Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra kybernetiky

Bakalářská práce

Virtuální laboratoře pro automatické řízení na základě nových webových standardů

Plzeň 2018 Michal Vraštil

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2017/2018

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Michal VRAŠTIL**
Osobní číslo:	**A15B0253P**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Počítačové řízení strojů a procesů**
Název tématu:	**Virtuální laboratoře pro automatické řízení na základě nových webových standardů**
Zadávající katedra:	**Katedra kybernetiky**

Zásady pro vypracování:

1. Analyzujte aktuální stav vývoje webových technologií, vyberte vhodné pro implementaci virtuálních laboratoří.
2. Vytvořte platformu na bázi architektury klient-server, která umožní efektivně spouštět virtuální laboratoře také na všech typech mobilních zařízení.
3. Vyviňte postup pro integraci SW modulů a výpočetních knihoven z prostředí Matlab a jazyka Java.
4. Vyviňte postup pro komunikaci virtuálních laboratoří se systémem REX, případnou integraci uživatelských rozhraní přímo do návrhových prostředků systému REX.
5. Ověřte možnosti provozování laboratoře na jednom počítači, tj. off-line bez výpočetního serveru.
6. Otestujte vyvinutou platformu na vybrané virtuální laboratoři.

Rozsah grafických prací: **dle potřeby**

Rozsah kvalifikační práce: **30-40 stránek A4**

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

1. Dokumentace funkčních bloků systému REX, REX Controls, http://www.rexcontrols.cz
2. Reitinger, J., Čech, M., Schlegel, M., Balda, P. New tools for teaching vibration damping concepts: ContLab.eu (2014) IFAC Proceedings Volumes (IFAC-PapersOnline), 19, pp. 10580-10585.
3. Reitinger, J., Cech, M., Goubej, M. Advanced input shaping filter 3D virtual laboratory (2013) Proceedings of the 2013 International Conference on Process Control, PC 2013, art. no. 6581465, pp. 528-533.
4. Severa, O., Čech, M., Balda, P. New tools for 3D HMI development in Java (2011) Proceedings of the 2011 12th International Carpathian Control Conference, ICCC'2011, art. no. 5945876, pp. 342-346.
5. Balda, P.; Čech, M. Java interface to REX control system. In Process control 2006. Pardubice: Technical University, 2006. s. 1-8. ISBN 80-7194-860-8.
6. Schlegel, M.; Čech, M. Internet PID controller design: www.pidlab.com. In IBCE '04. Grenoble: ENSIEG , 2004. s. 1-6.

Vedoucí bakalářské práce: **Ing. Martin Čech, Ph.D.**
Katedra kybernetiky

Datum zadání bakalářské práce: **1. listopadu 2017**

Termín odevzdání bakalářské práce: **18. května 2018**

Doc. Dr. Ing. Vlasta Radová
děkanka

L.S.

Prof. Ing. Josef Psutka, CSc.
vedoucí katedry

Prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne 16. května 2018

Michal Vraštil

Poděkování

Chtěl bych poděkovat svému vedoucímu bakalářské práce panu Ing. Martinu Čechovi, Ph.D. za jeho pomoc, odborné rady a připomínky při zpracování této práce. Dále bych chtěl poděkovat svým rodičům a přítelkyni za obětavou pomoc a podporu, kterou mi při psaní této práce poskytli.

Objectives

Presented bachelor thesis deals with the design and creation of a client-server platform for the implementation of a virtual laboratory for automatic control. The main objectives of the thesis were to create client-server architecture for virtual laboratory. The first chapter deals with the overview of selected and used technologies in the field of web and desktop technologies. The second chapter describes the implementation of the client-server architecture and applied design patterns as MVC or IoC/DI. The third chapter deals with the implementation of a Java computing server for the integration of existing computational libraries in Java and Matlab. The fourth chapter describe the virtual lab communication with the REXYGEN system, respectively how it searches in the REXYGEN web REST Api. In the last fifth chapter, the entire architecture is verified at a selected virtual lab, and the process of further development is described.

Key words

virtual laboratory, automatic control, ASP.NET core, Java, MySQL, Matlab, REXYGEN, web application, TCP/IP server, REST Api, client-server architecture, HTML, CSS, JavaScript, BootStrap, jQuery, responsive web

Anotace

Předkládaná bakalářská práce se zabývá návrhem a vytvořením platformy klient-server pro virtutální laboratoř pro automatické řízení. Hlavním cílem práce je vytvoření vzorové zjednodušené architektury virtuální laboratoře a ověření několika metod a programovacích postupů pro budoucí práci v této oblasti na Katedře kybernetiky. První kapitola se zabývá přehledem použitých technologií z oblasti webových a desktopových technologií. Ve druhé kapitole je popsána samotná realizace architektury klient-server a jsou představeny použité návrhové vzory jako MVC nebo IoC/DI. Třetí kapitola se zabývá vytvořením výpočetního serveru v jazyce Java pro integraci již existujících výpočetních knihoven v jazyce Java a Matlab. Čvtrtá kapitola řeší komunikaci virtuální laboratoře se systémem REXYGEN, resp. vyhledávání ve webovém REST Api programu REXYGEN. V poslední páté kapitole je celá architektura ověřena na vybrané virtuální laboratoři.

Klíčová slova

virtuální laboratoř, automatické řízení, ASP.NET core, Java, MySQL, Matlab, REXYGEN, webová aplikace, TCP/IP server, REST Api, architektura klient-server, HTML, CSS, JavaScript, BootStrap, jQuery, responzivní web

Obsah

1 Úvod

Webové technologie zažívají v posledních několika letech nebývalý boom způsobený především zkvalitněním, rozšířením a zlevněním poskytovaného internetového připojení a zvýšením dostupnosti osobních počítačů pro širokou veřejnost. Spolu s tímto trendem můžeme pozorovat velký přesun běžných aktivit, jako nakupování, obchodování, zábava nebo komunikace na internet. Příkladem budiž sociální sítě jako Facebook, Google+ nebo Twitter, velké e-shopy jako Amazon nebo E-bay, burzovní obchodování nebo dokonce čistě virtuální měny, jako bitcoin. Díky tomu došlo k velkému technologickému pokroku v rámci webových technologií a postupem času se i dříve pouze desktopové aplikace přesouvají na internet. Například balík Microsoft Office je již přístupný i přes prohlížeč. Aplikace pro řízení podniků nebo konfigurační části systémů pro řízení výrobních linek se též přesouvají do nejrůznějších webových Api nebo přímo do webových aplikací. Dále pak i bioinformatické výpočetní systémy například RepeatExplorer pro analýzu genetických sekvenčních dat je plně implementován jako webová aplikace. [16]

Na tento trend se snaží reagovat i předkládaná bakalářská práce. Hlavním cílem je vyzkoušet a ověřit postupy a dostupné prostředky pro realizaci webové virtuální laboratoře přístupné pouze přes prohlížeč. To s sebou přináší řadu výhod jako multiplatformita, možnost spouštět na velkém množství nejrůznějších mobilních zařízení jako tablety nebo chytré telefony, minimální zatížení hardware daného zařízení a možnost komunikace s celým obsáhlým webovým ekosystémem, například autentizace uživatelů pomocí účtů na sociálních sítích. Další výhodou je možnost připojit se přímo na běžící proces, autentizovat se a provádět měření, nebo změny parametrů v reálném čase. Nebo například možnost načíst data a spustit náročný výpočet bez nutnosti asistovat u výpočetního zařízení, jelikož konec výpočtu je ohlášen například e-mailem nebo zprávou na sociální síti a výsledky je možné zkontrolovat i na mobilním telefonu.

2 Analýza a popis použitých technologií

Webové technologie se v základu dělí na front-endové a back-endové. Front-endové technologie se zabývají vším co vidí uživatel. Back-endové jsou většinou uživateli skryty a starají se o generování obsahu. Dnes již existuje i určitý přesah, kdy dříve čistě front-endová technologie JavaScript zasahuje, díky projektu Node.js, i do back-endových technologií. [15]

2.1 Webové front-end technologie

2.1.1 Hypertext markup language (HTML)

Základním kamenem při tvorbě front-endové části webové aplikace zůstává především HTML (Hypertext Markup Language). HTML spatřilo světlo světa v roce 1989 v CERN (Centre Européenne pour la Recherche Nucléaire) a to díky Timu Berners-Leeovi, který jej vyvinul jako publikační jazyk pro sdílení dokumentů mezi různými vědeckými pracovišti po celém světě. Postupem času se myšlenkou HTML začali zabývat další významní lidé, kteří k jeho vývoji přispěli, jako například Dave Raggett, který napsal Arena browser, Eric Bina a NCSA organizace, kteří vyvynuli Mosaic browser nebo Marc Andreessen, který přišel s myšlenkou IMG tagu, bez kterého bychom se v dnešní době neobešli. V roce 1994 se v Ženevě uskutečnila první World Wide Web konference, kde byl představen jazyk HTML+, který přinesl nové prvky, jako obtékání textu, tabulky s nastavením velikosti nebo např. obrázková pozadí. V listopadu roku 1995 byla představena HTML verze 2.0 a přinesla především tagy list, anchor a například podporu pro obrázky v jedné řádce.

V průběhu let bylo HTML různě upravováno a vyvíjeno a právě druhá verze HTML měla sjednotit všechny tyto modifikace napříč komunitou. V listopadu 1994 byla založena společnost Netscape Inc., která stojí za známým Netsacpe prohlížečem a koncem téhož roku bylo založeno The World Wide Web Consortium. Do HTML se postupně připojovaly další tagy a funkčnost. V lednu roku 1997 vyšlo HTML verze 3.2 a přispěla například obtékáním obrázku textem nebo nastavitelnou barvou textu a pozadí. A roku 1998 spatřilo světlo světa HTML verze 4, která přinesla nové tagy jako tbody, label nebo frame a nové atributy pro kaskádové styly jako id, class a

style. [21] [11] [37]

Nejnovější verze, čili HTML 5 přináší velkou sadu novinek a vylepšení. Jednou z oblastí, na kterou se vývojáři zaměřili je sémantika HTML stránky a přehlednost kódu. Byly zavedeny nové značky jako section, article, main, nav, figure a další. Zjednodušení se dočkal také DOCTYPE a meta tag charset. V HTML 5 však přibylo mnoho funkčnosti, co se grafické stránky, videa, zvuku a například offline aplikací týče.

V prvé řadě jsme se dočkali kreslícího plátna v podobě prvku canvas, na které lze kreslit objekty pomocí skriptovacího jazyka JavaScript. Co se týče videa, HTML 5 obsahuje tag video, který je schopný přímo v prohlížeči přehrát video ve formátech MP4, WebM, nebo Ogg. Obdobně se chová tag audio, který přehrává hudbu ve formátech MP3, Wav, a Ogg. HTML 5 přináší také možnost psát offline aplikace, jednoduše je možné nastavit takzvaný cache manifest, to jest které soubory budou přednostně stažené a nahrané do cache v prohlížeči a v případě výpadku sítě dostupné i offline. [57] [36] [40]

2.1.2 Cascading style sheet (CSS)

Když se poprvé objevila technologie HTML jako platforma pro vydávání článků a publikací na internetu, nebyla zde žádná možnost, jak tyto dokumenty stylovat. První, kdo se této volné příležitosti ujal, byl pan Håkon Wium Lie v roce 1994, jež tou dobou pracoval v CERN, kdy publikoval první koncept kaskádových stylů. Nebyla to však jediná technologie, která cílila na stylování HTML, konkurovaly jí například jazyk pana Pei Weie z prohlížeče Viola nebo Document Style Semantics and Specification Language (DSSSL). CSS však díky myšlence kaskádování jednotlivých stylů, jejich oddělení od HTML, zohledňování rozlišení displeje a velikosti okna prohlížeče, měly navrch. Jedním z průkopníků byl například Microsoft, jež implementoval podporu CSS stylů do svého Internet Exploreru 3.

Oficiálně se CSS level 1 objevily ve W3C doporučení v prosinci 1996. Poté byla ustanovena skupina uvnitř W3C (World Wide Web Consorcium), která se kaskádovými styly začala zabývat a v roce 1998 bylo vydáno doporučení na CSS level 2. Bez webových prohlížečů by se však kaskádovými styly zabývali jen akademičtí pracovníci. Díky nim došlo k popularizaci a vývoji této technologie. První z prohlížečů, jež implementovaly CSS byly Internet Explorer verze 3 a Netscape Navigator verze 4. Ale kvůli chybějící standardizaci si každý přizpůsoboval kaskádové styly podle svého a implementoval jen určité části. Postupem času se přidala například Opera, která

již implementovala CSS level 1. Jednou z oblastí, kde se Opera uplatňovala, byly mobilní telefony, díky podpoře @media tagu, který umožňoval vykreslovat internetové stránky i pro malé obrazovky. Dále následovaly Mozzila Firefox a Safari od firmy Apple. Toto konkurenční prostředí velmi napomohlo následnému vývoji.

Ale historie kaskádových stylů není jen o vývojářích a prohlížečích, je především o lidech, dobrovolnících a webmasterech, kteří tuto technologii používali každý den a pomohli ji zdokonalit. Postupem času vyvstala potřeba testování kaskádových stylů, jelikož se na každém prohlížeči zobrazovaly trochu odlišně. A proto v roce 1998 vytvořil Todd Fahrner takzvaný "acid" test. Tento test má za cíl prověřit chování prohlížečů v zobrazování CSS a poskytovat zpětnou vazbu o jeho kvalitě. V dnešní době existují celkem tři acid testy a jsou přístupné na webové stránce www.acidtests.org. Další ze zajímavých projektů zabývajících se testováním kaskádových stylů a jejich zobrazování je práce pana Davida Shea, který vytvořil Zen zahradu CSS, aby ukázal ostatním kolegům sílu kaskádových stylů při vytváření různorodých designů webových stránek. [58] [13] [47] [14]

Jak již bylo řečeno, CSS level 1 byly poprvé standardizovány v roce 1996. Umožňovaly především barvit text, pozadí a další elementy. Přidaly také možnost stylování textu jako například nastavitelné mezery mezi slovy a řádkování. Další velmi důležitou součástí byla podpora zarovnání, odsazení okrajů a rámečků textu a elementů. V neposlední řadě přidaly unikátní identifikátory pro sdružení stylů do skupin. CSS level 2 spatřily světlo světa v roce 1998 a přinesly především absolutní, relativní a fixní pozicování. Velmi důležitou součástí CSS level 2 byla podpora různých fontů i možnost vložit je externě. Nejnovější CSS verze 3 představila například nové selektory. Nyní je možné vybrat DOM elementy na základě atributů, které obsahují. Dále pak bylo zjednodušeno vytváření zaoblených rohů u elementů přidáním atributu border-radius. Bylo přidáno box-shadow a text shadow, což jsou efekty stínů u textu a rámečků. K dalším z vizuálních vylepšení přispívají velkým dílem možnosti Animation a Transition. Umožňují nově vytvářet animace a přechody přímo v kaskádových stylech. Do této doby to byla doména JavaScriptu. A v neposlední řadě byla představena funkce calc, která dovoluje jednoduché aritmetické operace přímo v kaskádových stylech. [1] [54] [59]

2.1.3 JavaScript

Poslední front-endovou technologií, která je použita v této bakalářské práci je skriptovací jazyk JavaScript. Kolem roku 1992 začali první snahy o vytvo-

ření skriptovacího jazyka jako například C–, jenž byl později přejmenován na ScriptEase, nebo například CEnvi. Tyto pokusy však neměly valný výsledek. S rostoucí popularitou webových stránek, rostla i potřeba vytvořit skriptovací jazyk, který by mohl běžet jako aplikace na straně klienta. V roce 1995 byl ve společnosti Netscape vytvořen takový skriptovací jazyk panem Breandanem Eichem a byl pojmenován LiveScript. Původní účel tohoto jazyka bylo zjednodušení implementace Java appletů do internetové stránky. Před vydáním v prohlížeči Navigator verze 2.0 však marketingové oddělení rozhodlo o přejmenování na JavaScript.

