Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra ... Počítačové systémy a sítě

Diplomová práce

Simulátor akciové burzy

Bc. Jan Jůna

Vedoucí práce: Mgr. Jan Starý

14. dubna 2014

Poděkování

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či spracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 14. dubna 2014 .....

Odkaz na tuto práci

Jůna, Jan. *Simulátor akciové burzy.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2014.

Abstract

Sem doplňte ekvivalent abstraktu Vaší práce v angličtině.

Keywords stock market, realtime application, node.js, angular.js, mongodb, socket.io

Abstrakt

Práce se zabývá tvorbou burzovního systému od základních kamenů jako je order matching algoritmus pro spárování burzovních příkazů až po klientské rozhraní umožňující nakupovat nebo prodávat akcie. Aplikace je také tvořena pomocí moderních technologií, jako je Node.JS, Angular.JS, noSQL databáze MongoDB a mnoho dalších. Vše je tvořeno jednodušše a co nejvíce transparentně.

Klíčová slova akciová burza, realtime aplikace, node.js, angular.js, mongodb, socket.io

Obsah

Seznam obrázků

Úvod

Kapitola 1

Cíl práce

V této práci si kladu za cíl prozkoumat a otestovat pro mě nové a zajímavé technologie, naučit se vytvářet single page aplikace, pochopit základní principy burzovních systémů a jeden takový pak také implementovat. Aplikace, kterou budu tvořit a zde také popisovat bude sice pouze jen základní a velmi ořezanou verzí takové opravdové burzy, bude však základním a především i funkčním stavebním kamenem, který se může dále rozvýjet a vylepšovat až do stavu současných burzovních systémů, případně ještě dál. Krom pocitu z nové burzovního systému a získaných vědomostí na poli nových technologií však získám také referenci v a zajímavou poznámku do životopisu.

Mých cílů je tedy doopravdy mnoho, 3 hlavní z nich si ted dúkladněji popíšeme.

1.1 Burza

1.2 Nové technologie

1.3 Testování výkonu

Kapitola 2

Analýza a návrh

2.1 Požadavky

2.2 Koncept systému

Jak už bylo řečeno výše, celý systém je rozdělen do několika samostatných celků, které spolu komunikují přes síť. Koncept jednotlivých částí si teď zde popíšeme podrobněji.

2.2.1 Klientská aplikace

Klientem se rozumí webová aplikace, která slouží koncovým uživatelům k nakupování a prodávání jejich akcií. Zobrazují se zde aktuální informace o dění na burze a detaily jednotlivých obchodovatelných společností. Klient si tak může zobrazit vývoj ceny akcie a pak podle svého uvážení nakoupit nové nebo prodat jím vlastněné akcie. Kromě aktuálního stavu akcií zde uživatel má také možnost vidět své nakoupené akcie, podané a zatím nevyřízené příkazy a samozřejmě historii všech proběhlých akcí týkajících se uživatele. Další funkcí je zde registrace, kde se po zadání základních uživatelských údajů vytvoří účet pro obchodování. Při vytváření účtu se uživateli také na účet připíše základní virtuální finanční obnos, za který bude teprve moci nakupovat akcie. V opravdovém systému by toto bylo realizováno ještě nějakou formou platební brány, přes kterou by si klient nabil peníze z reálného účtu. Protože je tato práce zaměřená především na obchodování, v implementaci se zaměříme spíše na základní a klíčové části burzy, jako je samotné obchodování. Po úspěšné registraci uživatele máme možnost příhlásit se. Tato funkce se jako jediná provádí na straně serveru, kde se data z přihlašovacího formuláře nejprve odešlou na REST API umístěnou u brokera. Ten přihlašovací údaje ověří, odešle zpět výsledek přihlašování. V případě neúspěšného přihlášení klient obdrží chybovou hlášku, v opačném případě pak klíč, který bude dále používat při volání dotazů na REST API brokera. Při přihlášení si klientská aplikace tento klíč

