ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Kulečníkové počítadlo
Student:	Jan Kotrlík
Vedoucí:	Ing. Jan Kubr
Studijní program:	Informatika
Studijní obor:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2015/16

Pokyny pro vypracování

Seznamte se s počítáním bodů a statistik v různých typech kulečníkových her. Navrhněte a implementujte distribuovanou aplikaci usnadňující počítání bodů v kulečníku. Aplikace bude mít serverovou část a klienty. Serverová část umožní vytvářet statistiky. Klientské části umožní zadávat body a sledovat průběžné výsledky. Klientská část musí umožnit dávkový (offline) provoz. Server bude implementován na OS Linux. Klient bude implementován na platformě Android a PC s OS Linux nebo Windows. PC klient může být nahrazen webovým rozhraním. Součástí návrhu bude návrh komunikačního protokolu a výsledky testování.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof.Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 12. ledna 2015

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Kulečníkové počítadlo

Jan Kotrlík

Vedoucí práce: Ing. Jan Kubr

1. července 2016

Poděkování

Chtěl bych poděkovat svému vedoucímu práce, panu Ing. Janu Kubrovi, za neutuchající optimismus a cenné rady. Dále bych chtěl poděkovat své rodině a svým přátelům, kteří při mně vždy stáli a byli mi oporou.

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 zpracováním 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 1. července 2016 .....

Odkaz na tuto práci

Kotrlík, Jan. *Kulečníkové počítadlo*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Tato bakalářská práce má za cíl navrhnout a implementovat distribuovanou aplikaci pro sledování a zaznamenávání kulečníkových výsledků. Bude provedena analýza trhu na již existující řešení a navrhne a naimplementuje se aplikace pro Android a její desktopový ekvivalent, respektive webové rozhraní. Tato práce by měla mířit na pohodlné užívání jejích výsledných produktů a nabídnout hráčům kulečníku nové možnosti.

Klíčová slova Java, Android, server-side, backend, klient, PHP, postgresql, MVC, API, REST, statistika, kulečník, UML

Abstract

This bachelor thesis has set goal of designing and implementing of distributed application for watching results and keeping track of various billiard games. An analysis of the current market will take place for already existing solutions and the will be design and implementation of Android application and it's desktop equivalent, resprectively. This project should aim for comfortable usage of it's products and it should offer new options to billiard players.

Keywords Java, Android, server-side, client, backend, PHP, postgresql, MVC, API, REST, statistics, billiard, UML

Obsah

Úvod **3**

1 Analýza **5**
1.1 Analýza existujích řešení 5
1.2 Analýza potenciální uživatelské skupiny 6
1.3 Analýza procesů 6
1.4 Funkční požadavky 8
1.5 Nefunkční požadavky 9
1.6 Model případů užití 10

2 Návrh **13**
2.1 Architektura 13
2.2 Návrh komunikačního protokolu 14
2.3 Server 15
2.4 Klient 15
2.5 Bezpečnost 16
2.6 Databázový model 16

3 Implementace a testování **17**
3.1 Nasazení 17
3.2 Implementace 17

Závěr **25**

Literatura **27**

A Seznam použitých zkratek **29**

B Obsah přiloženého CD **31**

Seznam obrázků

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Kulečníkové počítadlo
Student:	Jan Kotrlík
Vedoucí:	Ing. Jan Kubr
Studijní program:	Informatika
Studijní obor:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2015/16

Pokyny pro vypracování

Seznamte se s počítáním bodů a statistik v různých typech kulečníkových her. Navrhněte a implementujte distribuovanou aplikaci usnadňující počítání bodů v kulečníku. Aplikace bude mít serverovou část a klienty. Serverová část umožní vytvářet statistiky. Klientské části umožní zadávat body a sledovat průběžné výsledky. Klientská část musí umožnit dávkový (offline) provoz. Server bude implementován na OS Linux. Klient bude implementován na platformě Android a PC s OS Linux nebo Windows. PC klient může být nahrazen webovým rozhraním. Součástí návrhu bude návrh komunikačního protokolu a výsledky testování.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof.Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 12. ledna 2015