Tento jazyk byl velice chytře navržen, byla to velmi užitečná a schopná technologie a přitom si zachoval jednoduchost a díky syntaxi podobné jazyku C nebylo pro programátory tolik obtížné se jej naučit. Díky těmto předpokladům začal JavaScript brzy žít svým vlastním životem. Každá společnost ho však implementovala trochu odlišně, díky chybějící standardizaci. Například první verze Internet Exploreru od Microsoftu, tedy verze 3, neimplementovala document.images array. To přimělo společnosti Sun a Netscape ke standardizaci JavaScriptu s pomocí European Manufactures Association (ECMA) roku 1997. Tím se oficiálním jménem standardu stal ECMAScript. Standard přijala i společnost Iternational Organization for Standardization (ISO) roku 1998. [41] [39]

ECMAScript 1 byl tedy představen roku 1997 a obsahoval základ jazyka jako například práci s čísly či základní proměnné a funkce pro práci
s nimi. Rok poté následovala verze 2, která přinesla možnost ovládat obsah díky Dynamic HTML, a která zavedla DOM (document object model). Document object model je do stromu zřetězená struktura HTML dokumentu, ke které se dalo pomocí JavaScriptových funkcí přistupovat. V roce 1999 byl vydán ECMASCript 3, který obsahoval předprogramované funkce pro XMLHttpRequest. Tento aplikační interface umožňoval skriptům na straně klienta odesílat a přijímat HTTP a HTTPS requesty. Formát dat v nich použitý byl například XML nebo HTML. Toto aplikační interface bylo poprvé implementováno do Internet Exploreru verze 5. V roce 2001 spatřil světlo světa JSON (Javascript based data exchange format). JSON je strukturovaný textový formát používaný pro reprezentaci objektů, polí a proměnných. Například právě při výměně dat skrze HTTP request.

Za další významnou událost v historii JavaScriptu lze považovat rok 2004 a 2005. V roce 2004 byl představen Dojo Toolkit. Jednalo se o framework, který usnadnil práci s JavaScriptem ve velkém. Hlavním přínosem byla například potřebná infrastruktura, aplikační interface pro desktop-style grafické widgety a modulový systém. Rok poté byl představen Ajax (Asynchronous

JavaScript and XML). Ajax znamenal průlom ve wedesignu. Díky němu mohl JavaScript na straně klienta asynchronně načítat HTML obsah a ten pak dynamicky vykreslovat v prohlížeči. Nejznámějším představitelem tohoto směru jsou například Google maps. Vše fungovalo přes HTTP requesty a přenášel se obsah v XML nebo později oblíbenějším JSON. V roce 2006 byl představen dnes velmi populární framework pro manipulaci s DOM, framework jQuery.

jQuery velmi zjednodušil práci s DOM, na který se jde díky němu odkazovat podle identifikátorů a jednoduše ho měnit. Dokázal také poskytnout odladěné multiplatformní Api. Dalším milníkem v historii JavaScriptu byl příchod WebKit. Jedná se o HTML engine, který byl představen společností Apple v roce 2003. V roce 2005 byl uvolněn pod open source licencí a v roce 2007 se objevil v prohlížeči chytrého telefonu iPhone. Krátce nato se stal mainstreamem pro mobilní internetové aplikace.

Další historickou událostí na časové ose je představení projektu Node.js v roce 2009. Jednalo se o webový server psaný v JavaScriptu a postavený na V8 enginu od společnosti Google. A ve stejném roce také došlo k vydání ECMAScript 5. Novinkou byl takzvaný Strict mode. Tento mód umožňuje vývojáři přepnout do striktní varianty JavaScriptu. Výhoda je v tom, že například v tomto módu JavaScript zahlásí chybu, kterou by v normálním módu přešel, čehož je využito především při ladění kódu. Následovala verze ECMAScript 5.1 v roce 2011, která přinesla menší změny a vylepšení funkčnosti. Například rozšíření konstruktoru Object nebo prototypu Array. V roce 2015 byl představen ECMAScript 2015, ten přinesl především klíčové slovo let pro deklaraci proměnných, nové metody objektu String a nové template literals. V roce 2016 byl poté představen ECMAScript 2016, který zavedl například exponenciální operátor “ ** “ nebo možnost deklarování proměnných jako konstant. [41] [39] [10] [12]

Nejnovější specifikace ECMAScriptu byla vydána v červnu roku 2017 a nese označení ECMAScript 2017. Mezi představené nové prvky patří například asynchronní funkce. Taková funkce může obsahovat výraz await, který přeruší provádění asynchronní funkce a počká, dokud nedostane odpověď ve formě objektu Promise. Poté pokračuje provádění asynchronní funkce. Dále přibyla možnost využití sdílené paměti díky nově implementovanému SharedArrayBuffer, ten může být využit pro alokování sdílené paměti, nad kterou lze provádět atomické operace. Object.values a Object.entries jsou nové metody prototypu Object, které vrací buď pole dvojic parametr, hodnota v případe Object.entries, nebo jen pole hodnot v případě Object.values. Nově je také možné formátovat odsazení výstupu JavaScriptu pomocí metod pad-

Start a padEnd. Object.getOwnPropertyDescriptors je nová metoda, která zjednoduší kopírování dat objektů mezi sebou. [6] [25]

Dále jsou v práci použité dvě knihovny založené na skriptovacím jazyce JavaScript, a sice knihovny JQuery.js a Bootstrap. Knihovna JQuery je charakterizována jako open-source, cross-platform JavaScriptová knihovna určená především pro zjednodušení práce při psaní webových aplikací na straně klienta a pro obsluhu a ovládání HTML prvků v DOM, například formulářů a nejrůznějších animačních efektů. Její hlavní přínos je tedy ve zjednodušení přístupu k DOM struktuře, jejím elementům a eventům. Obsahuje množství dostupných a odladěných animací, přechodů a efektů.
V JQuery je také psáno nepřeberné množství pluginů, například různé slidery, responsivní prvky stránek, jako třeba formuláře, tabulky a mnoho dalšího. V neposlední řadě usnadňuje práci s AJAX technologií a obsahuje nástroje i pro práci s JSON. [52]

Bootstrap je open-source front-end knihovna pro vývoj responzivních stránek a jejich prvků. Pracuje jednak s HTML, CSS, a jednak s jazykem JavaScript. Umožňuje vytváření témat, vzhledu jednotlivých prvků a obsahuje předpřipravené a odladěné prvky, jako například tlačítka, formuláře CSS identifikátory pro předpřipravené chování. Velice zjednodušuje práci při psaní mobile first aplikací. Což je způsob návrhu webových stránek, který dává stejnou důležitost správnému zobrazení stránky na mobilu, jako na desktop zařízení. [34]

2.2 Webové back-end technologie

Jako hlavní back-endová technologie byl použit Framework .NET core (jazyk C#) pro webovou aplikaci, pro interakci s uživatelem. Dále pak databázový systém MySQL, který je zde využíván pro ukládání a poskytování dat. Vyžití JAVA (jdk.1.7.75) bylo nutné, kvůli již existujícím výpočetním knihovnám Katedry kybernetiky a kvůli spouštění a interakci se zkompilovanými Matlab skripty. Následně pak bylo nutné použít Matlab runtime (MCR v. 9.1), který je vyžadován pro běh zkompilovaných skriptů v jazyce Matlab.

2.2.1 .NET core

.NET core framework je open-source projekt (licence MIT) vydaný Microsoftem v roce 2014 (verze 1.0) ve spolupráci s veřejnou .NET komunitou vývojářů, která se podílí velkou částí na vývoji. Nejaktuálnější je verze 2.0.5.

Jednou z jeho předností je, oproti .NET frameworku, multiplatformita. Lze jej provozovat na všech hlavních operačních systémech, tedy na Windows 7+, Windows Server 2008+, Mac Os X 10.12+ a z Linuxových distribucí jsou podporované například Ubuntu 17.10, 16.04 a 14.04, Debian 8.7+ a centOs 7. [17] [20]

Mezi hlavní výhody .NET core frameworku, které ho také nejlépe charakterizují, tedy patří jeho vydání pod open-source licencí MIT a Apache 2, je tedy možné si jej upravovat a používat volně pro komerční i soukromé účely. Cross-platform kompatibilita, což znamená, že jeden program je možné spustit na všech hlavních operačních systémech. Nejsme již limitováni jen na Microsoft Windows a Windows Server. .Net core je také kompatibilní s projekty Xamarin, Mono a s klasickým .NET frameworkem prostřednictvím .NET standard. V .NET core je tedy možné psát nativní aplikace pro Android, iOS, Mac OS X, Linuxové aplikace i Windows aplikace. A v neposlední řadě obsahuje účinné command-line nástroje, které umožňují provádět veškeré úpravy projektů pomocí příkazové řádky. [17] [56]

.NET core framework se skládá z několika hlavních částí, předně z .NET Core Common Language Runtime (CoreCLR), .NET Core foundational libraries CoreFX, .NET core Command-line interface (CLI), .NET Compiler Platform (Roslyn) a ASP.NET Core. [17]

.NET Core podporuje díky .NET Compiler Platform (Roslyn) celou řadu jazyků, především C#, dále pak F# a v neposlední řadě Visual Basic .NET. .NET Compiler Platform je sada open-source kompilátorů a code analysis Api pro podporu jednotlivých jazyků. Při kompilaci zdrojového kódu v jednom z těchto jazyků je zdrojový kód zkompilován do Intermediate Language Assembly (IL Assembly), ve kterém se nachází také všechny závislé projekty a NuGet balíky. IL Assembly je pak dále zkompilován do knihoven a *executables*, které obsahují instrukce jazyka Microsoft Intermediate Language (MSIL). Po spuštění těchto knihoven nebo *executables* je prozkoumána hlavička souboru a je zavolán .NET core Common Language Runtime (coreCLR), což je multiplatformní upravená verze CLR z .NET framework. Core CLR lze přirovnat k Java Virtual Machine pro jazyk Java. Ten je odpovědný, resp. jeho součást Ryu Just in Time compiler (RyuJIT), za převod MSIL jazyka do strojového kódu pro daný počítač.

Core CLR obsahuje i další nástroje pro běh aplikace jako třeba Garbage Collector, Thread manegement, Security manegemnt a Exception management. Existuje i další varianta kompilace, a sice .NET Native resp. pro .NET

core nově vyvíjený .NET core RT. Jedná se o nativní kompilaci, která s sebou přináší zvýšení výkonu a všechny další výhody nativní kompilace bez nutnosti psát program například v C++. .NET native je sada nástrojů, které kompilují MSIL kód do strojového kódu (x64 instrukcí). Pro tuto kompilaci je použit RyuJIT kompilátor jako ahead-of-time (AOT) kompilátor. Core RT je .NET core běhové prostředí optimalizované pro AOT kompilaci. Základem je malý nativní execution engine, který poskytuje základní služby jako Garbage collector. Ostatní součásti jsou psané v C#, jako například typový systém. Výhody této nativní kompilace jsou následující. Výstupem nativní kompilace je jeden soubor obsahující aplikaci, běhové prostředí a vše potřebné. Nativně kompilované programy startují rychleji, nepotřebují totiž volat JIT kompilátor a převádět MSIL do strojového kódu. [42] [19]

.NET Core foundational libraries (CoreFX) je, obdobně jako .NET Base Class Library (BCL) a Framework Class Library (FCL) pro .NET framework, souhrn knihoven v .NET prostředí, správněji řečeno jmenných prostorů, které poskytují nejrůznější funkcionalitu, například pro práci se soubory, pro práci s XML nebo pro práci s HTTP protokolem. Je však důležité zdůraznit, že ne všechna funkcionalita plného .NET frameworku je obsažena i v .NET core FX. Ta je z důvodu multiplatformity omezena, avšak postupem času přibývá další a další funkčnost. [17] [18]

.NET core Command-line interface je sada nástrojů, přístupná z příkazové řádky, pro správu projektů v .NET core. Všechny příkazy začínají driverem pojmenovaným dotnet. Mezi nejpoužívanější patří například "dotnet new", tento příklad založí nový projekt s konfiguračními soubory podle zadaného schématu. Dále pak "dotnet run", ten slouží pro spuštění projektu. A "dotnet build", pro sestavení projektu se všemi jeho závislostmi. [55]

2.2.2 ASP.NET core

ASP.NET Core je open-source webový framework pro tvorbu webových aplikací, IoT aplikací a back-end k mobilním aplikacím. Jedná se o redesignovanou verzi známější ASP.NET 4.x, která se dočkala architektonických změn, které přispěly k větší modularitě. Použití ASP.NET core přináší několik podstatných výhod. Mezi ně patří například jednotný styl psaní webových UI a webových APIs, jednoduchá integrace JavaScriptových, CSS a jiných frameworků na straně klienta, vestavěná Dependency Injection, vysoká výkonnost a v neposlední řadě možnost hostování programu v IIS, Nginx nebo Apache serveru.

ASP.NET core vyžívá vzoru MVC čili model-view-controller. Každý z těchto členů má svůj vlastní účel a pole působnosti. Model je většinou reprezentován třídou a poskytuje data controlleru. Controller přijímá HTTP požadavky, využívá modelu pro získání dat a odesílá jako odpověď například JSON nebo View čili HTML stránku. View je HTML stránka pro zobrazení dat. ASP.NET core využívá takzvaných Razor pages, čili stránky, kde je spojen C# kód s HTML a dohromady kooperují obdobně jako třeba PHP a HTML. [45]

2.2.3 Databázový systém MySQL

Databázový systém MySQL byl v této práci použit pro ukládání a management dat. Jedná se o open-source systém pro správu relační databáze, anglicky relational database management system (RDBMS). Její jméno je složeninou "My", což je jméno dcery spoluzakladatele Michaela Wideniuse a "SQL", což značí Structured Query Language (SQL). MySQL je dílo švédské firmy MySQL AB založené Davidem Axmarkem, Allanem Larssonem a Michaele Wideniusem v roce 1995. V témže roce firma představila první vydání MySQL, jejíž Api bylo psané v jazyce C a C++. Roku 1998 byla vydána verze 3.21, kterou podporoval i Windows 95 a Windows NT. V roce 2000 byla MySQL uvolněna pod open-source licencí GPL. Roku 2002 byla vydána MySQL verze 4.0 beta, která přinesla nové výrazy jako UNION nebo TRUNCATE TABLE.