uloží do session a odešle uživateli do prohlížeče aplikaci napsanou v javascriptovém frameworku angular.js. Tato aplikace si pak sama obsluhuje routování (přechod mezi stránkami), zobrazování jednotlivých pohledů (view v MVC frameworku), zpracování a načítání dat z externích zdrojů. Jednou z dalších možností, jak udělat uživatelské přihlášení je odesílat přihlašovací údaje rovnou z uživatelského prohlížeče na API brokera. Důvodem, proč v mé práci data nejprve odesílám na server klientské aplikace a až poté odtud kontaktuji brokera je, že vlastní aplikaci na obchodování posílám klientovi až při úspěšném přihlášení. Pro nepřihlášené uživatele je zobrazena pouze stránka s přihlášením a registrací. Nepřihlášení uživatelé tak nemohou získat potenciálně citlivé informace, jako jsou například volané adresy na REST API u brokera, bez toho, aniž by se nejprve přihlásili. V klientské aplikaci také pro ukládání sessions používám key-value databázi Redis, která v případě více spuštěných nodů poslouží jako společné úložiště sjednocující všechny sessions do jednoho místa. Více o výhodách použití Redis databáze si řekneme v sekci škálování. Při přihlášení klient začne komunikovat s REST API brokera, přes kterou si načítá veškerá data o uživateli. Protože může býkace s REST APt klientská aplikace umístěna na jiné doméně, než broker, probíhá komuniI pomocí CORS (Cross-origin resource sharing) mechanismu. To není nic jiného, než posílání ajax requestů na externí adresu, která však musí ve svých hlavičkách tento mechanismus povolit. Více k této metodě využívání externích zdrojů si uvedeme v popisu aplikace brokera. Přes tento komunikační kanál pak probíhá také zadávání BUY a SELL pokynů. Kromě RESTu zde používám také Socket.IO, což je nástroj pro obousměrnou komunikaci klienta i serveru. V praxi tak může server posílat push notifikace (zprávy o změně stavu) bez předchozího klientského dotazu (donedávna se toto řešilo cyklickým dotazováním klienta na nový stav, kdy velká zátěž byla tvořena neustálým a často i zbytečnými klientskými dotazy. Takto si server s clientem udržuje stálé spojení a až v případě změny stavu server sám upozorní klienta. V klientské aplikaci je tato technologie použita například při načítání nových provedených příkazů. Veškerá komunikace by se dala předělat jen na použití Socket.IO nebo opačně jen na REST, ale protože si v této práci kladu za cíl i naučit se nové technologie, využil jsem obě možnosti podle vhodnosti každé z nich. Pro vizuální stránku jsem použil free a open source šablonu AdminLTE, která svým přehledným rozhraním a responzivním chováním poskytuje uživateli příjemný pocit z používání aplikace.

2.2.1.1 Robot

2.2.1.2 Propojení s brokerem

2.2.2 Broker

2.2.2.1 Robot

2.2.3 Burza

2.2.3.1 Jádro

2.2.3.2 Rozhraní

2.2.4 Komunikace mezi celky

2.3 Použité technologie

2.3.1 Angular.js

Počátek vývoje této technologie sahá pouze do roku 2010, jde tedy o relativně novou technologii, která však již dnes přináší mnoho nástrojů a funkcí pro jednoduchý a především rychlý vývoj SPA (single page application) aplikací. Základní vlastností je, že se zobrazení výsledné stránky a jednoduché transformace dat provádí u klienta. To umožňuje odlehčit komunikaci na přenos pouze základní aplikace, která si poté už dotahuje pouze potřebná data. Ta pak na jednotlivých částech aplikace postupně zobrazuje. Kromě snížení přenášených dat zde dochází také k výraznému odlehčení zátěže na serveru, kdy se většina výpočtů deleguje na klienta. Tím se operace prováděné na serveru zredukují pouze na nejzákladnější práci s databází a podobně. Prostředí pro klientské výpočty, renderování vzhledu a mnoho dalšího zajišťuje právě open source MVC (model-view-controller) framework angular.js běžící v prohlížeči na straně klienta. Pomocí tohoto nástroje můžeme nadefinovat zobrazení jednotlivých stránek (view) a k nim pak controllery, které tyto zobrazení obsluhují. Controllery se dotazují nejčastěji přes nějakou formu API (v našem případě REST api realizované pomocí Node.JS) na data (model) a ty poté předávají do view, kde se teprve vykreslují a prezentují uživateli. Most mezi view a controllerem zajišťuje komponenta nazvaná “scope”, která zajišťuje obousměrné předávání dat mezi těmito dvěma vrstvami. Tato vlastnost je v terminologii angularu nazvaná two way data binding a je jednou z klíčových vlastností v tomto frameworku. Při změně dat v controlleru (například při dotažení nových dat z REST API) dojde totiž k automatickému promít-