Úvod

Každý z nás někdy hrál, nebo alespoň zkusil partii kulečníku. Biliardový stůl v polozakouřené místnosti vždycky vyvolával zvláštní atmosféru. Napadlo nás ale někdy - hrajeme to vlastně vůbec dobře? Znám někoho, kdo by hrál s vášní? A dolehla na něj povznášející atmosféra modernizace, virtualizace a minimalizace všeho, co se nebrání zuby nehty? I na tyto otázky se moje bakalářská práce pokusí odpovědět. Nejprve provedu průzkum trhu a zjistím, zda si někdo z IT světa gentlemanů s tágy všiml a zda se moderní doba otiskla i na jejich koníčku. Poté provedu návrh na ditribuovanou aplikaci pro sledování výsledků různých kulečníkových her, seznámým se s jejich hodnocením a statistikami a poté implementuji aplikaci pro Android a OS Linux, nebo webový portál se serverovou stranou běžící na OS Linux, která je bude propojovat. Nakonec tento ekosystém vezmu do světa a vyzkouším ho v reálných podmínách, s reálnými lidmi, reálnými hráči. Přínosem mé práce bude stmelení kulečníkářské komunity a její obohacení o možnosti, které mnou implementovaný systém přinese.

Kapitola 1

Analýza

Tato kapitola se věnuje analýze již existujících řešení, požadavků na systém a samotné systémové analýze. Ta je reprezentována UML (Unified Modeling Language - unifikovaným modelovacím jazykem). Ten je mezi vývojáři velmi oblíbený, protože pomocí něj lze jasným a přehledným způsobem popsat i obsáhlé a robustní systémy. Svou oblíbenost si jistě vysloužil, jak jeho název napovídá, svou unifikací, čili sjednocením nejefektivnějších a nejúspěšnějších principů při návrhu softwarových systémů. Developeři nemusí mluvit stejným jazykem, stačí když „hovoří v UML“. V nepřeberném množství specifikací a standardů se firmě Rational Software v 90. letech podařilo vytvořit jednoduchý, přehledný a standard, o který se díky své popularitě začalo starat konsorcium největších hráčů na poli informačních technologií a ti mu tak zajišťují pokračující prvenství ve světě developerských návrhů.[1]

1.1 Analýza existujích řešení

Pokud se rozhodneme pustit se do něčeho nového, zřejmě první činem většiny z nás bude prozkoumání již existujících řešení, ať už za účelem získání inspirace, vyvarování se jejich předchozích chyb, nebo pro posouzení, zda má naše počínání vůbec smysl. Průzkum trhu jsem provedl při zadání práce, na začátku roku 2015 a tento rok na jaře a musím konstatovat, že pozice pro systém sledující výsledky kulečníkových her a usnadňující jejich intepretaci se výrazně zlepšila. Před rokem existovala aplikace pro iOS, která vyhodnocovala výsledky kulečníkové hry Karambol, v roce 2016 se mi ji nicméně již nepodařilo dohledat. Co se týče her jako takových, najde se značné množství různých kulečníkových her jako pro mobilní, tak webové a desktopové platformy. Systém sledující pouze výsledky a vyhodnocování výsledků jako takový zatím chybí, čili se nacházíme ve slušné výchozí poizi.

1.2 Analýza potenciální uživatelské skupiny

Po průzkumu trhu je vhodný čas položit si dotaz, kdo je naše cílová skupina. Z podstaty věci by systém měl být stavěný na míru oficiálním pravidlům kulečníkových her. Toto omezení se při průzkumu vzorku obyvatelstva sestaveného především ze studentů a IT odborníků ovšem nejeví jako nejšťastnější. Z tohoto vzorku všichni některou kulečníkovou hru hráli nebo hrají pravidelně, ale její pravidla se oficiálním vzdaluje na míle daleko. Drtivá většina z nich hrála nebo hraje osmičku, ovšem v tradiční české pohostinské úpravě, která nepřístojným způsobem znevýhodňuje hráče, krerý má mít výhodu faulu po utopení bíle. Podle oficiálních pravidel má v případě, že protihráč utopil bílou právo opravy polohy bíle po celém hracím poli, a ne jen po přímce kolmé na délku stolu, procházející bodem, ze kterého se bílá začíná hrát na začátku hry. Vzhledem k tomuto aspektu je vhodné implementovat řešení jako pro oficiální hráčskou komunitu, tak pro rekreační hráče. Jak moc lukrativní pro ně bude možnost vést si statistiky pro své soukromé účely ukáže až čas. Z této pozice se tedy jako nadějnější skupina jeví oficiální hráčská komunita. Pokud se ovšem proklikáme jak českými[2], tak celosvětovými[3] asociacemi, které se zabývají pořádání turnajů, nebo i celosvětovou asociací[4] zabývající se pouze osmičkou, zjistíme, že jediné statistiky, které se u kulečníkových her sledují a evidují jsou body za vyhrané hry na turnajích a vyhrané turnaje samotné. Zde se tedy naskýtá prostor pro profesinální hráče, kterým nestačí statistiky o výhrách a prohrách, ale možná by ocenili i něco detailnějšího.