Firma poté nadále rostla, například v roce 2003 dosáhla 4 milionů aktivních instalací. V roce 2005 byla vydána verze 5.0, která přidala podporu pro Stored Procedures a Stored Funcitons. Roku 2008 byla společnost MySQL AB koupena společností Sun Microsystems. Vývoj pokračoval verzí 5.1, která opravovala několik bugů a chyb. V roce 2010 byl Sun Microsystems koupen společností Oracle. Tentýž den vznikla větev MySQL vytvořená Michaelem Wideniusem a nazvaná MariaDB. Následovaly verze 5.5 v roce 2010, přinesla vylepšení InnoDB I/O sybsystému a vylepšení podpodry pro Symetric Multiprocessing, a verze 5.6 v roce 2013, která přispěla především vylepšeným Partitioningem, Optimalizerem a změn doznal také systém logování. Nejnovější verzí vyvíjenou již společností Oracle je verze 5.7 vydaná v říjnu. Ta přináší například nativní podporu pro datový typ JSON a JSON funkce, vylepšení bezpečnosti díky jednodušší a bezpečnější inicializaci instancí a v neposlední řadě až třikrát vyšší rychlost v porovnání s verzí 5.6. [26] [49] [29] [22] [28] [30] [27]

2.2.4 Java

Poslední back-endová technologie, která byla použita pro tuto práci, je programovací jazyk Java a běhové prostředí Java Virtual Machine (JVM). Java je široce rozšířený multiplatformní, objektový, kompilovaný jazyk. Historie Javy se začala psát roku 1991 ve společnosti Sun Microsystems, založené v roce 1982 pány Vinodem Khoslem, Andy Bechtolsheimem, Billem Joyem a Scottem McNealym, kterou v roce 2010 koupila společnost Oracle a převzala tím i její produkty. Tedy i Javu a databázový systém MySQL, které dodnes provozuje. V roce 1991 tedy vznikl v Sun Microsystems tým, který měl za cíl vytvořit nový programovací jazyk jako nástupce zastaralým programovacím jazykům jako C a C++. Vznik tohoto týmu iniciovali James Gosling, Mike Sheridan, a Patrick Naughton a byl pojmenován Green Team. Tento jazyk, tehdy pojmenovaný anglicky Oak, byl původně navrhován pro malé embedded systémy elektronických zařízení, jako například set-top boxy. V roce 1995 byl Oak přejmenován na Java podle indonéského ostrova. Tentýž rok John Gage, ředitel vědecké kanceláře Sun Microsystems, a Marc Andreesen, spoluzakaldatel a výkoný ředitel Netscape, představili Javu veřejnosti a oznámili, že bude zapracovaná do Netscape Navigatoru (webový prohlížeč firmy netscape).

V roce 1996 bylo představeno první vydání Javy verze 1.0, označovaná jako Java 1, pro operační systém Solaris. Windows, Mac OS Classic a Linux, které obsahovalo Java Development Kit 1.0 (JDK) a nástroje pro vývoj. Sun Microsystems také vydali běhové prostředí pro běh Java programů Java Runtime Enviroment (JRE). O dva roky později vyšla verze 1.2 označovaná jako Java 2, ve které byl přepsán systém obsluhy eventů a byly představeny JIT-kompilery (Just In Time). V roce 2000 následovala Java Kestrel (Java 1.3), ve které byly představeny například Java Sound, Java Naming and Direcotry Interface a Java Platform Debugger. V roce 2002 byla vydána Java Merlin (Java 1.4), ta přispěla například funkčností pro zpracovávání XML, neblokujícími I/O operacemi a kryptogtafickým systémem. V roce 2006 spatřila světlo světa Java Mustang (Java 1.6.0) a obsahovala Web Servisy, možnost mixování s JavaScriptem a dalšími Desktop Api. V roce 2011 přišla na scénu Java Dolphin (Java 1.7.0) která rozšířila podporu JVM pro dynamické jazyky a novou knihovnou pro práci se soubory. Nejnovější verze, vydaná v roce 2014 je verze Java Spider (Java 1.8.0) a přináší několik novinek například podporu pro lambda výrazy, další vylepšení bezpečnosti a úprav se dočkala také Java FX, kde bylo přidáno nové schéma a nové ovládací prvky. [50] [31] [32]

Po napsání zdrojového kódu programu přichází na řadu kompilace. V Javě vše funguje následovně, nejprve je zavolán Java Compiler, který zdrojový kód převede do takzvaného byte kódu. Ten je nezávislý na platformě, protože ke spuštění potřebuje Java Virtual Machine, dále jen JVM, dostupnou na všech základních operačních systémech. Ta je po kompilaci spuštěna a stará se o přeložení byte kódu do strojového jazyka pro daný procesor. Toto dělení přináší řadu výhod, kromě již zmíněné multiplatformity, také zvýšení bezpečnosti, protože mezi programem a operačním systémem běží JVM, a také možnosti využití dalších nástrojů, které JVM poskytuje při běhu programu, jako například Java Virtual Machine Monitoring, Troubleshooting, a Profiling Tool. [33]

2.3 Výpočetní technologie

2.3.1 REXYGEN

Jedním z úkolů, kterým se tato práce věnuje, je komunikace virtuální laboratoře se systémem pro řízení strojů, technologií a procesů REXYGEN od firmy Rex Controls. Tento software, na jehož vývoji úzce spolupracuje Katedra kybernetiky, je v této práci použit jako simulační nástroj, který obsahuje bloky reprezentující reálný proces a řídící systém. A je využíván pro poskytování přístupu k jednotlivým bloků skrze jeho webové REST Api. Řídící systém REXYGEN se skládá z několika součástí, předně z REXYGENCore, REXYGENDraw a REXYGENComp. Je možné ho provozovat na operačním systému Windows a Linux i na jejich real time distribucích. Je také kompatibilní s programem Matlab - Simulink. [43] [4]

REXYGENCore je runtime jádro systému REXYGEN, které běží na cílovém zařízení a stará se o běh, spuštění a časování programů. REXYGENDraw je grafické prostředí systému REXYGEN, ve kterém se vytváří algoritmy pro řízení a simulaci procesů. Na výběr je z velkého množství již existujících algoritmů. Každý algoritmus představuje blok, které je možné spojovat a vytvářet tak daný řídící systém. Je také možné vytvářet vlastní bloky. REXYGENComp je překladač, který má na starosti přeložení vytvořených algoritmů do binární podoby. Mezi další jeho součásti patří například REXYGENOPCsv pro komunikaci přes protokol OPC, REXYGENView pro diagnostiku a v neposlední řadě REXYGENHMI pro zobrazení a ovládání procesu spuštěném v REXYGENu přes webový prohlížeč. [43] [44] [4]

2.3.2 Matlab - Simulink

Jako další z technologií je v této práci použit matematický software a skriptovací programovací jazyk Matlab. Byl zvolen z několika důvodů. Nejpádnější z nich je, že je vyučován na Katedře kybernetiky, tudíž již existuje mnoho prací psaných v tomto programu. Další z jeho přínosů je možnost kompilace do knihoven například v jazyce Java, C++, C# a následné spuštění na všech hlavních operačních systémech díky Matlab Runtime core.

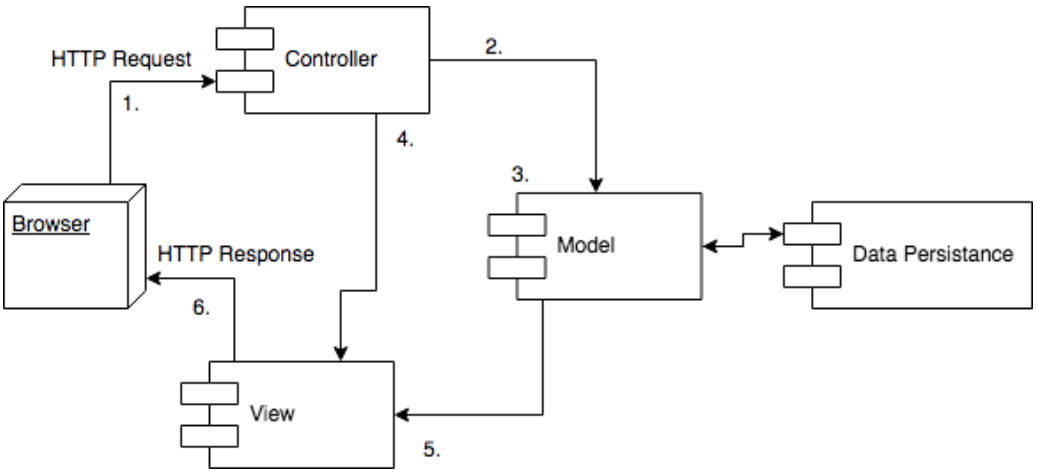
První verzi Matlabu napsal Cleve Moler ke konci 70. let jako knihovnu pro své studenty v jazyce Fortran, která využívala knihovny LINPACK a EISPACK, a obsahovala pouze 80 funkcí a jediným datovým typem byla matice. V roce 1981, když IBM ohlásilo vydání prvního PC, Jack Little, inženýr z MIT, vytušil příležitost využít Matlab na PC pro technické výpočty. Spolu s kolegou Stevem Bangertem přepsali Matlab do jazyka C a přidali podporu M-souborů, toolboxy a vylepšili grafickou stránku programu.

V roce 1984 Jak Little, Cleve Moler a Steve Bangert založili společnost MathWorks. V témže roce byla vydána verze 1.0, která byla představena na IEE Conference on Design and Control v Las Vegas, Nevada. V roce 1986 byla ve verzi 2 přidána podpora pro UNIX systémy. Do té doby byl Matlab doménou MS-DOS. Po dalším růstu společnosti byl v roce 1990 představen Simulink 1.0. V roce 1992 byla představena nová verze Matlab 4, která obsahovala barevnou grafiku ve 2 a 3D. Následující roky byla přidána podpora pro Microsoft Windows (1993) a pro Linux (1995). V roce 1996 byla vydána verze 5, která obsahovala datové typy, propracovanější vizualizaci, debugger, profiler a GUI builder. V roce 2000 byla představena verze 6, která obsahovala LAPACK a Fastest Fourier Transformation in the West (FFTW). Další verze číslo 7, vydaná v roce 2004, přispěla například single-precision a celočíselnou matematikou, anonymními funkcemi a vykreslovacími nástroji. Následovala verze 8 v roce 2012, ta přinesla Toolstrip interface, Matlab Apps a redesignovaný systém dokumentace. Verze 9, vydaná v roce 2016, přinesla například nové prostředí pro App Designer a možnost pozastavit provádění běžících programů. Nejaktuálnější verze je verze 9.3. Tato verze z roku 2017 přináší například Live Editor, Matlab Drive cloudovou službu nebo Add-On Manager. [23] [24]

3 Architektura virtuální laboratoře

3.1 ASP.NET core MVC model

Pro realizaci hlavního programu pro navrhovanou virtuální laboratoř byla zvolena technologie ASP.NET core MVC a architektura klient-server s tenkým klientem, který je ovšem schopen provádět drobné dílčí výpočty. ASP.NET core, jak již bylo představeno v první kapitole, je webový back-end framework z prostředí .NET core. Jeho hlavním účelem je tedy příjem HTTP requestů, příprava dat a zobrazení těchto dat například v HTML stránce. Model MVC, jak bylo nastíněno v sekci 2.2, je model rozdělení kódu do tříd a celků, které mají specifické pole působnosti a specifickou funkčnost viz obrázek 3.1.



Obrázek 3.1: Diagram modelu MVC

"M" v modelu MVC znamená model, většinou je reprezentován třídou nebo skupinou funkčně propojených tříd, které poskytují data (komunikují s data persistory) pro naši aplikaci, viz obrázek 3.1 bod č. 3. Jedná se tedy o nejspodnější stavební kámen. Setkáme se zde například s repozitáři pro přístup k databázi, entitami pro reprezentaci dat nebo třídami pro práci se soubory.

"C" v modelu MVC znamená kontroler, tedy třída, která v případě ASP.NET core musí dědit z abstraktní třídy `Controller`. Kontroler se stará

o příjem HTTP requestu, viz obrázek 3.1 cesta č. 1, od uživatele a o poskytování, úpravu nebo změnu dat vyžádanou uživatelem viz obrázek 3.1 cesta č. 2. Jedná se tedy o základní díl naší aplikace. Výchozí struktura takového kontroleru je ukázána v následujícím zdrojovém kódu. [7]

```
namespace PIDlab.controller
{
        //Atribut pro autorizaci
    [Authorize]
    public class HomeController : Controller
    {
        // GET: /Home/Home
        public IActionResult Home()
        {

        /* Navratovy typ je IActionResult a v tomto
         pripade se jedna o View Index.cshtml
         ve slozce Home */
            return View();
        }
    }
}
```

Zavolání správné metody kontroleru zvané action dochází na základě routování HTTP requestu. URL je parsována podle předem definované šablony, například /api/controller/action, a ASP.NET core se postará o zpracování HTTP requestu. Následně je tedy na základě URL zavolána akce kontroleru, které jsou předána data, jež mohou být buďto v těle HTTP requestu, nebo jako parametry v adrese URL. ASP.NET core se o všech kontrolerech dozví při startu aplikace, kdy jsou všechny třídy, které dědí z abstraktní třídy `Controller` nalezeny a zaregistrovány.

"V"čili view znamená pohled nebo zobrazení. Jedná se tedy o tu část, kterou přímo vidí a s níž pracuje uživatel. Po zpracování HTTP requestu a připravení všech dat je *view*, česky pohled, připraveno, viz obrázek 3.1 cesta č. 4, a odesláno do prohlížeče jako HTTP response, viz obrázek 3.1 cesta č. 6. Pohled se také může dotazovat na data, viz obrázek 3.1 cesta č. 5, nemůže je však měnit nebo provádět databázové operace, omezuje se pouze na čtení z již naplněných a připravených entit. Technologií je zde možno použít nepřeberné množství, avšak asi nejpoužívanější je kombinace HTML, CSS a JavaScript. V předkládané bakalářské práci je navíc použita nová funkčnost

a sice Razor Pages. Jedná se kombinaci HTML a C# kódu podobně, jako v případě jazyka PHP, který je taktéž možno kombinovat s HTML kódem. Takový C# kód je pak nutné psát za znak @, viz následující zdrojový kód. [2] [7]

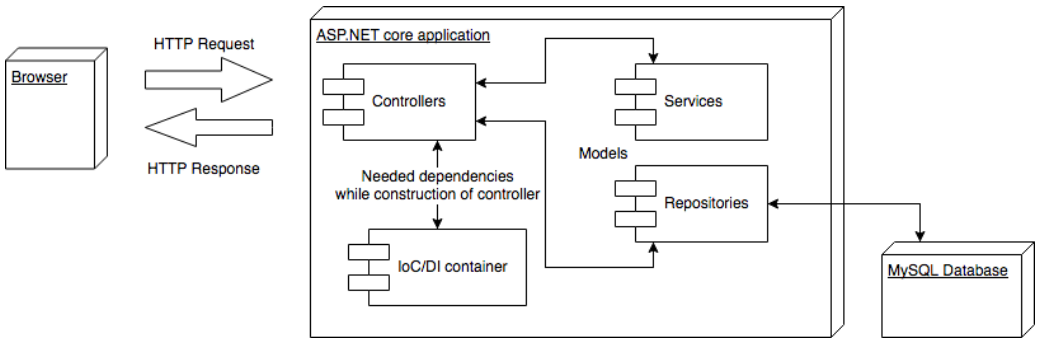
```
@model SystemsHolder

@for( int i = 0 ; i < Model.listOfSystems.Count();
   i++)
{
    <div class="myBorder  col-lg-4 col-md-6
       col-sm-12 col-xs-12">
        <div class="sys countHeight well">
          <!-- zjednoduseno pro ukazku -->
                <h5>System id:
                <span
                   class="sysId">@Model.listOfSystems[i].id
                </span>
                </h5>
            <div class="cleaner"></div>
            <div>
            <br>
                <p>\[f(s) =
                   \@Model.listOfSystems[i].Formula\]</p>
            </div>
                <br>
                <div class="checkbox checkBottom" >
                <label><input type="checkbox"
                   value="">Render
                   characteristics</label>
            </div>
        </div>
    </div>
}
```

3.2 Architektura ASP.NET core aplikace

Architektura celé ASP.NET core aplikace je následující, viz obrázek 3.2. Hlavní článkem je Kestrel server, jedná se o multiplatformní, open source,

event-based HTTP server vyvinutý firmou Microsoft, založený na libuv, což je multiplatformní knihovna pro asynchronní I/O operace. Ten v sobě hostí ASP.NET core aplikace. Přijímá HTTP requesty od jiného HTTP serveru, na operačním systému Windows se ve většině případů jedná o server IIS, na Linuxu nebo Mac OS X potom Nginx nebo Apache web server, které fungují jako reverzní proxy, čili jako vstupní brána ke Kestrel serveru a naší aplikaci. Po zpracování HTTP requestu je jeho URL parsována a je předán příslušné akci daného kontroleru. Tak, jak bylo popsáno v sekci 3.1. [8]



Obrázek 3.2: Architektura ASP.NET core

Projekt ASP.NET core aplikace je v kořenové složce virtuální laboratoře umístěn ve složce `PIDlab` a je organizován do solution .NET core frameworku, které se dělí na tři dílčí projekty, a sice na hlavní projekt aplikace ve složce `PIDlab`, projekt pomocných nástrojů ve složce `Tools` a projekt TCP/IP klienta ve složce `TCPClient`.