nutí i do view vrstvy a naopak, při změně dat ve view (například úpravě položky ve formuláři) se data ihned promítnou i v controlleru. Svou jednoduchostí a schopností raketově urychlit vývoj výsledné aplikace si angular.js oblíbila spousta webových vývojářu. Jeho použití však není omezeno pouze na webové aplikace. S příchodem technologií jako je Phonegap nebo Titanium se dá nasadit také na mobilních zařízeních a tabletech. Phonegap zde slouží jako prostředník mezi mobilní API a aplikací napsanou v technologiích HTML5 a JavaScript, což může být právě aplikace napsaná v angularu. Nejedná se pak sice o nativní aplikaci a takové použití rozhodně není určeno pro náročné aplikace jako jsou například hry a nebo nástroje provádějící složité výpočetní operace. Technologie jako je například již zmíněný Phonegap však sjednocuj API mnoha mobilních zařízeních a systémů a jedna aplikace napsaná v HTML a JavaScriptu se tak může použít na všech takto podporovaných zařízeních. Nejen to, ale také možňost vytvořit jednu společnou aplikaci pro mobily a pro web výrazně urychlí vývoj a sníží náklady na hromadu programátorů, kteří by jinak tvořili samostatné nativní aplikace na každé požadované prostředí. Angular.JS se tedy používá také jako první otestování bussines plánu, kdy se rychle vytvoří prvotní verze aplikace a až podle úspěchu a uživatelského zájmu se začne pracovat na nativních aplikacích. Z těchto a mnoha dalších důvodů jsem se rohodl v mé práci použít právě angular.JS. Spolu s dalšími technologiemi tvoří právě již zmíněný MEAN stack, což je čtveřice v současnosti hojně používaných technologií pro tvorbu startup aplikací.

2.3.2 Nginx

Nginx je další z použitých open source technologií, jejíž jedno možné použití je nasazení jako web server zajišťující vysoký výkon. V naší práci jej však použijeme jako Load balancer, který bude sloužit jako prostředník mezi uživatelem a backendem realizovaným pomocí node.js. Nejedná se o jediný možný použitelný Load balancer, je však jeden z nejznámějších, a proto ho použijeme i v našem případě. V základní verzi naší aplikace bude sloužit pouze jako proxy čekající na HTTP spojení na portu 80. Traffic jdoucí na tento port bude přesměrovávat na konkrétní zadaný port používaný naší aplikací. V reálném nasazení se nginx používá také pro load balancing, kdy se backend aplikací spustí více, popřípad se v aplikacích použije ještě clustering, kdy se se node.js aplikace rozprostře na více jader a těchto aplikací se spustí více na více počítačích a pomocí load balancingu se pak rozhazují jednotlivé požadavky mezi tyto nody. To nám zajistí vysoký výkon naší aplikace a především také vysokou horizontální škálovatelnost, kdy pro zvýšení výkonu nám

stačí pouze přidat další server a zapojit ho do load balancingu. O možnostech škálování si budeme více povídat v kapitole Škálování.

