

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE  
FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

Evidenčné číslo: FEI-5384-6041

**TVORBA VIRTUÁLNEHO LABORATÓRIA S  
VYUŽITÍM JAVASCRIPT-U NA STRANE SERVERA  
DIPLOMOVÁ PRÁCA**

**2016**

**Bc. Erich Stark**

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE**  
**FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

Evidenčné číslo: FEI-5384-6041

**TVORBA VIRTUÁLNEHO LABORATÓRIA S**  
**VYUŽITÍM JAVASCRIPT-U NA STRANE SERVERA**  
**DIPLOMOVÁ PRÁCA**

Študijný program: Aplikovaná informatika  
Číslo študijného odboru: 2511  
Názov študijného odboru: 9.2.9 Aplikovaná informatika  
Školiace pracovisko: Ústav informatiky a matematiky  
Vedúci záverečnej práce: Ing. Pavol Bisták, PhD.

**Bratislava 2016**

**Bc. Erich Stark**



## ZADANIE DIPLOMOVEJ PRÁCE

Študent: **Bc. Erich Stark**  
ID študenta: 6041  
Študijný program: Aplikovaná informatika  
Študijný odbor: 9.2.9. aplikovaná informatika  
Vedúci práce: Ing. Pavol Bisták, PhD.  
Miesto vypracovania: Ústav automobilovej mechatroniky

Názov práce: **Tvorba virtuálneho laboratória s využitím JavaScript-u na strane servera**

Špecifikácia zadania:

Cieľom práce je preskúmanie možností JavaScript-u pre realizáciu serverovej časti virtuálneho laboratória a vytvorenie funkčného virtuálneho laboratória, kde klient aj server budú realizovaný s využitím JavaScript-u.

Úlohy:

1. Naštudujte problematiku virtuálnych laboratórií a zoznámte sa s existujúcimi riešeniami.
2. Navrhnite objektový model virtuálneho laboratória. Definujte jednotlivé subsystémy a ich rozhrania. Využite možnosti JavaScript-u.
3. Implementujte serverovú časť aplikácie pre virtuálne laboratórium s využitím Node.js.
4. Pre overenie funkčnosti vytvorte klientskú časť aplikácie.
5. Aplikácie otestujte a porovnajte s predošlými riešeniami založenými na technológiách Java a .NET. Vypracujte technickú dokumentáciu.

Zoznam odbornej literatúry:

1. 1. Davoli, F.; Meyer, N.; Pugliese, R.; Zappatore, S.: Remote Instrumentation and Virtual Laboratories. Springer Verlag, 2010. ISBN 978-1-4419-5597-5
2. 2. G. Rauch: Smashing Node.js: JavaScript Everywhere. Wiley, 2. vydanie, 2012, ISBN 978-1119962595.

Riešenie zadania práce od: 21. 09. 2015

Dátum odovzdania práce: 20. 05. 2016



**Bc. Erich Stark**

študent



**prof. RNDr. Otokar Grošek, PhD.**

vedúci pracoviska



**prof. RNDr. Gabriel Juhás, PhD.**

garant študijného programu

# SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE  
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:	Aplikovaná informatika
Autor:	Bc. Erich Stark
Diplomová práca:	Tvorba virtuálneho laboratória s využitím JavaScript-u na strane servera
Vedúci záverečnej práce:	Ing. Pavol Bisták, PhD.
Miesto a rok predloženia práce:	Bratislava 2016

Diplomová práca sa zaoberá využitím možností modernej platformy Node.js v oblasti virtuálnych laboratórií a vytvorením referenčnej aplikácie v spojení ďalších technológií ako Matlab, Simulink, Angular.js a MongoDB. V úvode práce sú popísané vlastnosti virtuálnych laboratórií a jeho možných komponentov. Taktiež pojednávame o možnostiach interakcie s experimentami. V ďalšej časti boli porovnané už existujúce riešenia a ich možné nedostatky v súčasnosti. Nasleduje sekcia, kde sme si definovali technológie a ich hlavné vlastnosti, ktoré sme plánovali využiť. Implementácia riešenia prebiehala vytvorením menších častí. Ako prvú sme implementovali referenčnú simuláciu šikmého vrhu do Matlabu a Simulinku. Bolo potrebné získať dáta zo Simulinku do Matlab workspace. Ten ich následne posiela do Node.js pomocou RESTful služieb. Na strane Node.js čaká na dáta Socket.io, ktorý ich pošle do webového prehliadača. Posledná časť hovorí o vizualizácii dát v prehliadači vo forme grafu, animácie a tabuľky a následné zapísanie do databázy. Výsledkom práce je funkčné riešenie, kde je možné implementovať vlastnú simuláciu.

Kľúčové slová: virtuálne laboratórium, web, html, javascript, angular.js, json, mongodb, node.js, npm, express.js, restful, matlab, simulink

# ABSTRACT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

Study Programme:	Applied Informatics
Author:	Bc. Erich Stark
Diploma Thesis:	Virtual laboratory using JavaScript on the server side
Supervisor:	Ing. Pavol Bisták, PhD.
Place and year of submission:	Bratislava 2016

This thesis deals with possibilities of using modern Node.js platform in virtual laboratories and create a reference application in combination of other technologies such as Matlab, Simulink, Angular.js and MongoDB. In the introduction we described the characteristics of virtual laboratories and its possible components. We also discussed the possibilities of interaction with the experiment. In the next section we compared existing solutions and their possible lack in nowadays. The following is a section where we have defined the technology and their main characteristics that we planned to use. Implementation of solution was carried out by creating smaller parts. At first we have implemented a simulation motion of projectile in Matlab and Simulink. It was necessary to get data from Simulink to Matlab workspace. Then we had to send them to Node.js using RESTful web services. On the side of Node.js was waiting Socket.io for data receiving that were sent to the web browser. The last part refers to the visualization of data in the browser in the form of graphs, animations, data table and subsequently write data into the database. The result of this thesis is a functional solution called StarkLab where is possible implement own simulation.

Keywords: virtual laboratory, web, html, javascript, angular.js, json, mongodb, node.js, npm, express.js, restful, matlab, simulink

## Vyhlásenie autora

Podpísaný Bc. Erich Stark čestne vyhlasujem, že som diplomovú prácu Tvorba virtuálneho laboratória s využitím JavaScript-u na strane servera vypracoval na základe poznatkov získaných počas štúdia a informácií z dostupnej literatúry uvedenej v práci.

Vedúcim mojej diplomovej práce bol Ing. Pavol Bisták, PhD.

Bratislava, dňa 9.5.2016

.....  
podpis autora

## Podakovanie

Ďakujem môjmu vedúcemu práce Ing. Pavlovi Bistákovi, PhD. za jeho odbornú pomoc, snahu, zhovievavosť, pripomienky a cenné rady, ktoré mi boli poskytnuté pri vypracovaní diplomovej práce. Tiež sa chcem podakovať Petrovi Širkovi, ktorý mi vnúkol myšlienku využiť modernú technológiu Node.js. Moja veľká vďaka patrí tiež rodičom za podporu počas celého štúdia a priateľke za trpezlivosť a porozumenie v čase písania tejto práce.



# Obsah

Úvod	14
<b>1 Cieľ práce</b>	<b>15</b>
<b>2 Virtuálne laboratória</b>	<b>16</b>
2.1 Prehľad existujúcich virtuálnych laboratórií . . . . .	18
2.1.1 Nevýhody existujúcich riešení . . . . .	18
2.2 Komponenty virtuálneho laboratória . . . . .	19
<b>3 Použité technológie</b>	<b>20</b>
3.1 Matlab R2015b . . . . .	20
3.1.1 Simulink . . . . .	20
3.1.2 Komunikácia medzi Matlabom a Node.js . . . . .	20
3.2 Node.js . . . . .	24
3.2.1 História . . . . .	24
3.2.2 Architektúra . . . . .	24
3.2.3 Event loop . . . . .	27
3.2.4 Možnosti a využitie . . . . .	28
3.3 Node Package Manager . . . . .	28
3.3.1 Použitie modulov . . . . .	29
3.3.2 Vstavane moduly . . . . .	29
3.3.3 Vytvorenie web servera pomocou HTTP modulu . . . . .	29
3.4 Express.js . . . . .	30
3.4.1 Web server . . . . .	30
3.4.2 Objekty Request a Response . . . . .	31
3.4.3 Routing . . . . .	31
3.4.4 Middleware . . . . .	32
3.5 MongoDB . . . . .	32
3.6 Angular.js . . . . .	33
3.6.1 Koncept . . . . .	33
3.6.2 Scope . . . . .	34
3.6.3 Expressions . . . . .	36
3.6.4 Data Binding . . . . .	36
3.6.5 Controller . . . . .	37
3.6.6 Module . . . . .	37

3.6.7	Service . . . . .	38
3.6.8	Bootstrap aplikácie . . . . .	39
<b>4</b>	<b>Návrh a implementácia StarkLab</b>	<b>40</b>
4.1	Referenčný model simulácie v Matlabe . . . . .	40
4.2	Diagramy . . . . .	43
4.2.1	Prípad použitia . . . . .	43
4.2.2	Diagram "tried" pre StarkLab . . . . .	44
4.2.3	Sekvenčný diagram . . . . .	45
4.3	Databázový model MongoDB . . . . .	48
4.4	Node.js s frameworkom Express.js . . . . .	49
4.4.1	Vlastný middleware prihlasovania . . . . .	49
4.4.2	Spustenie Matlabu z príkazového riadku . . . . .	50
4.4.3	Práca s dátami simulácie . . . . .	51
4.4.4	Zápis dát do MongoDB . . . . .	51
4.5	Webový klient s frameworkom Angular.js . . . . .	54
4.5.1	Grafy s Chart.js . . . . .	55
4.5.2	Animácia pomocou html canvas . . . . .	56
4.5.3	Tabuľka dát . . . . .	57
4.5.4	Zobrazenie existujúcich simulácií . . . . .	58
	<b>Záver</b>	<b>61</b>
	<b>Zoznam použitej literatúry</b>	<b>62</b>
	<b>Prílohy</b>	<b>I</b>
	<b>A Štruktúra elektronického nosiča</b>	<b>II</b>
	<b>B Algoritmus</b>	<b>III</b>

## Zoznam obrázkov a tabuliek

Obrázok 1	Porovnanie medzi počítačovo riadeným experimentom vľavo a vzdialene riadeným experimentom vpravo. . . . .	17
Obrázok 2	HTTP request. . . . .	21
Obrázok 3	Architektúra Node.js platformy. . . . .	25
Obrázok 4	Udalostná slučka. . . . .	27
Obrázok 5	Spustenie web servera na porte 8081. . . . .	31
Obrázok 6	Objekt req a res ako parametre callback funkcie. . . . .	31
Obrázok 7	Vlastnosti middleware funkcie. . . . .	32
Obrázok 8	JSON dokument v MondoDB. . . . .	33
Obrázok 9	Koncept frameworku Angular.js. . . . .	34
Obrázok 10	Možnosti sledovania modelu v Angular.js. . . . .	35
Obrázok 11	Data binding v Angular.js. . . . .	36
Obrázok 12	Návrh komunikácie medzi jednotlivými komponentami. . . . .	40
Obrázok 13	Inicializačná funkcia šikmého vrhu v Matlabe. . . . .	41
Obrázok 14	Hodnoty v Matlab workspace po spustení funkcie. . . . .	42
Obrázok 15	Diagram prípadu použitia na prácu zo systémom. . . . .	44
Obrázok 16	Diagram "tried" pre náš systém. . . . .	45
Obrázok 17	Sekvenčný diagram pre prihlásenie pomocou LDAP. . . . .	46
Obrázok 18	Komunikácia medzi jednotlivými komponentami systému. . . . .	48
Obrázok 19	Model objektu v JavaScripte, pre vytvorenie záznamu v MongoDB. . . . .	48
Obrázok 20	Príklad záznamu simulácie v MongoDB. . . . .	49
Obrázok 21	Middleware kód v Express.js na kontrolu prihlásenia. . . . .	50
Obrázok 22	Spustenie príkazu v OS pomocou shelljs modulu. . . . .	51
Obrázok 23	Prijímanie dát a zasielanie pomocou socket.io do prehliadača. . . . .	51
Obrázok 24	Vloženie záznamu do MongoDB pomocou JavaScriptu. . . . .	52
Obrázok 25	Vyhľadanie záznamu v MongoDB pomocou JavaScriptu. . . . .	53
Obrázok 26	Volanie vloženia záznamu do MongoDB. . . . .	53
Obrázok 27	Volanie vyhľadania záznamu v MongoDB. . . . .	54
Obrázok 28	Prihlásenie do LDAP pomocou STUBA údajov. . . . .	54
Obrázok 29	Parametre simulácie - počiatočná rýchlosť a uhol v stupňoch. . . . .	55
Obrázok 30	Inicializácia hodnôt pre Chart.js. . . . .	56

Obrázok 31	Graf vykresľujúci závislosti $[x, y]$ v šikmom vrhu. . . . .	56
Obrázok 32	Animácia vykresľujúca závislosti $[x, y]$ v šikmom vrhu. . . . .	57
Obrázok 33	Tabuľka dát času, $x$ , $y$ a smer rýchlosti vy. . . . .	58
Obrázok 34	Zoznam uložených simulácií pre prihláseného užívateľa. . . . .	59
Obrázok 35	Graf a animácia pre vybranú vzorku dát. . . . .	60
Tabuľka 1	Porovnanie fyzických, virtuálnych a vzdialených a laboratórií . . .	17
Tabuľka 2	Porovnanie virtuálnych laboratórií vytvorených mimo FEI STU. .	18
Tabuľka 3	Porovnanie virtuálnych laboratórií vytvorených na FEI STU. . .	18

## **Zoznam skratiek a značiek**

VL - Virtual Laboratory

StarkLab - centrálna webová aplikácia nad Node.js serverom

REST - Representational state transfer

URL - Uniform Resource Locator

NPM - Node package manager

LTS - Long term support

API - Application programming interface

I/O - input and output

SSL - Secure Sockets Layer

TLS - Transport Layer Security

RDN - Relative Distinguished Name

COM - Component Object Model

MVC - Model View Controller

XML - eXtensible Markup Language

JSON - JavaScript Object Notation

HTTP - HyperText Transfer Protocol

URL - Uniform Resource Locator

CDN - Content Delivery Network

DOM - Document Object Model

LDAP - Lightweight Directory Access Protocol

## Zoznam algoritmov

1	Príklad POST requestu. . . . .	22
2	Príklad GET requestu. . . . .	23
3	Ukážka controlleru v Angular.js a jeho volanie na HTML elemente. . . . .	38
4	Ukážka vytvorenia modulov a pridávania funkcionality do nich. . . . .	38
B.1	Ukážka algoritmu . . . . .	III

# Úvod

Praktické cvičenie v laboratóriu je dôležitá súčasť pri procese vzdelávania technicky založených ľudí. Ako aj raz povedal starý čínsky filozof Confucius: *"Povedz mi a ja zabudnem, nauč ma a ja si spomeniem, ale nechaj ma zúčastniť sa a ja pochopím."* Zo skúseností už vieme, že človek sa najrýchlejšie učí tak, že si danú vec niekoľko krát sám vyskúša a tak najlepšie pochopí ako to funguje. Nanešťastie nie je možné vždy zabezpečiť výskumníkom alebo študentom priamy prístup k reálnym zariadeniam pre vykonanie experimentu. Problémov môže byť viacero: vysoká cena vybavenia laboratória, bezpečnosť na pracovisku v závislosti od experimentu prípadne nedostatok kvalifikovaných asistentov.

V posledných rokoch sa vývoj virtuálnych systémov zvýšil hlavne vďaka technologickej evolúcii softwarového inžinierstva. Pokrok moderných technológií nám dáva solídny základ pri tvorbe, či už všeobecne virtuálnych systémov nápomocných pre online výučbu, alebo konkrétnych virtuálnych laboratórií, kde sa simulujú fyzikálne javy a procesy. Pri experimentoch vykonávaných vo virtuálnom prostredí je možné zdieľať zdroje tohto prostredia na to, aby sa k nemu pripojilo viac užívateľov, ktorí chcú vykonávať rovnaký experiment, čo by pri reálnom zariadení nebolo možné. Vďaka tomu je virtuálne laboratórium vhodným doplnkom štúdia aj výskumu, kde je možné si skúsiť rôzne variácie experimentu bez ohrozenia na zdraví, prípadne zničenia zariadenia a až potom skúšať na reálnom zariadení ak je to potrebné.

# 1 Cieľ práce

Všeobecným cieľom diplomovej práce je analyzovať existujúce riešenia virtuálnych laboratórií a možností Node.js pre vytvorenie nového. Na základe zistených možností je potrebné vytvoriť virtuálne laboratórium ako klient-server architektúru, kde server bude Node.js a klienti matlab a webová aplikácia v prehliadači. Experiment vrámci virtualného laboratória prebieha ako simulácia v Matlabe cez rozšírenie Simulink. Táto aplikácia nebude obmedzená len na lokálnu sieť, ale bude prístupná aj z internetu. Klient aj server bude vytvorený v dynamicky typovanom jazyku JavaScript. Údaje z experimentu sa budú zasielať z Matlabu na server cez websockets, kde môžu byť následne spracované a uložené do databázy, alebo zaslané klientovi do prehliadača.