Základní třídou a vstupní branou našeho programu je třída a stejnojmenný soubor `Program.cs` ve složce `PIDlab`. Ten obsahuje metodu `main`, kde je sestaven objekt `WebHost` s danou konfigurací a ten je následně spuštěn. Tato konfigurace je uložena v souboru a třídě `Startup.cs`. Dochází zde k registraci Middleware a jednotlivých servisních tříd a repozitářů. Důležitou součástí je také nastavení Route, čili vzoru, podle kterého je pak parsována URL HTTP requestu. Dále pak je možné ve složce `PIDlab` nalézt složky jako `controller`, `models`, `services`, `views` a `wwroot`.

Ve složce `controller` se nachází všechny třídy kontrolerů, které jsou v této práci použity. Dále ve složce `models` jsou obsaženy všechny třídy, která v sobě uchovávají a data a jsou použita k jejich přenosu. Ty nesou označení DTO čili anglicky *data transfer object*. Složka `View` zase obsahuje všechny .cshtml soubory, což jsou zmiňované *Razor Pages*, kde je spojen C# kód s HTML a slouží k zobrazení dat uživateli. Složka `services` obsahuje všechny servisní třídy a repozitáře, která se starají a získávání dat a jejich zpracování. Dále jsou u těch, které to vyžadují, entity, které slouží k uchování dat. Tyto

servisy a repozitáře jsou registrovány v *Dependency injection Containeru* a jejich závislosti jsou vkládány do kontrolerů při jejich inicializaci. V poslední řadě složka `wwwroot` obsahuje všechny kaskádové styly, JavaScript a další statické soubory, jako obrázky, nutné pro zobrazení jednotlivých *views* čili pohledů jako webových stránek.

Další důležitou součástí ASP.NET core aplikace je autentizace uživatelů. Uživatel musí být zaregistrován v databázi a musí se přihlásit pomocí hesla a jména, aby dostal přístup k dalšímu obsahu. Při přihlášení na přihlašovací stránce jsou zadané údaje ověřeny a následně je vytvořen zašifrovaný `Cookie` soubor s uživatelskými údaji. Ty jsou vyžadovány při přístupu ke každé akci kontroleru, který je dekorován atributem `Authorize`. V případě vypršení platnosti `Cookie` nebo jeho neexistenci, je po takovém přístupu uživatel přesměrován na přihlašovací stránku. Autorizační `Cookie` soubor obsahuje zašifrované nezbytné údaje. Je však možné přidat do tohoto `Cookie` souboru vlastní data a získávat je zpět, jak je patrné na následující ukázce kódu. [38]

```
//zapis jmena a id uzivatele do Authnetication
   cookie
var claims = new List<Claim>
{
        new Claim(ClaimTypes.Name, logs.Name),
        new Claim("Id", userId.ToString())
};

//ziskani id uzivatele z Authentication cookie
//struktury jsou prochazeny prikazy LINQ
int userId = int.Parse(User.Claims.Where(x =>
   x.Type.Equals("Id")).ToList()[0].Value);
```

3.2.1 Dependency Injection v ASP.NET core aplikaci

Dependency Injection je technika pro vkládání závislostí mezi jednotlivými komponentami programu. Hlavní výhoda je v tom, že závislosti jsou vkládány za běhu programu podle potřeby a nikoliv při jeho sestavování. Odděluje se tak životní cyklus komponenty (inicializace, běh a destrukce) od programu. Program je implementován proti rozhraním, anglicky *interface*, potřebných objektů, takže i následná úprava nebo *refactoring* takového objektu nepřináší další potřebu přepisovat kód, v němž je používán.

Dependency Injection může být několika druhů, například *Contructor Dependency Injection* nebo *Property Dependency Injection*. Při *Construtor Dependency Injcetion* je v konstruktoru třídy daná závislost vyžádána díky parametru konstruktoru, který má jako typ rozhraní daného objektu. Dependency Injection Controller, což je třída nebo soubor tříd, která se stará o zakládání objektů, údržbu a destrukci, pak takovou referenci vyhledá a vloží potřebnou závislost viz ukázka kódu konstruktor třídy SystemController. [48]

```
namespace PIDlab.controller
{
    [Authorize]
    public class SystemController : Controller
    {
        //privatni field do ktereho je ulozena
        //injectnuta instance
        private ISystemService _systemService;

        //diky tomuto argumentu, ktery ma typ
           Interface dane
        //servisy, dojde k injectnuti zavislosti
        public SystemController(ISystemService
           systemService)
        {
            _systemService = systemService;
        }

        // GET: /System/GetSystem/
        public IActionResult GetSystem()
        {
```

Další možnost je místo parametru konstruktoru využít obdobně členskou proměnnou dané třídy, výsledek je poté stejný. Aby vše fungovalo, musíme nejprve nastavit a inicializovat *Dependency Injection Container* a zaregistrovat všechny naše potřebné objekty (nazývané servisní třídy). *Dependency Injection Containerů* je dostupné nepřeberné množství, v této bakalářské práci je použit výchozí a nejjednodušší *DI Container*, dodávaný spolu s ASP.NET core, který je reprezentován rozhraním `IServiceProvider`.

Dále pak mezi nejpoužívanější patří Castle Windsor, Structure Map, Autofac nebo Unity. Opět existuje i více způsobů, jak zaregistrovat servisní

třídu v závislosti na požadovaném chování. Například registrace jako *Singleton* vytvoří jednu instanci a ta jediná je po celou dobu vkládána, registrace jako *Scoped* vytvoří pokaždé novou instanci při každém vyžádání závislosti ale jen jednou pro každý HTTP request. Registrace typu *Transient* vytvoří pro každý request svou vlastní instanci. Ukázka registrace viz následující kód. [48]

```
//services
      services.AddScoped<ILogInService,LogInService>();
      services.AddScoped<IBlockService,
         BlockService>();

//repositories
       services.AddScoped<ILogInRepository,LogInRepository>();
       services.AddScoped<IBlockRepository,
          BlockRepository>();
```

3.2.2 Responsive, mobile first web layout

Jeden ze zadávajících požadavků bylo implementovat možnost spouštět virtuální laboratoř také na všech typech mobilních zařízeních. To je z části splněno použitím internetového prohlížeče jako hlavního programu pro komunikaci s virtuální laboratoří. Hlavní část řešení pak tkví v použití responzivního, mobile-first, open-source, JavaScriptového a CSS frameworku Bootstrap a JavaScriptového open-source frameworku JQuery. Díky němu jsou všechny HTML stránky používané v naší virtuální laboratoři připraveny pro bezchybné zobrazení také na mobilních zařízeních se schopností připojení k internetu a spuštění prohlížeče, což je v dnešní době naprostá většina mobilních zařízení, ať už chytré telefony nebo tablety s operačními systémy Android, IOS nebo Windows Phone.

Základ frameworku bootstrap tvoří kaskádové styly a JavaScriptový soubor, které je nutné přiložit k HTML stránce. Poté je možné využít předprogramovaných kaskádových tříd pro potřebné chování HTML elementů. Bootstrap má předpřipravené třídy pro většinu scénářů, pro stylování a úpravu chování obalovacích HTML elementů slouží třída `col`. Bootstrap rozděluje obrazovku vždy na dvanáct pomyslných sloupců, třída `col-6` určuje, že `div` bude široký přes 6 sloupců. Je možné nastavit také různou šířku podle velikosti okna potažmo zařízení, které stránku zobrazuje. Například `col-lg-4` znamená, že na šířce okna nad 1200 pixelů bude `div` široký přes 4 sloupce a

například `col-sm-10` znamená, že Div bude široký přes 10 sloupců v okně širokém od 576 do 768 pixelů. Viz následující ukázka kódu. [35]

```
<div class="myBorder  col-lg-4 col-md-6 col-sm-12
   col-xs-12">
 <div class="sys countHeight well">
   <table class="sysRemove">
     <tr>
     <td>
      <button type="button" class="btn btn-primary
         sysEdit" >&#9881</button>
```

Bootstrap obsahuje celou řadu optimalizovaných tříd například `btn` pro tlačítko nebo `well` pro dlaždici, použitou v předešlém kódu. Obsahuje ale i mnoho dalších například pro formuláře, obrázky a další.

3.3 MySQL databázový systém

Jeden ze základních úkolů naší hlavní ASP.NET core aplikace, je komunikace s databázovým systémem MySQL, viz obrázek 3.2, a dále pak získávání, úprava, přidávání a mazání dat v něm. Firma Oracle poskytuje Nuget package pro .Net core, která obsahuje veškerou potřebnou funkčnost pro komunikaci s MySQL databází. Viz následující ukázka kódu ze třídy `SystemRepository`. Funkce `GetConnection` slouží k získání objektu `MySqlConnection`, který obsahuje spojení s databází. Po jeho otevření funkcí `Open` je pak možné pomocí třídy `MySqlCommand` provádět SQL dotazy na data databáze a zpracovávat výsledky pomocí třídy `MysqlDataReader`.

```
using (MySqlConnection conn = GetConnection())
{
 conn.Open();
 MySqlCommand cmd = new MySqlCommand("select * from
    System where UserId = "+ id.ToString(), conn);

  using (var reader = cmd.ExecuteReader())
  {
   while (reader.Read())
   {
     ListOfSystems.Add(new SystemHolder()
```

```
{
    id = Convert.ToInt32(reader["Id"]),
    Numerator = reader["Numerator"].ToString(),
    Denumerator =
        reader["Denumerator"].ToString(),
```

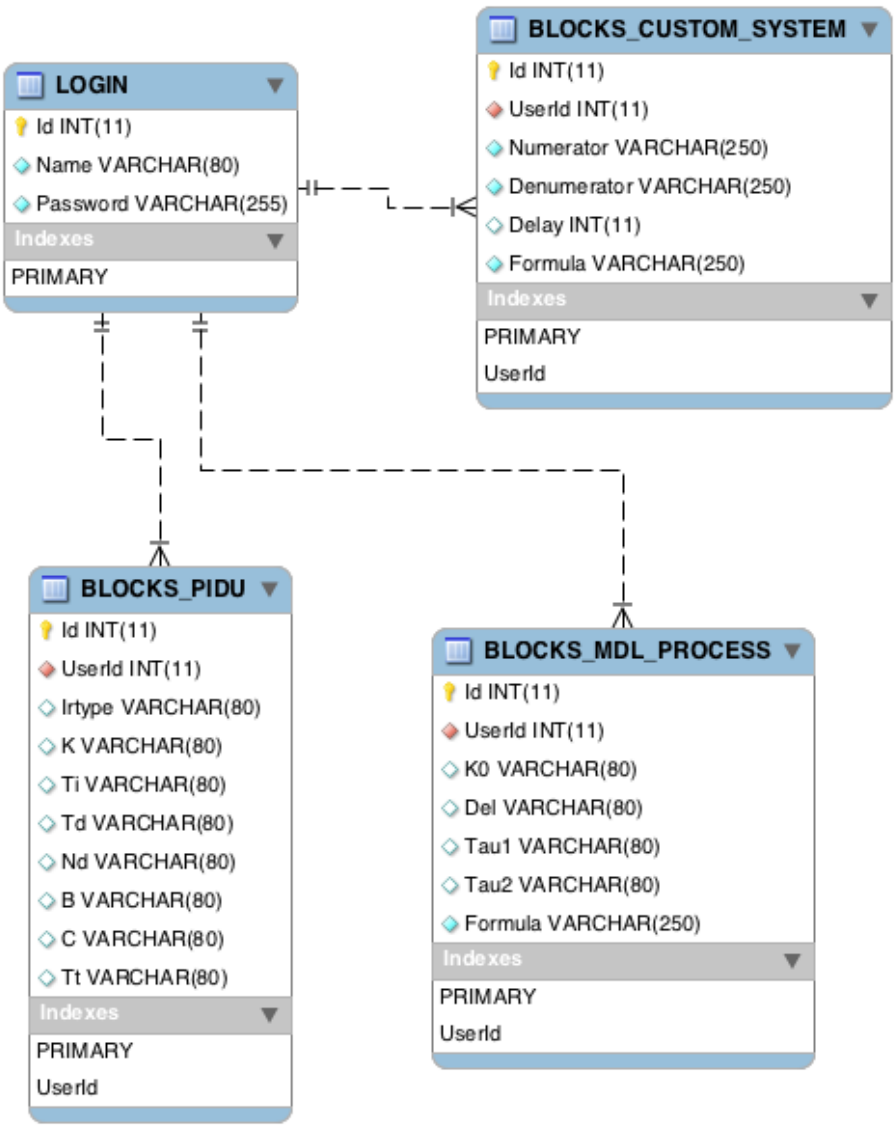
Cestu dat z kontroleru do databáze nebo obráceně lze popsat následovně. Poté, co je zpracován HTTP request od uživatele, je zavolána příslušná akce kontroleru a rozhodnuto o vyžádání dat. Dále je zavolána příslušná metoda servisní třídy, jejíž závislost byla vložena do kontroleru. Ta v sobě drží vloženou závislost daného repozitáře, ve kterém je implementace přístupu, zápisu do databáze nebo čtení dat z databáze. Výhoda tohoto prostředníka jako servisní třídy tkví v tom, že v případě použití více databázových systémů lze například podle záznamů v konfiguračním souboru rozhodnout o využití toho daného repozitáře.

Zakládající skript pro databázi potřebnou pro běh této virtuální laboratoře je umístěn ve složce `SqlScripts` v souboru `CreateFull.sql`. Obsahuje definice databáze, všech potřebných tabulek a jejich vlastností, dále pak i minimální potřebná data pro provoz. Schéma databáze prozatím není rozsáhlé. Databáze je pojmenována `PIDLAB` a obsahuje tabulky `LOGIN`, `BLOCKS_CUSTOM_SYSTEM`, `BLOCKS_MDL_PROCESS` a `BLOCKS_PIDU`. Tabulka login slouží pro uchování přihlašovacích údajů uživatele jako například jméno a heslo. Tabulky, jejichž název začíná slovem blocks jsou tabulky které v sobě uchovávají data jednotlivých bloků jako třeba numerator. Schéma celé databáze bylo vytvořeno v programu Mysql Workbench a je zobrazeno na obrázku 3.3.

3.4 Uživatelské rozhraní virtuální laboratoře

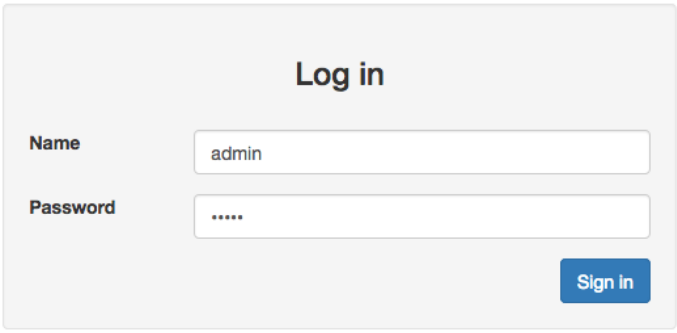
Pohledy ve složce `views` jsou rozděleny do několika složek podle svého určení, například pohledy pro `RexController` jsou ve složce `Rex`. Jsou zde navíc dvě speciální složky a sice `Shared` a `Partials`. Složka `shared` obsahuje pohled s názvem `_Layout`, ten představuje hlavní pohled, který zavádí kostru HTML dokumentu. Jednotlivé dílčí pohledy, nazývané `Partial Views`, jsou vykreslovány do tohoto hlavního pohledu. Složka `Partials` poté obsahuje pohled s menu ASP.NET core aplikace. To je zde umístěno, jelikož přímo nepřísluší žádnému kontroleru. Výhodou v jeho umístění ve zvláštním souboru je lepší udržovatelnost v případě jeho změn. Následující obrázek 3.4 zobrazuje vstupní pohled pro přihlášení uživatele.

A na obrázku 3.5 je zobrazena hlavní stránka ASP.NET core aplikace,

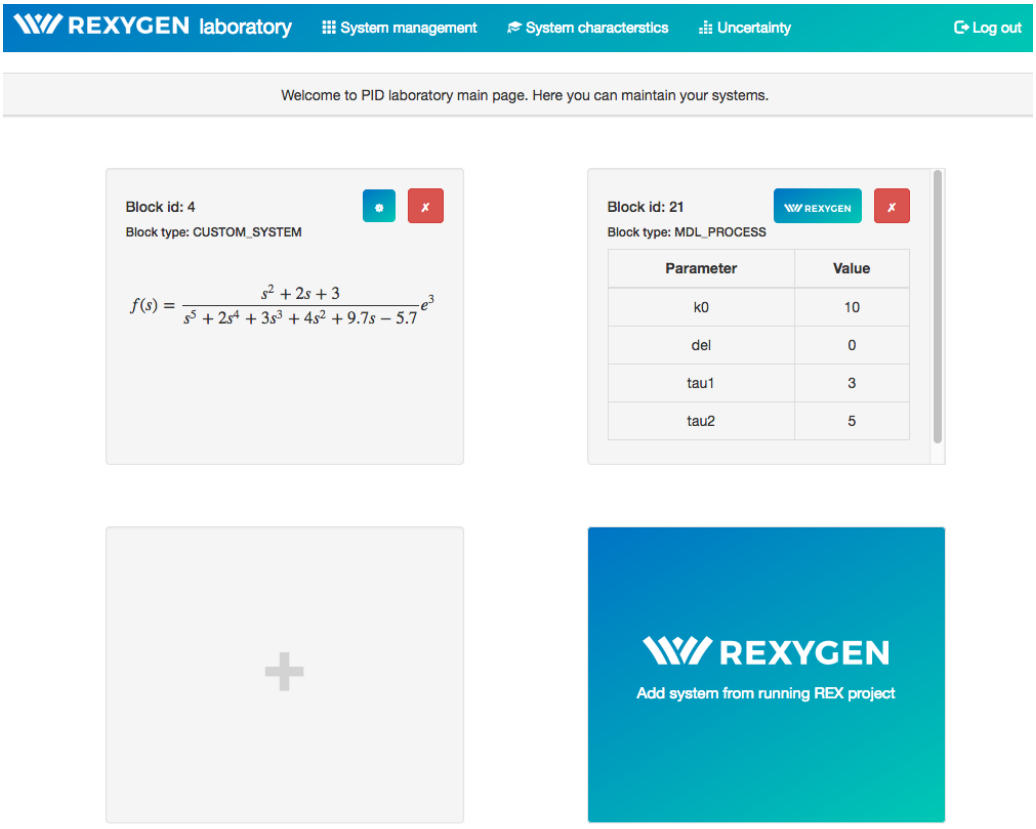


Obrázek 3.3: Schéma databáze PIDLAB

která umožňuje spravovat vlastní bloky a prohledávat a přidávat nalezené bloky z webového REST Api běžící simulace programu REXYGEN.



Obrázek 3.4: LogIn view pro přihlášení uživatele

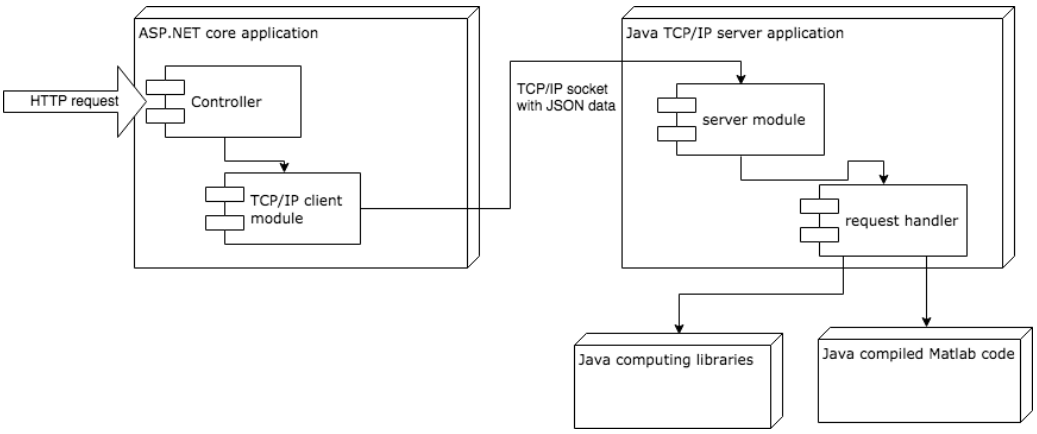


Obrázek 3.5: GetSystem view pro zobrzení všech bloků a jejich správu

4 Postup pro integraci SW modulů a výpočetních knihoven z prostředí Matlab a jazyka Java

Použití frameworku .Net core přineslo mnoho výhod, stejně jako i několik nevýhod, které však lze elegantně vyřešit. Pro tuto práci bylo potřeba využít výpočetních knihoven třetích stran, především ty, z Katedry kybernetiky. V naprosté většině případů jsou však implementovány v jazyce Java, případně v jazyce C nebo C++. Dále pak bylo potřeba použít zkompilované skripty jazyka Matlab. Ty však lze zkompilovat pouze do jazyka Java, Python, C, C++ a do knihovny prostřední .Net, nelze ji však zkompilovat do prostředí .Net core.

Obojí se podařilo vyřešit použitím TCP/IP protokolu a TCP/IP serveru psaného v jazyce Java, kde mohou být volány jednak již naprogramované knihovny a jednak zkompilované skripty programu Matlab. Architekturu takové komunikace naší webové aplikace psané v ASP.Net core, ve které byl implementován TCP/IP klient s výpočetní serverem v jazyce Java, znázorňuje obrázek 4.1.

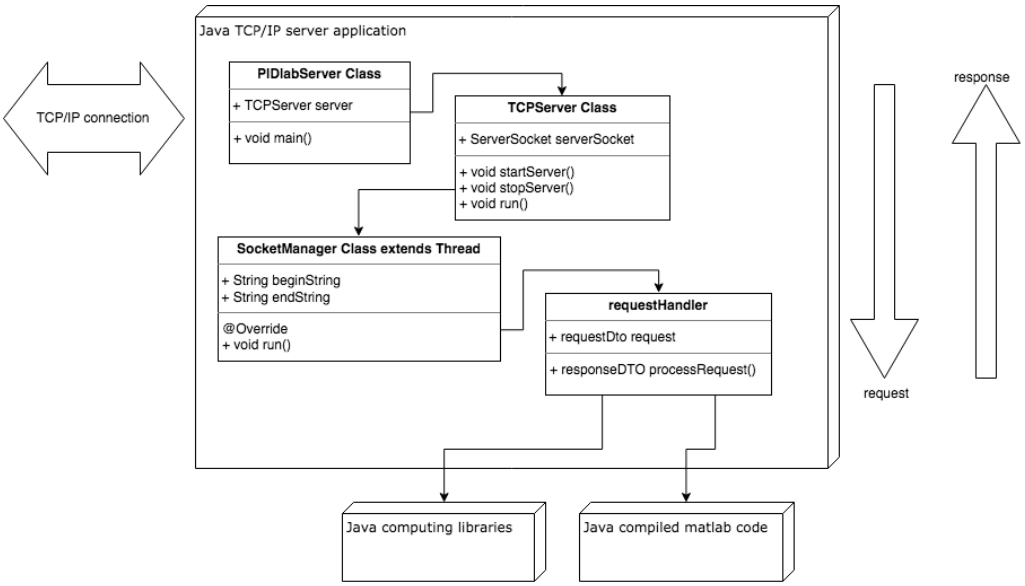


Obrázek 4.1: Diagram TCP/IP komunikace mezi ASP.NET core aplikací a výpočetním Java serverem

4.1 TCP/IP Java server

Základní třídou, která v sobě obsahuje metodu main, je třída `PIDlabServer`. Ta má za úkol nastavit `port` a `waitTime`, čili dobu, po kterou server čeká na příchozí socket (v případě že je nastavena na 0, čeká neomezenou dobu). Dále pak vytváří novou instanci třídy `TCPServer`, nastaví jí dané parametry v konstruktoru a zavolá její členskou metodu `startServer`.

V této metodě je vytvořena nová instance objektu `ServerSocket` s daným portem a je zavolána metoda `run`. Ta obsahuje nekonečnou smyčku serveru, kde se čeká na příchozí spojení. V případě nového spojení je vytvořeno nové vlákno pro toto spojení reprezentované objektem `ScoketManager`. To obstarává přijímání a odesílání zpráv od a na klienta. V případě přijetí zprávy kontroluje, zda přišel otevírací řetězec `"STARTMSG|"` a přijímá, dokud nepřijde ukončovací řetězec `"|ENDMSG"`. Z důvodu většího množství přenášených dat mohou být odeslána ve více socketech. Z přijatých dat vytvoří objekt JSONObject a založí novou instanci třídy `requestHandler`, které JSONObject předá jako parametr konstruktoru, následně je zavolána metoda `processRequest`.



Obrázek 4.2: Diagram struktury TCP/IP serveru v jazyce Java

V kontruktoru třídy `requestHandler` je z JSON dat vytvořen objekt `requestDto`, který parsuje vložený JSON na jednotlivé členské proměnné. V metodě `processRequest` je poté podle typu `requestDto` zavolán daný výpočet. Po jeho dokončení je opačným způsobem v objektu `responseDto` parsován na `JSONObject` a ten je pak vrácen jako odpověď na příchozí socket.

Celá tato hierarchie je znázorněna na zjednodušeném diagramu 4.2.

4.2 TCP/IP klient v ASP.NET core aplikaci

Pro implementaci TCP/IP klienta byla v hlavní ASP.NET core aplikaci vytvořena třída `TCPClient`. V případě jednodušší, například konzolové aplikace, by stačilo vytvořit její novou instanci, a začít komunikovat. V této práci je však uvažováno, že hlavní ASP.NET core aplikace bude obsluhovat velké množství požadavků najednou, a proto by použití jedné instance klienta nebylo z hlediska výkonnosti postačující.

K efektivnější implementaci byla využita *Dependency Injection* a následující algoritmus. Hlavní stavební kámen ASP.NET core aplikace je kontroler. V něm by měla být vytvořena nová instance pro potřebu komunikace. Nastává zde ovšem problém, jelikož kontroler je znovu inicializován pro každý HTTP request a pokud by byl pokaždé založen i `TCPClient` zbytečně by byla zatížena paměť i server nadbytečnými připojeními.

Z tohoto důvodu je v ASP.NET core aplikaci zaregistrována servisní třída `TCPClientService` jako *Singleton*, existuje tedy po celou dobu běhu programu jen jedna její instance a ta v sobě uchovává seznam instancí všech aktivních TCP/IP klientů pro dané uživatele. Poskytuje také administraci nad těmito jednotlivými klienty a sice v případě, že je některý již určitou dobu neaktivní, spojení ukončí a klient je smazán. Tato implementace byla zvolena z důvodu minimalizace přenosového času pro jednotlivé výpočty.

Celý proce je tedy následující. Na kontroler přijde HTTP request s dotazem na vypočtená data. Kontroler si vyžádá od `TCPClientService` instanci `TCPCLient`, která mu podle jeho `ID` náleží. Ta mu ho poskytne, v případě, že žádnou vytvořenou ještě nemá, ji založí. A klient může komunikovat. Při každé komunikaci je zvýšen čas do smazání TCP/IP klienta kvůli neaktivitě. Když již není TCP klient potřeba, je po určité době smazán. Následující ukázka zdrojového kódu ukazuje implementaci právě `TCPCLientService` servisní třídy.

```
namespace PIDlab.services.TcpClient
{
    public class TCPClientService :
       ITCPClientService
    {
        private List<ClientHolder> _listOfCLients;
```

```
private Timer _timer;

public TCPClientService()
{
    _listOfCLients = new
       List<ClientHolder>();

    _timer = new Timer(maintainClients,
       null, 0, 50000);
}

public IPIDClient getClient(int userId)
{
    var key = _listOfCLients.Where( client
       => client.userId == userId).ToList();

    if(key.Count > 0)
    {
        var PIDcl = key.FirstOrDefault();

        PIDcl.deleteTime =
           PIDcl.deleteTime.AddSeconds(20);

        return PIDcl.client;
    }
    else
    {
        //je zalozen novy objekt
           ConnectinHoler a pridan do
           seznamu
    }
}

private void maintainClients(object state)
{
    //every 5 second delete inactive
       connections
    var timeNow = DateTime.Now;
```

```
            var listToRemove = new
                List<ClientHolder>();
                    // for each pro nalezeni
                       neaktivnich klientu
            listToRemove.ForEach( item =>
                _listOfCLients.Remove(item));
        }
```

4.3 Integrace výpočetních knihoven v jazyce Java

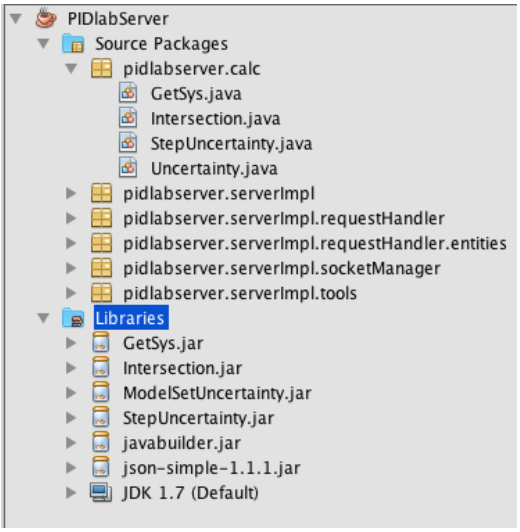
Díky zvolené architektuře, tedy hlavní aplikace naprogramovaná v ASP.NET core a vedle ní běžící výpočetní server v jazyce Java, je import knihoven, psaných v tomto jazyce a jejich využití o mnoho zjednodušeno. Daný JAR soubor, který je potřeba importovat do výpočetního serveru, musí být nejprve nahrán do projektu `PIDLabMathAPi`, viz obrázek 4.3, do složky Libraries a poté stačí importovat tento JAR soubor do dané třídy. Následně je možno jej začít používat viz následující zdrojový kód, kde dochází k importu výpočetního balíčku `ModelSetUncertainty`.

```
import com.mathworks.toolbox.javabuilder.*;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import ModelSetUncertainty.*;
```

Kvůli omezení Matlab runtime je nutné dodržet verzi Java Virtual Machine stejnou, jako obsahuje program Matlab, ve kterém jsou balíčky vytvořeny, v našem případě tedy Java Virutal Machine 1.7_75.

Integrace knihoven z jazyka C nebo C++ je taktéž možná díky Java Native Interface (JNI). Pro použití knihoven jazyka C, je tedy nejprve nutné naprogramovat v Javě takzvanou *wrapper*, česky obalující třídu. Následně příkazem `javah` dojde k vytvoření zdrojového souboru jazyka C s deklarací hlaviček potřebných funkcí. V dalším odděleném souboru by měla být implementace těchto funkcí. Nakonec již stačí příkazem `gcc` zkompilovat a slinkovat oba soubory, které je následně možné jako knihovnu importovat a použít v Java výpočetním serveru. Obdobným způsobem lze importovat i knihovny psané v jazyce C++. Další možností by bylo napsat jednoduché TCP/IP servery v jazyce C a C++ a zřetězit je za Java výpočetní server. Vyslaný socket reprezentující požadavek na výpočet by poté byl zachycen

správným serverem, vypočítán a odeslán zpět. ASP.NET core aplikace by v tomto případě nemusela o těchto serverech vůbec vědět. Toto řešení by však bylo náročnější na konfiguraci. Oproti tomu by, díky nativním C a C++ knihovnám, mohlo být výpočetně rychlejší. [46] [51]



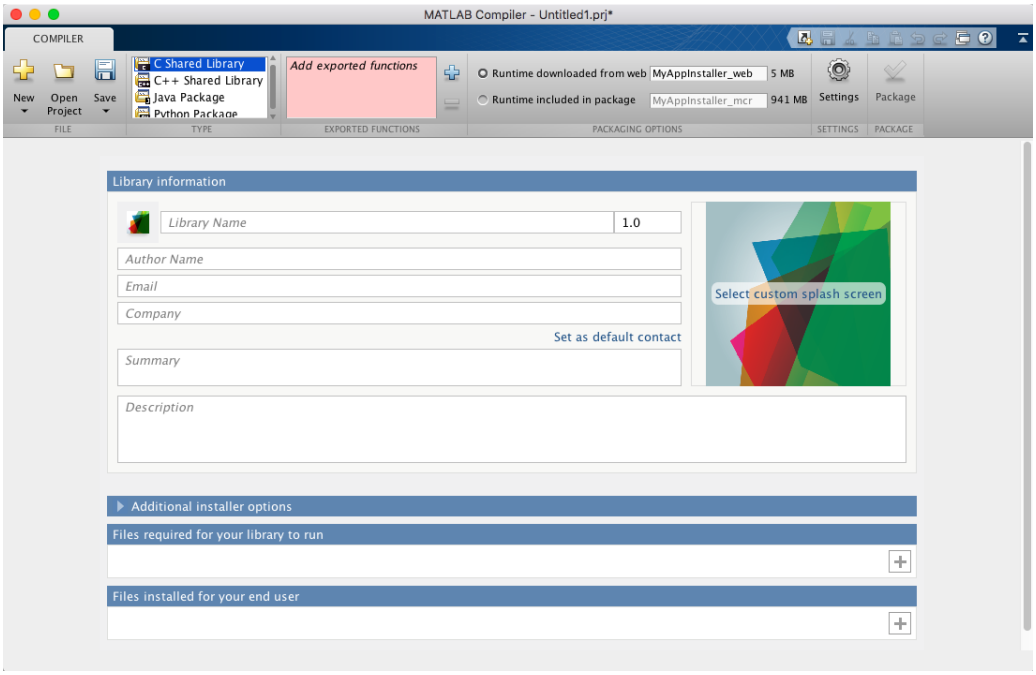
Obrázek 4.3: Ukázka importu knihoven v jazyce Java do pojektu výpočetního serveru

4.4 Integrace výpočetních modulů jazyka Matlab

Dalším úkolem, kterým se tato práce zabývá je vytvoření postupu pro integraci výpočetních modulů jazyka Matlab do naší aplikace. Program Matlab nabízí několik možností, jak má být jeho zdrojový kód zkompilován. Jednou z možností je kompilace jako *stand alone* aplikace pomocí programu Application Compiler. Další možností je kompilace do knihoven pro různá prostředí pomocí programu Library Compiler. V této bakalářské práci se budeme zabývat druhou možností, čili kompilace do výpočetní knihovny. Postup je následující. V nabídce `type` programu Library Compiler je možné vybrat platformu, pro kterou je knihovna kompilována. Na výběr je možnost kompilace jako knihovna pro Python, Javu, C, C++ a .NET framework viz obrázek 4.4.

V této práci využijeme naprogramovaný výpočetní server v jazyce Java, a tudíž budeme kompilovat Matlab moduly do balíčku v jazyce Java. K rozběhnutí takového balíčku je třeba nainstalovat Matlab runtime a to přesně ve verzi, která odpovídá verzi Matlabu, ve kterém byl balíček vytvořen.

Dále je potřeba mít nainstalovanou Java Virtual Machine přesně ve verzi,



Obrázek 4.4: Ukázka GUI aplikace Matlab Compiler

kterou interně obsahuje program Matlab, ve kterém byl balíček vytvořen. Následně je nutné zaregistrovat Matlab runtime do systémových proměnných na Windows do Path, na Linuxu do LD_LIBRARY_PATH a na MAC OS X do DYLD_LIBRARY_PATH a do LD_LIBRARY_PATH. Aby program po importu námi vytvořeného balíčku mohl provést výpočet v Matlab runtime, je potřeba importovat předpřipravený balíček Javabuilder v kořenovém adresáři instalace Matlab runtime, do složky `Libraries` v projektu výpočetního serveru viz obrázek 4.3.

Ten nám poskytuje datové typy pro práci s daty, která do Matlab Runtime vkládáme, a která vrací. Po instalaci a nastavení prostředí je možné kterýkoliv modul zkompilovaný do balíčku v jazyce Java použít v našem programu, jedinou podmínkou je, že verze programu Matlabu musí neustále odpovídat verzi Matlab runtime. Následuje ukázka zdrojového kódu v jazyce Java a sice použití třídy `MWNumericArray`, která obaluje vstup a výstup z výpočetního balíčku, tuto třídu nám poskytuje právě balíček `javabuilder`. Po inicializaci `MWNumericArray`, což je obalující třída pro data, je vytvořena instance naší výpočetní třídy `Class1`. Po zavolání výpočetní metody, kterou obsahuje je, výsledek uložen do pole objektů `result`.

```
double inputValue = 2.0;
MWNumericArray n = new MWNumericArray(inputValue,
```

```
    MWClassID.DOUBLE);

Class1 ourMatlabFunction = new Class1();
Object[] result =
    ourMatlabFunction.magicFunction(1, n);
```

5 Postup komunikace virtuální laboratoře se systémem REXYGEN

V následujícím úkolu se tato práce zabývá komunikací mezi virtuální laboratoří a systémem REXYGEN. K tomuto účelu bude využito webové REST API, které poskytuje běžící simulace v systému REXYGEN. To bude konzumováno klientem, implementovaným ve výpočetním serveru virtuální laboratoře.

5.1 Popis webového REST API systému REXYGEN

Program REXYGEN obsahuje jednoduché webové REST API, ke kterému je možné přistupovat jak přes prohlížeč, tak programově přes HTTP protokol, již od verze 2.50. Díky tomu je možné nahlížet jednak na parametry a data simulace, a jednak přímo za běhu tyto parametry měnit. Výchozí port, kde REXYGEN Web server naslouchá je 8008 pro HTTP a 8009 pro HTTPS (HTTP pře SSL) protokol. Adresa přes kterou je REST API přístupné, je potom ve tvaru `http://host:port/api`. Všechny zdroje jsou v REST API organizované do stromové struktury a každý zdroj obsahuje podpoložky, které jsou přístupné jako HTML, XML, text nebo JSON. Každý uzel tohoto stromu obsahuje následující vlastnosti.

1. data: reprezentují aktuální hodnotu, kvalitu a časovou značku konfiguračního parametru nebo signálu, který může být vstupem, výstupem, parametrem a dalším.

2. meta data: reprezentují informace o vlastnostech dat a mohou být například typ, rozmezí a další.

3. seznam subitemů: reprezentuje informace o dalších zdrojích, které jsou dětmi daného zdroje, seznam subitemů je součástí metadat.

První úroveň stromu zdrojů, kterou obsahuje cílové běhové prostředí, která je přístupná vždy na adrese `http://host:port/api`, a vypadá následovně.

1. modules: poskytuje seznam modulů.
2. archives: poskytuje seznam archivů.
3. drivers: poksytuje seznam ovladačů.
4. levels: poskytuje seznam úrovní.
5. tasks: poskytuje seznam úloh.
6. data: poskytuje přístup k signálům v běhovém prostředí.
7. system: poskytuje informace o cílovém zařízení.
8. info: poskytuje informace o běhové prostředí.

Ke každému dalšímu zdroji nižší úrovně se dostaneme přes stejnou adresu, za kterou následuje lomítko a jméno daného zdroje. Například `http://host:port/api/tasks`. K meta datům se poté dostaneme například přes následující URL `http://host:port/api/tasks?typVlastnosti&mime=mimeType`.

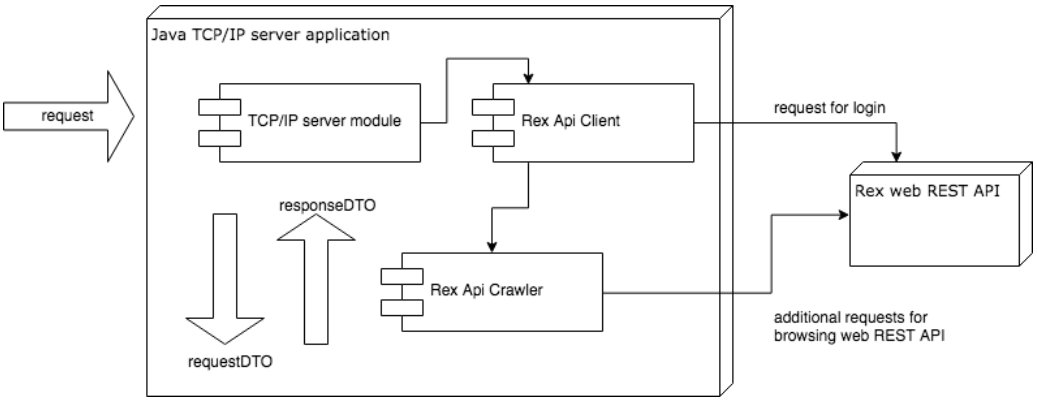
Webové REST API je zabezpečeno proti neautorizovaným přístupům. Je tedy nutné se nejdříve přihlásit. To lze provést buď přes HTTP Basic Authentication, nebo přes HTTP Cookie. Při přihlášení přes HTTP Basic Authentication se přistupuje k API přes adresu `http://user:password@host:port/api/tasks`. Při přihlášení přes HTTP Cookie je třeba nejprve odeslat na webový server HTTP request typu POST na adresu `http://host:port/login` s tělem, které obsahuje vyplněné údaje username, password a cílovou adresu, tu lze ponechat vyplněnou znakem /. Následně server ověří přihlašovací údaje a odešle zpět Cookie, se kterým se lze dotazovat na jednotlivé zdroje.

5.2 Implementace klienta pro konzumaci webového REST API ve virtuální laboratoři

Klient, který obsahuje implementaci pro připojování, přihlášení a získání dat z webového REST API systému REXYGEN, je obsažen ve výpočetním serveru, který je implementován v jazyce Java. Hlavní jeho třída, která se jmenuje `RexApiClient`, se stará o přijetí požadavku reprezentovaným třídou `requestDTO` a jeho zpracováním. Následně je vytvořen HTTP request typu POST pro přihlášení a odeslán na Webový server programu REXYGEN. Po úspěšném přihlášení je uložen vrácený soubor Cookie a následně je zakomponován do třídy `HttpClient`, která je poskytována balíčkem `org.apache.`

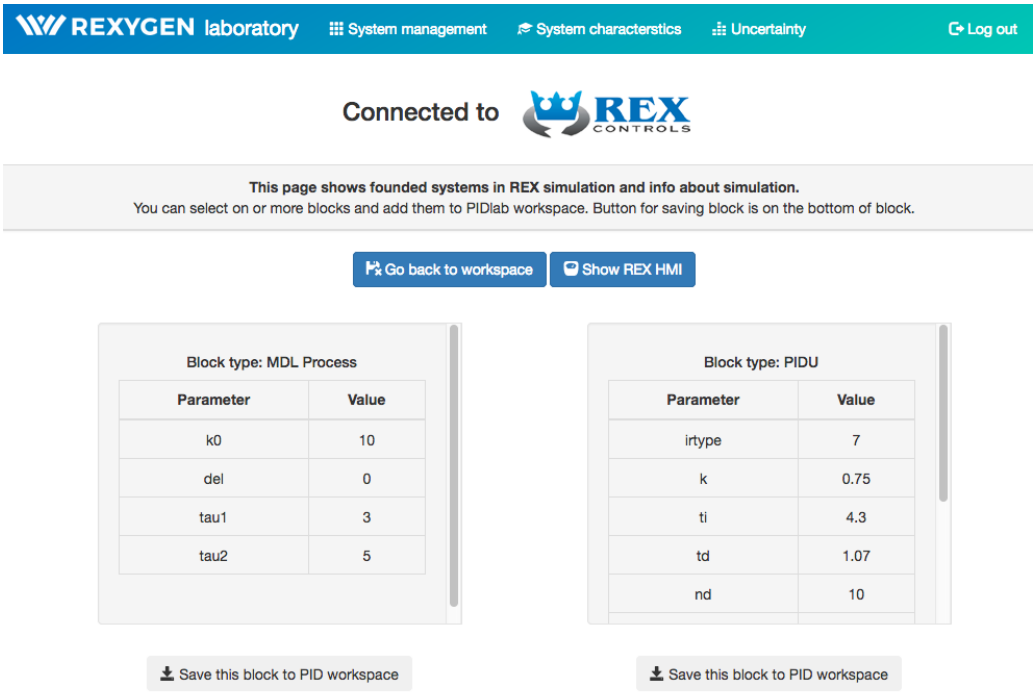
`http.impl.client.HttpClients`. Ta je poté předána jako parametr do třídy `RexApiCrawler`.

Tato třída se poté stará o procházení celého stromu REST API systému REXYGEN a vybrání námi hledaných bloků, které vrátí jako `LinkedList`, který obsahuje jejich jednotlivé adresy. Celé prohledávání stromu zdrojů je implementované jako prohledávání grafu do šířky s dynamickým načítáním sousedních uzlů, v našem případě reprezentované seznamem subitems v metadatech zdroje. Po nalezení a vrácení zmíněného objektu `LinkedList`, se provádění programu přesune zpět do třídy `RexApiClient`, která vytvoří dané entity podle adres ve vráceném seznamu a naplní je daty. Následně je vytvořena odpověď na dotaz, čili je vytvořena instance třídy `responseDTO` a vyplněné entity, jež reprezentující nalezené bloky, jsou vráceny do ASP.NET core aplikace. Celý proces je zobrazen na následujícím diagramu 5.1.



Obrázek 5.1: Diagram prohledávání webového REST API v Java výpočetním serveru

Po nalezení a odeslání bloků do ASP.NET core aplikace jsou bloky přijaty servisní třídou `RexRestClientService`, zpracovány v kontroleru `RexController` a předány do pohledu s názvem `ShowRexBlocks.cshtml`. To je zobrazeno na následujícím obrázku 5.2.



Obrázek 5.2: ShowRexBlocks view pro zobrazení nalezených bloků v simulaci programu REXYGEN

6 Možnosti provozování virtuální laboratoře v offline režimu na jednom počítači

Následující kapitola se zabývá popisem spuštění virtuální laboratoře na jednom počítači v offline režimu. To znamená, že všechny komponenty jako ASP.NET core aplikace, výpočetní server a databázový server poběží na lokálním serveru čili na `localhost`. Dále je popsána možnost provozování jednotlivých komponent na jednom serveru, nebo distribuovaně na více serverech.

6.1 Instalace prostřední nutných pro běh virtuální laboratoře

V prvé řadě je nutné stáhnout a nainstalovat .Net Core verze 2.0 a vyšší pro překlad a běh hlavní aplikace ASP.NET core. To je možné stáhnout na adrese `https://www.microsoft.com/net/download`. Následně je nutné stáhnou a nainstalovat přesnou verzi Java Virtual Machine a sice verzi 1.7_75. Tu je možné získat na adrese `http://www.oracle.com/technetwork/java/javase/downloads/`.

A poslední běhové prostřední, které virtuální laboratoř potřebuje ke svému běhu, je Matlab runtime přístupné na adrese `https://www.mathworks.com/products/compiler/matlab-runtime.html`. Je třeba nainstalovat verzi 9.0.1. Ta totiž musí odpovídat verzi programu Matlab, ve kterém byly vytvořeny výpočetní knihovny. Detailním popisem nastavení Matlab Runtime se věnuje podkapitola 4.4.

To jsou všechna běhová prostřední nutná pro běh virtuální laboratoře. Jedná se o multiplatformní běhová prostředí, uživatel si tak může vybrat, na jaké platformě je chce provozovat. Zbývá již jen stáhnout a nainstalovat databázový server MySQL. Zde lze doporučit použití předpřipravených balíků, které v sobě MySQL databázový server obsahují jíž nakonfigurovaný. Pro Mac OS to je například balík MAMP, pro platformu Windows například balík Vertrigo a konečně na Linux lze doporučit, při použití jedné z nejrozšířenějších distribucí Ubuntu, balíček LAMP, jehož instalací provádí následující článek [5]. Je však samozřejmě možné použít jakýkoliv libovolný

balík pro libovolnou podporovanou platformu. [3] [53]

6.2 Nastavení virtuální laboratoře pro běh v offline režimu na jednom počítači

Při spuštění virtuální laboratoře lokálně na jednom počítači postupujeme následujícím způsobem. Nejprve si stáhneme a extrahujeme projekt virtuální laboratoře. Poté přejdeme do složky PIDlab v kořenovém adresáři virtuální laboratoře. Otevřeme zde příkazovou řádku nebo terminál a zadáme příkaz `dotnet run`, tím dojde k přeložení a spuštění programu a spuštění Kestrel serveru, který hostí aplikaci virtuální laboratoře a naslouchá na portu 5000. Čili přistupovat k virtuální laboratoři je možno přes prohlížeč, zadáním následující adresy `http://localhost:5000`.

Následně je nutné spustit výpočetní server psaný v jazyce Java. Ten je možné spustit buď pomocí programu Ant, nebo editoru NetBeans, podrobně v kroku 8 Uživatelské příručky. Výchozí port pro TCP/IP komunikaci mezi virtuální laboratoří a výpočetním serverem je nastaven na 8989. Pokud by jej z nějakého důvodu nebylo možné použít, lze ho přenastavit viz kroky 4 a 5 Uživatelské příručky.

Poté je potřeba spustit databázový server MySQL a upravit takzvaný connection string v nastavení virtuální laboratoře viz krok 3 Uživatelské příručky. Kompletní uživatelská příručka, která krok po kroku provádí nastavením a spuštěním virtuální laboratoře je obsažena v příloze A v závěru této práce.

6.3 Nastavení laboratoře pro běh na jednom či více serverech

Virtuální laboratoř je možné provozovat na jednom nebo více serverech. Odděleně mohou běžet virtuální laboratoř, výpočetní server i databázový server. Nastavení je obdobné jako 6.2. Před server Kestrel, ve kterém běží virtuální laboratoř, je však doporučeno postavit jiný webový server jako reverzní proxy. K tomuto účelu je možné využít například server IIS, Apache nebo Nginx. Tato možnost je popsána v následujícím článku. [9]

7 Ověření vyvinuté platformy na vybraném příkladu

Vyvinutá platforma byla otestována v práci kolegy Michala Brabce, který se věnoval vizualizaci a výpočtu modelu neurčitosti z několika vstupních proměnných. První graf na stránce Uncertainty pod záložkou MF se zabývá výpočtem modelu neurčitosti ze zadaného bodu frekvenční charakteristiky systému. Pod záložkou MM je vypočítáván model neurčitosti z momentů impulsní odezvy systému a pod záložkou Step je vypočítáván model neurčitosti z ohraničené přechodové charakteristiky.

Všechny výpočty byly nejprve realizovány v programu Matlab. Poté byly zkompilovány do balíčku v jazyce Java a importovány do výpočetního serveru. Po implementaci `Views` čili pohledů bylo možné ověřit funkčnost výpočtů. Tím byla celá tato cesta ověřena. Jediná vada na kráse je rychlost výpočtů a přenosu dat do hlavní ASP.NET core aplikace. První výpočet trvá vždy déle, kvůli inicializaci a startu Matlab Runtime. Poté jsou výpočty znatelně rychlejší. Nicméně by bylo potřeba dále zrychlit celý proces. Nyní je virtuální laboratoř schopna přepočítat výsledky v řádu jednotek vteřin. Tento čas by bylo možné nadále zkrátit implementací Java výpočetního serveru pomocí NIO.2 Api pro asynchronní příjímání a odesílání soketů.

Tím však byla ověřena jen část implementované funkčnosti. Administrace bloků a jejich vyhledávání bylo ověřeno na jednoduchém scénáři. V programu REXYGEN byla lokálně spuštěna vzorová simulace a v té byly úspěšně vyhledány bloky `MDL Process` a `PIDU`, viz obrázek 5.2. A dále byly oba v pořádku přidány do pracovního prostředí virtuální laboratoře, viz obrázek 3.5, kde byl dále úspěšně smazán blok PIDU. Tím byla ověřena funkčnosti klienta pro konzumaci webového REST Api, komunikace s databází MySQL a jejich vizualizace.

8 Závěr

V předkládané bakalářské práci byla navržena a implementována architektura virtuální laboratoře, jež byla rozdělena na hlavní ASP.NET core aplikaci, která komunikuje s uživatelem a databázovým systémem, a na výpočetní server psaný v jazyce Java. Pro uchovávání dat byl použit databázový systém MySQL. ASP.NET core aplikace komunikuje s výpočetním serverem psaným v jazyce Java přes protokol TCP/IP, jehož sokety přenáší informace mezi oběma servery reprezentované řetězci JSON. Architekturou a použitými programovacími technikami se zabývá třetí kapitola této práce.

Výpočetní server má na starosti provádění výpočtů a komunikaci se systémem REXYGEN. Díky implementaci výpočetního serveru v jazyce Java bylo možné pro výpočty využít již existující výpočetní knihovny na Katedře kybernetiky, které jsou ve většině případů implementovány právě v tomto jazyce. Dále bylo možné importovat a pro výpočty použít zkompilované skripty programu Matlab. Ty však potřebují pro svůj běh Matlab Runtime. Ověřením těchto dvou cest možných výpočetních zdrojů byla získána velká variabilita při implementaci a okamžitém použití matematických knihoven. Detailním popisem integrace těchto knihoven se zabývá čtvrtá kapitola této práce.

V neposlední řadě se podařilo ve výpočetním serveru implementovat klienta pro konzumaci webového REST API v běžící simulaci v systému REXYGEN. Po přihlášení je tedy možné se připojit, zkrze virtuální laboratoř, na běžící experiment v programu REXYGEN, získat z něj potřebné bloky tohoto experimentu a přidat je do pracovního prostředí virtuální laboratoře k dalšímu zpracování. Samozřejmě lze jednoduše doimplementovat také opačnou cestu, tedy že například nastavený regulátor bude nahrán do tohoto běžícího experimentu v reálném čase. Komunikaci se systémem REXYGEN je věnována pátá kapitola předkládané práce.

V předposledním šestém úkolu byl popsán způsob zprovoznění a spuštění virtuální laboratoře jednak na jednom počítači v režimu offline a jednak na serveru či serverech, protože některé součásti laboratoře je možné provozovat odděleně. Například ASP.NET core aplikaci, Java výpočetní server i MySQL server mohou běžet odděleně.

Na závěr v sedmém úkolu byla ověřena funkčnost implementované laboratoře díky bakalářské práci kolegy Michala Brabce, který vyvinutou platformu využil pro implementaci front-endové části výpočtů v podobě grafů pro model neurčitosti a dále pak ověřil funkčnost komunikace ASP.NET core

aplikace s výpočetním serverem. Podařilo se mu implementovat výpočetní moduly v programu Matlab, které pak zkompiloval do balíčku v jazyce Java a úspěšně je použil ve výpočetním serveru virtuální laboratoře. Tím byla celá platforma ověřena.

Seznam použitých zkratek

- Ajax - Asynchronous JavaScript and XML
- AOT - Ahead of time
- Api - Application Programming Interface
- ASP - Active Server Pages
- BCL - Base Class Library
- CERN - Centre Européenme pour la Recherche Nucléaire
- CLI - Command line interface
- CLR - Common Language Runtime
- CoreFX - Core foundational libraries
- CoreCLR - Core Common Language Runtime
- CSS - Cascading style sheet
- DB - Database
- DOCTYPE - Document Type
- DOM - Document Object Model
- DOS - Disk Operating System
- DSSSL - Document Style Semantics and Specification Language
- DTO - Data transfer object
- ECMA - European Manufactures Association
- FCL - Framework Class Library
- FFTW - Fastest Fourier Transformation in the West
- GPL - General Public License
- GUI - Graphical User Interface
- HMI - Human Machine Interface
- HTML - Hypertext markup language
- HTTP - Hypertext Transfer Protocol
- HTTPS - Hypertext Transfer Protocol Secure
- IIS - Internet Information Server
- IL/MSIL Inermediate Language/ Microsoft Intermediate Language
- IMG - image

- IoC/DI - Inversion of Controll /Dependency Injection
- ISO - International Organization for Standardization
- I/O - Input Output
- JAR - Java Archive
- JDK - Java Development Kit
- JIT - Just in time
- JNI - Java Native Interface
- JRE - Java Runtime Enviroment
- JSON - JavaScript Object Notation
- JVM - Java Virtual Machine
- MCR - MATLAB Runtime
- MIT - Massachusetts Institute of Technology
- MVC - Model-View-Controller
- NCSA - The University of Illinois' National Center for Supercomputing Applications
- NT - New Technology
- OPC - Open Platform Communications
- PHP - Personal Home Page
- RDBMS - Relational database management system
- REST - Representational State Transfer
- RT - Real time
- SQL - Structured Query Language
- SSL - Secure Sockets Layer
- SW - Software
- TCP/IP - Transmission Control Protocol/Internet Protocol
- UI - User Interface
- URI - Uniform Resource Identifier
- URL - Uniform Resource Locator
- XML - Extensible Markup Language

Seznam obrázků

Literatura

[1] QuinStreet Inc. *Top 10 New Features in CSS 3* [online]. WebReference. Web Development and Design Tutorials, Tips and Reviews. Dostupné z: http://www.webreference.com/authoring/css3/index.html.

[2] Anderson, R. *Úvod do stránky Razor v ASP.NET Core* [online]. 2017. Microsoft Docs. Dostupné z: https://docs.microsoft.com/cs-cz/aspnet/core/mvc/razor-pages/?tabs=visual-studio.

[3] appsolute GmbH. *MAMP and MAMP PRO* [online]. 2004 - 2018. MAMP: My Apache - MySQL - PHP. Dostupné z: https://www.mamp.info/en/.

[4] Balda, P. Programové prostředky řízení, 2017. Řídící systém reálného času REX.

[5] Bearnes, B. *How To Install Linux, Apache, MySQL, PHP (LAMP) stack on Ubuntu 16.04* [online]. 2016. DigitalOcean: Cloud Computing, Simplicity at Scale. Dostupné z: https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu-16-04.

[6] Bevacqua, N. *ECMAScript® 2017 Language Specification* [online]. 2015. Mozilla Hacks – the Web developer blog. Dostupné z: https://hacks.mozilla.org/2011/01/ecmascript-5-strict-mode-in-firefox-4/.

[7] Torre, C. *Web Applications with ASP.NET Core Architecture and Patterns guidance (Updated for .NET Core 2.0)* [online]. 2017. .NET Blog A first-hand look from the .NET engineering teams. Dostupné z: https://blogs.msdn.microsoft.com/dotnet/2017/08/09/web-apps-aspnetcore-architecture-guidance/.

[8] Dykstra, T. *Kestrel web server implementation in ASP.NET Core* [online]. 2018. Microsoft Docs. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers/kestrel?tabs=aspnetcore2x.

[9] Dykstra, T. et al. *Implementace webového serveru v ASP.NET Core Microsoft Docs* [online]. 2018. Web server implementations in ASP.NET Core. Dostupné z: https://docs.microsoft.com/cs-cz/aspnet/core/fundamentals/servers/?tabs=aspnetcore2x.

[10] Ecma International. *ECMAScript Language Specification* [online]. 2011. ECMAScript Language Specification - ECMA-262 Edition 5.1. Dostupné z: https://www.ecma-international.org/ecma-262/5.1/.