2.3.3 Node.JS

Node.JS je jednou z klíčových technologií v této práci, jedná se o nástroj umožňující stavět multiplatformní škálovatelné aplikace běžící na systému Windows, Mac OS X i Linux. Jádro zde představuje V8 JavaScript engine vytvořený společností Google, který byl použit mimo jiné i v prohlížeči Chrome pro zpracování JavaScriptu. V roce 2009 byl tento interpretr vzat a spolu s pár dalšími knihovnami nasazen na serveru, čímž se vytvořilo prostředí pro spuštění javascriptového kódu jako server-side aplikace. Kromě V8 enginu se node skládá z knihoven, které jsou napsány buď v jazyce C a C++ a nebo přímo v JavaScriptu. Psaní programů v Node.JS je téměř srovnatelná s tvorbou JavaScript aplikací v prohlížeči. Rozdílem je zde, že byla odebrána podpora pro práci s DOM modely. Přidáno pak bylo rozhraní pro práci se síťovou komunikací a především také funkcionalita na snadné vytvoření TCP/IP nebo přímo HTTP serveru. Jednou ze základních principů nejenom nodu, ale i celého javascriptu je orientace na eventy (event driven programming), kdy se spíše než používání návratových hodnot z funkcí pomocí klíčového slova return používají callback funkce. Vše můžeme uvést na příkladu se čtením ze souboru, kdy místo toho, abychom zkoumali návratovou hodnotu z čtecí funkce, pošleme spolu s příkazem na čtení také speciální funkci (callback funkce), která se zavolá až dojde k načtení dat. Pomocí vlastnosti neblokujících i/o operací (nonblocking i/o) se v nodu zpracování nezastaví a nečeká se na načtení dat. Místo toho se pokračuje dál a zatím se provádí další operace. Při načtení dat se teprve zavolá callback funkce, která teprve data obslouží. Právě I/O operace zabírají nejvíce času a node.js svým neblokujícím zpracováním toto zbytečné čekání odstranil. Tyto vlastnosti a fakt, že je z velké části psán v jazyce C/C++ mu dávají ohromnou rychlost a výkonnostní náskok před jeho ekvivalenty. Mezi velké přednosti Nodu patří i jeho modulárnost, kdy se celá aplikace rozděluje do modulů, které se až poté spojují do funkčního celku. Těchto modulů je na světě ohromné množství, z nichž téměř 70 tisíc bylo uvolněno samotnými uživateli také pro veřejnost. Pokud pak někdo tvoří aplikaci například s uživatelským přihlášením, může se nejprve podívat, zda již někdo nevytvořil modul, který by tuto funkčnost zajišťoval. Pro snadné

hledání a indexaci vytvořených modulů vznikla také stránka www.npmjs.org, která umožňuje rychlé hledání ve veřejných modulech.

2.3.4 Express.JS

2.3.5 Redis

Redis je jedna z klíčových složek škálovatelnosti Node.JS aplikací. Jejím hlavním cílem je sloužit jako malá a jednoduchá, ale zároveň i velmi rychlá databáze slučující data z více serverů. Do této databáze se tak nejčastěji ukládají "cache", které server používá, aby odlehčil a zmírnil přílišné dotazování se do databáze. Jako dalším častým příkladem dat ukládaných do Redisu jsou uživatelské sessions, což řeší problém, kdy se například při rozhazování zátěže pomocí load balancingu přepošlou data s uživatelskou registrací na jeden server, zde dojde k přihlášení, ale na druhém serveru již uživatel není znám. Pokud by pak load balancer přesměroval uživatelův požadavek na jiný server, z důvodu absence session by byl pak považován za nepřihlášeného a jeho požadavek by mohl být zamítnut. Takto mají všechny aplikační servery společné úložiště, které zajišťuje, že pokud se uživatel přihlásí na jednom serveru, bude přihlášen i na všech ostatních.

2.3.6 MongoDB

2.3.7 Socket.IO

2.4 Bezpečnost

2.5 Škálovatelnost

Kapitola 3

Realizace

3.1 Klient

3.2 Broker

3.2.1 Broker robot

3.3 Burza

3.3.1 OrderMatching

Kapitola 4

Testování

4.1 Testování kódu

4.1.1 Nastavení unit testů

4.1.2 Unit testy

4.2 Zátěžové testy

4.2.1 Testované prostředí

na jakém stroji probíhá testování

4.2.2 Testovací scénář

jaké jsou testovaní scénáře

4.2.2.1 Scénář 1 - rychlost odezvy

text a grafy

4.2.2.2 Scénář 1 - rychlost odezvy

text a grafy

4.2.2.3 Scénář 1 - rychlost odezvy

text a grafy

4.2.2.4 Scénář 1 - rychlost odezvy

text a grafy

4.2.2.5 Scénář 1 - rychlost odezvy

text a grafy

4.2.3 Výsledky měření

4.2.4 Závěr měření

Kapitola 5

Deployment

5.1 Správa kódu

5.2 Nasazení na serveru

5.2.1 Instalace závislostí

5.2.2 Stažení zdrojového kódu

5.2.3 Nastavení prostředí

5.2.4 Spuštění

Kapitola 6

Monitoring

6.1 Monitorování serveru

6.2 Debugging běžícího serveru

Kapitola 7

Škálování

Závěr

Práce žádný závěr nemá...

Příloha **A**

Seznam použitých zkratek

GUI Graphical user interface

XML Extensible markup language

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS