zdroj REST API služby, který pro vykonání svých metod a generování výstupu vyžaduje vstupní data. V takovém případě webová aplikace s pomocí vstupních parametrů a příslušné DTO třídy vygeneruje objekt jazyka C#. Ten je následně převeden na JSON objekt a dochází k vytvoření HTTPS žádosti. Tělo žádosti tvoří zmíněný JSON. Poté, co je žádost přijata příslušným kontrolerem REST API služby, je JSON objekt v těle žádosti převeden na požadovaný C# objekt, odvozený od příslušné DTO třídy. Nakonec REST API služba tato data využije jako vstup do volaných metod.

Dále se používá **DTO odpovědí REST API služby**. Pokud webová aplikace na kontroler služby odešle validní žádost, s pomocí metod služby je žádost zpracována a dojde k vygenerování výsledného objektu nesoucího data odpovědi. Následuje tvorba objektu odpovědi odvozeného od příslušné DTO třídy. Objekt odpovědi ve svých atributech zapouzdřuje jednak objekt s daty odpovědi, jednak další informace o tom, zda byla žádost v pořádku zpracována a pokud ne, tak i příslušnou chybovou hlášku.

Jmenný prostor `CassandraApi.Controllers` obsahuje třídy API kontrolerů. Ty slouží k obsluze HTTPS žádostí ze strany webové aplikace. V samotných kontrolerech se nenachází implementace zdrojů, které jsou jimi poskytovány. Ta je obsažena v samostatných třídách, jejichž metody jsou v kontrolerech volány. Zde se pro realizaci vazby mezi kontrolery a třídami s implementací logiky uplatňuje návrhový vzor DI. Nyní budou podrobněji popsány jednotlivé kontrolery spolu s třídami realizujícími implementaci jimi poskytovaných zdrojů.
`AAController` poskytuje služby autentizace a autorizace uživatelů systému pro detekci rizikových dopravních situací. Samotná implementace služeb je zajištěna s pomocí tříd jmenného prostoru `CassandraApi.AAService`. V rámci těchto tříd program pracuje s databázovou tabulkou `users`, nesoucí informace o uživatelích. `AAController` podporuje následující HTTPS POST metody:

- **RegisterUser** slouží k vytvoření záznamu nového uživatele. Toto API může volat pouze oprávněný uživatel systému s přiřazenou příslušnou administrátorskou rolí. Vstupem jsou uživatelské údaje a vracena je informace o úspěchu operace.
- **LoginUser** je použit pro vystavení přístupového tokenu. API metoda může být volána kýmkoliv. Parametrem je uživatelské jméno a hash hesla. Návrato-

vou hodnotu může tvořit v případě, že se uživatel nachází v systému, validní JWT přístupový token. V opačném případě obecná informace o neúspěchu operace.

- **EditUser** slouží pro změnu údajů uživatele, uložených v databázi. Může být volána pouze administrátorem.
- **DeleteUser** realizuje smazání uživatele dle předaného uživatelského jména. Může být volána pouze administrátorem.
- **GetUser** umožňuje administrátorovi systému získat základní informace o uživatelích za účelem správy systému.
- **RenewToken** slouží pro obnovení časové platnosti validního JWT tokenu. Lze volat libovolným oprávněným uživatelem.
- **Test** realizuje testovací rozhraní kontroleru.

`AlertsController` zajišťuje webové aplikaci data o nejnovějších událostech, klasifikovaných jako výstraha. Implementace se nachází v třídách jmenného prostoru `CassandraApi.AlertInfoService`. Dostupné metody zajišťují stránkování pro potřeby webové aplikace. Stránky zde reprezentují skutečné stránky záznamů v rámci uživatelského rozhraní. Pokud je například webovou aplikací požadováno číslo stránky dva a počet záznamů na stránku pět, je webové aplikaci vrácen JSON objekt, obsahující pouze šestý až desátý nejnovější záznam. Tímto je sníženo množství komunikace mezi REST API službou a webovou stránkou. Poskytovány jsou tyto HTTPS POST metody:

- **GetAlertsBasic** vybírá z databázových tabulek definované množství nejnovějších výstrah pro každou z kamer systému. Této metodě je předáván jako parametr objekt, nesoucí informace o minimální úrovni závažnosti výstrahy, o počtu záznamů na stránku a také o čísle stránky. Vracen je JSON objekt s vybranými výstrahami.

`EventsController` poskytuje grafickému uživatelskému rozhraní data jednak o veškerých zaznamenaných událostech, tedy i těch, které nebyly klasifikovány jako výstraha. Dále pak detailní data vybrané zájmové události. Podobně jako v případě výstrah, i zde je zajištěno stránkování. Se zdroji tohoto kontroleru může nakládat libovolný oprávněný uživatel. Podporovány jsou tyto HTTPS metody:

- **GetEventsBasic** umožňuje získat základní data o událostech zaznamenaných definovanou zdrojovou kamerou v určeném časovém rozsahu. Ve výsledném JSON objektu se vždy nachází jak události klasifikované jako výstraha, tak i ty ostatní.
- **GetEventDetail** umí pro událost určenou jejím jednoznačným identifikátorem z databáze získat a ve formě JSON objektu webové aplikaci předat veškerá dostupná data.

`CamerasController` implementuje veškeré operace s daty kamer. Jeho metody jsou dostupné oprávněným uživatelům. Podporovány jsou následující HTTPS metody:

- **GetCamerasBasic** slouží ke generování JSON objektu s daty pro přehled o stavu veškerých připojených kamer.
- **GetCameraDetail** pro konkrétní kameru vygeneruje JSON objekt, obsahující jak současné, tak i historické záznamy o jejím stavu.
- **RenameCamera** slouží pro změnu uživatelsky definovaného jména kamery.

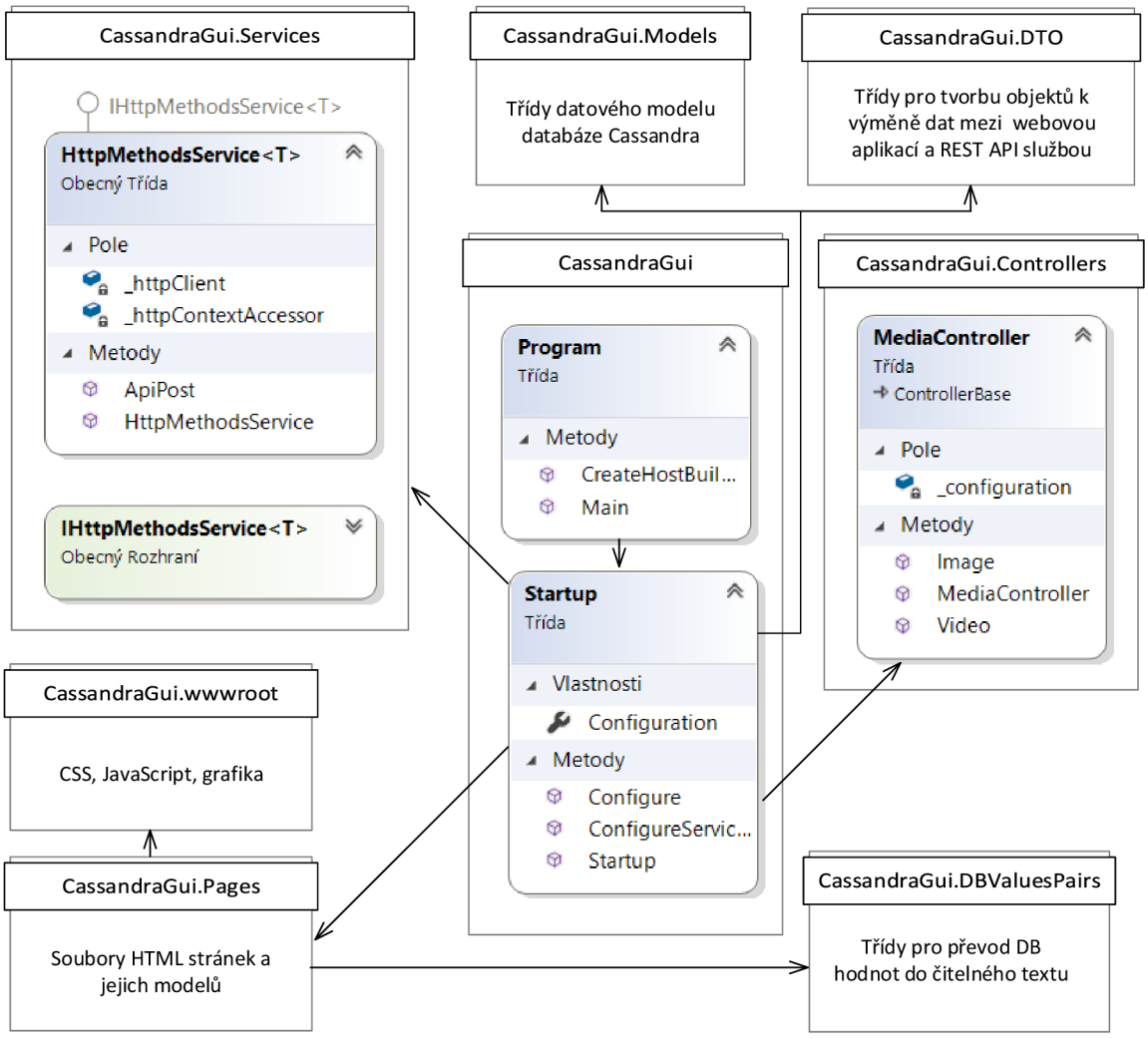
2.3.2 Webová aplikace

Webová aplikace slouží jako grafické uživatelské rozhraní pro prezentaci dat poskytovaných REST API službou. Operátorovi systému umožňuje přehledně analyzovat zaznamenané události a stavy připojených kamerových modulů. Aplikace je vytvořena s pomocí .NET Core 3.1 a technologie Razor Pages. Diagram na obrázku 2.10 znázorňuje architekturu projektu webové aplikace. Grafické kontejnery reprezentují jmenné prostory a složky se soubory.

Třída `Program` je vstupním bodem v procesu spouštění webové aplikace. Nachází se zde statická metoda `Main()`. V ní jsou provolány metody pro tvorbu objektu typu `IHostBuilder` a inicializaci celého programu. K vytvoření `IHostBuilder` se využívá obsahu třídy `Startup`.

Tato třída nese reference služeb dostupných v .NET Core aplikaci. V tomto případě je zde přidána podpora technologie Razor Pages. Ta zajistí, aby byly HTTPS žádosti obsluhovány s pomocí jednotlivých HTML stránek, definovaných v souborech jmenného prostoru `CassandraGui.Pages`. Autentizační a autorizační služby jsou zde taktéž konfigurovány. Autentizace je zajištěna díky .NET Core modulu pro práci s Cookies. Jde o malý soubor, nesoucí informace o uživateli a jeho roli, zabezpečený s pomocí .NET Core Data Protection modulu. Cookie je při úspěšném přihlášení uživatele vygenerována a uložena do jeho prohlížeče. Její platnost je časově omezena. V případě této webové aplikace dojde z důvodu bezpečnosti po jedné hodině k automatickému odhlášení uživatele. Uživatel je o této skutečnosti informován prostřednictvím notifikace.

Autorizace přístupu k jednotlivým HTML stránkám a zdrojům je zajištěna s pomocí rolí a politik. Role je uživateli přiřazena v rámci procesu vytváření uživatelského účtu správcem systému. REST API služba, popsaná v předchozí podkapitole, zajišťuje webové aplikaci data o uživatelích a jejich rolích. Razor Pages webová aplikace pro řízení přístupu ke svým stránkám, členěným do složek, pracuje s politikami. Politika je pravidlo, které určuje, jaké uživatelské role jsou vyžadovány, aby byl uživateli umožněn přístup k obsahu chráněnému danou politikou. `Startup` obsahuje definice dvou různých politik. Administrátorská politika je konfigurována



Obr. 2.10: Zjednodušený diagram tříd a složek zdrojových kódů webové aplikace.

tak, aby povolila přístup pouze uživatelům s rolí administrátora. Je uplatňována pro kontrolu přístupu k modulu administrace v rámci webové aplikace. Zbylé moduly jsou zajištěny uživatelskou politikou. Ta umožní přístup každému subjektu, který má přiřazenou roli administrátora či uživatele.

Ve třídě `Startup` je také přidána podpora Sessions (relací). Protokol HTTPS, který slouží ke komunikaci mezi webovou stránkou zobrazenou v prohlížeči klienta a serverem, je bezstavový. Aby bylo možné v době, kdy je uživatel do webové aplikace přihlášený, uchovávat dočasně některá data, potřebná po celou dobu trvání jeho práce se systémem, existují právě tyto relace. Jakmile uživatel přistoupí k webové aplikaci, je vygenerována Cookie s unikátním identifikátorem relace. S pomocí tohoto identifikátoru umí webová aplikace na straně serveru ukládat data v rámci relace. Tento mechanismus je použit například k uchování JWT tokenu uživatele.

Během přihlášení do systému dochází k volání na autentizační rozhraní REST API služby. Pokud jsou webovou aplikací předány správné přihlašovací údaje, REST API služba vrátí validní JWT token. Ten je poté uložen do dat relace a použit webovou aplikací k autentizaci v rámci budoucího využívání zabezpečených zdrojů REST API služby. Doba, po kterou jsou data relace uchovávána, je konfigurovatelná a činí jednu hodinu. Poté dojde k jejich odstranění.

`Startup` také obsahuje volání metody přidávající REST API kontrolery. Webová aplikace sice většinu klientských HTTPS požadavků obsluhuje prostřednictvím Razor stránek. Avšak v jejích některých modulech je třeba prezentovat i dynamická multimediální data. Jedná se o snímky a videa událostí zaznamenaných kamerovými moduly. Takový obrazový materiál se označuje jako dynamický, protože vzniká průběžně po celou dobu běhu webové aplikace. Pokud kamera detekuje zájmovou událost, vytvoří kromě zprávy ve formátu JSON i krátké video a snímek, zachycující událost. Tato multimediální data jsou pak přenesena díky protokolu SFTP do definované složky na serveru hostujícím webovou aplikaci. K jejich prezentaci slouží REST API kontroler ve třídě `MediaController`. Ten implementuje dvě HTTPS GET metody. `Image` umí s pomocí předaného parametru prohledat zmíněnou složku a vrátit klientovi patřičný snímek. Pokud se ve složce nenachází, tak zástupný snímek s informací o dočasné nedostupnosti daného obrázku. `Video` funguje obdobně pro video soubory. Obě tyto metody provádí validaci vstupů, aby je nebylo možné zneužít k neoprávněnému přístupu k ostatním souborům na serveru. Současně jsou dostupné pouze pro autentizované uživatele systému. Pro zobrazení multimediálních dat je ve výsledné HTML stránce URL odkaz na popsané REST API, kterému je jako parametr předáván název daného obrázku či videa.

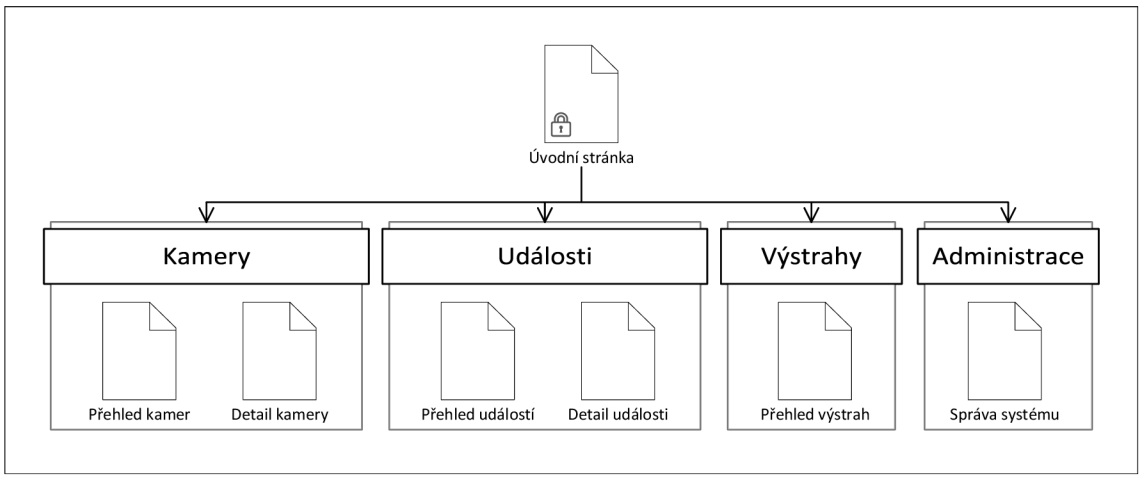
V rámci jmenného prostoru `CassandraGui.Services` se nachází obecná třída `HttpMethodsService<T>`. V jazyce C# se takové třídy používají pro implementaci generalizovaných operací, jež mohou probíhat nad objekty různých datových typů. V tomto konkrétním případě se jedná o třídu s implementací asynchronní metody `ApiPost(object postBody, string uri)` pro komunikaci s REST API službou popsanou v předchozí podkapitole. Této metodě se jako parametr předává C# objekt a URI požadovaného zdroje REST API služby. Objekt je vstupním parametrem pro využití daného zdroje. Aby mohl být přenesen k REST API službě, je provedena jeho konverze do formátu JSON. Následně je vygenerován HTTPS POST, který ve své hlavičce obsahuje JWT token uživatele, pod kterým je vytvořen. V těle zprávy je zapouzdřen JSON objekt. Kontroler na straně REST API provede zpracování žádosti a vrátí odpověď. Jedná se o JSON objekt odpovědi, zapouzdřující další objekty s daty odpovědi. Po jejím přijetí převede popisovaná metoda přijatý JSON na C# objekty. K této konverzi jsou využity třídy jmenného prostoru `CassandraGui.DTO` a také `CassandraGui.Models`. Posledním krokem je předání získaných dat prostřednictvím

návratové hodnoty metody. Jelikož jsou získaná data tvořena objekty různých typů, je tato metoda i třída obecná. A při vytváření její instance je v konstruktoru předána i třída, jež je předlohou návratové hodnoty. Aby bylo možné s objekty odvozenými od této třídy pracovat v rámci zdrojových kódů jednotlivých Razor stránek, jsou její konstruktory s pomocí techniky vkládání závislostí inicializovány ve třídě `Startup`.

Složka `CassandraGui.wwwroot` slouží k uchování statických souborů. Tedy takových, k jejichž změnám po dobu běhu aplikace nedochází. Soubory v ní uložené jsou webovou aplikací automaticky poskytovány internetovému prohlížeči klienta. `wwwroot` je dále členěn na podsložky, obsahující grafiku, zdrojové kódy jazyka JavaScript, soubory stylů CSS a softwarové knihovny. Grafika zahrnuje zejména ikony použité ve webové aplikaci. CSS knihovna Bootstrap je použita k vytvoření konzistentního grafického designu, nicméně vzhled jednotlivých elementů je upraven na míru této konkrétní webové aplikace s pomocí definic kaskádových stylů CSS obsažených v podsložce CSS.

Soubory jazyka JavaScript jsou spouštěny přímo v prohlížeči klienta. Pro účely této webové aplikace je s pomocí nich implementován interaktivní boční navigační panel pro orientaci v aplikaci. Zdrojový kód v případě najetí kurzorem myši na panel rozšíří s využitím animace velikost panelu a k permanentně zobrazeným navigačním ikonám připojí textový popis. JavaScriptem je realizováno také vykreslování grafů, obsažené na stránce detailního přehledu kamery. Ve vlastních zdrojových kódech je k renderování grafů použita knihovna Chart.js. Jak uvádí její dokumentace [34], jedná se o open source knihovnu jazyka JavaScript, dostupnou pod licencí MIT. S její pomocí lze generovat grafy různých druhů, například spojnicový nebo radarový.

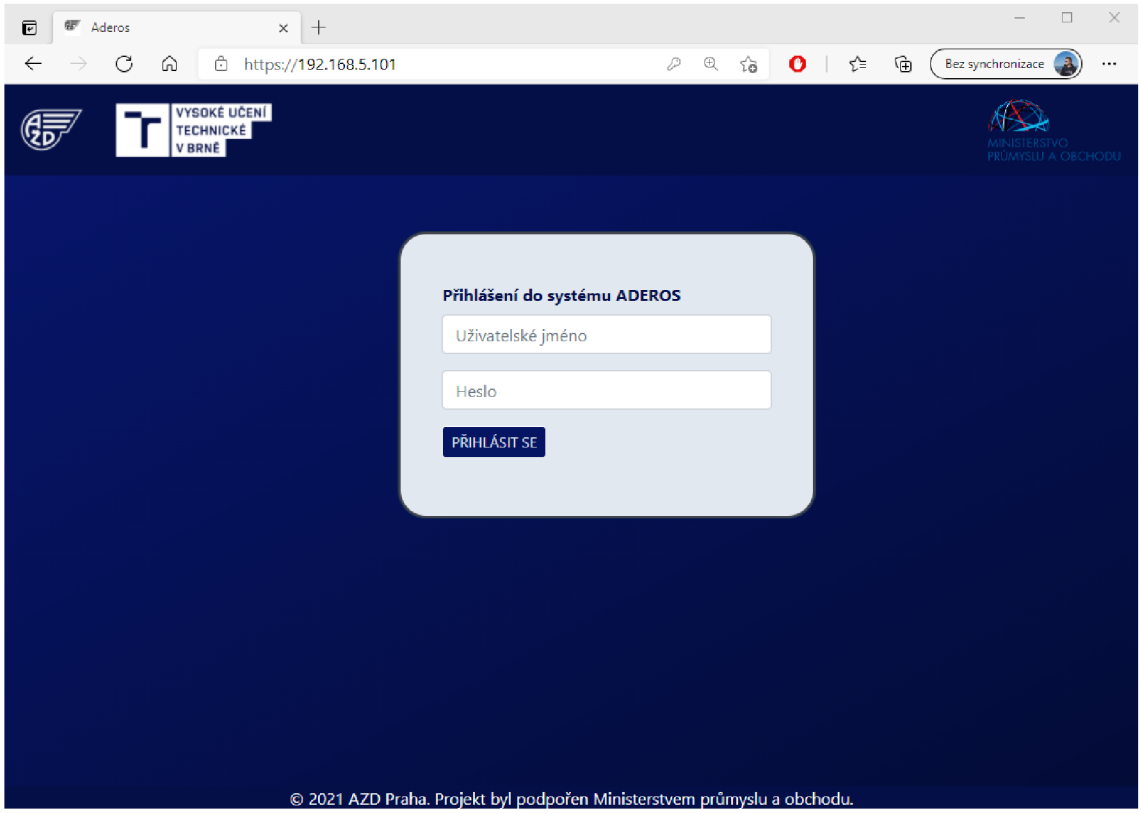
Z pohledu uživatele obsahuje webová aplikace úvodní stránku a dále stránky s obsahem, které prezentuje logicky členěné do modulů. Její schéma je na obrázku 2.11. Technologie Razor renderuje výsledné HTML stránky s pomocí šablon v rámci



Obr. 2.11: Schéma stránek webové aplikace.

souboru stránky. Jelikož jsou některé HTML prvky společné pro více stránek, je možné definovat šablonu. V případě této aplikace je šablona vytvořena a nachází se ve složce `CassandraGui.Pages.Shared`. Implementuje například záhlaví stránky, boční navigační panel nebo spodní informační lištu. Nyní budou popsány funkcionality jednotlivých modulů.

Úvodní stránka je dostupná komukoliv a slouží k autentizaci uživatele. Je zobrazena na obrázku 2.12. Skládá se z formuláře pro zadání uživatelských údajů a přihlášení do systému. Její implementaci obsahuje soubor stránky `Index.cshtml` a model stránky `Index.cshtml.cs`. Po kliknutí na tlačítko přihlášení webová aplikace provede volání na REST API službu, které předá data vyplněná uživatelem do přihlašovacího formuláře. Ta data ověří, pokud se uživatel v databázi nachází, vrátí webové aplikaci validní JWT token. Webová aplikace uloží token v rámci relace, vygeneruje autentizační Cookie a tu vrátí prohlížeči klienta. Současně dojde k jeho přesměrování do samotných zabezpečených modulů webové aplikace. Odhlášení ze systému je implementováno v rámci tlačítka horního navigačního panelu, definovaného ve výše popsané šabloně stránek.



Obr. 2.12: Úvodní stránka pro přihlášení do systému.

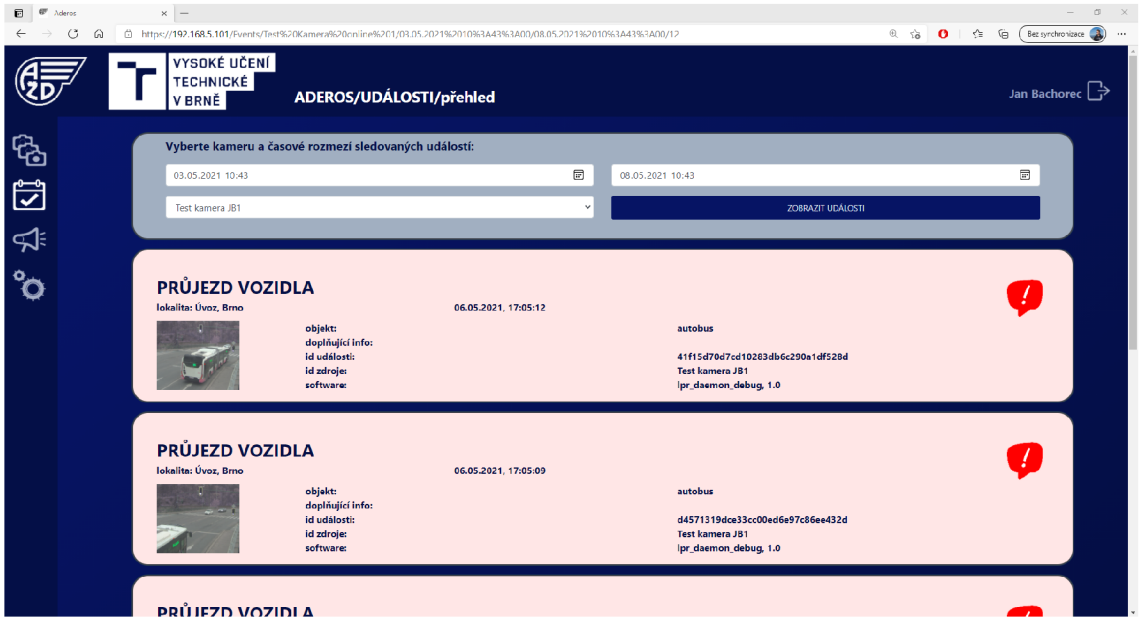
Modul kamery zprostředkovává stav kamer uživateli systému. Jmenný prostor, kde se nachází jeho implementace, se jmenuje `CassandraGui.Pages.Cameras`.

Jsou zde stránky a modely stránek jak souhrnného přehledu kamer, tak i detailu vybrané kamery. **Přehled kamer** zobrazuje základní informace o každé z kamer systému. Webová aplikace jeho prostřednictvím prezentuje data získaná z databáze přes REST API službu. Pro přehlednost je aplikováno stránkování, kdy každá stránka obsahuje informaci až o pěti kamerových modulech. Současně jsou dostupné výběrové filtry, s nimiž lze zobrazit veškeré kamery, nebo pouze ty, co jsou ve stavu připojené. Také je možné kamery řadit, a to dle času poslední komunikace, nebo abecedně na základě názvu. Element reprezentující kameru v přehledu kamer je možné rozkliknout. Uživatel je poté přesměrován na stránku **detail kamery**. Ta má za úkol, opět díky využití zdrojů REST API služby, poskytnout operátorovi systému detailní přehled jednak o aktuálním stavu kamery, jednak o historickém vytížení jejích hardwarových zdrojů. K tomu je využito různých grafů vytížení procesoru, grafické karty nebo třeba stavu baterie, zobrazeného na obrázku 2.13. A také radarového diagramu se souhrnným přehledem všech aktuálních naměřených hodnot o využití hardwaru. Uživatel zde současně může nastavit kameře uživatelsky definovaný název pro její snažší identifikaci v ostatních přehledech.

Obr. 2.13: Graf historického využití baterie na stránce detailu kamery.

Modul události informuje operátora o veškerých událostech zaznamenaných kamerami. Tedy i těch, které nebyly kamerou klasifikovány jako výstraha. Soubory a modely stránek přehledu a detailu událostí se nachází ve jmenném prostoru `CassandraGui.Pages.Events`. **Přehled událostí** je na obrázku 2.14. Operátorovi umožňuje procházet události a vybírat si z nich ty zájmové. Pro zjednodušení a zpřehlednění celé operace se zde nachází filtr. S pomocí něj uživatel specifikuje kame-

rový modul, jímž zaznamenané události mají být zobrazeny. Dalším parametrem filtru je sledovaný časový interval. K jeho zadání si operátor může zobrazit kalendář a požadované datum v něm vybrat. Po stisku tlačítka pro zobrazení událostí dojde k prezentaci událostí splňujících podmínky definované filtrem. Tyto události jsou stránkovány po maximálně pěti záznamech na jednu stránku. Každý HTML element reprezentující událost je aktivní. Jeho stisknutím lze přejít na zobrazení detailu události.



Obr. 2.14: Přehled událostí v rámci modulu události.

Detail události umožňuje operátorovi vhodnou analýzu zaznamenané události. Nachází se zde veškeré dostupné informace o incidentu, a to včetně obrazových podkladů. V rámci detailu události je uvedena kamera, která je jejím zdrojem. Název této kamery funguje jako odkaz. Uživatel se tak jednoduše z detailu události dostane na detail kamery a může si prohlédnout její aktuální stav i stav v době detekce. Jestliže se z detailu kamery chce operátor dostat zpět na detail události, může k tomu použít příslušné tlačítko zpět. K přehrávání videa události se používá HTTPS GET metoda `Video`, popsaná v odstavci pojednávajícím o třídě kontroleru a samotné video je poskytováno prohlížeči klienta jako proud dat.

Implementaci **modulu výstrahy** pro prezentaci výstrah, tedy událostí klasifikovaných jako událost s výstrahou, zajišťuje jmenný prostor zvaný `CassandraGui.Pages.Alerts`. Nachází se zde Razor stránka **přehled výstrah**. Jejím úkolem je prezentovat definovaný počet nejnovějších výstrah ze všech dostupných kamerových modulů. Pro omezení zobrazených výsledků může operátor využít dvou filtrů. Prvním z nich je filtr zdroje. V něm lze definovat sledovanou kameru,

či vybrat sledování všech dostupných kamer. Druhým je pak filtr závažnosti. Dopravní kamera každé události přiřadí stupeň závažnosti a uživatel si na této stránce může vybrat, zda bude sledovat všechny výstrahy bez ohledu na stupeň závažnosti, nebo zda ho zajímají pouze ty s například středním či vyšším stupněm rizika. Přehled výstrah je stránkovaný s maximálně pěti záznamy na stránku. Každý HTML element pro reprezentaci výstrahy obsahuje její základní informace a náhledový snímek, pokud je dostupný. Při rozkliknutí prvku výstrahy je uživatel přesměrován na zobrazení jejího detailu.

Pro správu systému Aderos slouží modul **Administrace**. Ten obsahuje stránku **správa systému**. Její implementace je řešena soubory v rámci jmenného prostoru `CassandraGui.Pages.Administration`. Tento modul je dostupný pouze uživatelům s administrátorskou rolí. Běžným uživatelům přístupný není a v bočním navigačním panelu se jim jeho ikona ani nezobrazuje. V rámci správy systému je možné přidávat, editovat a mazat uživatele.

`CassandraGui.DBValuesPairs` je jmenný prostor s pomocnými třídami, jejichž úkolem je provádět překlad dat získaných prostřednictvím REST API služby z databáze. Některá data z kamer jsou totiž reprezentována ve formátech, v nichž je není možné z důvodu přehlednosti či jazykové lokalizace ve webové aplikaci zobrazit přímo. Namísto toho dochází s pomocí metod popisovaných tříd k jejich překladu do snadno čitelné podoby a následné prezentaci v rámci HTML stránky.

2.3.3 Vývojové prostředí a testování

K vývoji a testování webové aplikace a REST API služby bylo nezbytné modifikovat virtualizované vývojové prostředí, realizované v rámci vývoje serverového řešení. Jeho architekturu znázorňuje obrázek 2.4. Úprava prostředí spočívala v dodatečném nasazení webové aplikace i REST API služby na server IIS Express. Tento krok byl realizován s pomocí nástrojů vývojového prostředí Visual Studio. Následně bylo nutné s pomocí Postman a serverové služby přidat do databáze testovací hodnoty zpráv událostí a zpráv o stavu kamery.
Samotné testování probíhalo v tomto případě manuálně. Webová aplikace ani API služba nebude v praxi pod takovou zátěží, jako serverové řešení, které bude neustále dostávat zprávy z kamer. Proto se zde testování zaměřovalo zejména na hledání chyb ve webové aplikaci s pomocí různých testovacích scénářů. A také na manuální vytváření dotazů na REST API službu a následnou kontrolu správnosti dat odpovědí na tyto dotazy.

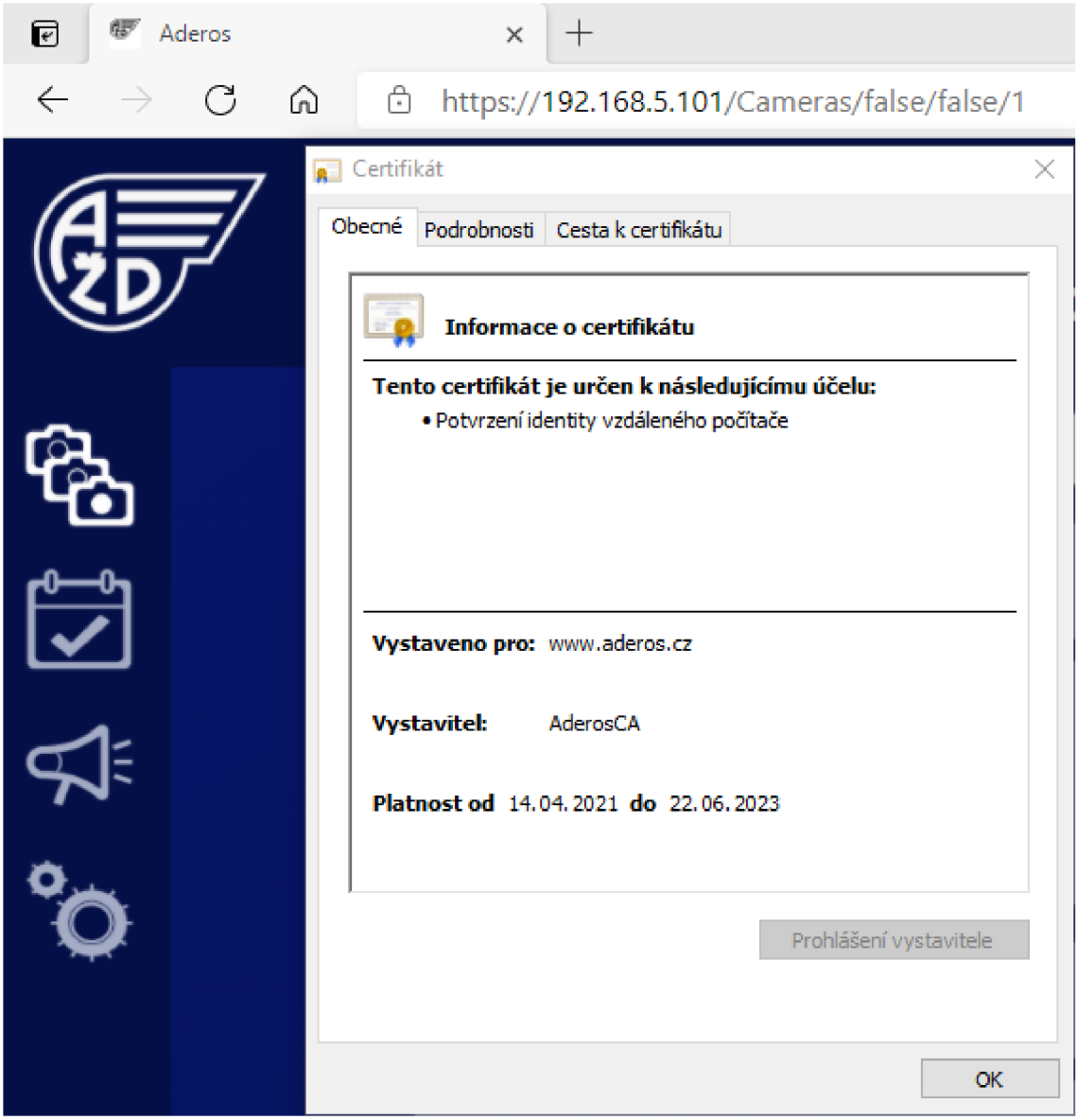
2.4 Nasazení řešení na experimentální pracoviště

Součástí této diplomové práce je i realizace experimentálního pracoviště. Jeho účelem je demonstrovat funkčnost vyvinutého softwaru v praxi a umožnit tak analýzu dopravních incidentů, zachycených testovacími kamerovými moduly. K realizaci experimentálního pracoviště byl využit nástroj ESXi, jenž běží na fyzických serverech na FEKT VUT v Brně. Dle zdroje [35] jde o hypervizor prvního typu. Tedy o software, umožňující na jednom fyzickém hostitelském stroji spouštět a spravovat více virtualizovaných počítačů, který běží přímo na hostitelském hardwaru.

Do ESXi byly nasazeny dvě virtuální stanice s operačním systémem Ubuntu 18.04 LTS. K dispozici mají 8 GB operační paměti a dvě jádra procesoru. Na nich běží databáze Cassandra, jež je součástí softwarového balíčku Elassandra 6.2.3.31. V rámci databázového clusteru byl s pomocí vyvinutých skriptů vytvořen keyspace, schéma tabulek a potřebné materializované pohledy databáze. Dále byly upraveny konfigurační soubory databáze, aby mezi sebou jednotlivé uzly správně komunikovaly.

Dále je v ESXi vytvořena virtuální stanice s operačním systémem Windows 10 verze 20H2, dvěma procesorovými jádry a deseti GB paměti RAM. Na ní je instalován webový server IIS 10. V rámci něj běží veškerý vyvinutý software. Serverová služba ke sběru dat z kamer, REST API služba poskytující data z databáze webové aplikaci a samotná webová aplikace pro grafické prezentovaní dat operátorovi systému.

V rámci komunikace mezi kamerou a serverovou službou se uplatňuje protokol HTTPS s ověřováním klientských certifikátů. HTTPS se používá i pro přístup k webové aplikaci. Z tohoto důvodu bylo nezbytné vytvořit základní PKI (Public Key Infrastructure) pro generování certifikátů potřebných za účelem ověření identity komunikujících stran v rámci protokolu HTTPS. Ke generování certifikátů byl využit nástroj XCA - X Certificate and key management [36]. Jde o aplikaci, která jako kryptografické jádro využívá známou knihovnu openSSL a umožňuje vytvářet a spravovat certifikáty. Jako šifrovací algoritmus byl zvolen RSA s délkou klíčů 2048 bitů. Nejprve byl vytvořen kořenový certifikát pro certifikační autoritu. S jeho tajným klíčem pak došlo k podpisu veškerých dalších certifikátů, tedy certifikátu webového serveru i certifikátu kamery. Následně byly certifikáty nasazeny do IIS i do modulu kamery, a došlo k úspěšnému zahájení testovacího provozu. Aby se operátor mohl bezpečně připojit do webového uživatelského rozhraní, je nezbytné do jeho počítače přidat mezi certifikáty důvěryhodných certifikačních autorit i certifikát vytvořené kořenové certifikační autority. Na snímku 2.15 je zobrazen webový prohlížeč klienta, který je do webové aplikace bezpečně připojen přes HTTPS, přičemž k autentizaci webového serveru byl úspěšně použit jeho certifikát.

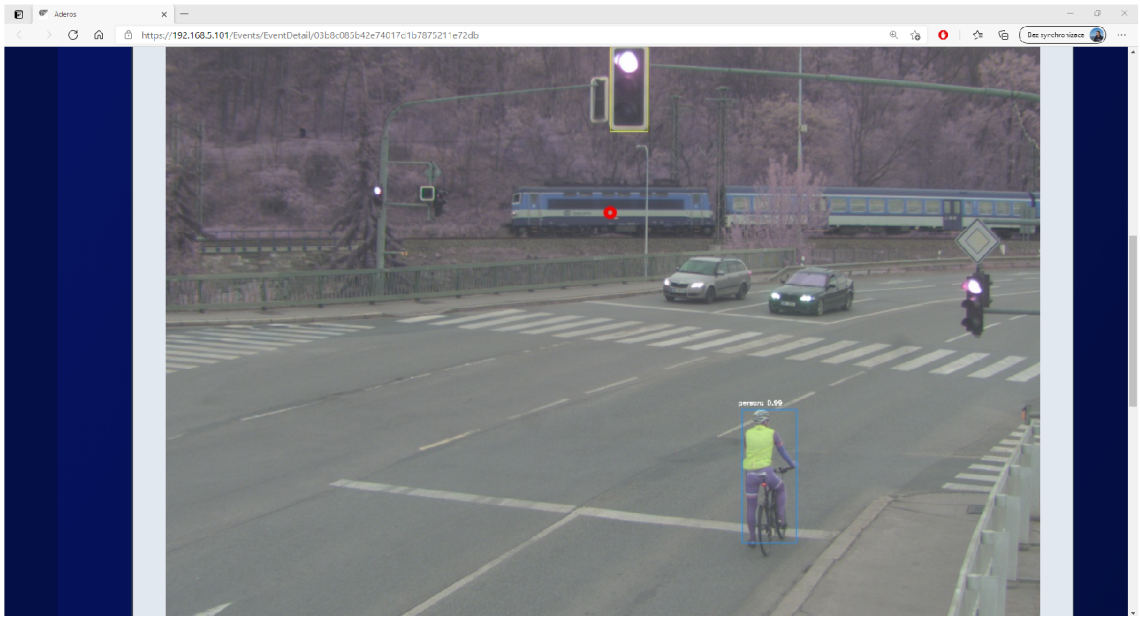


Obr. 2.15: HTTPS připojení klienta do vytvořené webové aplikace.

Multimediální data jsou na server posílána prostřednictvím SFTP. Proto byl na stanici s Windows 10 zapnut OpenSSH server, jenž je nově součástí tohoto operačního systému. Následně byla vytvořena složka pro příjem obrázků a videí. Dále došlo k vytvoření uživatele pro přihlašování kamerových modulů v rámci přidávání dat do složky. Tomu byla odebrána s pomocí správce uživatelských politik práva pro přihlašování do systému a jeho přístup byl z důvodů bezpečnosti omezen pouze na zmiňovanou multimediální složku.

Kamerový modul, jenž slouží jako zdroj dat pro vytvořené experimentální pracoviště, je fyzicky umístěn na dopravní světelné křižovatce na ulici Křižíkova v Brně. Jde o testovací provoz kamery, během nějž jsou však kamerou rozpoznávány reálné

události. Zprávy o nich předává kamera s pomocí svého softwarového a hardwarového vybavení přes bezpečný protokol HTTPS vlastní serverové službě. Současně kamera serveru poskytuje i multimediální data, tedy fotky a videa detekovaných dopravních událostí. Obrázek 2.16 je snímkem grafického rozhraní vytvořené webové aplikace. Konkrétně jde o zobrazení obrazového záznamu události v rámci modulu události a stránky detailu události.



Obr. 2.16: Snímek detekované události v rámci detailu události.

Závěr

Hlavním cílem diplomové práce bylo navrhnout a implementovat serverové řešení a grafické uživatelské rozhraní pro autonomní systém detekce rizikových dopravních situací na železničním přejezdu. Grafické rozhraní mělo mít formu interaktivní webové stránky a umožnit operátorovi systému analyzovat zaznamenané rizikové situace. Serverové řešení by mělo sloužit pro sběr a uchovávání dat zaznamenaných připojenými dopravními kamerami. Mezi další cíle patřilo seznámení se s technologiemi pro tvorbu jak serverového řešení, tak i interaktivní webové aplikace.

Všechny stanovené cíle práce byly splněny. Bylo vytvořeno serverové řešení pro sběr, zpracování a uchovávání dat z autonomních kamerových modulů. To se skládá ze serverové služby a databáze. Služba je implementována s pomocí .NET Core a programovacího jazyka C#. Komunikace mezi kamerou a službou probíhá prostřednictvím REST API přes protokol HTTPS, přičemž je vyžadována z důvodu bezpečnosti autentizace každé z komunikujících stran. A to prostřednictvím certifikátů. Databáze je realizována clusterovou technologií Cassandra, přičemž bylo navrženo kompletní schéma potřebných tabulek a materializovaných pohledů.

Dále vznikla vlastní interaktivní webová stránka, realizovaná s pomocí následujících komponent. REST API služby, postavené na technologii .NET Core, psané v jazyce C#. A také webové aplikace, jež kromě zmíněných technologií navíc používá ke generování samotných HTML stránek i Razor Pages. REST API služba implementuje veškeré logické operace, tedy například práci s databází, pro potřeby webové aplikace. Ta má pak za úkol data získaná od REST API služby graficky prezentovat a umožňovat uživateli, připojenému přes HTTPS, jak analyzovat zaznamenané rizikové situace na železničním přejezdu, tak i provádět konfiguraci systému a spravovat připojené dopravní kamery. Tento modulární přístup, kdy je oddělená logická vrstva od uživatelského rozhraní, byl zvolen z důvodů lepší přehlednosti zdrojových kódů a větších možností případného budoucího rozšíření aplikace.

Vytvořené řešení bylo nasazeno na experimentální pracoviště FEKT VUT. Jeho funkčnost byla úspěšně ověřena. Experimentální pracoviště umožňuje prostřednictvím vlastní interaktivní webové aplikace a serverového řešení operátorovi systému provádět analýzu dat zaznamenaných kamerovým modulem, umístěným v Brně na světelné křižovatce na ulici Křižíkova. Současně byly v teoretické části diplomové práce rozebrány technologie jak pro tvorbu uživatelského rozhraní, tak i serverového řešení.

Literatura

[1] *Ředitelství silnic a dálnic zahájí v červenci Celostátní sčítání dopravy. In: Ministerstvo dopravy [online]. 2020, 24. 6. 2020 [cit. 2020-10-09].* Dostupné z: <https://www.mdcr.cz/Media/Media-a-tiskove-zpravy/Reditelstvi-silnic-a-dalnic-zahaji-v-cervenci-Celo>.

[2] *Statistika nehodovosti 2019. In: Policie [online]. 2019 [cit. 2020-10-09].* Dostupné z: <https://www.policie.cz/clanek/statistika-nehodovosti-900835.aspx?q=Y2hudW09Mg%3d%3d>.

[3] *Železniční přejezdy. Besip - Statistiky nehodovosti v Ceske republice [online]. 2019, , 28 [cit. 2020-10-09].* Dostupné z: <https://www.ibesip.cz/getattachment/Statistiky/Statistiky-nehodovosti-v-Ceske-republice/Dopravni-nehodovost-v-roce-2019/Zeleznicni-prejezdy/Zeleznicni-prejezdy.pdf>.

[4] *FREEMAN, Jonathan. What is an API? In: InfoWorld [online]. 2019 [cit. 2020-10-10].* Dostupné z: <https://www.infoworld.com/article/3269878/what-is-an-api-application-programming-interfaces-explained.html>.

[5] *What is REST? In: Codecademy [online]. [cit. 2020-10-10].* Dostupné z: <https://www.codecademy.com/articles/what-is-rest>.

[6] *HTTP Headers. In: Developer Mozilla [online]. [cit. 2020-10-11].* Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers>.

[7] *JSON. In: Json.org [online]. [cit. 2020-10-11].* Dostupné z: <https://www.json.org/json-en.html>.

[8] *Úvod do technologie .NET. In: Docs Microsoft [online]. 28. 9. 2020 [cit. 2020-10-11].* Dostupné z: <https://docs.microsoft.com/cs-cz/dotnet/core/introduction>.

[9] *Rest Security. In: OWASP [online]. [cit. 2021-04-03].* Dostupné z: <https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html>.

[10] *Transport Layer Security. In: Www.cloudflare.com [online]. [cit. 2021-04-03].* Dostupné z: <https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/>.

[11] *SMITH, Steve. Architect Modern Web Applications with ASP.NET Core and Azure. Microsoft Documentation [online]. 2021 [cit. 2021-04-05].* Dostupné z: <https://docs.microsoft.com/cs-cz/dotnet/architecture/modern-web-apps-azure/>.

[12] *An Introduction To Razor Pages. Learn Razor Pages [online]. [cit. 2021-04-05].* Dostupné z: <https://www.learnrazorpages.com/>.

[13] *Bootstrap. In: Bootstrap [online]. [cit. 2021-04-05].* Dostupné z: <https://getbootstrap.com/docs/5.0/getting-started/introduction/>.

[14] *Kafka Documentation. In: Kafka Apache [online]. [cit. 2020-10-30].* Dostupné z: <https://kafka.apache.org/documentation/#connectapi>.

[15] *CARTER, Michael. Apache Kafka Architecture. In: Instaclustr [online]. [cit. 2020-10-30].* Dostupné z: <https://www.instaclustr.com/apache-kafka-architecture/>.

[16] *What is Confluent Platform? In: Confluent [online]. [cit. 2020-10-31].* Dostupné z: <https://docs.confluent.io/current/platform.html>.

[17] *Apache Avro. Apache Avro [online]. [cit. 2020-10-31].* Dostupné z: <http://avro.apache.org/docs/current/>.

[18] *Apache Spark [online]. [cit. 2020-11-01].* Dostupné z: <https://spark.apache.org/>.

[19] *What is a database? In: Https://www.oracle.com [online]. [cit. 2020-10-16].* Dostupné z: <https://www.oracle.com/database/what-is-database.html>.

[20] *Co je datový sklad? In: Oracle [online]. [cit. 2020-10-17].* Dostupné z: <https://www.oracle.com/cz/database/what-is-a-data-warehouse/>.

[21] *NoSQL. In: Searchdatamanagement [online]. [cit. 2020-10-16].* Dostupné z: <https://searchdatamanagement.techtarget.com/definition/NoSQL-Not-Only-SQL>.

[22] *What is NoSQL? In: Https://www.oracle.com [online]. [cit. 2020-10-16].* Dostupné z: <https://aws.amazon.com/nosql/>.

[23] *KNIGHT, Michelle. What is a Wide Column Database? In: Dataversity [online]. August, 2017 [cit. 2020-10-17].* Dostupné z: <https://www.dataversity.net/wide-column-database/>.

[24] *Apache Cassandra. In: Apache Cassandra [online]. [cit. 2020-10-17].* Dostupné z: <https://cassandra.apache.org/doc/latest/architecture/overview.html#features>.

[25] *Cassandra Architecture. In: Guru99 [online]. [cit. 2020-10-24].* Dostupné z: <https://www.guru99.com/cassandra-architecture.html#1>.

[26] *What is elasticsearch. In: Elastic.co [online]. [cit. 2020-10-28].* Dostupné z: <https://www.elastic.co/what-is/elasticsearch>.

[27] *Architecture. In: Medium [online]. [cit. 2020-10-17].* Dostupné z: <https://elassandra.readthedocs.io/en/latest/architecture.html>.

[28] *MS IIS. Microsoft Documentation [online]. [cit. 2020-11-27].* Dostupné z: <https://bit.ly/3vXig0y>.

[29] *Ubuntu. Ubuntu [online]. [cit. 2020-11-27].* Dostupné z: <https://ubuntu.com/#download>.

[30] *Operating Cassandra. Cassandra Documentation [online]. [cit. 2020-11-27].* Dostupné z: <https://cassandra.apache.org/doc/latest/operating/index.html/>.

[31] *VirtualBox [online]. [cit. 2020-11-27].* Dostupné z: <https://www.virtualbox.org/>.

[32] *Postman [online]. [cit. 2020-11-27].* Dostupné z: <https://www.postman.com/>.

[33] *Get started with Swashbuckle and ASP.NET Core. In: Microsoft Docs [online]. 26.6.2020 [cit. 2021-04-11].* Dostupné z: <https://bit.ly/3vYiV1p>.

[34] *Chart.JS [online]. [cit. 2021-04-15].* Dostupné z:
<https://www.chartjs.org/>

[35] *ESXi. In: Wmware [online]. [cit. 2021-04-17].* Dostupné z:
<https://www.vmware.com/products/esxi-and-esx.html>

[36] *XCA - X Certificate and key management documentation. [online]. [cit. 2020-08-11].* Dostupné z:
<https://www.hohnstaedt.de/xca/index.php/documentation/manual>.

Seznam symbolů, veličin a zkratek

API	Application Programming Interface
CLR	Common Language Runtime
CSS	Cascading Style Sheets
CQL	Cassandra Query Language
CRUD	Create, Read, Update, Delete
DBMS	Database Management System
GC	Garbage Collector
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTPS	Hypertext Transfer Protocol Secure
HTTP	Hypertext Transfer Protocol
JIT	Just In Time
JWT	JSON Web Token
JSON	JavaScript Object Notation
MVC	Model View Controller
RAM	Random Access Memory
REST	REpresentational State Transfer
SFTP	Secure File Transfer Protoco
SPA	Single Page Application
SQL	Structured Query Language
SSH	Secure Shell
TCP	Transmission Control Protocol
TLS	Transport Layer Security
URL	Uniform Resource Locator

A Vytvořené manuály

A.1 Instalační manuál

Instalace serverového řešení a interaktivní webové stránky se dělí na dvě základní části. První z nich je instalace a konfigurace databázového clusteru Cassandra. Druhou pak nasazení vyvinutých aplikací na webový server a jejich konfigurace. Základní softwarové a hardwarové vybavení, nezbytné pro instalaci systému, je:

- Stroj s operačním systémem Windows 10 nebo Windows Server 2016 a novějším. Tato stanice by měla mít dispozici alespoň dvě jádra procesoru a 8 GB RAM. V operačním systému je třeba mít zapnutý webový server IIS.
- Dva a více strojů s operačním systémem Ubuntu verze 18.04 nebo novějším. Každý z nich by měl disponovat alespoň osmi GB RAM a minimálně dvěma jádry procesoru. Na každém z nich je třeba mít nainstalovaný Java 8 JDK (Java Development Kit), případně open source variantu implementace Javy OpenJDK, taktéž ve verzi 8.

Instalace a konfigurace databázového clusteru:

1. Na každou z Ubuntu stanic je potřeba stáhnout balíček Elassandra, který obsahuje databázi Cassandra a Elasticsearch. Balíček je dostupný na tomto odkazu: `https://github.com/strapdata/elassandra/releases`
2. Na každé Ubuntu stanici Elassandra archiv rozbalit do vhodné složky. Jeho rozbalením vznikne složka `/elassandra-verze`. Toto je domovská složka databázového systému.
3. Na všech Ubuntu stanicích editovat soubor `/etc/environment` přidáním řádku `CASSANDRA_HOME="/cesta/elassandra-verze`.
4. Na každé Ubuntu stanici provést konfiguraci Cassandry prostřednictvím editace souboru `/cassandra-verze/conf/cassandra.yaml`:
   - `cluster_name: 'názevClusteru'`
   - `seeds: 'ipAdresaSeedStanice'`
   - `listen_address: ipKonfigurovanéStanice`
   - `rpc_address: ipKonfigurovanéStanice`
   - `enable_materialized_views: true`
5. Provést restart všech Ubuntu stanic pro načtení proměnné `CASSANDRA_HOME`.
6. Přepnout se do domovské složky Cassandry na každé Ubuntu stanici a databázový uzel spustit příkazem: `bin/cassandra -e`.
7. Na jedné libovolné stanici Ubuntu se přepnout do domovské složky Cassandry a s pomocí příkazu `bin/cqlsh ipStanice` se připojit do CQL konzole pro práci s databází.

8. Posledním krokem je spuštění veškerých skriptů pro generování schématu databáze prostřednictvím CQL konzole. Tyto skripty jsou dodávány v rámci instalačních souborů tohoto systému.
9. Pro kontrolu funkčnosti databázového clusteru slouží příkaz `bin/nodetool status`, spouštěný z domovské složky Cassandry.

Nasazení aplikací na IIS v rámci Windows stanice:

1. Stáhnout a nainstalovat .NET Core hostující sadu ve verzi 3.1 a .NET Core SDK (Software Development Kit) také ve verzi 3.1.
2. Na Windows nakopírovat složky se soubory serverové služby, REST API služby a webové aplikace a pro tyto složky nastavit oprávnění pro přístup. Popisované složky jsou dodávány v rámci instalačních souborů systému.
3. V rámci složky serverové služby se nachází soubor `app.config`, v tomto souboru je potřeba nastavit IP adresu jednoho z uzlů Cassandra clusteru.
4. Nyní je třeba všechny tři aplikace nasadit do IIS. Dále je třeba do IIS přidat PKI certifikát serveru. Tento certifikát je použit pro konfiguraci HTTPS v rámci všech tří aplikací. Při nastavování vazeb webů, tedy adres a portů, pod kterými jsou weby dostupné, je nezbytné pro webovou aplikaci zvolit standardní HTTPS port 443, ostatní služby mohou běžet na libovolném volném TCP portu, avšak také přes protokol HTTPS. Pro serverovou službu je nutné HTTPS nastavit takovým způsobem, aby bylo vyžadováno i ověření klientského certifikátu.
5. V rámci správce služeb systému Windows je potřeba zapnout OpenSSH server. Dále je třeba vytvořit omezeného uživatele systému Windows, kterému jsou s pomocí editoru místní zásady zabezpečení odebrány práva pro lokální i vzdálené přihlašování do Windows.
6. Následujícím krokem je vytvoření složky pro nahrávání multimediálních dat z kamer. Přístup do této složky je omezen na výše popisovaného systémového uživatele. Jeho účet slouží kamerám pro připojení do této složky a k následnému přenosu jejich obrazových dat.
7. Konfigurace REST API služby se provádí v její domovské složce v souboru `appsettings.json`. Zde je potřeba nastavit správnou IP adresu jednoho z uzlů databázového clusteru.
8. Nastavení webové aplikace se provádí v její domovské složce úpravou souboru appsettings.json. Zde je třeba nastavit adresu REST API služby, aby s ní mohla webová aplikace komunikovat. A také upravit cestu ke sdílené složce s multimediálními daty z kamer.
9. Posledním krokem je konfigurace firewallu. Ten musí obsahovat pravidla, která umožní TCP komunikaci jak na portu 443 pro potřeby připojení klienta do webové aplikace, tak i na portu, jenž je přiřazen serverové službě pro komuni-

kaci s kamerou. Následuje spuštění serverové služby, webové aplikace i REST API služby v rámci IIS. Nyní server přijímá data z připojených kamer, které jsou vybaveny patřičným certifikátem. Současně se klient může připojit a sledovat tato data v přehledné webové aplikaci.

A.2 Uživatelský manuál

Operátor se k interaktivní webové stránce pro analýzu zaznamenaných dopravních incidentů může připojit zadáním IP adresy serveru, na kterém je webová aplikace hostovaná, do webového prohlížeče. Vhodné je použít například Microsoft Edge nebo Google Chrome. Poté je mu zobrazena úvodní stránka s oknem pro přihlášení. Po úspěšném přihlášení dojde k přesměrování uživatele do modulu kamery. Nyní je možné procházet jednotlivé moduly systému a pracovat s nimi. Podrobnější popis jednotlivých funkcionalit se nachází v kapitole 2.3.2 o webové aplikaci. Elektronická příloha diplomové práce obsahuje video, v rámci kterého jsou demonstrovány funkce webové aplikace a ukázány základní komponenty vytvořeného systému.