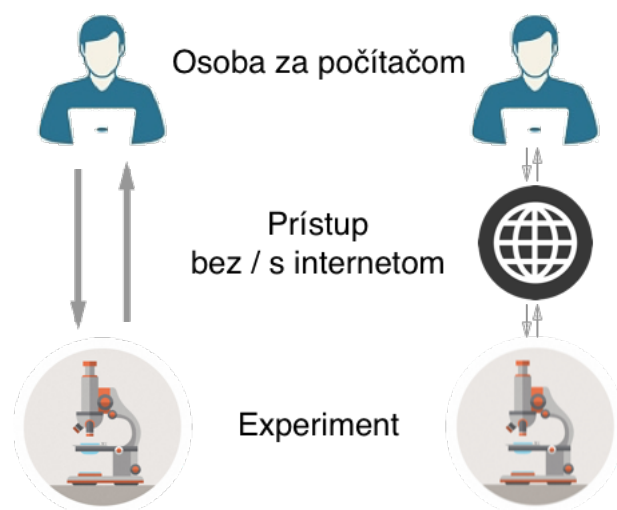


## 2 Virtuálne laboratória

V dobe keď internet ešte nebol rozšírený, experimenty sa robili vo fyzických laboratóriách. Bolo potrebné dodržiavať isté bezpečnostné predpisy, kvôli možnému úrazu osoby alebo možnému poškodeniu nástrojov.

Vzdialenosť a hlavne nedostatok finančných zdrojov nám sťažuje podmienky pri testovaní experimentov, hlavne v prípadoch keď je potrebné mať pokrokové sofistikované nástroje. Ďalší problém s ktorým sa stretávame je nedostatok kvalitných lektorov. V dnešnej dobe existujú online kurzy, ktoré poskytujú aj video ukážky, ale to tento problém rieši len čiastočne. Vždy bolo výzvou vykonávanie spoločných experimentov viacerými inštitúciami súčasne a zároveň zdieľanie nákladov na prostriedky. V súčasnej dobe internetu a počítačových technológií už tieto obmedzenia nemusia trápiť študentov ani výskumníkov. Internet umožnil to, že experimenty môžu byť štruktúrované tak, aby boli ovládané a prezerané na diaľku. Práve to by pomohlo v učení v základných ale aj pokročilých konceptoch prostredníctvom vzdialeného experimentovania. V súčasnosti veľa vybavenia už poskytuje rozhranie pre pripojenie počítača a spracovanie dát z neho. Vďaka tomu je možné navrhnúť experimenty, ktoré pomôžu študentom pri učení. Experimentovanie cez internet umožňuje využívanie zdrojov, znalostí, software a dát z internetu narozdiel od fyzických experimentov, ktoré by vznikali súčasne na rôznych miestach.[26]

V tejto práci sa budeme zaoberať tvorbou virtuálneho laboratória (ďalej len VL). Predtým ako si popíšeme detailné fungovanie technológií pre vytvorenie VL, si musíme vysvetliť čo považujeme za VL, pochopiť aké hodnoty nám môže priniesť, ale samozrejme aj tie ktoré nemôže. Vo všeobecnosti môžeme povedať že VL je počítačový program, kde študenti sú v interakcii s experimentom prostredníctvom počítača. Typický príklad je simulácia experimentu, kedy je študent v interakcii s naprogramovaným rozhraním. Ďalšia možnosť je diaľkovo ovládaný experiment, kde študent je v interakcii s reálnym zariadením cez počítačové rozhranie, napriek tomu že sa nenachádza pri ňom. Keď vylúčime druhú variantu tak si môžeme utvoriť definíciu nasledovne: *Virtuálnym laboratóriom voláme to, keď je študent študent v interakcii s experimentom, ktorá je od neho fyzicky vzdialená alebo nemá za sebou žiadnu fyzickú realitu.*[6]



Obrázok 1: Porovnanie medzi počítačovo riadeným experimentom vľavo a vzdialene riadeným experimentom vpravo.

Po vysvetlení čo je VL sa pozrieme na výhody, ktoré nám môže priniesť. Sú popísané v bodoch v *tabuľke č.1*. Človek často radí medzi "výhody" to, že môže nahradiť fyzické laboratória. Lenže to medzi výhody nepatrí. Nie je možné nahradiť skúsenosti z fyzickej práce zo zariadením VL aj keď je to lepšie ako žiadna skúsenosť. VL by nemalo byť vnímané tak, že poskytuje maximálnu možnú skúsenosť.

Typ laboratória	Výhody	Nevýhody
<b>Fyzické</b>	realistické dáta interakcia s reálnym zariadením lepšia spolupráca interakcia s lektorom	obmedzenia na čas a mieste potrebné plánovanie prístupu nákladnosť experimentu potrebný lektor
<b>Virtuálne</b>	dobré pre vysvetlenie konceptu bez obmedzenia na čas a miesto interaktívne médium nízke náklady	idealizované dáta nedostatok spolupráce bez interakcie s reálnym zariadením
<b>Vzdialené</b>	interakcia s reálnym zariadením kalibrácia realistické dáta bez obmedzenia na čas a miesto stredné náklady	"virtuálna" prítomnosť v laboratóriu

Tabuľka 1: Porovnanie fyzických, virtuálnych a vzdialených a laboratórií

## 2.1 Prehľad existujúcich virtuálnych laboratórií

V dobe písania tohto dokumentu existuje množstvo rôznych virtuálnych/vzdialených laboratórií, ktoré používajú zahraničné školy pre výučbu alebo výskum. V práci [8] je zoznam veľmi používaných laboratórií, ktoré sú prístupné cez internet. Porovnanie funkcionality a využitých technológií je možné vidieť v *tabuľke č.2*.

Názov	Klient	Server	Prevedenie
Weblab-DEUSTO	AJAX, Flash, Java applets, LabVIEW, Remote panel	Web services, Python, LabVIEW, Java, .NET, C, C++	Xilinx-VHDL, LabView
NCSLab	AJAX, Flash	PHP	Matlab, Simulink
ACT	HTML, Java Applets	PHP	Matlab, Simulink
LabShare Sahara	AJAX, Java applets	Web services, Java	Java
iLab	HTML, ActiveX, Java applets	Web services, .NET	LabVIEW
RECOLAB	HTML	PHP	Matlab, Simulink
SLD	AJAX, HTML	Web services, PHP	Matlab, Simulink

Tabuľka 2: Porovnanie virtuálnych laboratórií vytvorených mimo FEI STU.

Následne som preskúmal možnosti existujúcich riešení v *tabuľke č.3*, ktoré boli vytvorené na Fakulte elektrotechniky a informatiky STU.[3][1][7][27][25]

Rok vypracovania	Autor	Prevedenie	Spôsob komunikácie	Klient	Server
2011	Roman FARKAŠ	Matlab Simulink Reálna sústava	JMI, sockets	Java	Java
2012	Tibor BORKA	Matlab Simulink Reálna sústava	WCF	.NET, WPF	.NET
2014	Michal KUNDRÁT	Matlab Simulink	JMI, SOAP	HTML, JS	Tomcat, Java, JSF, EJB3, MySQL
2014	Tomáš ČERVENÝ	Matlab Simulink	JMI, HTTP	Mobilné HTML, JS	Jetty, Java
2015	Štefan VARGA	Matlab Simulink	COM, HTTP	HTML, JS	PHP, .NET

Tabuľka 3: Porovnanie virtuálnych laboratórií vytvorených na FEI STU.

### 2.1.1 Nevýhody existujúcich riešení

Pri tvorbe softwarového systému, či už všeobecne, alebo v našom prípade virtuálneho laboratória je vhodné preskúmať možnosti existujúcich riešení. Robí sa to kvôli tomu, aby sme sa pri návrhu vyvarovali rôznym chybám ktoré môžu nastať, alebo technológiám, ktoré už časom zastarali. V dnešnej dobe je vývoj nových technológií neskutočne rýchly. Takúto analýzu existujúcich riešení sme spravili v predchádzajúcej sekcii. Naša téma je zameraná na vytvorenie multiplatformového riešenia, kde nie je možné využiť WCF ani COM technológie ako v predchádzajúcich riešeniach. JMI je zase vhodné len pre riešenie, kde sa využíva Java. Pre server nie je možné využiť technológie LabVIEW, .NET (momentálne je vo vývoji multiplatformová verzia). Čo sa týka klientských riešení tak Flash,

ActiveX, Java applets už nie sú podporované v prehliadačoch, taktiež ich nie je vhodné použiť.

## 2.2 Komponenty virtuálneho laboratória

Počet existujúcich laboratórií je veľký, ale väčšinou nie je možné zaručiť kompatibilitu, pretože tu neexistuje žiadny štandard. Ale vždy je možné identifikovať základné komponenty, ktoré tieto VL využívajú. Niektoré z nich môžu byť dokonca využité viac krát.[19]

1. Samotný experiment
2. Zariadenie umožňujúce kontrolu experimentu a získavanie hodnôt z neho.
3. Laboratórny server, ktorý zabezpečí kontrolu, monitorovanie a spracovanie dát z experimentu.
4. Server, ktorý zabezpečí prepojenie medzi vzdialenými užívateľmi a laboratórneho servera, zvyčajne prostredníctvom internetu.
5. Webová kamera pripojená k serveru, ktorá môže byť použitá pre vzdialeného užívateľa ako vizuálna a zvuková spätná väzba o stave experimentu.
6. Nástroje umožňujúce viac užívateľské audio, video a chat komunikáciu.
7. Klientské zariadenia, ktoré sa pripoja vzdialene k experimentu. Väčšinou sa jedná o webovú aplikáciu alebo java aplikáciu.

Je ale dôležité si uvedomiť, že na vytvorenie laboratória nepotrebujeme všetky tieto komponenty, resp. môžeme využiť aj iné, ktoré sa nám dokonale hodia. Niekedy sa používa napr. aj databázový server ak chceme experimenty ukladať a spracovávať neskôr. Tak isto je potrebné uvedomiť si, aký typ VL chceme vytvoriť. Určite budú rozdiely pri návrhu jednougávateľského VL narozdiel od viacougávateľského dokonca s viacerými experimentami súčasne. Treba myslieť na to, ako vhodne vyriešiť škálovateľnosť, možné problémy s bezpečnosťou, viacougávateľský prístup, ostatné problémy prístupnosti a podobne.

## 3 Použité technológie

V predchádzajúcich kapitolách sme popísali ciele, čo chceme vlastne dosiahnuť. Ďalej sme analyzovali možnosti virtuálnych laboratórií a porovnali už s existujúcimi riešeniami. V tejto sekcii budú v krátkosti rozpísané použité technológie. Je samozrejmé, že nie je možné ku každej popísať všetky jej možnosti, ale budeme sa venovať hlavne tým, ktoré plánujeme využiť aj v implementácií.

### 3.1 Matlab R2015b

Milióny inžinierov a vedcov na celom svete používajú MATLAB na analýzu a návrh systémov a produktov ktore menia náš svet. MATLAB sa používa v automobilových systémoch, vesmírnych staniciach, smart siete, mobilých sieťach LTE alebo aj v škole na štúdium. Ďalej sa používa pre strojové učenie, spracovanie signálu, spracovanie obrazu, počítačové videnie, komunikácie, finančníctvo, riadenie, robotika a mnoho ďalších využití. Celá platforma MATLAB je optimalizovaná pre riešenie inžinierskych a vedeckých problémov. Jazyk MATLABu je založený na práci s maticami. Je považovaný za najprirodzenejší spôsob ako počítať matematické úlohy. Pomocou integrovanej grafickej knižnici je možné vizualizovať a získať výsledky z dát. MATLAB integruje aj množstvo toolboxov, ktoré pomáhajú priamo začať s algoritmami, ktoré potrebujeme pre našu doménu.[9]

#### 3.1.1 Simulink

Matlab obsahuje viacero integrovaných nástrojov a jeden z nich je aj Simulink. Je to grafické rozhranie, v ktorom je možné modelovať, simulovať a potom aj analyzovať dynamické systémy. Jeho hlavné rozhranie je grafické plátno, kde môžeme spájať jednotlivé bloky do diagramu. Simulink vie úzko spolupracovať s matlabom, dokonca môže byť odtiaľ aj skriptovaný, resp. počiatočná inicializácia hodnôt. Matlab a simulink sú vlastne dve prostredia integrované do jedného software. Čiže je možné simulovať a analyzovať náš model v každom kroku simulácie v oboch prostrediach. Simulink väčšinou spúšťame priamo z Matlabu.

#### 3.1.2 Komunikácia medzi Matlabom a Node.js

Simulácia dynamickej sústavy, ktorá sa spustí v Simulinku môže odosielať výsledné dáta do Matlab workspace. Odtiaľ ich budeme chcieť posilať do Node.js na ďalšie spracovanie. V matlabe existuje viacero možností získania dát z workspace.

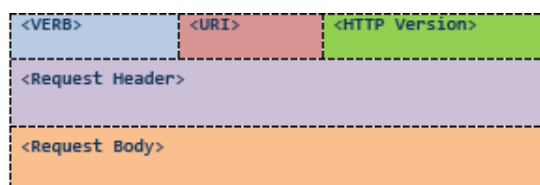
**COM** (Component Object Model) je prvá z technologických možností. COM bolo vytvorené spoločnosťou Microsoft, čiže toto riešenie je obmedzené len na Windows platformu. Použí-

vajú sa na prepojenie rôznych aplikácií podporujúcich túto technológiu. Tieto objekty môžu byť vytvorené pomocou rôznych programovacích jazykov ako napr. C++ alebo Java.[13]

Kým idea COM je celkom jasná, terminológia až toľko nie. *COM object* je softwarový komponent, ktorá zodpovedá Component Object Model. COM vnucuje zapuzdrenie objektu a tým predchádza pred priamym prístupom do dát a implementácie. COM objekt poskytne rozhranie, ktoré obsahuje premenné, metódy a udalosti. *COM client* je program, ktorý používa COM objekty. COM objekty, ktorý poskytujú funkcionality pre použitie sú volané COM server. COM server môže byť in-process alebo out-of-process. Príklad out-of-process servera je napríklad Microsoft Excel. *Microsoft ActiveX control* je typ in-process COM servera, ktorý vyžaduje kontajner. ActiveX zvyčajne poskytuje užívateľské rozhranie. Príkladom môže byť Microsoft Calendar control. Control container je aplikácia schopná poskytovať ActiveX prvky. Matlab figure okno alebo Simulink model su tiež príklady control kontajnerov. Matlab môže byť použitý aj ako COM klient aj ako COM automation server.[10]

**Websockets** popisat matlab webscokety – narocnejsie implementovat, pretoze nie je nativna implementacia

**RESTful web service** Medzi najmodernejšie možnosti komunikácie Matlabu s vonkajškom patria jednoznačne RESTful služby. REST (representational state transfer) je softwarový architektonický štýl, pomocou ktorého vieme posielať a získavať dáta zo serveru. REST komunikuje pomocou HTTP/HTTPS protokolu a zväčša sa používa na CRUD aplikácie, čiže tam kde chceme robiť CREATE, READ, UPDATE, DELETE operácie. Dáta je možné vymieňať medzi klientom a serverom cez *JSON* alebo *XML* správ. Pre jednoduchšie projekty sa používajú skôr JSON, hlavne ak sa majú spracovávať JavaScriptom. Výhodou RESTful v tomto riešení je to, že je priamo implementované v Matlabe a nie je potrebné inštalovať ďalšie knižnice a toolboxy.



Obrázok 2: HTTP request.

Na obrázku č.2 vidíme ako vyzerá HTTP request. V položke *<VERB>* sa nachádza jedna z HTTP metód GET, PUT, POST, DELETE, OPTIONS, ... V *<URI>* je zase linka na zdroj, nad ktorým sa bude vykonávať daná operácia. *<HTTP version>* je verzia HTTP, vo všeobecnosti to bude "HTTP v1.1" ale môže byť aj "HTTP v2.0" v novších systémoch.

*<Request Header>* obsahuje metadáta ako kolekciu párov "key" : "value" hlavičky. Tieto nastavenia obsahujú informáciu o správe a jej odosielateľovi ako typ klienta, aký formát podporuje klient, typ formátu správy tela, nastavenia cache pre odpoveď a mnohé ďalšie informácie.

*<Request Body>* je aktuálny obsah správy. V RESTful službách je toto miesto, kde sa nachádza obsah správy, ktorý sa vymieňa medzi klientom a serverom. V tejto časti nie sú žiadne tagy ani značky pre učenie začiatku alebo konca správy.[24]

---

**Algoritmus 1** Príklad POST requestu.

---

```
POST http://MyService/Person/
Host: MyService
Content-Type: text/xml; charset=utf-8
Content-Length: 123
<?xml version="1.0" encoding="utf-8"?>
<Person>
  <ID>1</ID>
  <Name>M Vaqqas</Name>
  <Email>m.vaqqas@gmail.com</Email>
  <Country>India</Country>
</Person>
```

---

---

## Algoritmus 2 Príklad GET requestu.

---

```
HTTP/1.1 200 OK
Date: Sat, 23 Aug 2014 18:31:04 GMT
Server: Apache/2
Last-Modified: Wed, 01 Sep 2004 13:24:52 GMT
Accept-Ranges: bytes
Content-Length: 32859
Cache-Control: max-age=21600, must-revalidate
Expires: Sun, 24 Aug 2014 00:31:04 GMT
Content-Type: text/html; charset=iso-8859-1
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns='http://www.w3.org/1999/xhtml'>
<head><title>Hypertext Transfer Protocol -- HTTP/1.1</title></head>
<body>
...
```

---

V ukážke POST requestu č.1 zasielame serveru XML s údajmi o novej osobe. V ukážke GET requestu č.2 je vidieť, že server vráti obsah HTML stránky na ktorú bola požiadavka vytvorená.

Teraz keď sme si vysvetlili v stručnosti ako fungujú RESTful služby, tak sa dostávame k tomu ako ich je možné využiť z Matlabu. Matlab poskytuje viacero funkcií na prácu s REST ako *weboptions*, *webread*, *webwrite*.

Objekt *weboptions* slúži na špecifikáciu parametrov pre RESTful službu. V Matlabe sa volá príkazom *options = weboptions* alebo *options = weboptions(Name, Value)* pričom *Name* je názov parametra, ktorý chceme nastaviť a *Value* jeho hodnota.

Je možné nastaviť tieto parametre: *CharacterEncoding*, *UserAgent*, *Timeout*, *Username*, *Password*, *KeyName*, *KeyValue*, *ContentType*, *ContentReader*, *MediaType*, *RequestMethod*, *ArrayFormat*. Ak chceme zobrazíť objekt *weboptions*, tak namiesto hesla tam budú hviezdičky. Každopádne, objekt ukladá heslo ako čistý text. V prípade keď zavoláme túto vlastnosť v Matlabe cez *options.Password* tak heslo bude viditeľné.??

Objekt *webread* číta obsah z REST služby, na ktorú sme mu poskytli URL cestu a vráti obsah ako štruktúru do požadovanej premennej. Existujú tri najpoužívanejšie možnosti použitia. *data = webread(url)* kde parameter *url* je refazec, v ktorom sa nachádza odkaz na REST službu. *data = webread(url, QueryName1, QueryValue1, ..., QueryNameN, QueryValueN)* s parametrami *url* a *QueryName*, *QueryValue* ktoré pridá ako parametre do *url* volania. *data = webread(\_\_\_\_, options)* kde môžeme špecifikovať aj *weboptions* objekt.??



A posledný objekt *webwrite*. Ako aj v predchádzajúcom prípade obsahuje rovnaké metódy s parametrami. *response = webwrite(url, PostName1, PostValue1, ..., PostNameN, PostValueN)* zapíše obsah na špecifikovanú url a vráti response. *response = webwrite(url, data)* zapíše obsah na špecifikovanú url a vráti response. Vstupný parameter data špecifikuje obsah, ktorý je uložený ako pole. *response = webwrite(\_\_\_\_, options)* zapíše obsah na špecifikovanú url a vráti response.

## 3.2 Node.js

Na stránke platformy (<http://www.nodejs.org>) je definovaný Node ako "platforma založená na JavaScript runtime, ktorý je v Chrome pre jednoduchú tvorbu rýchlych, škálovateľných sieťových aplikácií. Node.js používa udalosťami riadený, neblokujúci I/O model, ktorý ho robí nenáročný a efektívny, perfektný pre real-time aplikácie." V súčasnosti patrí medzi najpopulárnejšie JavaScript technológie.

Vďaka jeho súčasnej stabilite ho používa v produkcii mnoho svetových firiem, napríklad eBay, GoDaddy, Microsoft, PayPal, Uber, Yahoo...

### 3.2.1 História

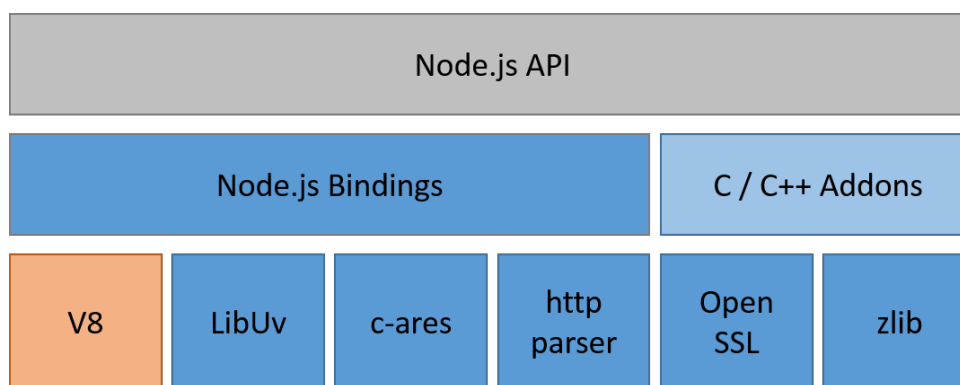
Vytvoril ho Ryan Dahl v roku 2009 a bol dostupný iba pre Linux. Vývoj a údržba bola vedená jeho zakladateľom a neskôr sponzorovaná firmou Joyent. Node.js sa skladá z JavaScriptového engine V8 od Google, event loop a nízko úrovňové I/O API. V roku 2011 bol vytvorený správca balíkov pre Node.js volaný NPM (Node Package Manager). Umožňuje programátorom publikovať, zdieľať zdrojový kód modulov a bol navrhnutý tak, aby zjednodušoval inštaláciu, aktualizáciu alebo odinštalovanie modulov. Neskôr v júni 2011, Microsoft a Joyent spolupracovali na implementácii natívnej verzii Node.js pre Windows. V roku 2014 vznikli nezhody pri vývoji, tak Fedor Indutny spravil fork Node.js a vytvoril ioJS. Na rozdiel od Node.js autorov, chcel udržiavať ioJS aktuálny súčasne s poslednou verziou V8 JavaScript enginu. Po dohode bola vytvorená Node.js Foundation, ktorá zastrelila vývoj, a spojila Node.js v0.12 a ioJS v3.3 do Node.js 4.0, aby spojila znovu komunitu. Táto verzia priniesla V8 ES6 novinky do Node.js a súčasne sa vytvorila aj verzia LTS vhodná pre produkčné nasadenie, ktorá má dlhší vývojový cyklus a príjma len opravy chýb.[15]

### 3.2.2 Architektúra

V prvom rade by som spomenul kľúčové vlastnosti Node.js. **Asynchronné a udalosťami riadené API**, čo znamená, že neblokuje vykonávanie nasledujúcich volaní. V podstate to znamená, že server založený na Node.js nikdy nečaká, kým API vráti data. Server sa presunie na ďalšie volanie a vďaka pomoci notificačného mechanizmu udalostí získa

server odpoveď z prechádzajúceho volania. Je **jednovláknový a vysoko škálovateľný** vďaka udalostnej slučke. Udalostný mechanizmus pomôže serveru vrátiť odpoveď tak aby nebol blokujúci a robí server vysoko škálovateľný oproti tradičným serverom, ktoré vytvárajú limitovaný počet vlákien na spracovávanie požiadaviek. Node.js spustí jednovláknový program, ktorý môže poskytnúť službu omnoho väčšiemu počtu požiadaviek ako tradičný server Apache HTTP. Aplikácie sú **bez vyrovnávajúcej pamäte**, čiže jednoducho posielajú údaje na výstup v malých blokoch.

Ako sme už spomenuli, tak platforma Node.js sa skladá z viacerých častí. Bolo by možné ju rozdeliť ešte na menšie časti, ale toto je základný pohľad na obrázku č.3.



Obrázok 3: Architektúra Node.js platformy.

Na vrchole obrázku máme základné Node.js API. Je napísané v JavaScripte a je priamo dostupné programátorom, pre využitie v ich aplikáciach. Pod základným API je knižnica, ktorá viaže C/C++ s JavaScriptom. Node.js tiež poskytuje doplnky (addons), čo sú dynamicky linkované zdieľané objekty. Tie sa viažu na C/C++ knižnice. To znamená, že môžeme využiť skoro hocikakú C/C++ knižnicu a vytvoriť z nej doplnok, ktorý použijeme v Node.js.

Pod týmto všetkým máme už len natívne knižnice vytvorené v C/C++:

**V8** je open source JavaScript engine, ktorý bol vytvorený pre Google Chrome. Je napísaný v C++ a je možné ho spustiť samostatne alebo v hocikakej C++ aplikácii. V podstate kompiluje JavaScript kód do natívneho strojového kódu, namiesto toho, aby bol interpretovaný.

**Libuv** je multiplatformová podporná knižnica so zameraním na asynchrónne I/O operácie. Zo začiatku Node.js začal používať *libuv* ako abstrakčnú vrstvu pre *libev* a *libio*, ale neskôr sa libuv stala robustnejšou a nahradila túto funkcionálnosť, aby sa mohla

stať multiplatformovou. Keď V8 spravuje vykonávanie JavaScriptu, tak libuv spravuje udalostnú slučku (event loop) a asynchrónne I/O operácie. V tomto zozname vidíme všetky možnosti *libuv*:

- plnohodnotnú udalostnú slučku, ktorú tvorí epoll, kqueue, IOCP a udalostné porty,
- asynchrónne TCP a UDP sockety,
- asynchrónne DNS,
- asynchrónne operácie nad súborom a súborovým systémom,
- udalosti nad súborovým systémom,
- preklad ANSI kódov kontrolovaných cez TTY,
- IPC so zdieľaním socketu s využitím Unix socketov alebo pomenované kanály (Windows),
- detské procesy,
- vlákna a synchronizáciu,
- riadenie signálov,
- hodiny s presným časovaním (high resolution clock).

**c-ares** je C knižnica pre asynchrónne DNS žiadosti vrátane prekladanie názvov. Je určená pre aplikácie, ktoré potrebujú vykonávať dotazy na DNS bez blokovania, alebo je potrebné vykonať niekoľko dotazov paralelne.

**http\_parser** je parser pre požiadavky a odpovede HTTP napísaný v C. Nerobí žiadne systémové volania ani alokácie, neukladá dáta do vyrovnávacej pamäte, môže byť zrušený okamžite. Jeho hlavné vlastnosti sú:

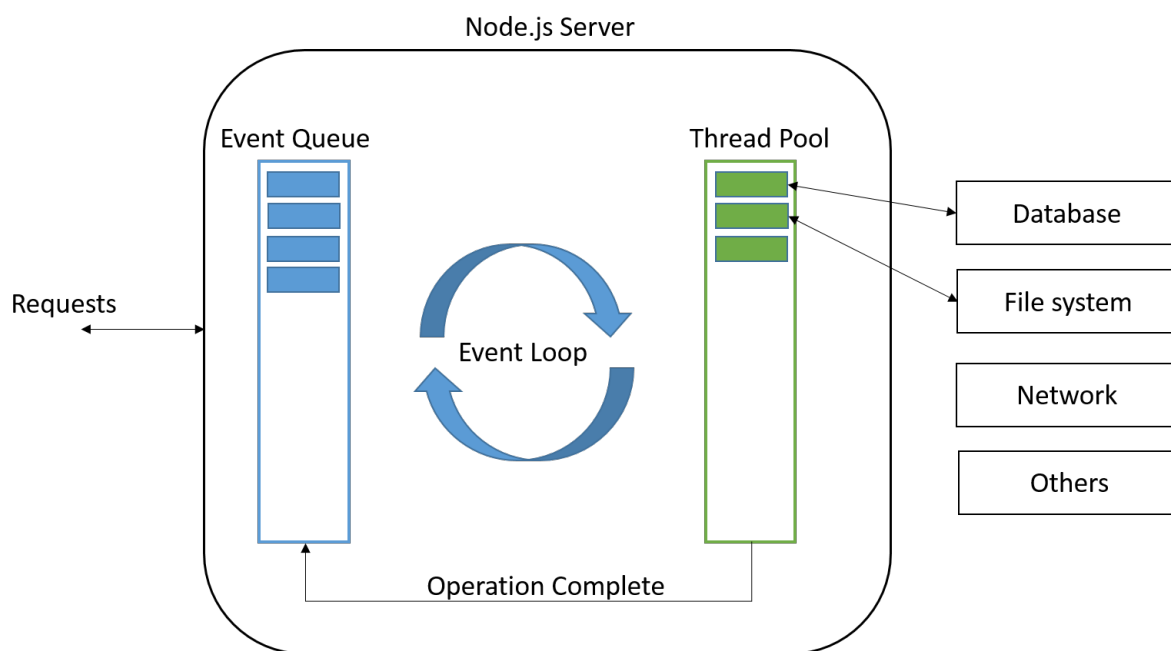
- žiadne závislosti,
- udržiava trvalý stream
- dekodovanie blokového kódovania,
- chráni buffer proti útokom.

**OpenSSL** je open source implementácia SSL v2/v3 a TLS v1 protokolov ako aj kryptografickú knižnicu pre všeobecné účely. Je založená na SSLeay knižnici. Tá poskytuje všetky potrebné kryptografické metódy ako je hash, hmac, cipher, decipher, sign a verify metódy.

**Zlib** je všeobecná knižnica na kompresiu dát napísaná v C.[16]

### 3.2.3 Event loop

Udalostná slučka dáva Node.js možnosť zvládnuť veľké množstvo súčasných požiadaviek aj keď je spustený v "jednom vlákne". V každej udalostne riadenej aplikácii je vo všeobecnosti hlavná slučka ktorá počúva a čaká na udalosti a keď udalosť je zaregistrovaná tak sa zavolá callback funkcia. Na obrázku č.4 je zjednodušený pohľad na to, ako to funguje vrámci Node.js.



Obrázok 4: Udalostná slučka.

Udalostná slučka jednoducho prechádza cez frontu čo je vlastne zoznam udalostí a callbackov vykonaných operácií. Všetky I/O operácie sú vykonané asynchrónne vláknami vo "vláknovom stacku" (thread pool). Tu zohráva dôležitú rolu libuv. Ak nejaká položka vyžaduje I/O operáciu, tak udalostná slučka jednoducho prenechá operáciu do vláknového stacku. Udalostná slučka pokračuje vo vykonávaní položiek v udalostnej fronte. Keď je I/O operácia hotová, tak callback je zaradený na spracovanie. Udalostná slučka vykoná callback a poskytne požadované výstupy. A takto sa celý proces opakuje.[17]

### 3.2.4 Možnosti a využitie

Na základe popísaných základných častí Node.js v predchádzajúcich sekciách si ukážeme možnosti využitia, resp. výhody a nevýhody.[16]

#### Výhody

- asynchrónne I/O - vhodné pre webové a sieťové aplikácie,
- rôzne možnosti škálovania,
- programovací jazyk je JavaScript, čiže rovnaký jazyk aj pre backend aj frontend,
- jednoducho je možné s ním začať na rozdiel od Javy alebo .NET, stačí len zmeniť myslenie na asynchrónne,
- veľmi aktívna komunita, ktorá zdieľa množstvo kódu na verejných repozitároch ako github,
- rýchlo rastúca NPM komunita s množstvom modulov pripravených na použitie.

#### Nevýhody

- veľmi neefektívne pre úkony náročné na CPU, ako generovanie reportov, analýzy, výpočty...
- použitím udalostami riadenej metodológie bez pochopenia prístupu môže viesť k nevhodne napísaným kódom (napr. "callback hell"),
- neexistuje veľa štandardných knižníc ako pri Java, .NET platforme ako sú napr. XML parsery, alebo zložitejšie dátové štruktúry.

Teda Node.js nie je vhodný na všetko, vždy záleží od konkrétnej potreby. Čiže ak potrebujeme streaming alebo rýchly upload súborov, real-time údaje, single page aplikácie, websockety tak je na takéto úlohy vhodný. Vo všeobecnosti všade kde sa používa I/O operácie, tak vie zvládnuť veľké množstvo súčasných spojení.[21]

## 3.3 Node Package Manager

Node.js po nainštalovaní obsahuje aj *NPM* (Node Package Manager). NPM je program, ktorý sa spúšťa z príkazového riadku, pomocou ktorého vieme sťahovať moduly z centrálného repozitára (<https://www.npmjs.org>). Rovnako aj môžeme vytvoriť vlastný modul a uložiť ho do repozitára.

Každý modul by mal mať vlastný adresár, ktorý tiež obsahuje súbor s metadátami volaný

*package.json*. V tomto súbore musí byť nastavené aspoň tieto dve vlastnosti: *name*, *version*.<sup>[20]</sup>

```
{
  "name": "my-awesome-nodejs-module",
  "version": "0.0.1"
}
```

### 3.3.1 Použitie modulov

Vo všeobecnosti sú tri možnosti ako použiť moduly, ktoré už existujú. Všetky tri zahŕňajú manažéra na spravovanie balíkov:<sup>[20]</sup>

- môžeme nainštalovať špecifický modul manuálne tak, že sa presunieme do požadovaného priečinka a zavoláme v termináli *npm install nazov-modulu*. Správca balíkov automaticky nainštaluje poslednú verziu modulu a vloží ju do priečinka *node\_modules*. Ak ho chceme použiť, nepotrebujeme volať celú cestu, ale len *require(nazov-modulu)*.
- inštalácia modulu globálne vďaka *-g* značke v príkaze. *npm install nazov-modulu -g*. Je to vhodnejšie používať skor na vývojové nástroje ako na bežné moduly.
- posledná možnosť je zavolať *npm install* nad priečinkom kde sa nachádza už predtým spomínaný *package.json* a obsahuje v sebe vlastnosť *dependencies*

```
{
  "name": "another-module",
  "version": "0.0.1",
  "dependencies": {
    "my-awesome-nodejs-module": "0.0.1"
  }
}
```

### 3.3.2 Vstavane moduly

Node.js je považovaný za technológiu, pomocou ktorej môžeme tvoriť backend aplikácie. Často potrebujeme robiť rôzne úlohy. V Node.js existujú veľmi nápomocné moduly, ktoré môžeme využiť.<sup>[20]</sup>

### 3.3.3 Vytvorenie web servera pomocou HTTP modulu

Modul *http* patrí medzi veľmi používané, pretože vďaka nemu spustíme web server na konkrétnom porte.

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(9000, '127.0.0.1');
console.log('Server running at http://127.0.0.1:9000/');
```

Máme dostupnú metódu *createServer*, ktorá vracia nový web server objekt. Vo väčšine prípadov zavoláme aj metódu *listen*. Ak je potrebné tak pomocou *close* zastavíme príchod nových požiadaviek. Callback funkcia, ktorú tam využívame, vždy má *request* (req) a *response* (res) objekty. Prvý objekt môžeme použiť na získanie informácií o prichádzajúcich požiadavkách ako napr. *GET*, *POST* parametre.[20]

### 3.4 Express.js

Express.js sa zaraďuje medzi minimalistické a flexibilné Node.js webové frameworky. Poskytuje robustné možnosti pre vývoj webových a mobilných aplikácií. Uľahčuje vývoj webových aplikácií založených na Node.js.[23]

Medzi jeho hlavné vymoženosti patria:

- umožňuje nastaviť middleware pre odpovede na HTTP požiadavky,
- definuje smerovaciu tabuľku, ktorá je použitá na vykonanie rôznych požiadaviek založených na HTTP,
- umožňuje dynamicky vykreslovať HTML stránky, tak aby bolo možné vkladať argumenty do šablóny.

Často pri inštalácii Expressu inštalujeme aj *body-parser*, ktorý slúži ako middleware pre narábanie s JSON, Raw, Text a URL kódovanými údajmi.

#### 3.4.1 Web server

Následujúca ukážka na obrázku č. 6 je základná kostra Express aplikácie, ktorá spustí server a počúva na porte 8081. Po príchode na stránku bude vždy odpovedať textom **Hello word!**.[23]

```

var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.send('Hello World');
})

var server = app.listen(8081, function () {

  var host = server.address().address
  var port = server.address().port

  console.log("Example app listening at http://%s:%s", host, port)

})

```

Obrázok 5: Spustenie web servera na porte 8081.

Na spustenie tohto kódu ho musíme vložiť do súboru s koncovkou .js a následne spustiť `node app.js` a potom dostaneme výstup v termináli *Example app listening at http://0.0.0.0:8081*.

### 3.4.2 Objekty Request a Response

Webová aplikácia používa objekty request a response z Node.js, ktoré sú dostupné ako callback funkcia.[23]

```

app.get('/', function (req, res) {
  // --
})

```

Obrázok 6: Objekt req a res ako parametre callback funkcie.

### 3.4.3 Routing

Routing, teda smerovanie slúži na určenie ako bude aplikácia reagovať na požiadavku klienta, čo zahŕňa URI a špecifické metódy (GET, POST, PUT, DELETE, ...) HTTP požiadavky.[23] Každý **route** môže mať viacero callback funkcií, ktoré sa vykonajú ak je zavolaná daná route.

Definícia route má nasledujúcu štruktúru `app.METHOD(PATH, CALLBACK)`, kde:

- `app` je inštancia `express`,
- `METHOD` je jedna z dostupných HTTP metód,
- `PATH` je cesta na serveri,
- `CALLBACK` je funkcia, ktorá sa vykoná v prípade, že sa zhoduje cesta.[?]



### 3.4.4 Middleware

Funkcie označované middleware, sú také, ktoré majú prístup do request (**req**), response (**res**) objektu a následujúcej middleware funkcie. Následujúca funkcia sa bežne označuje názvom **next**.<sup>[2]</sup>

Úlohy middleware funkcií:

- vykonávať akýkoľvek kód,
- robiť zmeny na **request** a **response** objektoch,
- ukončiť request-response cyklus,
- zavolať ďalšiu middleware funkciu v poradí.

Na obrázku č. 21 nižšie sú popísane vlastnosti tejto funkcie.<sup>[2]</sup>



Obrázok 7: Vlastnosti middleware funkcie.

V prípade, že middleware funkcia neukončí volanie, musí posunúť obsluhu ďalšej middleware funkcií zavolaním **next()**. Inak sa request zasekne a klient bude stále čakať na odpoveď a po určitom omeškaní sa ukončí.<sup>[2]</sup>

## 3.5 MongoDB

MongoDB je multi platformová dokumentovo orientovaná databáza. Je klasifikovaná ako NoSQL databáza, čiže nepoužíva klasickú štruktúru ako v relačnej databáze na báze tabuliek, ale objekty vo forme JSON dokumentov s dynamickými schémami (v MongoDB sa volá tento formát BSON). Vďaka ukladaniu JSON dokumentov je možné ukladať dáta z rôznych aplikácií rýchlejšie a jednoduchšie.<sup>[14]</sup>

```

{
  name: "sue",
  age: 26,
  status: "A",
  groups: [ "news", "sports" ]
}

```

← field: value  
← field: value  
← field: value  
← field: value

Obrázok 8: JSON dokument v MondoDB.

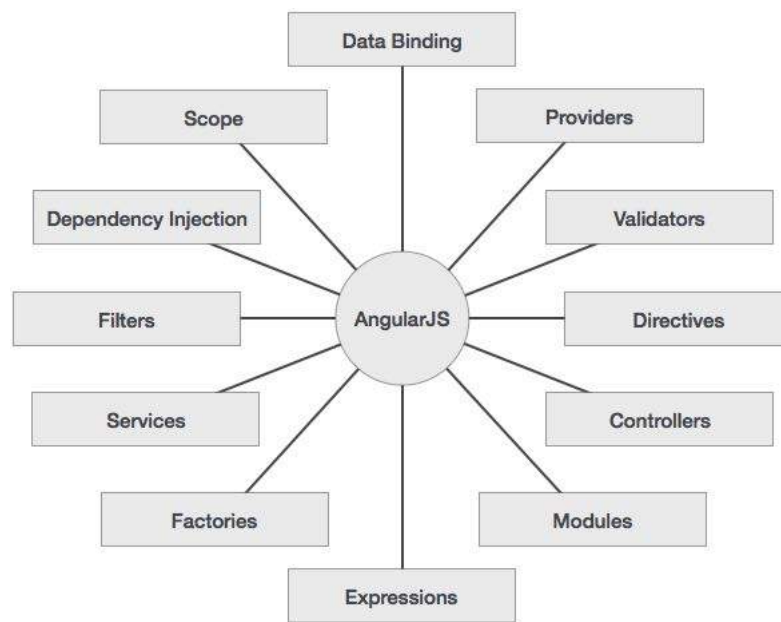
Na obrázku č.8 je príklad dokumentu, ktorý sa ukladá do MongoDB, v jednoduchých prípadoch nie je vidieť žiadny rozdiel oproti klasickému JSON.

## 3.6 Angular.js

Angular.js, bežne označovaný aj ako Angular je open-source framework pre web vytvorený Slovákom Miškom Hevery do verzie 1.0. Neskôr vývoj a údržbu frameworku vzal pod seba Google. Bol vytvorený za účelom tvorby SPA (single page applications), čiže webových aplikácií, v ktorých je možné aktualizovať svoj obsah bez toho, aby si užívateľ toho vo väčšej miere všimol. Cieľom je zjednodušiť vývoj, ale aj testovanie front-end web aplikácií na báze MVC (model-view-controller) alebo MVVM (model-view-view-model) architektúre. Angular.js framework funguje tak, že pri prvom načítaní HTML stránky vloží k elementom vlastné atribúty. Angular interpretuje tieto atribúty ako direktívy a viaže ich vstupno-výstupné časti ako model, ktorý je reprezentovaný štandardnými premennými JavaScriptu. Hodnoty týchto premenných môžu byť manuálne nastavené v kóde, alebo získané z statických (väčšinou uložené na súborovom systéme), dynamických (získané z RESTful služby) JSON súborov. Je front-end súčasť MEAN vývojarského stacku, čo je vlastne MongoDB databáza, ExpressJS web server framework, Angular a NodeJS ako aplikačný server, resp. prostredie.[4][5]

### 3.6.1 Koncept

Angular teda ako framework sa skladá z viacerých častí, ktoré robia svoju časť práce. Na obrázku je možné vidieť jeho základné komponenty, z ktorých sa skladá. Nižšie si popíšeme hlavne tie, ktoré využívame v práci.[22]



Obrázok 9: Koncept frameworku Angular.js.

**Directives** Jednoducho povedané, directívy sú značky na DOM elemente (napr. atribút, element, komentár alebo CSS trieda), ktorá povie Angular HTML compileru (*\$compile*), aby priradil špecifické správanie na daný DOM element (napr. pomocou event listenerov), alebo dokonca transformoval DOM element ako svoje dieťa. V šablónach sa spájajú informácie z modelu a controlleru pre renderovanie dynamického obsahu, ktorý potom vidí užívateľ v prehliadači. Angular je dodávaný so vstavanými directívami, od ktorých sa očakáva, že budú využívané často. Napr. *ngBind*, *ngModel*, *ngClass*. [5]

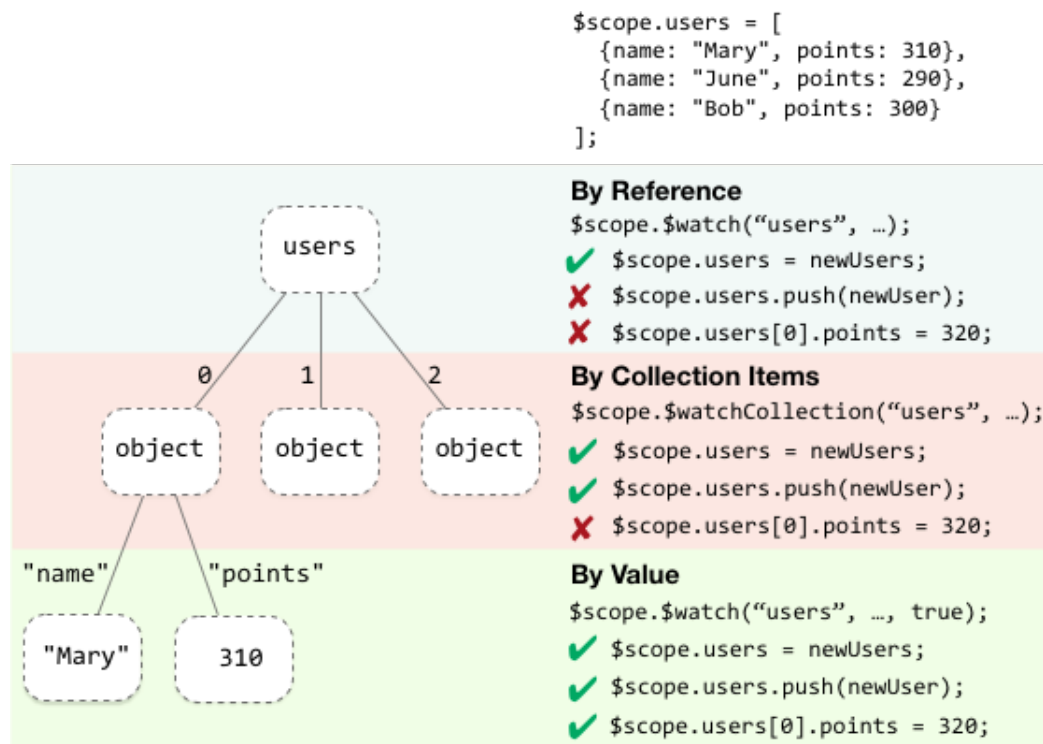
### 3.6.2 Scope

Scope je objekt, ktorý odkazuje na model aplikácie. Je to kontext, na ktorom sa vykonávajú výrazy. Scopes sú usporiadané do hierarchickej štruktúry, ktorá napodobňuje DOM štruktúru aplikácie. V scope je možné sledovať model pomocou *textit\$watch* a vykonávať udalosti (*\$apply*) cez celý systém do view. Je považovaný ako lepidlo medzi aplikačným controllerom a view. Aj controller aj directívy majú prístup k scope, ale nie navzájom k sebe. Vďaka tomu je controller izolovaný od directívy a tak isto aj od DOM. Každá aplikácia má práve jeden *\$rootScope*, ale môže mať viacero detských *\$scope*. Aplikácia môže mať viacero scopes, pretože niektoré directívy môžu vytvoriť nový detský scope, ak to potrebujeme. Keď je nový scope vytvorený, tak je priradený ako dieťa k rodičovskému scope. Toto vytvára stromovú štruktúru paralelnú k DOM, kde sú priradené. [5]

Dirty checking v scope pre zmeny vlastností objektu je často používaná operácia a

preto by mala byť efektívna. V závislosti od potreby môže byť dirty checking využité týmito troma stratégiami: referenciou na objekt, obsah poľa alebo na hodnoty objektu. Líšia sa v spôsobe zaregistrovania zmeny aj výkonostnými rozdielmi.

- Sledovanie podľa referencie `$scope.$watch(watchExpression, listener)`: detekuje zmenu, keď sa do sledovanej hodnoty nastaví nová. Ak sledovaný objekt, alebo pole, zmeny vnútri nie su detekované. Toto je najúčinnnejšia stratégia.
- Sledovanie na celej kolekcii (poli) `$scope.$watchCollection(watchExpression, listener)`: detekuje zmeny, ku ktorým dochádza vo vnútri poľa alebo objektu. Napr. keď sú položky pridané, odstránené alebo sa v nich zmenilo poriadie. Detekcia je plytká, čiže nesleduje vnorené kolekcie. Sledovanie celého obsahu kolekcie je výkonovo náročnejšie ako sledovanie podľa referencie, pretože treba uchovávať kópie obsahu. Avšak, táto stratégia sa snaží minimalizovať množstvo potrebných kopírovaní.
- Sledovanie všetkých položiek objektu `$scope.$watch(watchExpression, listener, true)`: detekuje akúkoľvek zmenu v ľubovolnej vnorenej štruktúry. Táto stratégia má najväčšie možnosti detekovania zmien, ale za to aj výkonovo a pamäťovo náročnejšia. Je potrebné uchovávať kópiu celej vnorenej štruktúry a pri každej zmene, sa musí nakopírovať do pamäte.



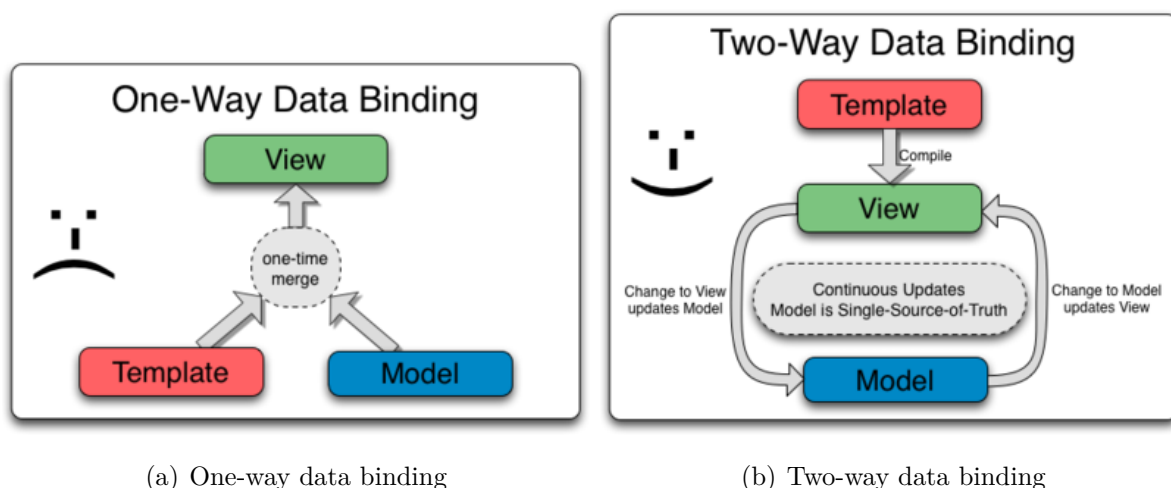
Obrázok 10: Možnosti sledovania modelu v Angular.js.

### 3.6.3 Expressions

Angular výrazy sú kúsky kódu v JavaScripte, ktoré sú umiestnené medzi dvojitémi kučeravými zátvorkami. Príklad: `<span title="{{ attrBinding }}">{{ textBinding }}</span>`. Vyhodnotenie môže rovnako prebehnúť aj vo funkcii na kliknutie `ng-click="functionExpression"`. Pre príklad pridávam zopár ďalších platných výrazov, ktoré sa často môžu používať: `{{ 1 + 2 }}`, `{{ a + b }}`, `{{ user.name }}` a `{{ items[index] }}`.<sup>[5]</sup>

### 3.6.4 Data Binding

Data binding v Angulare funguje ako automatická synchronizácia dát medzi modelom a view. Ak sa zmení model, tak zmena sa automaticky prejaví aj do view, alebo aj naopak. Na obrázkoch môžeme vidieť pre porovnanie aký je rozdiel medzi One-way a Two-way data binding.



(a) One-way data binding

(b) Two-way data binding

Obrázok 11: Data binding v Angular.js.

Mnoho šablónovacích enginov nastavujú dáta len v jednom smere: spoja spolu šablónu a komponenty modelu do view. Keď sa dokončí spojenie, zmeny v modeli alebo jeho príslušných sekcií vo view sa neprejavujú automaticky v zobrazení. Horšie je, že zmeny ktoré užívateľ v zobrazení sa neprejaví do modelu. To znamená, že vývojár musí napísať kód, ktorý sústavne synchronizuje view s modelom a model s view.

Šablóny v Angulari fungujú inak. Šablóna (čo je neskompilovaný HTML kód spolu s ďalšími značkami a directívami) je najprv kompilovaná v browseri. Tento krok produkuje živý view. Akékoľvek zmeny vo view sa okamžite prejaví v modeli, a akékoľvek zmeny v modeli sú poslané do view. Vďaka tomuto spôsobu môžeme o view hovoriť ako o okamžitej projekcii modelu.

Pretože view iba je projekcia modelu, tak controller je kompletne separovaný od view

a nevedia o sebe. Vďaka tomu je testovanie jednoduchšie, pretože je možné testovať controller izolovane od view.[5]

### 3.6.5 Controller

V Angulari je controller definovaný funkciou konštruktoru JavaScriptu, ktorý je používaný pre rozšírenie Angular Scope. Keď je controller pripojený k DOM cez *ng-controller* directívu, tak bude vytvorená inštancia objektu konštruktoru. Bude vytvorený detský Scope a je dostupný ako parameter to konštruktorovej funkcie controlleru ako *\$scope*. Vo všeobecnosti by controller nemal toho robiť veľa. Mal by obsahovať iba business logiku potrebnú pre jednotlivý view. Najlepší spôsob ako zachovať controller čo najmenší, je zabalenie niektorej úlohy do service, ktorá tam nepatrí. Potom tieto úlohy zo services voláme v controlleri ako závislosť.[5]

Controller je možné používať na:

- nastavenie počiatočného stavu v *\$scope* objekte.
- pridanie vlastností a správania do *\$scope* objektu.

Nepoužívať controller na:

- manipuláciu s DOM, pretože controller by mal obsahovať iba business logiku. Vložením hociakej prezentačnej logiky do controlleru vo veľkej miere ovplyvňuje jeho testovanie. Na manipuláciu slúžia directívy v sekcii 3.6.1.
- formátovanie vstupu - použite *angular from controls*
- filtrovanie výstupu - použite *angular filter*
- zdieľanie kódu alebo stavu naprieč controllermi - použite *angular service*
- spracovanie životného cyklu ostatných komponentov (napr vytváranie inštancií service).

### 3.6.6 Module

Module môžeme uvažovať ako kontajner pre rozličné časti našej aplikácie - controller, service, filter, directivy... Každý modul môže obsahovať zoznam iných modulov ako svoju závislosť. Keď závisíme na nejakom module, tak tento modul musí byť načítaný ešte pred našim modulom. Toto poriadie môžeme manuálne nastaviť v konfigurácii modulov. Každý modul môže byť načítaný iba raz aj keď na ňom závisí viacero modulov.[5]

---

**Algoritmus 3** Ukážka controlleru v Angular.js a jeho volanie na HTML elemente.

---

```
var myApp = angular.module('myApp', []);

myApp.controller('LabController', ['$scope', function($scope) {
    $scope.name = 'StarkLab!';
}]);

-----

<div ng-controller="LabController">
    {{ name }}
</div>
```

---

Pri programovaní modulov musíme dávať pozor na to, aby sme si neprepísali existujúci v pamäti.

---

**Algoritmus 4** Ukážka vytvorenia modulov a pridávania funkcionality do nich.

---

```
var myModule = angular.module('myModule', []);

// pridanie directivy a sluzby do modulu
myModule.service('myService', ...);
myModule.directive('myDirective', ...);

// toto volanie prepise uz vytvorene myService a myDirective vytvorenim
// noveho modulu
var myModule = angular.module('myModule', []);

// vyhodi chybu pretoze volame myOtherModule, ktory este nebol
// definovany
var myModule = angular.module('myOtherModule');
```

---

### 3.6.7 Service

Angular service je nahraditeľný objekt, ktorý je možné použiť v aplikácií pomocou dependency injection (DI). Services môžeme použiť pre lepšiu organizáciu a zdieľanie kódu naprieč celej aplikácie. Framework poskytuje mnoho užitočných services, ako napr. *\$http*, ale väčšina aplikácií má potrebu si vytvoriť vlastné.[5] Poznáme dva typy services:

- lazily instantiated - angular vytvorí inštanciu service iba v prípade, že na nej závisí komponent/modul

- singleton - každý komponent závisiaci na service získava referenciu na jedinou inštanciu generovanú z service factory.

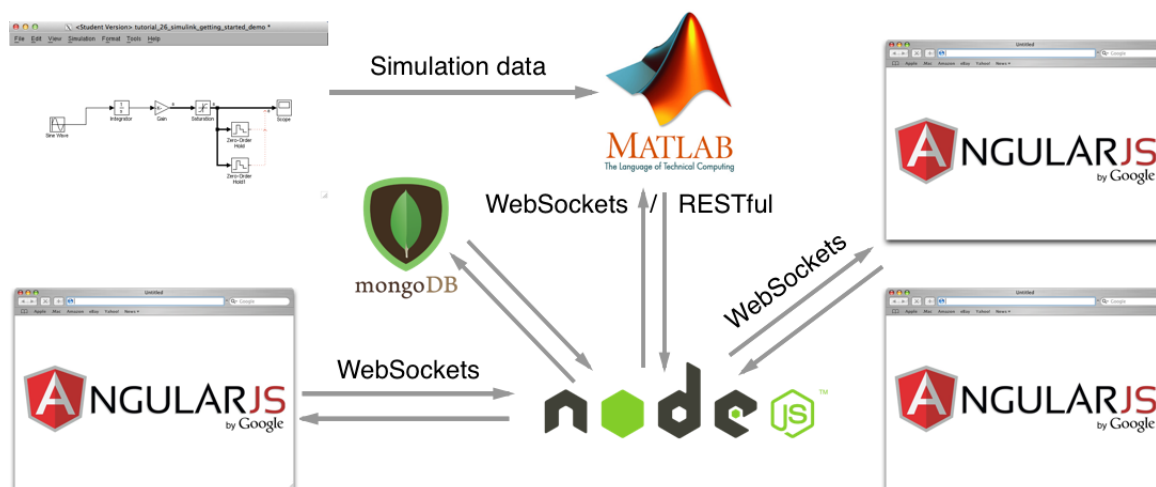
### **3.6.8 Bootstrap aplikácie**

možno lepšie pred konceptom



## 4 Návrh a implementácia StarkLab

Témou práce je navrhnuť a implementovať virtuálne laboratórium s využitím JavaScriptu na strane servera. V tejto kapitole si ukážeme predpokladaný návrh celého virtuálneho laboratória s využitím technológií na jednotlivých komponentoch. Ich presný popis a využitie si popíšeme viacej v sekcii č.3. Teda máme dané, že budeme využívať Node.js technológiu ako server. To je náš centrálny server (ďalej len StarkLab), ktorý spracováva dáta z Matlab workspace. Do Matlab workspace sú dáta posielané intervalovo zo Simulinku, v ktorom bola spustená referenčná schéma generujúca dáta. Zo začiatku nebolo isté či bude možné docieľiť spustenie Simulinku v reálnom čase multiplatformovo. Našli sme riešenie Simulink Real-time priamo od Mathworks, ktorá toto umožňuje ale bohužiaľ len pre operačný systém Windows. Lenže neskôr sme našli knižnicu *tos\_lib.mdl*, ktorá nám túto funkcionality poskytla aj pod MAC OS X. Na komunikáciu s workspace využívame volanie RESTful služieb, ktoré podporuje aj Matlab R2015b [11]. Keď si užívateľ spustí simuláciu cez klienta, tak sa údaje budú uchovávať v databáze mongoDB pre neskoršie spracovanie.



Obrázok 12: Návrh komunikácie medzi jednotlivými komponentami.

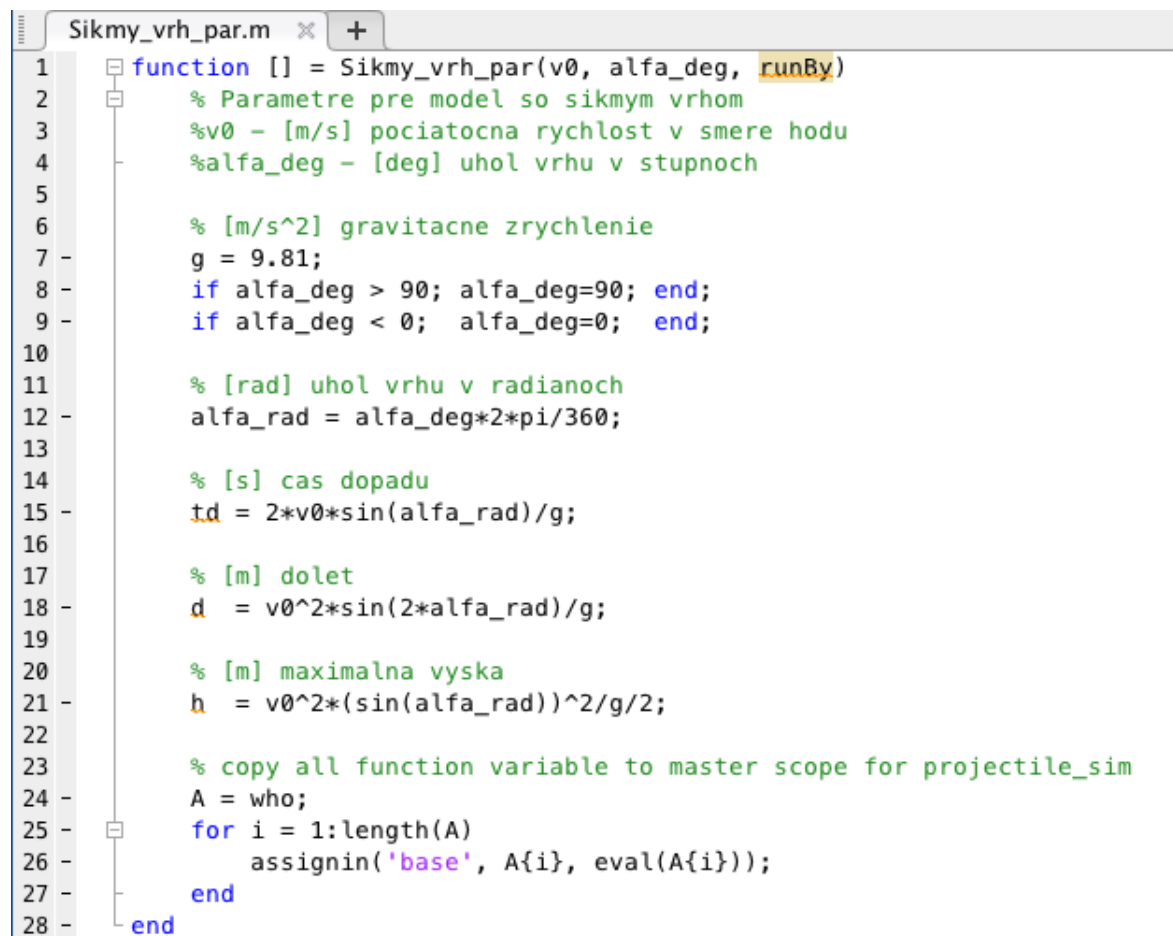
### 4.1 Referenčný model simulácie v Matlabe

Systém síce bude mať možnosť spolupracovať s reálnymi zariadeniami, pre vývoj sme použili simuláciu dynamického systému šikmého vrhu. Pre spustenie simulácie je potrebné zavolať dva súbory. Účel prvého je inicializácia premenných potrebných pre výpočet súradníc polohy bodu v šikmom vrhu. Na obrázku č.13 si všimnime, že je potrebné zadať aj parametre pre spustenie a funkcia nič nevracia. Prvý z parametrov  $v_0$  nastavuje

počiatočnú rýchlosť telesa v metroch za sekundu, akou bude vymrštené z bodu (0,0). Druhý parameter *alfa\_deg* zase určuje pod akým uhlom bude teleso vymrštené v stupňoch. Posledný parameter *runBy* nie je potrebný k samotnej simulácii, ale skôr k identifikácii, kto spustil simuláciu. Vďaka tomu je možné neskôr z databázy zistiť kto spustil simuláciu.

Keď sú parametre funkcie správne zadané a spustí sa telo funkcie, tak sa nastaví a vypočítajú hodnoty ako *alfa\_rad*, *td*, *d*, *h* ktorých popis vidíme na obrázku a sú potrebné ako vstupné parametre do simulácie.

Posledná časť kódu od riadku č. 23 slúži pre prekopírovanie premenných do hlavného scope matlabu, čiže workspace. Keby toto neurobíme, tak všetky hodnoty po skončení funkcie zaniknú keďže sa berú ako lokálne. Síce by bolo možné sa tomuto vyhnúť, tak že by sme nevolali funkciu *Sikmy\_vrh\_par(80, 70, 'xstark')* a volali len súbor, čo by vlastne zostali všetky hodnoty v hlavnom workspace. Lenže v tomto prípade by nebolo možné nastavovať inicializačné premenné z webovej aplikácie, ale by museli byť natvrdo nastavené v kóde matlabu.



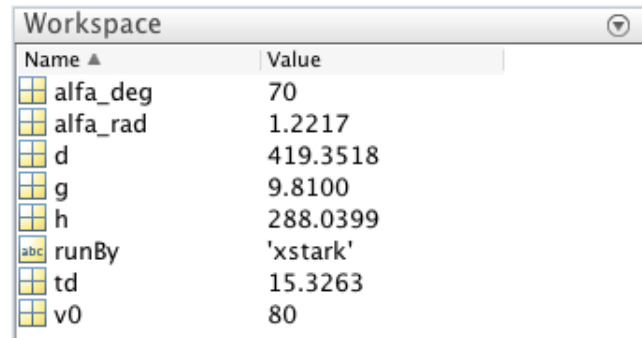
```

1  function [] = Sikmy_vrh_par(v0, alfa_deg, runBy)
2      % Parametre pre model so sikmym vrhom
3      %v0 - [m/s] pociatocna rychlost v smere hodu
4      %alfa_deg - [deg] uhol vrhu v stupnoch
5
6      % [m/s^2] gravitacne zrychlenie
7      g = 9.81;
8      if alfa_deg > 90; alfa_deg=90; end;
9      if alfa_deg < 0; alfa_deg=0; end;
10
11     % [rad] uhol vrhu v radianoch
12     alfa_rad = alfa_deg*2*pi/360;
13
14     % [s] cas dopadu
15     td = 2*v0*sin(alfa_rad)/g;
16
17     % [m] dolet
18     d = v0^2*sin(2*alfa_rad)/g;
19
20     % [m] maximalna vyska
21     h = v0^2*(sin(alfa_rad))^2/g/2;
22
23     % copy all function variable to master scope for projectile_sim
24     A = who;
25     for i = 1:length(A)
26         assignin('base', A{i}, eval(A{i}));
27     end
28 end

```

Obrázok 13: Inicializačná funkcia šikmého vrhu v Matlabe.

Ako sme si už povedali, že hodnoty sa po spustení úspešne uložia do Matlab workspace a to vidieť aj na obrázku č. 14. Tieto hodnoty boli vygenerované po zavolaní funkcie zo špecifickými hodnotami parametrov *Sikmy\_vrh\_par(80, 70, 'xstark')*.



Name	Value
alfa_deg	70
alfa_rad	1.2217
d	419.3518
g	9.8100
h	288.0399
runBy	'xstark'
td	15.3263
v0	80

Obrázok 14: Hodnoty v Matlab workspace po spustení funkcie.

Druhý súbor, ktorý potrebujeme spustiť po tomto je obslužný kód, vďaka ktorému vieme zaslať údaje do Node.js. Kvôli jeho dĺžke a vlastnej implementácii ho nie je možné zobraziť celý, tak si popíšeme len kľúčové vlastnosti.

Pri inicializácii sa nastavi URL cesta pre Express.js API, na ktorú ma komunikovať. Prednačítavanie modelu sa vykoná pomocou funkcie *load\_system('Sikmy\_vrh\_2')*; Táto funkcia vyhledá v aktuálnom priečinku súbor *Sikmy\_vrh\_2.mdl* a nastaví ho ako bdroot čo je aktuálne top-level model. Po nastavení musíme simuláciu aj spustiť. Ovládanie simulácie sa robí pomocou príkazu *set\_param(model, 'SimulationCommand', 'Start')*; kde posledný parameter znamená spustenie simulácie. Následne sa spustí nekonečný *while* cyklus, vďaka ktorému je možné zbierať údaje zo simulácie až do stavu kým nie je ukončená. Na začiatku cyklu sa volá *set\_param(bdroot, 'SimulationCommand', 'WriteDataLogs')*; čo vlastne pristúpi k aktuálne najvyššie spustenej simulácii a snaží sa zapísať do Matlab workspace aktuálne vypočítane hodnoty. Bez tohto príkazu by sa zapísali až po skončení simulácie.

Medzitým sa údaje upravujú na potrebný formát a pred odoslaním na REST API je vhodné ho zabaliť do JSON štruktúry. Tú vieme vytvoriť pomocou knižnice *jsonlab* v aktuálnej verzii 1.2. Pomocou sekvencie týchto dvoch príkazov vytvoríme požadovaný JSON formát a odošleme ho na API. Vytvorenie JSON *json = savejson('result', struct('user', runBy, 'status', 'running', 'sessionId', 'xxx72', 'data', struct('time', A, 'vy', B, 'y', C, 'x', D)))*; a zaslanie do služby *response = webwrite(url,json,options)*; Pomocou príkazu *get\_param(model, 'SimulationStatus')* vieme získať aktuálny stav simulácie. V prípade, že simulácia stále beží tak získame výstup *'running'* inak je ukočená a vráti reťazec

'stopped'. Hneď ako dostaneme status 'stopped' tak ukončíme cyklus pomocou *break* a vieme že všetky dáta su prenesené do Node.js.

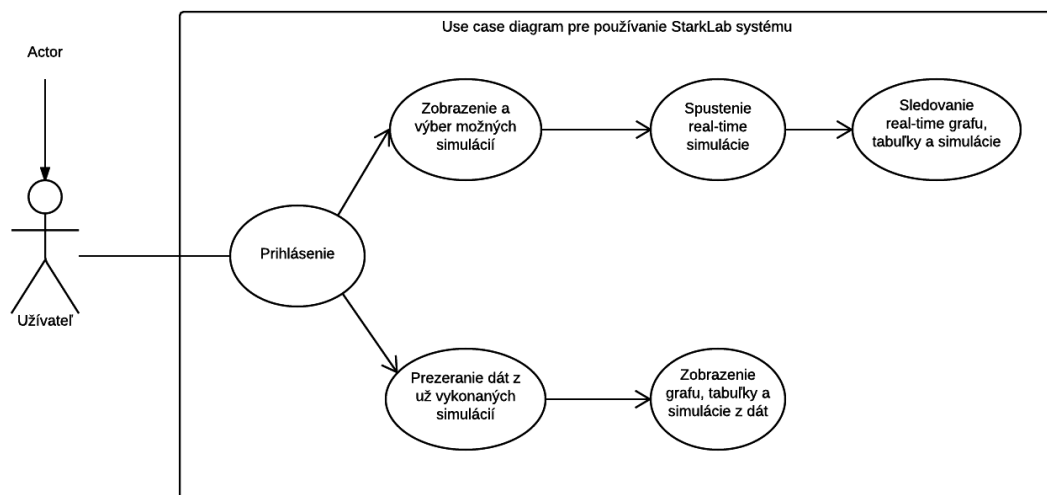
## 4.2 Diagramy

Pred tvorbou rôznych softwarových systémov zväčša nezačneme hneď programovať, ale najprv si zadefinujeme požiadavky, ktoré by mal spĺňať. Následne na základe požiadaviek vytvoríme diagramy, podľa ktorých sa neskôr budeme riadiť pri tvorbe systému. Niektoré nám ukážu správanie len zo všeobecnosti čo by sa mohlo dať v systéme robiť ako napr. "Prípady použitia". Potom existuje aj "Diagram tried", ktorý hovorí o jednotlivých triedach, ktoré sa vytvoria v kóde a jeho vlastnostiach. Diagram tried v našom prípade nie je možné vytvoriť, pretože JavaScript nie je klasický objektovo orientovaný jazyk. Ale zase môžeme využiť sekvenčné diagramy, pretože tie nám hovoria o interakcií medzi jednotlivými modelmi, prípadne časťami kódu.

### 4.2.1 Prípad použitia

Diagram prípadov použitia sa používa k popisu chovania systému z hľadiska užívateľa a zachytáva, ktoré typy užívateľov so systémom pracujú a aké činnosti v rámci systému vykonávajú.[18]

Na obrázku č. 15 vidíme možnosti používania systému na diagrame prípadov použitia. V systéme vystupuje len jeden typ používateľa a musí to byť člen STUBA LDAP. Bez prihlásenia sú jeho možnosti obmedzené a nemôže nič pozeráť ani vykonávať. Po prihlásení môže vykonať dve úlohy. Zobrazenie už existujúcich simulácií alebo vybrať simuláciu, ktorú chce spustiť s počiatočnými parametrami ak sú potrebné.

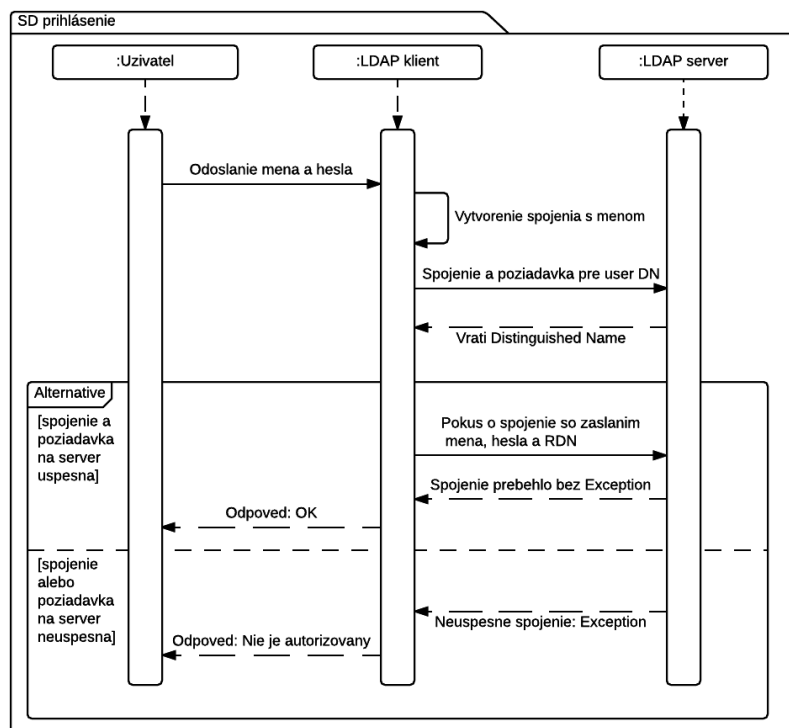


Obrázok 15: Diagram prípadu použitia na prácu zo systémom.

#### 4.2.2 Diagram "tried" pre StarkLab

Síce sme hovorili o tom, že klasický diagram tried nie je možné vytvoriť pre JavaScript aplikáciu, ale našli sme knižnicu *wavi* (<https://www.npmjs.com/package/wavi>), ktorá dokáže vygenerovať takýto diagram. Síce to nie je prehľadné ako pri klasickom diagrame, ale je vhodné ho vygenerovať už len pre to, aby mal programátor informáciu o základnej štruktúre aplikácie ako vidíme na obrázku č. 16.





Obrázok 17: Sekvenčný diagram pre prihlásenie pomocou LDAP.

Na obrázku č.17 si všimneme ako približne funguje prihlásenie pomocou mena a hesla do LDAP systému. Nižšie si popíšeme kód ako to máme implementované v JavaScripte. Toto je len časť kódu, ktorá sa určená pre prihlásenie do systému. Na začiatku súboru je volaná požadovaná knižnica pre prácu s LDAP ako `var ldap = require("ldapjs");` Keď užívateľ príjde na stránku a vyplní prihlasovacie meno a heslo do stuba ldap, tak formulár ho presmeruje na `/login`, kde už sa postaral o routovanie Express.js. Z `request` parametra získame zadané username - `req.body.username` a password - `req.body.password`, ktoré sme vyplnili pred odosielaním formulára. Čiže ak sú oba vyplnené dostaneme sa do vnútra podmienky, kde sa vytvorí spojenie na linku `ldap://ldap.stuba.sk`, vytvorí RDN reťazec v tvare `"uid=xstark", ou=People, DC=stuba, DC=sk`. V momente keď je zostavený reťazec a máme získané heslo užívateľa, tak zavoláme ldap funkciu `bind` s parametrami RDN a heslom. V prípade ak nenastala žiadna chyba pri overovaní, tak sme úspešne overený a vytvoríme `session` a `cookie` pre užívateľa a presmerujeme ho na stránku, kde už môže vidieť možné simulácie.

```

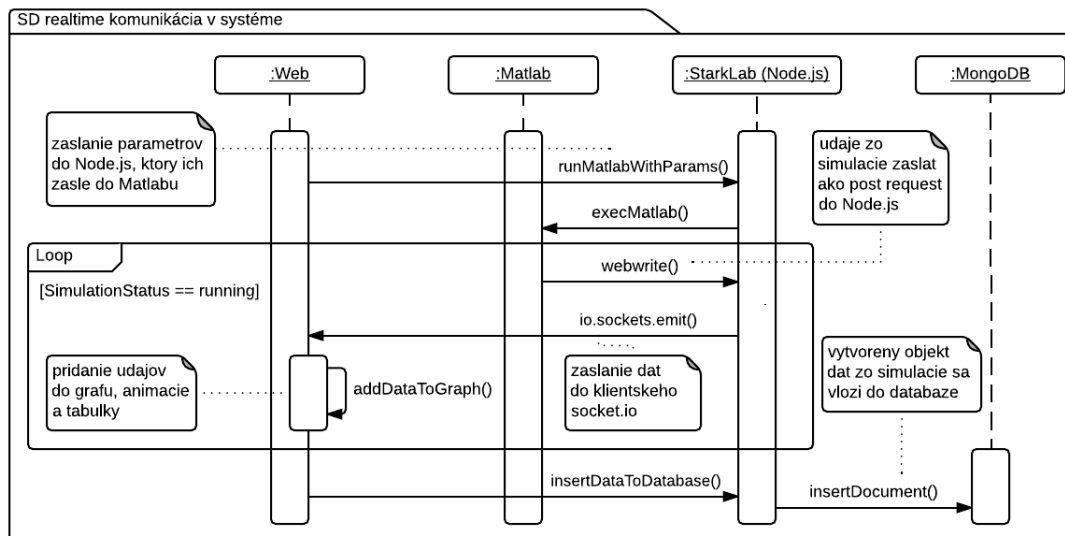
1  app.post('/login', function (req, res) {
2      if (req.body.username && req.body.password) {
3          var client = ldap.createClient({
4              url: 'ldap://ldap.stuba.sk'
5          });
6          var rdn = "uid=" + req.body.username + ", ou=People, DC=stuba, DC=sk";
7          var password = req.body.password;
8
9          client.bind(rdn, password, function (err) {
10             if (err != null) {
11                 if (err.name === "InvalidCredentialsError") {
12                     console.log("Credential error");
13                     res.redirect('/');
14                 }
15                 else {
16                     console.log("Unknown error: " + JSON.stringify(err));
17                     res.redirect('/');
18                 }
19             }
20             else {
21                 console.log("Login successful!");
22
23                 req.session.user = req.body.username;
24                 res.cookie('username', req.body.username);
25                 res.redirect('/matlab');
26             }
27         });
28     } else {
29         res.redirect('/');
30     }
31 });

```

## Komunikácia komponentov v systéme

Medzi jednotlivými komponentami systému prebieha komunikácia. Síce v každej časti prebieha inak, ale ich spoločný menovateľ je komunikácia založená na HTTP protokole. Na obrázku č. 18 vidíme, že komunikácia začína od webového klienta v prehliadači. Užívateľ zadá parametre simulácie a tie budú zaslané do StarkLab. Ten následne spustí Matlab na aktuálnom operačnom systéme s potrebnými súbormi simulácie a ich parametrami. Medzitým užívateľ čaká kým sa na pozadí spustí Matlab. Hneď na to začne vykonávať simuláciu a posielat údaje do StarkLab, ktorý ich pošle priamo webovému klientovi, kde bola spustená simulácia. Každé prijatie údajov sa prejaví zapísaním do grafu, animácie a tabuľky vo webovom rozhraní. Táto sekvencia sa opakuje pokiaľ platí podmienka, že je simulácia stále spustená. Po zastavení simulácie klient zašle požiadavku na zapísanie kompletných dát cez StarkLab priamo do dokumentovej databázy MongoDB.





Obrázok 18: Komunikácia medzi jednotlivými komponentami systému.

## 4.3 Databázový model MongoDB

V našom zadaní sme nepoužívali štandardnú SQL databázu, čiže nie je možné využiť bežné modelovanie cez ERD diagramy. Ako už vieme MongoDB neukladá dáta ako tabuľky, ale JSON dokumenty. Keďže sa jednalo o jednoduchý model šikmého vrhu, tak v tomto prípade nebude model extra zložitý. V tejto časti si ukážeme do akého objektu ho v JavaScripte ukladáme a potom konkrétny príklad z databázy.

```

1 function ProjectileDataObject(user, time, x, y, vy) {
2   this.user = user;
3   this.experiment = 'projectile';
4   this.executed = new Date();
5   this.time = time;
6   this.x = x;
7   this.y = y;
8   this.vy = vy;
9 }

```

Obrázok 19: Model objektu v JavaScripte, pre vytvorenie záznamu v MongoDB.

Ako vidíme v JavaScripte sa nepoužívajú žiadne typy. Hodnoty *user*, *experiment* budú vždy typu **String**, *executed* je typu **Date** vo formáte ISO-8601 a jeho formát je YYYY-MM-DDTHH:mm:ss.sssZ. Zvyšok hodnôt *time*, *x*, *y*, *vy* sú hodnoty závislé od simulácie a každá z nich je pole čísiel.

Na obrázku č.20 je záznam z MongoDB kde konkrétne hodnoty *time*, *x*, *y*, *vy* nie sú zobrazené, pretože simulácia vygenerovala až 788 hodnôt.

Key	Value	Type
▼ (1) {_id : 570976a056b5eff515993525}	{ 8 fields }	Document
_id	570976a056b5eff515993525	ObjectId
"user"	xstark	String
"experiment"	projectile	String
"executed"	2016-04-09T21:39:44.382Z	String
time	[ 788 elements ]	Array
x	[ 788 elements ]	Array
y	[ 788 elements ]	Array
vy	[ 788 elements ]	Array

Obrázok 20: Príklad záznamu simulácie v MongoDB.

## 4.4 Node.js s frameworkom Express.js

Hlavná časť tejto práce je sústredená na komunikáciu medzi Matlabom a Node.js, resp Express.js. Využívame ich posledné verzie Node.js verzie 6.0.0 a Express.js verzie 4.13.4. Je to vlastne len framework a umožňuje jednoduchšiu tvorbu REST služieb. Na začiatok si povieme čo sme použili ako *middleware*. Ako sme si už hovorili je to funkcia, ktorá má prístup do request objektu a response objektu. Použili sme *body-parser*, vďaka ktorému bolo možné spracovávať telo POST požiadavky, ktorá sa zasielala z Matlabu. Potom ho je potrebné ešte aj aktivovať ako middleware v kóde `app.use(bodyParser.json({limit: '2mb'}));`. Po viacerých testoch som zistil, že základný limit je nastavený na 1MB, kde sa vyskytoval problém s prijatím POST requestu, v prípade ak boli údaje v tele moc dlhé.

### 4.4.1 Vlastný middleware prihlasovania

V aktuálnom softwarovom riešení nebolo potrebné vytvoriť prihlasovanie, pretože neskôr bude integrované do LMS Moodle, ktorý si takéto veci už rieši svojím spôsobom. Ale pre ukážku možností frameworku Express.js som implementoval middleware *express-session*, vďaka ktorému môžeme vytvoriť reláciu pre úspešne autorizovaného užívateľa.

Pri implementácii prihlasovania cez nastavovania session som zistil, že v každom smerovaní (route), bolo potrebné kontrolovať na začiatku kódu, či je prihlásený používateľ alebo či je to požiadavka z Matlab služby. Pri väčšom projekte, prípadne viacerých route by to začalo byť neefektívne a neprehľadné. Vytvorili sme vlastný middleware na obrázku č. 21, ktorý nam overuje prihláseného človeka pri každom prístupe na hocikáku route.

```
// middleware to check logged user through browser or result from matlab
app.use(function (req, res, next) {
  if ((req.session && req.session.user) || req.body && req.body.result && req.body.result.user) {
    next();
  } else {
    res.redirect('/');
  }
});
```

Obrázok 21: Middleware kód v Express.js na kontrolu prihlásenia.

Volanie *app.use* nám zabezpečí, že sa táto funkcia zaregistruje ako middleware a bude volaná pri každom requeste. Prvá časť podmienky overuje, či prichádzame z webového prehliadača a je nastavená session. Keďže Matlabu nevieme nastaviť session, pretože tá sa ukladá len na webserveri tak sme to vyriešili iným spôsobom. V prichádzajúcom tele POST requestu od Matlabu je objekt *result*, kde sú uložené vygenerované údaje. Priradili sme mu ďalšiu property *user*, ktorá keď je nastavená, tak vieme, že požiadavka prichádza z Matlabu a vieme jej povoliť prístup. V prípade splnenej aspoň jednej podmienky sa zavolá callback *next()*, čo vlastne vyvolá buď nejaký ďalší zaregistrovaný middleware, alebo route, na ktorú sme posielali požiadavku. Ak nie je splnená ani jedna z dvoch podmienok, tak nás server presmeruje na hlavnú stránku, kde sa od nás požaduje znovu prihlásenie.

#### 4.4.2 Spustenie Matlabu z príkazového riadku

Nebolo isté ako budeme spúšťať simuláciu, pretože nebolo isté či Node.js umožňuje vykonávať príkazy nad operačným systémom, resp. spúšťať programy. Spočiatku sme riešili komunikáciu takým spôsobom, že Matlab sa spustil ručne a v ňom potrebné inicializačné súbory a následne samotná simulácia. Takéto riešenie nie je postačujúce z hľadiska automatizácie a samostatnosti.

Potom sme zistili, že Node.js má možnosti na spustenie hocijakého software, ktorý je možné spustiť cez terminál. Pre zjednodušenie práce som využil modul *shelljs*, ktorý túto funkcionality poskytuje. Na obrázku č. 22 je funkcionality, ktorá sa vykoná po prístupe na route */matlab/run*. Táto časť sa volá po odoslaní formuláru s parametrami počiatočnej rýchlosti a uhlu z webového prehliadača.

Ako vidíme vstupný reťazec, ktorý chceme vykonať, bolo potrebné zadať cestu k Matlabu kde sa nachádza na súborom systéme. Ďalej je vidieť, že sa spúšťa aj s istými parametrami. Všetky potrebné možnosti som našiel v online dokumentácii Matlabu. [12]

Keďže sa spúšťa z terminálu a nie je potrebné, aby nám ukazoval výsledky aj v okne, tak jeho spúšťanie vieme urýchliť použitím parametrov, ktoré vidíme na obrázku. Pomocou parametra *-r* vieme vykonávať príkazy v príkazovom riadku Matlabu, napr.

špecifikovať cestu, kde sa nachádza naša simulácia a potrebné súbory. Následne zavoláme funkciu *Sikmy\_vrh\_par()* kam sa pošlú parametre počiatočnej rýchlosti a uhla a zároveň prihlásený používateľ. Hneď ako sa skončí vykonávanie tejto funkcie, spustí sa súbor *projectile\_sim* čo je už konkrétna simulácia. Po skončení už môžeme zatvoriť aj Matlab cez príkaz *exit*;

Po zostrojení tohto reťazca ako parametra funkcie *exec* a jeho spustení sa čaká na vykonávanie Matlabu. Medzitým nás server presmeruje na stránku */dashboard* kde sa čaká na údaje zo simulácie. Hneď ako sa načíta simulácia zo vstupnými parametrami a začne vykonávať v momente sa údaje zobrazia aj vo webovom prehliadači realtime.

```
app.post('/matlab/run', function (req, res) {
  var cmd = '\\Applications\\MATLAB_R2015b.app\\bin\\matlab -nosplash -nodesktop -noFigureWindows ' +
    '-r \\\"cd \\Users\\Erich\\Desktop\\DP\\Matlab\\diploma-matlab\\;Sikmy_vrh_par(' +
    + req.body.v0 + ', ' + req.body.alfa_deg + ', \\\" + req.session.user + '\\\";projectile_sim;exit;\\\"';

  shell.exec(cmd, function (code, stdout, stderr) {
  });

  res.redirect('/dashboard');
});
```

Obrázok 22: Spustenie príkazu v OS pomocou shelljs modulu.

### 4.4.3 Práca s dátami simulácie

Keď je simulácia spustená a Matlab pripravený zasielať výsledky na route, ktorú vidíme na obrázku č. 23. Výsledky z Matlabu máme uložené v tele každého requestu, ktorý príjde. Následne môžeme k nim prístupiť pomocou *req.body*. Pôvodne socket.io odosielať údaje len na adresu "message". V tom prípade by sme ale nevedeli identifikovať, ku ktorému užívateľovi sa majú posilať údaje. Po implementácii session sme zvažili, že by ich bolo lepšie posilať priamo na adresu "message:xstark", teda s menom prihláseného užívateľa. Vždy po prijatí je potrebné odoslať status kód 200, čo v HTTP znamená OK pre prijatie dát.

```
app.post('/matlab/result', function (req, res) {
  console.log(req.body.result.user);
  io.sockets.emit("message:" + req.body.result.user, req.body);
  res.sendStatus(200);
});
```

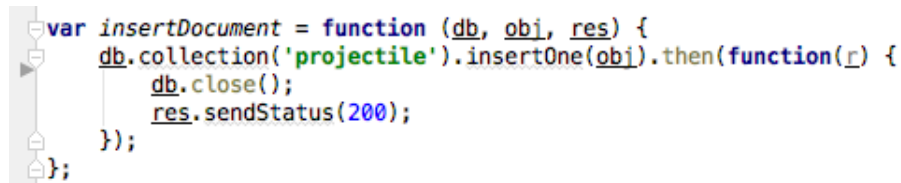
Obrázok 23: Prijímanie dát a zasielanie pomocou socket.io do prehliadača.

### 4.4.4 Zápis dát do MongoDB

Po prijímaní a zobrazovaní údajov nasleduje ich spracovanie. Ich spracovanie sa vykonáva na úrovni Express.js za pomoci MongoDB ovládača.

Na začiatku je potrebné najprv zavolať modul a následne jeho klientskú časť cez `var MongoClient = require('mongodb').MongoClient;`. Potom nasleduje definovanie URL, na ktorej máme spustenú databázu `mongodb://localhost:27017/test`.

Najprv si ukážeme funkcie, ktoré vykonávajú svoju úlohu, až potom ich použitie. Prvá z nich je vloženie dokumentu do databázy na obrázku č. 24. Táto funkcia na vytvorenie jedného záznamu, sa pokúsi vložiť `obj` do kolekcie `projectile` čo je vlastne náš kompletný záznam zo simulácie. Po úspešnom vložení zatvorí konekciu na databázu a vráti HTTP status 200.

The image shows a snippet of JavaScript code with syntax highlighting. It defines a function named 'insertDocument' that takes three arguments: 'db', 'obj', and 'res'. Inside the function, it calls 'db.collection('projectile').insertOne(obj).then(function(r) {' to insert the object into the 'projectile' collection. After the insertion, it calls 'db.close()' to close the database connection and 'res.sendStatus(200)' to send an HTTP 200 status response. The function is enclosed in curly braces and ends with a semicolon.

```
var insertDocument = function (db, obj, res) {  
  db.collection('projectile').insertOne(obj).then(function(r) {  
    db.close();  
    res.sendStatus(200);  
  });  
};
```

Obrázok 24: Vloženie záznamu do MongoDB pomocou JavaScriptu.

Druhú funkciu, ktorú využívame na prácu s databázou je na obrázku č. 27. Slúži na vyhľadanie záznamu v závislosti od parametrov. Jednu funkciu vieme využiť na viacero účelov. V prípade ak zadáme parameter funkcie `sim`, tak v podmienke sa skontroluje, či je dostupná vlastnosť `id`. Ak áno, tak vyhľadanie v databáze, prebehne na základe parametrov užívateľského mena a konkrétného `id`, čiže nám vyhledá iba jeden záznam. V opačnom prípade vyhledá všetky záznamy simulácií pre daného užívateľa. Táto funkcionality sa využíva pri zobrazení všetkých záznamov u klienta.

```

var findSimulation = function (db, sim, callback) {
  var query = {};
  var user = sim.user;
  var simType = sim.experiment;
  var simulationId = sim.id;
  if (simulationId) {
    query = {
      "_id": ObjectId(simulationId),
      "user": user
    }
  } else {
    query = {
      "user": user
    }
  }
  var cursor = db.collection("projectile").find( query ).toArray(function(err, results){
    callback(results);
  });
};

```

Obrázok 25: Vyhľadanie záznamu v MongoDB pomocou JavaScriptu.

Po ukážke implementácií funkcií, ktoré vykonávajú zápis/čítanie si ukážeme ich volanie. Na obrázku č. 26 je vloženie záznamu po prístupe na route `/mongo/insert/one`. Pre vykonanie nejakej operácie nad databázou musíme vytvoriť spojenie medzi klientom a MongoDB. Služi na to funkcia `connect(url, callback)`, ktorej prvej parameter je URL kde sa nachádza DB a druhý je callback, kde sa vracajú údaje.

```

app.post('/mongo/insert/one', function (req, res) {
  console.log("mongo insert one");

  MongoClient.connect(url, function (err, db) {
    if (err === null) {
      console.log("Connected correctly to server.");
      insertDocument(db, req.body, res);
    }
  });
});

```

Obrázok 26: Volanie vloženia záznamu do MongoDB.

Pre vyhľadanie záznamu na obrázku č. 26 sa s pripojením nič nemení. Rozdiel vidíme vo využití volania route. Dvojbodka pred názvom `:user` znamená, že tento parameter môže byť v URL meniteľný, ak je tam ešte otáznik ako pri `:id?` znamená to, že tento parameter nie je povinný, a URL môže byť volaná aj bez neho. Ak sa v URL nachádzajú parametre, prístupujeme k nim pomocou poľa `req.params`.

```

app.get('/mongo/:user/:simulation?/:id?', function (req, res) {
  MongoClient.connect(url, function (err, db) {
    if (err) {
      console.log(err);
      res.send(err);
    } else {
      console.log("Connected correctly to server.");
      var checkedId = undefined;

      if (req.params.id && req.params.id.length === 24) {
        checkedId = req.params.id;
        // should be send bad param
      }

      var simulationParams = {
        user: req.params.user,
        experiment: req.params.simulation,
        id: checkedId
      };

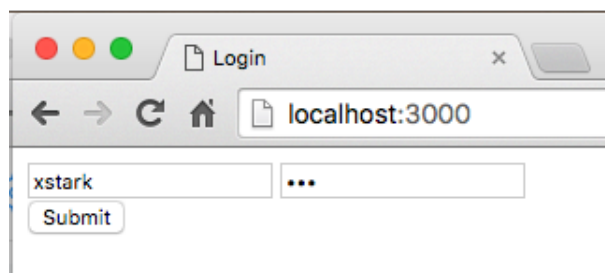
      findSimulation(db, simulationParams, function (results) {
        res.setHeader('Content-Type', 'application/json');
        res.send(JSON.stringify(results));
        db.close();
      });
    }
  });
});

```

Obrázok 27: Volanie vyhľadania záznamu v MongoDB.

## 4.5 Webový klient s frameworkom Angular.js

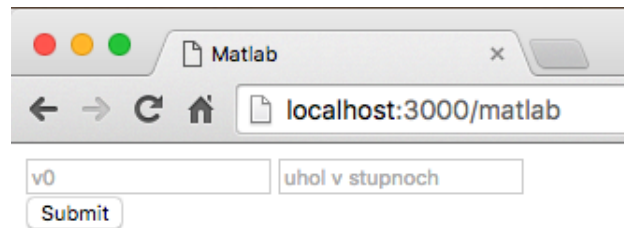
Na klientskú časť práce sme využili JavaScriptový framework Angular.js v poslednej stabilnej verzii 1.5.5. Úloha webového klienta, bolo overiť funkčnosť vytvoreného prostredia, ktorý generuje údaje. Funkčnosť sme overili a popíšeme si konkrétne obrazovky.



Obrázok 28: Prihlásenie do LDAP pomocou STUBA údajov.

Ako to funguje na pozadí už sme si popísali v sekcii 4.2.3, kde sa pojednávajú sekvenčné diagramy, konkrétne prihlásenie do LDAP. Po prihlásení sa zobrazí stránka vyhradená pre zadanie parametrov šikmého vrhu, počiatočnú rýchlosť  $v_0$  a uhol vystrelenia  $\alpha_{deg}$ . Pri zadaní rýchlosti  $v_0=40$  a  $\alpha_{deg}=60$  sme presmerovaní na stránku

/matlab.



Obrázok 29: Parametre simulácie - počiatočná rýchlosť a uhol v stupňoch.

#### 4.5.1 Grafy s Chart.js

Pre vizualizáciu prichádzajúcich dát používame knižnicu na grafy *Chart.js*, ktorá je vytvorená pomocou HTML canvas. Využívame poslednú verziu jednotkovej rady 1.1.1. Použiť ju môžeme ako skript v HTML stránke `<script src="/node_modules/chart.js/Chart.min.js"></script>`. Ako si môžeme všimnúť tak nie je závislá od žiadnej online CDN služby, ale jej inštalácia prebehla lokálne pomocou NPM.

Použije sa tak, že v HTML stránke sa zavolá ako canvas, ktorému priradíme id `<canvas id="updating-chart" width="1200" height="300"></canvas>`. Na obrázku č. 30 vidíme vytvorenie grafu a nastavenie hodnôt. Na začiatku je potrebné získať element v HTML DOM, pre ktorý sme si predtým vytvorili ID, získať kontext canvasu a vytvoriť objekt *startingData* s nastaveniami pre inicializačné dáta. Následne je potrebné vytvoriť inštanciu grafu v prehliadači, kde ako parameter grafu smeruje kontext canvasu, a vola sa funcia čiarového grafu, ktorý tam potrebujeme. Tá zase vyžaduje ako vstupný parameter inicializačné dáta.

Ak by sme chceli rozšíriť graf, resp. ak by bolo rozumné súčasne vykreslovať viacero závislých veličín, tak je možné zdefinovať ďalší dataset.



```

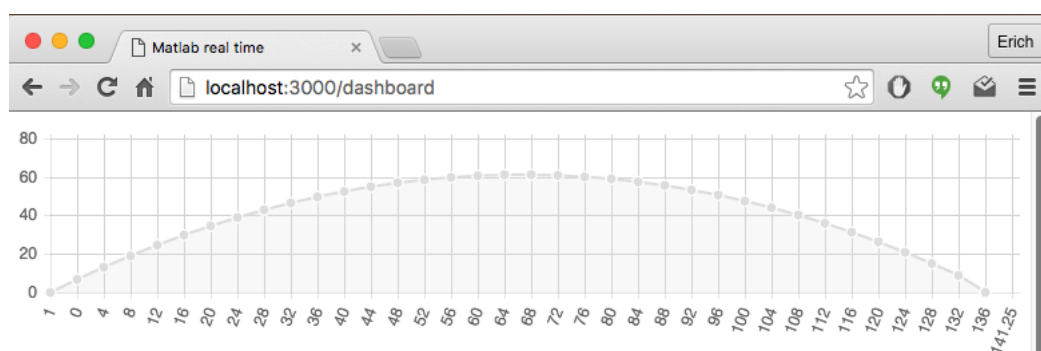
var canvas = document.getElementById('updating-chart'),
    ctx = canvas.getContext('2d'),
    startingData = {
      labels: [1],
      datasets: [
        {
          fillColor: "rgba(220,220,220,0.2)",
          strokeColor: "rgba(220,220,220,1)",
          pointColor: "rgba(220,220,220,1)",
          pointStrokeColor: "#fff",
          data: {}
        }
      ]
    },
    latestLabel = startingData.labels[0];

var myLiveChart = new Chart(ctx).Line(startingData, {
  responsive: true,
  animationEasing: 'easeOutBounce',
  animationSteps: 15,
  scaleGridLineColor: 'lightgray'
});

```

Obrázok 30: Inicializácia hodnôt pre Chart.js.

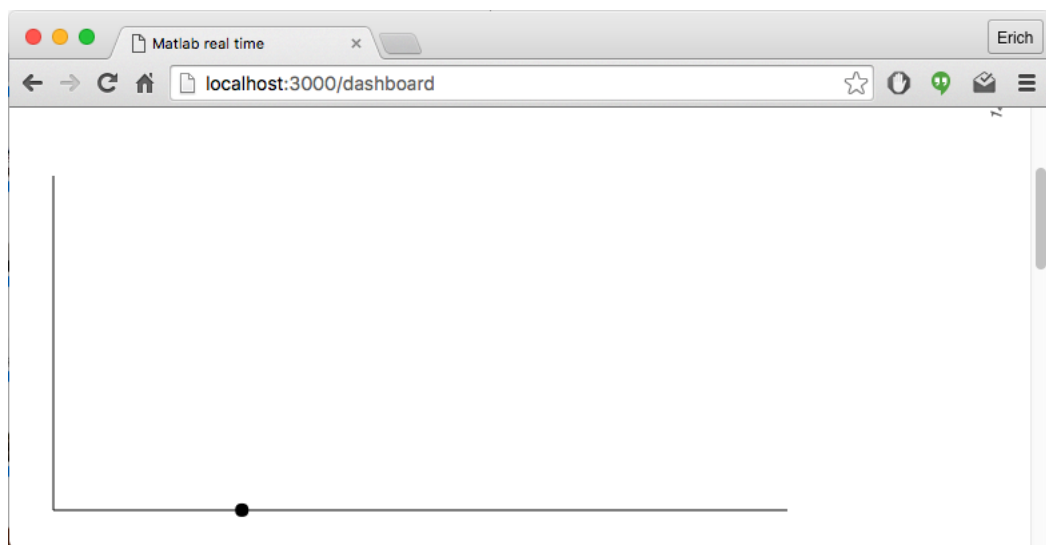
V sekcii 4.5 sme si hovorili o tom, čo musíme spraviť, aby nám generovalo údaje. Po presmerovaní na stránku /matlab treba počkať kým sa spustí Matlab zo simuláciou a následne sa začnú generovať údaje realtime do prehliadača. Táto časť by mohla byť zrýchlená tým, že sa Matlab bude nachádzať na výkonnom serveri. Na obrázku č. 31 je využitá knižnica Chart.js pre tvorbu grafov.



Obrázok 31: Graf vykresľujúci závislosti  $[x, y]$  v šikmom vrhu.

#### 4.5.2 Animácia pomocou html canvas

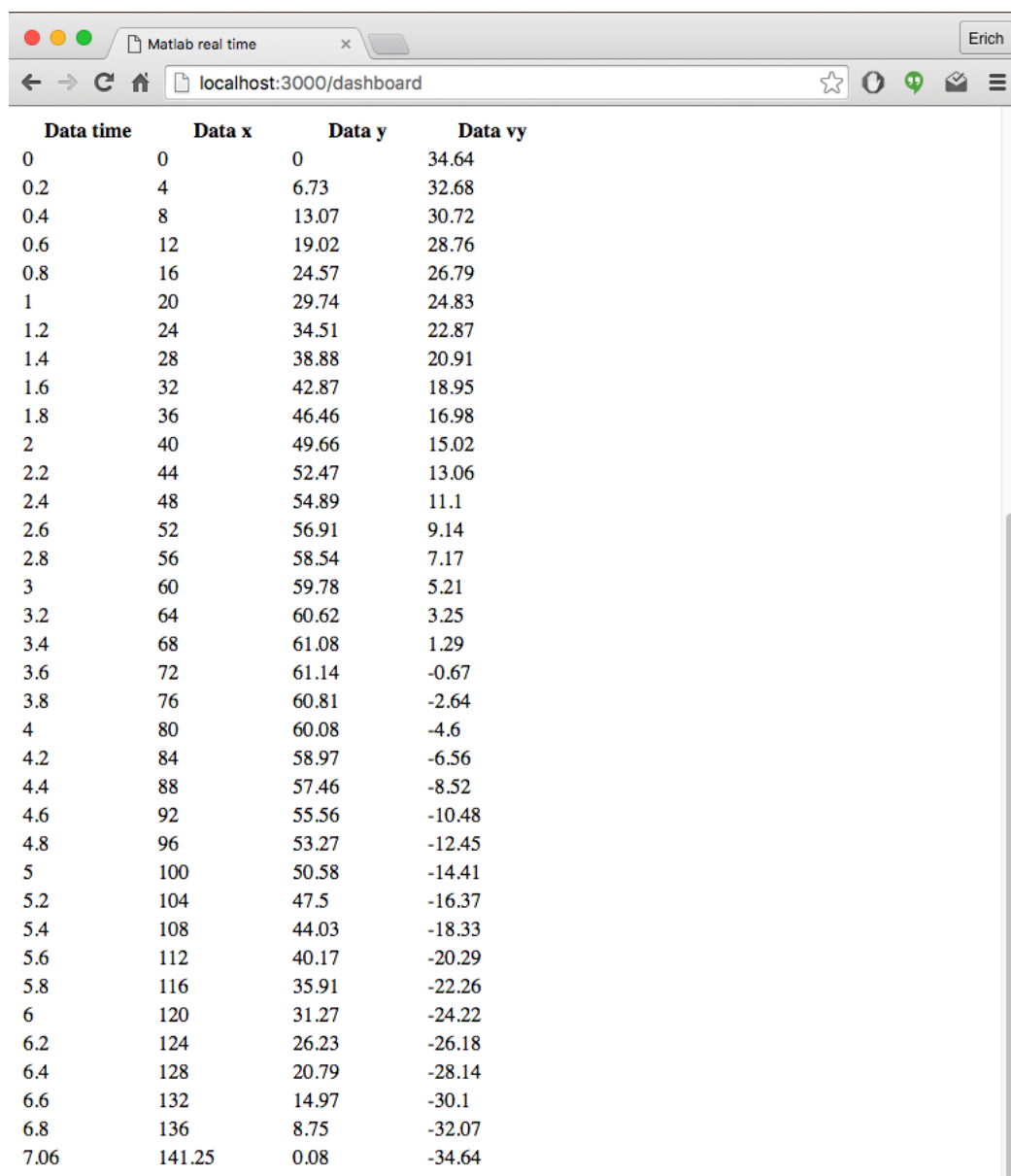
Na ďalšom obrázku č. 32 sa vykresľujú rovnaké údaje ako na grafe, len je to spôsobom animácie guľičky. Táto animácia bola vytvorená pomocou html canvas technológiie. Viac o canvas v sekcii 4.5.2.



Obrázok 32: Animácia vykresľujúca závislosti  $[x, y]$  v šikmom vrhu.

### 4.5.3 Tabuľka dát

Ako posledná časť sledovania dát je tabuľka, ktorej údaje sa generujú rovnako realtime ako aj pri grafe a animácii pred ním.

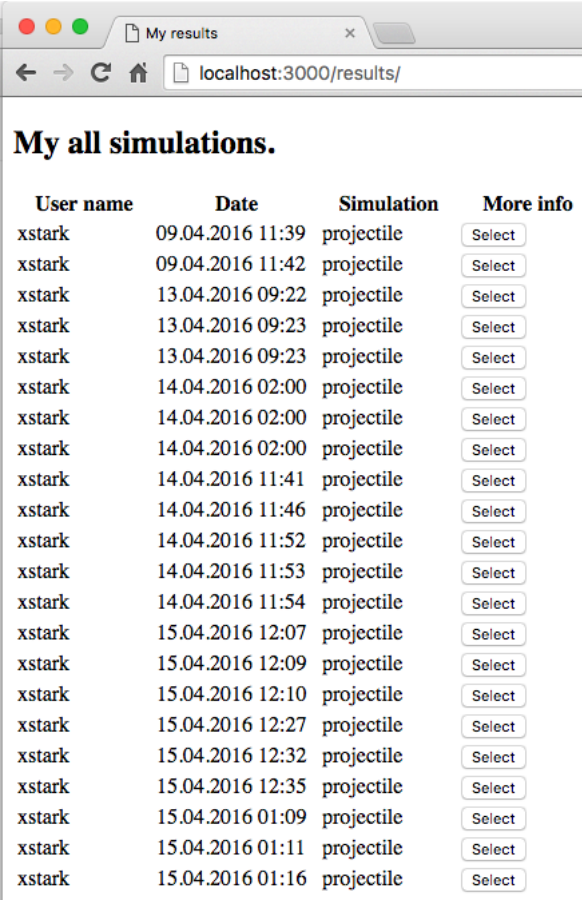


Data time	Data x	Data y	Data vy
0	0	0	34.64
0.2	4	6.73	32.68
0.4	8	13.07	30.72
0.6	12	19.02	28.76
0.8	16	24.57	26.79
1	20	29.74	24.83
1.2	24	34.51	22.87
1.4	28	38.88	20.91
1.6	32	42.87	18.95
1.8	36	46.46	16.98
2	40	49.66	15.02
2.2	44	52.47	13.06
2.4	48	54.89	11.1
2.6	52	56.91	9.14
2.8	56	58.54	7.17
3	60	59.78	5.21
3.2	64	60.62	3.25
3.4	68	61.08	1.29
3.6	72	61.14	-0.67
3.8	76	60.81	-2.64
4	80	60.08	-4.6
4.2	84	58.97	-6.56
4.4	88	57.46	-8.52
4.6	92	55.56	-10.48
4.8	96	53.27	-12.45
5	100	50.58	-14.41
5.2	104	47.5	-16.37
5.4	108	44.03	-18.33
5.6	112	40.17	-20.29
5.8	116	35.91	-22.26
6	120	31.27	-24.22
6.2	124	26.23	-26.18
6.4	128	20.79	-28.14
6.6	132	14.97	-30.1
6.8	136	8.75	-32.07
7.06	141.25	0.08	-34.64

Obrázok 33: Tabuľka dát času, x, y a smer rýchlosti vy.

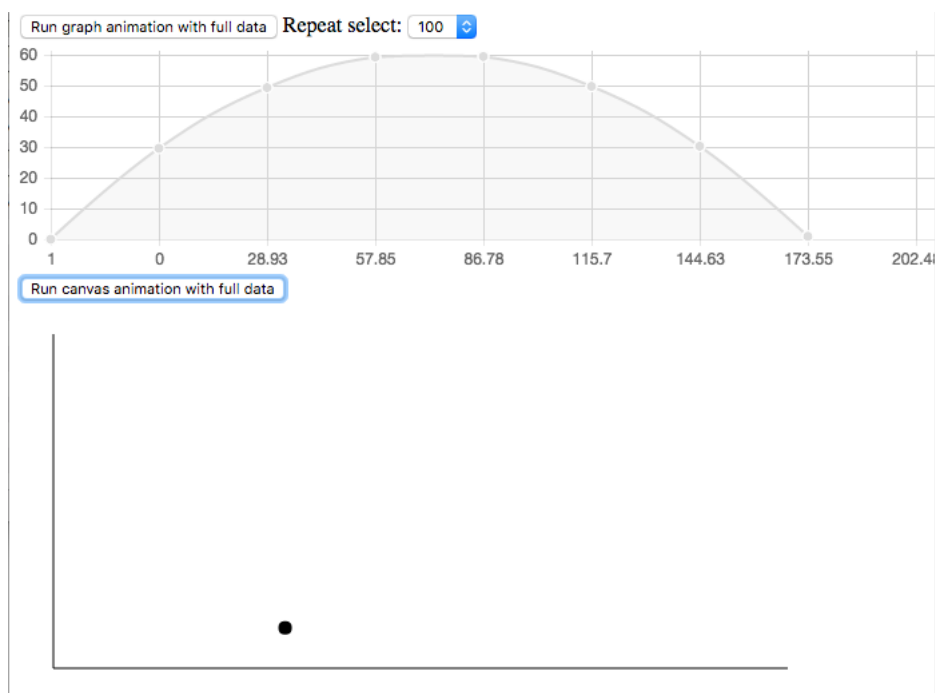
#### 4.5.4 Zobrazenie existujúcich simulácií

V tomto systéme nejde len o real-time vykreslenie dát, ale aj o ich neskoršie zobrazenie, spracovanie. Po prejdení na stránku /results, vidíme všetky spustené simulácie pre aktuálne prihláseného užívateľa.



User name	Date	Simulation	More info
xstark	09.04.2016 11:39	projectile	Select
xstark	09.04.2016 11:42	projectile	Select
xstark	13.04.2016 09:22	projectile	Select
xstark	13.04.2016 09:23	projectile	Select
xstark	13.04.2016 09:23	projectile	Select
xstark	14.04.2016 02:00	projectile	Select
xstark	14.04.2016 02:00	projectile	Select
xstark	14.04.2016 02:00	projectile	Select
xstark	14.04.2016 11:41	projectile	Select
xstark	14.04.2016 11:46	projectile	Select
xstark	14.04.2016 11:52	projectile	Select
xstark	14.04.2016 11:53	projectile	Select
xstark	14.04.2016 11:54	projectile	Select
xstark	15.04.2016 12:07	projectile	Select
xstark	15.04.2016 12:09	projectile	Select
xstark	15.04.2016 12:10	projectile	Select
xstark	15.04.2016 12:27	projectile	Select
xstark	15.04.2016 12:32	projectile	Select
xstark	15.04.2016 12:35	projectile	Select
xstark	15.04.2016 01:09	projectile	Select
xstark	15.04.2016 01:11	projectile	Select
xstark	15.04.2016 01:16	projectile	Select

Obrázok 34: Zoznam uložených simulácií pre prihláseného užívateľa.



Obrázok 35: Graf a animácia pre vybranú vzorku dát.

Na webovej stránke sa ešte nachádza pod animáciou sa nachádza rovnaká tabuľka ako v obrázku č. 33 s rozdielom, že v tejto tabuľke sú kompletne údaje.

# Záver

Diplomová práca nám ozrejmila pojmy ako virtuálne laboratórium, aké môže mať komponenty a jeho využitie v súčasnej dobe.

Cieľom tejto práce bolo naštudovanie problematiky virtuálnych laboratórií, vytvoriť stručnú analýzu existujúcich riešení a vytvoriť komplexnú aplikáciu, ktorá sa mala skladať z viacerých častí. Začali sme tvorbou a úpravou referenčnej simulácie šikmého vrhu pre Simulink podľa potrieb. Ďalej bolo treba zabezpečiť prenos dát medzi Matlabom a Node.js, čo sme vyriešili vďaka RESTful komunikácií. Údaje bolo potrebné posielat ďalej k web klientom. Táto časť sa vyriešila použitím JavaScript knižnice socket.io čo je obdoba websocketov s rozšírenou podporou aj pre staršie prehliadače. Na strane webu sa využil aktuálne populárny frontend JavaScript framework Angular.js. Medzi dôležité súčasti implementácie nebolo len zobrazovať realtime údaje v grafoch, ale ich aj ukladať pre neskoršie spracovanie. Údaje sa podarilo ukladať do dokumentovej databáze MongoDB. Ciele, ktoré sme si stanovili, sme aj splnili.

Za konečným výsledkom je vidieť mnoho práce. Síce súčasné riešenie nie je možné nasadiť do reálnej prevádzky bez istých úprav a integrácií, ale poslúži ako solídny základ, na ktorom je možné stavať a využiť ho minimálne v priestoroch FEI STU na simuláciu systému alebo na zber dát z reálneho zariadenia.

# Zoznam použitej literatúry

- [1] BORKA, T. Aplikácia typu klient-server pre virtuálne laboratórium s využitím platformy .net. Master's thesis, FEI STU, 2012. FEI-5406-30519.
- [2] EXPRESS.JS. Writing middleware for use in express apps, 2016. <http://expressjs.com/en/guide/writing-middleware.html>.
- [3] FARKAŠ, R. Vzdialené laboratórium riadenia reálnych systémov využívajúce technológie matlab a java. Master's thesis, FEI STU, 2011. FEI-5384-16514.
- [4] GOOGLE. Angularjs, 2016. <https://en.wikipedia.org/wiki/AngularJS>.
- [5] GOOGLE. What is angular?, 2016. <https://docs.angularjs.org/guide/introduction>.
- [6] HATHERLY, Paul. *The Virtual Laboratory and interactive screen experiments*. s. 1. <https://web.phys.ksu.edu/icpe/publications/teach2/Hatherly.pdf>.
- [7] KUNDRÁT, M. Aplikacia pre virtualne laboratorium vyuzivajuca java matlab interface. Master's thesis, FEI STU, 2014. FEI-5384-47855.
- [8] SANTANA I.; FERRE M.; IZAGUIRRE E.; ARACIL R.; HERNÁNDEZ L. *Remote Laboratories for Education and Research Purposes in Automatic Control Systems*. IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, VOL. 9, NO. 1, FEBRUARY 2013. s. 3. <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6140966>.
- [9] MATHWORKS. Matlab - the language of technical computing, 2016. <http://www.mathworks.com/products/matlab/>.
- [10] MATHWORKS. Matlab com integration, 2016. [http://www.mathworks.com/help/matlab/matlab\\_external/introducing-matlab-com-integration.html](http://www.mathworks.com/help/matlab/matlab_external/introducing-matlab-com-integration.html).
- [11] MATHWORKS. *Matlab RESTful web services*, r2015b ed. <http://www.mathworks.com/help/matlab/internet-file-access.html>, 2016.
- [12] MATHWORKS. Start matlab program from mac terminal, 2016. <http://www.mathworks.com/help/matlab/ref/matlabmac.html>.
- [13] MICROSOFT. What is com? <https://www.microsoft.com/com/default.mspx>.

- [14] MONGODB. What is mongodb, 2016. <https://en.wikipedia.org/wiki/MongoDB>.
- [15] NODEJS. History of nodejs, 2016. <https://en.wikipedia.org/wiki/Node.js>.
- [16] OLAKARA, A. R. Nodejs architecture, 2015. <http://abdelraoof.com/blog/2015/10/19/introduction-to-nodejs/>.
- [17] OLAKARA, A. R. Understanding nodejs event loop, 2015. <http://abdelraoof.com/blog/2015/10/28/understanding-nodejs-event-loop/>.
- [18] REJNKOVÁ, P. Diagram případů užití, 2009. [http://uml.czweb.org/pripad\\_uziti.htm](http://uml.czweb.org/pripad_uziti.htm).
- [19] S., G. L. B. Current trends in remote laboratories. *IEEE Transactions on Industrial Electronics* (december 2009). <http://ieeexplore.ieee.org/xpls/icp.jsp?arnumber=5280206>.
- [20] TSONEV, Krasimir. *Node.js By Example*. PACKT PUBLISHING.
- [21] TUTORIALSPPOINT. What is nodejs, 2015. [http://www.tutorialspoint.com/nodejs/nodejs\\_intro](http://www.tutorialspoint.com/nodejs/nodejs_intro)
- [22] TUTORIALSPPOINT. Angularjs concepts, 2016. [http://www.tutorialspoint.com/angularjs/angularjs\\_overview.htm](http://www.tutorialspoint.com/angularjs/angularjs_overview.htm).
- [23] TUTORIALSPPOINT. Express overview, 2016. [http://www.tutorialspoint.com/nodejs/nodejs\\_exp](http://www.tutorialspoint.com/nodejs/nodejs_exp)
- [24] VAQQAS, M. Matlab com integration, 2014. <http://www.drdobbs.com/web-development/restful-web-services-a-tutorial/240169069>.
- [25] VARGA, S. Virtuálne laboratórium riadenia dynamických systémov. Master's thesis, FEI STU, 2015. FEI-5396-5865.
- [26] vlab.co.in. *Philosophy of Virtual Laboratories*. <http://vlab.co.in/>.
- [27] ČERVENÝ, T. Mobilný klient pre účely virtuálneho laboratória využívajúci web rozhranie. Master's thesis, FEI STU, 2014. FEI-5384-29988.



# Prílohy

A	Štruktúra elektronického nosiča . . . . .	II
B	Algoritmus . . . . .	III

# A Štruktúra elektronického nosiča

```
\
\Bakalarska_praca.pdf
\FEIk_Identuty.xpi
\FEIkIdentity
\FEIkIdentity\chrome.manifest
\FEIkIdentity\install.rdf
\FEIkIdentity\content
\FEIkIdentity\content \function.js
\FEIkIdentity\content \options.xul
\FEIkIdentity\content \overlay.xul
\FEIkIdentity\content \window.js
\FEIkIdentity\content \window.xul
\FEIkIdentity\defaults
\FEIkIdentity\defaults\preferences
\FEIkIdentity\defaults\preferences \prefs.js
\FEIkIdentity\locale
\FEIkIdentity\locale \sk-SK
\FEIkIdentity\locale \sk-SK\options.dtd
\FEIkIdentity\locale \sk-SK\window.dtd
\FEIkIdentity\skin
```

# B Algoritmus

---

## Algoritmus B.1 Ukážka algoritmu

---

```
1  while 1
2
3  end
4  }
```

---