[11] Envato Pty Ltd. *The History of HTML* [online]. 2018. HTML 2.0 The history of HTML. Dostupné z: http://historyofhtml.web.unc.edu/home/html-2/.

[12] Heilmann, C. *ECMAScript 5 strict mode in Firefox 4* [online]. 2011. Mozilla Hacks – the Web developer blog. Dostupné z: https://hacks.mozilla.org/2011/01/ecmascript-5-strict-mode-in-firefox-4/.

[13] Hickson, I. *Acid Tests* [online]. Acid Tests. Dostupné z: http://www.acidtests.org.

[14] Hissom-Daugherty, A. E. *Introduction to HTML5 and CSS3* [online]. Hisotry of HTML and CSS. Dostupné z: http://amyhissom.com/HTML5-CSS3/history.html.

[15] Joynet, N. *Node.js* [online]. 2018. Node.js Home page. Dostupné z: https://nodejs.org/en/.

[16] Laboratory of Molecular Cytogenetics, Institute of Plant Molecular Biology, Biology Centre AS CR, Ceske Budejovice, Czech Republic. *RepeatExplorer Galaxy* [online]. RepeatExplorer for discover repeats in next generation sequencing data. Dostupné z: https://galaxy-elixir.cerit-sc.cz.

[17] Lander, R. *.NET Core Guide Micorosft Docs* [online]. 2016. .NET Core Guide. Dostupné z: https://docs.microsoft.com/en-us/dotnet/core/.

[18] Lander, R. *.NET Core Libraries (CoreFX)* [online]. 2017. GitHub - dotnet/corefx: Readme. Dostupné z: https://github.com/dotnet/corefx.

[19] Lander, R. *Intro to .NET Native and CoreRT* [online]. 2017. corert/intro-to-corert.md at master · dotnet/corert · GitHub. Dostupné z: https://github.com/dotnet/corert/blob/master/Documentation/intro-to-corert.md.

[20] Lander, R. *.NET Core 2.0 - Supported OS versions* [online]. 2018. core/2.0-supported-os.md at master · dotnet/core · GitHub. Dostupné z: https://github.com/dotnet/core/blob/master/release-notes/2.0/2.0-supported-os.md.

[21] Longman, A. W. *Chapter 2* [online]. 1998. A history of HTML. Dostupné z: https://www.w3.org/TR/2017/REC-html51-20171003/.

[22] MariaDB. *Old MySQL Versions* [online]. 2018. MariaDB Knowledge Base. MariaDB | Open Source Database (RDBMS) for the Enterprise. Dostupné z: https://mariadb.com/kb/en/library/old-mysql-versions/.

[23] Moler, C. *The Origins of MATLAB* [online]. 1994 - 2018. MathWorks - Makers of MATLAB and Simulink. Dostupné z: https://www.mathworks.com/company/newsletters/articles/the-origins-of-matlab.html.

[24] Moler, C. *MathWorks* [online]. 1993 - 2018. Makers of MATLAB and Simulink. Dostupné z: https://www.mathworks.com/content/dam/mathworks/tag-team/Objects/t/72887_92020v00Cleve_Growth_MATLAB_MathWorks_Two_Decades_Jan_2006.pdf.

[25] Mozilla and individual contributors. *MDN web docs* [online]. 2005-2018. async function. Dostupné z: https://hacks.mozilla.org/2011/01/ecmascript-5-strict-mode-in-firefox-4/.

[26] Oracle. *1.3.1 What is MySQL?* [online]. 2018. MySQL :: MySQL 5.7 Reference Manual. Dostupné z: https://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html.

[27] Oracle. *1.4 What Is New in MySQL 5.5.* [online]. 2018. MySQL :: MySQL 5.5 Reference Manua. Dostupné z: https://dev.mysql.com/doc/refman/5.5/en/mysql-nutshell.html.

[28] Oracle. *1.4 What Is New in MySQL 5.6.* [online]. 2018. MySQL :: MySQL 5.6 Reference Manual. Dostupné z: https://dev.mysql.com/doc/refman/5.6/en/mysql-nutshell.html.

[29] Oracle. *1.3.3 History of MySQL* [online]. 2018. MySQL :: MySQL 5.7 Reference Manual :: 1.3.3. Dostupné z: https://dev.mysql.com/doc/refman/5.7/en/history.html.

[30] Oracle. *Oracle Announces General Availability of MySQL 5.7.* [online]. 2018. New version of the world's most popular open source database is up to 3x faster than MySQL 5.6 in benchmark tests. Dostupné z: https://www.oracle.com/corporate/pressrelease/mysql-5-7-ga-101915.html.

[31] Oracle. *Oracle and Sun Microsystems* [online]. 2011. Strategic Acquisitions | Oracle. Dostupné z: https://www.oracle.com/sun/index.html.

[32] Oracle. *Java Timeline* [online]. 2018. Java Timeline. Dostupné z: http://oracle.com.edgesuite.net/timeline/java/.

[33] Oracle. *Java Virtual Machine Monitoring,Troubleshooting, and Profiling Tool* [online]. 1993 - 2018. jvisualvm - Java Virtual Machine Monitoring,Troubleshooting, and Profiling Tool. Dostupné z: https://docs.oracle.com/javase/7/docs/technotes/tools/share/jvisualvm.html.

[34] Otto, M. *Introduction Bootstrap* [online]. 2018. Get started with Bootstrap, the world's most popular framework for building responsive, mobile-first sites, with BootstrapCDN and a template starter page. Dostupné z: https://getbootstrap.com/docs/4.0/getting-started/introduction/.

[35] Otto, M. *Getting started* [online]. 2018. An overview of Bootstrap, how to download and use, basic templates and examples, and more. Dostupné z: https://getbootstrap.com/docs/4.0/getting-started/introduction/.

[36] Programujte.com. *Programujte.com — odborný web zaměřený na oblast vývoje, návrhu a designu webových, mobilních a desktopových aplikací* [online]. 2003. HTML5 – Články – Programujte.com. Dostupné z: http://programujte.com/clanky/125-html5/.

[37] Quinn, L. *What's New in HTML 4* [online]. 1998-2006. New Elements in HTML 4.0. Dostupné z: https://www.tutorialspoint.com/html/what_is_new_in_html4.htm.

[38] Rastogi, P. *Introduction to Identity on ASP.NET Core* [online]. 2018. Microsoft Docs. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?tabs=visual-studio%2Caspnetcore2x.

[39] Rauschmayer, A. *Chapter 6. Historical JavaScript Milestones* [online]. 2014. Speaking JavaScript: An In-Depth Guide for Programmers. Dostupné z: http://speakingjs.com/es5/ch06.html.

[40] Refsnes Data. *HTML5 Introduction* [online]. 1999-2018. W3Schools Online Web Tutorials. Dostupné z: https://www.w3schools.com/html/html5_intro.asp.

[41] Refsnes Data. *JavaScript Versions* [online]. 1999-2018. W3Schools Online Web Tutorials. Dostupné z: https://www.w3schools.com/js/js_versions.asp.

[42] Rendle, M. *A (Hitchhiker's) Guide To The .NET Core Projects on GitHub* [online]. 2016. The .NET Core Projects. Dostupné z: https://blog.rendle.io/a-guide-to-the-net-projects-on-github/.

[43] REX Controls. *Řídicí systém REX* [online]. 2000 - 2018. REX Controls - Pokročilá automatizace, měření a regulace. Dostupné z: https://www.rexcontrols.cz/rex.

[44] REX Controls. *Funkční bloky systému REX* [online]. 2017. REX Controls - Pokročilá automatizace, měření a regulace. Dostupné z: https://www.rexcontrols.cz/media/doc/CZECH/MANUALS.

[45] Roth, D. *Introduction to ASP.NET Core* [online]. 2017. Introduction to ASP.NET Core | Microsoft Docs. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/.

[46] Salonen, J. *Calling C From Java Is Easy* [online]. 2018. The Mindful Programmer. Dostupné z: http://jonisalonen.com/2012/calling-c-from-java-is-easy/.

[47] Shea, D. *CSS Zen Garden* [online]. The Beauty of CSS Design. Dostupné z: http://www.csszengarden.com/.

[48] Smith, S. *Dependency injection in ASP.NET Core* [online]. 2016. Microsoft Docs. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection.

[49] SSS IT Pvt Ltd. *MySQL History - javatpoint* [online]. 2011. Tutorials - Javatpoint. Dostupné z: Dostupnéz:https://www.javatpoint.com/mysql-history.

[50] SSS IT Pvt Ltd. *History of Java - Javatpoint* [online]. 2011. Tutorials - Javatpoint. Dostupné z: https://www.javatpoint.com/history-of-java.

[51] The Breakfast Post. *Wrapping a C++ library with JNI – introduction* [online]. 2018. The Breakfast Post. Dostupné z: Dostupnéz:https://thebreakfastpost.com/2012/01/21/wrapping-a-c-library-with-jni-introduction/.

[52] The Jquery Foundation. *Jquery Api Documentation* [online]. 2018. async function. Dostupné z: http://api.jquery.com.

[53] VertrigoServ WAMP Server. *VertrigoServ - PHP, Apache, MySQL* [online]. 2004 - 2018. Freeware WAMP Server. Dostupné z: https://www.vswamp.com/.

[54] WEBFLUX. *W3C Release Series of Updated CSS3 Specifications* [online]. 2009. All you ever needed to know about CSS3. Dostupné z: http://www.css3.info/w3c-release-series-of-updated-css3-specifications/.

[55] Wenzel, M. *.NET Core command-line interface (CLI) tools* [online]. 2017. .NET Core Command-Line Interface (CLI) Tools | Microsoft Docs. Dostupné z: https://docs.microsoft.com/en-us/dotnet/core/tools/?tabs=netcore2x.

[56] Wnzel, M. *.NET Core and Open-Source Microsoft Docs* [online]. 2017. .NET Core and Open-Source. Dostupné z: https://github.com/dotnet/core/blob/master/release-notes/2.0/2.0-supported-os.md.

[57] World Wide Web Consortium (W3C). *HTML 5.1 2nd Edition* [online]. 2017. HTML 5.1 2nd Edition Documentation. Dostupné z: https://www.w3.org/TR/2017/REC-html51-20171003/.

[58] World Wide Web Consortium (W3C). *A brief history of CSS until 2016* [online]. The CSS saga. Dostupné z: https://www.w3.org/Style/CSS20/history.html.

[59] Zine EOOD. *12 Awesome CSS3 Features* [online]. 2017. 12 Awesome CSS3 Features That You Can Finally Start Using. Dostupné z: https://tutorialzine.com/2013/10/12-awesome-css3-features-you-can-finally-use.

Přílohy

Příloha A

Uživatelská příručka

Následující seznam uvádí a popisuje jednotlivé kroky potřebné ke spuštění a provozu virtuální laboratoře.

1. **Instalace závislostí**: Všechny závislosti, které jsou nutné pro běh aplikací, jejich instalace a nastavení, jsou popsány v kapitole 6.

2. **Stažení a extrahování**: Po úspěšné instalaci a nastavení závislostí je třeba stáhnout, extrahovat a otevřít složku s virtuální laboratoří.

3. **Konfigurace databázového připojení ASP.NET core aplikace**: V kořenové složce přejdeme do složky `PIDlab` a v preferovaném editoru otevřeme soubor `appsettings.json`. Nyní je třeba upravit položku `MySqlConnectionString` tak, aby souhlasila s nastavením Vámi nainstalovaného databázového serveru MySQL. Je tedy nutné zadat server, port, jméno databáze, které je pidlab a dále jméno a heslo pro přihlášení do databázového serveru.

4. **Konfigurace ASP.NET core TCP/IP klienta**: Z kořenové složky virtuální laboratoře se přesuneme do složky `TCPClient` a dále do složky `client`, kde v editoru otevřeme soubor `TCPClient.cs`. Zde je třeba v metodě `ConnectToServer` nastavit `ipAddress` čili IP adresu Java výpočetního serveru a jeho `port`, kde poslouchá.

5. **Konfigurace Java výpočetního serveru**: Z kořenové složky virtuální laboratoře se přesuneme do složky `/PIDlabMathApi/PIDlabServer/src/pidlabserver/serverImpl/`. Zde v editoru otevřeme soubor `PIDlabServer.java`, kde je možné nastavit port, na kterém bude výpočetní server poslouchat. Pozor! Je nutné, aby port ve výpočetním serveru souhlasil s portem a IP adresou v nastaveni TCP/IP klienta z předchozího kroku.

6. **Spuštění MySQL serveru**: Nyní je potřeba spustit Vámi nainstalovaný MySQL server.

7. **Spuštění ASP.NET core aplikace**: Z kořenové složky virtuální laboratoře se přesuneme do složky `PIDlab`, kde otevřeme terminál a zadáme příkaz `dotnet run`. Ten zkompiluje a spustí ASP.NET core aplikaci hostovanou v Kestrel serveru, která poslouchá na serveru localhost portu 5000.

8. **Spuštění Java výpočetního serveru**: Pro sestavení a spuštění Java výpočetního serveru je možné postupovat dvěma cestami. Zaprvé Java výpočetní server je psaný v editoru NetBeans, je tedy možné ho v tomto programu otevřít a s jeho pomocí sestavit a spustit. Druhá možnost je využít terminál a program apache ant, který je možné stáhnout a nainstalovat z této URL: `https://ant.apache.org/bindownload.cgi`. Poté se z kořenové složky virtuální laboratoře přesuneme do složky `PIDlabMathApi` a dále do `PIDlabServer`, kde spustíme terminál a zadáme příkaz `ant run`. Poté dojde k sestavení a spuštění výpočetního serveru přímo v terminálu.

9. **Otevření virtuální laboratoře v prohlížeči**: Otevřeme preferovaný prohlížeč a přejdeme na adresu: `http://localhost:5000`. Zde se otevře úvodní obrazovka pro přihlášení.

10. **Přihlášení**: Pro přihlášení je nutné zadat výchozí přihlašovací údaje, které jsou vloženy do databáze ze zakládajícího skriptu. Přihlašovací jméno je admin a heslo je také admin.

11. **Přidávání, mazání a úprava bloků**: V hlavním okně virtuální laboratoře jsou výchozí dvě dlaždice, jedna s velkým symbolem plus a druhá pro přidávání bloků z webového REST Api simulace v programu REXYGEN. Pro přidání vlastního kroku klikněte na první dlaždici a vyplňte příslušné údaje. Po kliknutí na tlačítko save je zvalidována, uložena do databáze a ihned zobrazena v pracovním prostředí virtuální laboratoře. Poté je možné ji editovat, kliknutím na tlačítko s ozubeným kolečkem, nebo smazat, kliknutím na tlačítko s červeným křížkem.

12. **Vyhledání bloků ve webovém REST Api běžící simulace v programu REXYGEN**: Pro připojení a vyhledání bloků ve webové REST Api v programu REXYGEN je nutné kliknout na druhou dlaždici popisovanou o krok výše. Následně vyplnit potřebné údaje, kliknout na connect a chvíli počkat, než se laboratoř připojí k REXYGENU. Následně je ukázána stránka s vyhledanými bloky, které je možné přidat tlačítkem `Save this block to PID workspace` pod ka-

žým z nich nebo je možné se vrátit zpět do pracovního prostředí díky tlačítku v horní části obrazovky `Go back to workspace`.