1.3 Analýza procesů

Modelování procesů je nezbytou součástí návrhu aplikace. Je důležité, aby vývojář správně pochopil byznys logiku aplikace. Je to jediný způsob, jak se v budoucnu vyhnout nucené refaktorizaci kódu, protože implementované řešení neodpovídá původním představám zákazníka. V mojí roli je zákazník zatím pouze virtuální, takže zodpovědnost za ideu aplikace přebírám já, což je dvousečný meč. Na jedné straně si mohu napsat aplikaci přesně tak, jak mi to přijde nejlepší, což je obrovská vývojářská výhoda. Na straně druhé ovšem stojí uživatel, který má nějaké své preference a ideje a pokud se implementace produktu neprotne alespoň s nějakou množinou těchto představ, aplikace pozbývá smyslu. Každý uživatel je sice jiný, ale vzhledem k tomu, že patří do stejné zájmové skupiny, lze očekávat, že budou mít na věc podobný názor. Potom celá vývojářská svoboda ztrácí na smyslu.

Procesy se UML diagramech nejlépe zakreslují pomocí diagramu aktivit, kde jde každý proces namodelovat jako sérii aktivit. Diagram musí obsahovat všechny klíčové informace, aby tak bezpečně zachytil celý proces a při jeho následné implementaci nemohlo dojít k nepodloženým rozhodnutím. Diagram ovšem není případ užití, musí tedy zachovat příslušnou obecnost.

1.3.1 Registrace

Aby uživatel plně využil svou novou aplikaci, je nezbytné, aby se registroval. Neregistrovaný uživatel může hrát pouze lokální hry, nemůže se stát hostem pro ostatní hráče a nemá žádný přístup ke statistikám, které si sice může ukládat, ale nejsou mu nic platné. Uživatel spustí aplikaci a hned v úvodu ho čeká rozhodnutí, zda bude dále pouze lokálním nepřipojeným hráčem, nebo se registruje. Pokud se rozhodne pro registraci, je prezentován formulářem, kam může zadat jiný než defaultní server, vyplní uživatelské jméno, email pro validaci, dvakrát zadá stejné heslo a klikne registrovat. Jakmile backend požadavek zpracuje a ověří, že již v systému email ani uživatelské jméno není a nedochází tedy ke kolizi, oznámí aplikace uživateli, že mu na do schránky přišel validovací email, aby se skutečně ověřila identita uživatele. Jakmile uživatel zaslaný link navštíví, backend server jej označí jako aktivního a uživatel se může přihlásit.

1.3.2 Hotswap mód

Není nutné, aby všichni uživatelé měli aplikaci nainstalovanou na svých zařízeních a byli všichni registrovanými hráči. Aplikace podporuje hotswap mód, což znamená že všichni uživatelé hrají na jednom zařízení. Uživatel zvolí aktivitu „Hraj hru“. Pokud není přihášený, aplikace ho vyzve k vyplnění lokálního uživatelského jména. Pokud je přihlášený, je prezentován s nabídkou zvolení počtu spoluhráčů. Implementovaná hra osmička má pouze dva hráče, ale existují kulečníkové hry s možností vytvářet tými a nebo hrát všichni proti sobě. Po zadání počtu hráču je každý hráč vyzván, aby se přihásil, nebo si zvolil lokální uživatelské jméno. Pokud se hráč přihlásí, je přihlášen pouze po dobu této jedné hry a nemusí tak mít obavu, že by jeho přihlášení někdo zneužil. Naopak lokální hráči si své statistiky budou moci vyzvednout pouze v případě, že je majitel zařízení nechá na krátkou dobu přihlásit nechat je své statistiky synchronizovat. Hry se ovšem automaticky po ukončení nesynchronizují, každý hráč přihášený je vyzván, aby se rozhodl, zda hru do statistik zahrne, či nikoliv. Toto je svým zplsobem obrana před hráči, kteří by úmyslně znepříjemňovali hru a nehráli fér (byť by to byli přátelé majitele zažízení), ale zároveň pokušení pro každého uživatele nezahrnovat do statistik hry odehrané špatně, nebo dokonce prohrané. Zde spoléhám na chuť uživatelů po skutečné statistice.

1.3.3 Online hra

Pokud je uživatel přihlášen, může chtít hostovat hru ostatních uživatelů. Hostovat je zde v tom smyslu, že on učiní první krok ke hře a zviditelní se jako host. Ostatní pak vytvoří hru tím, že dají aplikace na vědomí, že chtějí začít hrát hru s oním iniciátorem. Samozřejmě se u toho předpokládá, že všichni

hráči budou kolem jednoho kulečníkového stolu, protože z logiky věci nelze hrát odděleně.

1.3.4 Trénink

V tomto případě uživatli stačí, když druhého hráče (pokud nehraje kulečníkovou hru určenou pro jednoho hráče) založí pouze virtuálně jako lokálního hráče. Na konci hry může doslova zahodit druhý výsledek a nebo je oba přidat ke svým statistikám, jako dvě samostatné hry.

1.3.5 Spravování starších, lokálně uložených her

Přihlášený uživatel může v aplikace navigovat do spravování anonymních her, kde pro každou hru ze seznamu rozhodne, zda si tuto hru chce přidat do statistik, zda ji chce zahodit a nebo, pokud je to hra jiného uživatele, který jí hrál lokálně offline, ji tomuto hráči nabídnout k přiřazení do statistik.

1.4 Funkční požadavky

Doslova požadavky na funkce. Tyto specifikuje zákazník na naprosto abstraktní úrovni. Není důležité, jak bude dosaženo výsledku z pohledu technologie, implementace nebo nasazení. Důležité je, že vytvořený systém bude splňovat každý z funkčních požadavků. Nesprávně nebo neúplně specifikované funkční požadavky jsou často kámenem úrazu celých systémů, kdy se nějaká vlastnost systému vyvine podle jiného nebo jinak myšleného požadavku, než jaký měl klient a to vede k navýšení nákladů na vývoj, kterým se mohlo předejít.

1.4.1 Uživatelské účty

Existují tři druhy uživatelských účtů:

- administrátorský účet, který není určený pro hraní, ale pro správu uživatelů,
- aktivní uživatelský účet, který může využívat všech možností, které aplikace poskytuje,
- neaktivní uživatelský účet. Tento může být neaktivní, protože ještě nebyl autorizován přes emai, protože bylo resetováno heslo a uživatel jej ještě neobnovil a nezměnil a nebo účet zablokovaný od adminitrátora, a
- lokální anonymní hráč, který se může přihlásit nebo registrovat.

Měnění uživatelských rolí je monžé, ale v praxi se moc neobjevuje. Uživatelé obvykle nezapomínají svá hesla dvakrát týdně, adminy se nestávají takřka

nikdy a když už administrátor udělí ban, není moc velká naděje, že si přes noc rozmyslí.

Stav uživatele Uživatel je buď přihášený, nebo lokální anonymní hráč. Ten se může přihlásit. Aplikace si pamatuje, že hráč je přihlášen a při obnovení aplikace jej přihlásí znovu.

1.4.2 Hry

Každou hru může hrát každý aktivní nebo lokální anonymní hráč. Aktivní hráč může být i Online hráč, to znamená hrající z jiného zařízení. Po dohrání hry jí přihlášený hráč může smazat, zahrnout do statistik a pokud to není jeho hra, ale anonymího lokálního uživatele, může ji poslat jinému přihlášenému uživateli k zahrnutí do statistik. Lokální anonymní hráč může hru smazat, nebo ji nechat uložit na disk. V tu chvíli k ní ztrácí přístup. Přihlášený uživatel má možnost ve speciálním módu prohlížet odehrané hry.

1.4.3 Uložené hry

Uložená hra může být zahrnuta do statistik lokálního přihlášeného hráče, smazána a nebo odeslána k zahrnutí jinému přihlášenému hráči.

1.4.4 Statistiky

Přihlášený hráč má právo přidávat do statistik nové hry a sledovat je

1.5 Nefunkční požadavky

Můžeme je chápat jako omezení na aplikaci, nebo její formování do žádaného stavu. Obvykle mívají rozhodující vliv na architekturu. Stejně jako funkční požadavky definovány abstraktně.

1.5.1 Bezpečnost

Pokud bude mít aplikace úspěch, může její databáze obsahovat až desítky tisíc uživatelských emailů a hesel.

1.5.2 Výkon versus cena

Protože si musíme uživatelskou základnu vybojovat, nemá cenu hned od začátku stavět hardwarově náročné řešení. Stačí takové, které zvládne až 100 registrovaných uživatelů.

1.5.3 Rozšiřitelnost

Aby aplikace vzkvétala, je vhodné implenetovat ji tak, aby byla snad co možná nejjednodušeji rozšiřitelná. Nutnost je snadná rozšiřitelnost funkčnosti, ideální je možnost rozšířit o dosud nepodporované platformy.

1.5.4 Požadavky na architekturu

na architekturu je požadavek pouze na užití principu klient-server. Architektuře se budu více věnovat ve druhé kapitole - návrhu.

1.5.5 Požadavky na dostupnost

Pro začátek by aplikace měla být dostupná na Android a desktop a to na OS Linux nebo Windows, případně nahrazení formou webového rozhraní

1.5.6 Požadavky na technologie

Na technologie nebyly vzneseny žádné požadavky.

1.5.7 Požadavky na komunikaci

Na komunikaci byl vznesen požadavek na návrh komunikačního protokolu mezi klientem a serverem.

1.6 Model případů užití

Tvorbou tohotu modelu jistý způsobem dokumentujeme požadavky očima uživatele. Tento uživatel může být i sám systém. Postihují pouze funkční požadavky a popisuje interakci mezi uživateli.

1.6.1 Účastníci

Jak vyplývá s předchozího textu, aplikaci mohou využívat jak přihlášení tak nepřihlášení účastníci. Přihlášení účastníci mohou být administrátory.

1.6.2 Případy užití

Případy pokrývají funkce, které systém poskytuje svým účastníkům.

1.6.2.1 Vstup do aplikace

Tento případ zachycuje, co může uživatel dělat, když spustí aplikaci v závislosti na tom, jestli je přihášený, nebo ne.

Přihlásit se - anonymní užiatel se může přihlásit.

Registrovat se - anonymní uživatel se může registrovat.

Hrát offline - uživatel může používat aplikaci, aniž by se musel přihlašovat, nebo odhlašovat.

Vybrat aktivitu - tato akce je společná pro oba uživatele, jenom se jinak jmenuje.

Odhlásit se - uživatel se může odhlásit.

1.6.2.2 Výběr aktivity

Začít hrát - přejít k hraní samotnému.

Ukázat statistiku - hlavní důvod existence aplikace - statistika.

Přiřadit anonymní hry - rozhodnout, co s hrami, které momentálně nepatří nikomu.

1.6.2.3 Rozhodování o lokálně uložené hře

Přidat do statistik - přidá hru do statistik.

Smazat hru - smaže hru.

Poslat hru - nabídnout hru na sdílení.

1.6.2.4 Výběr aktivity při rozstřelu osmičky

Otevřít stůl - úspěšný náběh.

Faul - opakovat náběh - soupeř se rozhodl nechat hráče opakovat neúspěšný náběh.

Faul - předat náběh - soupeř chce začít sám.

1.6.3 Doménový model

Doménový model slouží jako virtuální popis tříd. Je platformově nezávislý. Třídy v doménovém modelu jsou značně zjednodušené, nemají metody, pouze důležité atributy. Ty někdy do konce nemají ani datové typy. Protože se pohybujeme na extrémně abstraktní úrovni, můžeme si dovolit benefit nadhledu.

Authorization Třída obsahující registrační náležitosti nutné k úspěšnému přihlášení.

Game Třída představuje jednotlivé implementované hry.

Move Tato třída obstarává jeden celý tah hry, a když je na konci, je poslána případnému online hráči, kde jí applikace začlení to hry na jeho zařízení a umožní mu pokračovat s novými daty.

Statistics Třída obstarávacjící statistiku.

Player Třda obsahující hráče a jeho přihlašovací údaje.

Teams Třída spravující teamy.

Network Třída starající se o síťování.

Registration Třída vyřizující registrace.

Core Třífa řídící celý backend

Model Třída starající se o přístup k databázi

View Třída, která vytváří GUI pro uživatele webového rozhraní

Kapitola 2

Návrh

Tato kapitola se zaměřuje na již konkrétní principy, technologie a metody. Jejím cílem je přesně určit implementační detaily a závislosti, které nakladly otázky během analýzy. Během návrhu se rozhoduje o strategicky důležitých věcech, jako jsou principy programování, technologie, výběr architektury a tak podobně. V ideálním případě se z tohoto návrhu vyjde při implementaci a ta se podle něj zrealizuje, což bude známka toho, že to bylo dobrý návrh.

2.1 Architektura

Návrh architektury je největší rozhodnutí, které musí vývojář během plánování udělat. Pokud zvolí špatně, nepomůže nic. Jednou z nefunkčních podmínek byla i architektura, a to klient-server. Požadavek na archietkturu není úplně nejlepší podmínka, ale v tomto požadavku si návrh doslova říká o API (Application Programming Interface) společně s REST (Representational State Transfer). V následující kapitole budou rozvedeny principy, technologie a budou učiněna rozhodnutí,která budou mít na aplikaci nezanedbatelný vliv. Před pár lety byl a pořád ještě je oblíbený webový návrhový vzor MVC (Model, View, Controller). Zvlášť výhodný je, pokud je klient pouze webová stránka dodávají obsah a reagující na uživatele. Logika tohoto návrhu spočívá v tom, že je rozdělena do 3 logických vrstev. Model je modelem přístupu k datům, což může být třeba ORM, nebo přímo SQL příkazy volané pomocí dedikované třídy a ovladače. View má na startosti pouze zobrazování obsahu uživateli a přenesení jeho interakce ke Controlleru, který obstarává veškerou byznys logiku aplikace. V mnou navržené architektuře je tento model zachován a trochu upraven dle potřeb systému. Serverová část je prakticky téměř jen rozhraní pro dostávání dat z databáze, ale i tam musí být nějaká ta logika. A v zájmu bezpečnosti je jí tam docela dost. Naproti tomu klient se o data nestará, a když, tak o malé lokální soubory. Visí na něm ovšem tíha obstarávat celé View, tedy interakci s uživatelem a většina byznys logiky aplikace se ode-

hrává tam. Ona koneckonců musí, protože jeden z požadavků je offline provoz, což by jen s View nemělo naprosto žádný efekt.

2.1.1 Klient - server model

Pod tímto označením se vždy ukrývá dvojice (minimálně) strojů, která mezi sebou neustále (ideálně v reálném čase) komunikuje a vyměňuje si data. V dřívějších dobách, kdy nebyl internet a sítě obecně tak dostupné se vyvíjeli jednouzlové systémy. V jedné aplikaci byla veškerá logika, všetně databáze a všech dat, na jednom stroji. S nástupem internetu a webových technologií se začalo přicházet na to, že efektnější je mít společné uživatelské rozhraní (GUI - Graphical User Interface) pro všechny veřejně dostupné a data oddělit od zbytku autentizací a silnou bezpečnostní politikou.

2.2 Návrh komunikačního protokolu

Jak již bylo řečeno, pro architektonický model klient-server je vhodné použít REST API. Toto je založeno na HTTP (HTTPS v případě zabezpečeného spojení), kdy klient pomocí 4 typů dotazů komunikuje se serverem a získává od něj, nebo mu zasílá data. Tento princip se schovává pro akronymem CRUD, neboli Create, Read, Update, Delete, respektive typy HTTP dotazů POST, GET, PUT, DELETE. Pomocí vhodně stanovených pravidel se z této čtveřice stává mocný komunikační nástroj, který má masové uplatnění ve webových technologiích, a toto místo si drží už od devasátých let a zatím to nevypadá, že by se tato pozice otřásla. Základní návrh komunikačního protokolu tedy máme - data můžeme ze serveru dostat (GET - READ), poslat je tam (CREATE - POST), zaktualizovat (UPDATE - PUT) a nebo i zažádat o smazání(DELETE - DELETE). Je tohle už plnohodnotný komunikační protokol?

Není. K akcím, které ovlivňují data ještě potřebujeme vědět, která data chceme ovlivnit. I to se stará REST. Přístup ke správným datům zajišťuje URI (Uniform Based Resource). Společně tyto dvě věci zajišťují spolehlivý komunikační protokol pro procesy jakéhokoliv druhu. Z tohoto důvodu je využívá i Kulečníkové počítadlo. Logika URI přístupu je implementována na serveru a klienti tak při potřebě přistoupit k datům volají API s příslušnou URI. Server navíc vždy odpovídá nejen číselným kódem, jak je zvykem (např 200 ok), ale automatiky zasílá s odpovědí ještě další data, která slouží buď k indikaci problému, nebo jsou v něm data. Jako komunikační formát jsem zvolil JSON (Javascript Object Notation), protože je přímočarý, rozšířený a prakticky každý jazyk už s ním dnes umí slušně pracovat, takže si člověk ušetří spoustu práce s parsováním a laděním něčeho vlastního.

2.3 Server

2.3.1 Architektura

Jak již tedy bylo řečeno, server je ta část klasického MVC, kde se pracuje s daty a pouští se do hry trochu logiky. Z architektonického hlediska vlastně potřebuje přístup do databáze, nějaký interface se kterým by mohl pracovat a Controller. Tohle všechno zajistí PHP s Apachem a postgresql databází. Za pár desítek korun měsíčně tak vyjde stroj, který uživatelů nezvládne 100, ale 10000.

2.3.2 Technologie

Jazyků pro webově orientované služby je přehršel. Díky požadavku výkon versus cena je vhodné použít něco méně náročného, co utáhnou cenově poměrně optimální virtuální servery s kapacitou RAM 1GB. V případě použití například Javy a jejích Netbeanů bychom na jedno API takové virtály potřebovali 4. Node.js se jako javascriptový backend framework momentálně hodně tlačí dopředu, ale pro naše účely je zbytečně robustní. Moje volba padla na PHP. Je intuitivní, Rychle si ho člověk osahá a jednoduše se s ním pracuje.

2.4 Klient

2.4.1 Architektura

Klient architektonicky dokresluje MVC. Controller na serveru stále zprostředkovává komunikaci s Modelem (taky by bylo zvláštní, kdyby klient bral data přímo od Modelu). To se týká androidu. Webové rozhraní je standalone aplikace, která má svou vlastní vnitřní logiku a server tak využívá pouze jako interface pro databázi a autorizaci uživatelů, stejně jako Android. Server je tudíž úplně odstíněn od byznys logiky systému a uplatňuje se tak asi jako chytřejší interface k databázi.

2.4.2 Technologie

Ač by otázka technologie pro Android mnohé zarazila, je docela logická. Už nějakou dobu exitují překladače, které z jazyka jiného než java vyrobí spustitelý kód pro Android nebo iOS. Node.js v tomto poli zřejmě hraje prim. Existuje na něj například Cordova, která byla zakoupena Adobe a která umí z javascriptu udělat spustitelný kód. Pak je tu nový horký kandidát od Facebooku React-Native, který se v současné době veze na vlně znovuobjeveného funkcionálnáho programmování. Nicméně jsem se stálo rozhodl pro javu a android a jQuery pro web. Ve hře byla i java ve swingu s tín, že by spolu s adnroidím kódem sdíleli skrz stejný interface byznys logiku aplikace a UI prvky si každý ovládal sám.

2.5 Bezpečnost

Vzhledem ukládání citlivých údajů (a email, uživatelské jméno a nějaká forma hesla už pro mě osobně citlivé údaje jsou) má bezpečnost nejvyšší prioritu. Klientské aplikace sice benevolentně dovolují komunikaci po nešifrovaném HTTP protokolu, ale je to silně nedoporučeno. Co se bezpečnosti týče, rozhodl jsem se i další citlivé údaje (token získaný úspěšnou autentifikací) posílat buď ve hormě hlaviček a tak například z logů nejde zjistit, jaké má ten který uživatel přihlašovací údaje/heslo/aktivní token a podobně, jak tomu jde u jiných služeb, Server poslouchá pouze na portu 443, což znamená že přístup z prohlížeče bez „https://“ v URL skončí chybou - nebude automaticky přesměrován. Na serveru je aktivní fail2ban démon, který na úrovni IP tables zakáže každou IP, která se pokusí připojit bez správného klíče. Svou bezpečnost bereme opravdu vážně. Hesla, jež uživatel zadává např. do Android aplikace jsou okamžitě hashovaná MD5 (MessageDigest5). Ta je sice dávno prolomená, ale efekt to má. Mimo zařízení jde (alespoň z mnou provozovaného serveru) pouze bezpečná šifrovaná komunikace

2.6 Databázový model

Kapitola 3

Implementace a testování

Po zevrubné analýze a návrhu se přistoupilo k implementaci jednotlivých částí. Z možných her jsem si jako první vybral osmičku s tím, že další budu postupem času dodávat. V dalších kapitolách o implementaci ukáži některé zajímavé kusy kódu.

3.1 Nasazení

Server momentálně běží na mnou předplacené doméně carambolcounter.eu. Běží přesně na naplánované konfiguraci. Android apk soubor se nachází na přiloženém CD a je podepsaný mým osobním certifikátem. Kdyby se aplikace jednou dostala na GooglePlay, potěšilo by to nejednoho hráče kulečníku. Server provoz zatím zvládá (také o něm nikdo neví), takže ještě nebyl vyzkoušen pod reálnou zátěží.

3.2 Implementace

Následují zajímavé ukázky kódu ze serveru a z Androidu

3.2.1 Implementace serveru - ukázka

```
<?php
class Core
{
    protected $username;
    protected $userId;
    protected $token;
    protected $password;
    protected $remoteIP;
    protected $request;
    protected $args;
```

```
protected $dbConn;
protected $requestType;
protected $clasName;
protected $coreArray;
protected $data;
protected $rawData;
public function __construct()
{
    if (array_key_exists('HTTP_USERNAME', $_SERVER)) {
        $this->username = $_SERVER['HTTP_USERNAME'];
    } else $this->username = "";
    if (array_key_exists('HTTP_TOKEN', $_SERVER)) {
        $this->token = $_SERVER['HTTP_TOKEN'];
    } else $this->token = "";
    if (array_key_exists('REMOTE_ADDR', $_SERVER)) {
        $this->remoteIP = $_SERVER['REMOTE_ADDR'];
    } else $this->remoteIP = "";
    if (array_key_exists('REQUEST_METHOD', $_SERVER)) {
        $this->requestType = $_SERVER['REQUEST_METHOD'];
    } else {
        throw new Exception("No request method found");
    }
    if (array_key_exists('HTTP_USERID', $_SERVER)) {
        $this->userId = $_SERVER['HTTP_USERID'];
    } else $this->userId = "";
    if (array_key_exists('HTTP_PASSWORD', $_SERVER)) {
        $this->password = $_SERVER['HTTP_PASSWORD'];
    } else $this->password = "";
    $this->request = $_SERVER['REQUEST_URI'];
    $this->args = Array();
    $this->dbConn = null;
    $this->coreArray = Array();
    $this->rawData = file_get_contents("php://input");
    $this->data = SupportingFunctions::convertJSONToArr($this->rawData);
}
public function run()
{
    try {
        $this->checkCredentials();
        $this->formatUri();
        $this->getDb();
        $this->wrapToArr();
        if ( file_exists( API_PATH . $this->className . '.php')) {
            $apiCall = new $this->className($this->coreArray);
```

```
            $apiCall->processApi();
        } else {
            http_response_code(404);
            echo "Uknown api call\n";
            exit(0);
        }
    } catch (Exception $e) {
        echo json_encode(Array('status' => 'error','message' => $e->getMessage())
        exit (1);
    }

}
protected function checkCredentials()
{
    if ( $this->username != "" ) {
        $username = filter_var($this->username, FILTER_VALIDATE_EMAIL);
        if ( is_bool($username)  ) {
            throw new Exception("username in forbidden format");
        }
    }
    if ( $this->token != "" ) {
        if ( ! SupportingFunctions::checkTokenFormatAndLength($this->token, 36) )
            throw new Exception("token in forbidden format");
        }
    }
    if ( $this->userId != "" ) {
        if ( ! SupportingFunctions::checkNumerosityAndLength( $this->userId, 9) )
            throw new Exception("userid in forbidden format");
        }
    }
}
protected function formatUri()
{
    $lowrequest = strtolower($this->request);
    $trimmed = trim($lowrequest, '/');
    $formatedUri = explode('/', $trimmed );
    if ( $formatedUri[0] != 'api' ) {
        var_dump($formatedUri);
        throw new Exception("Unsuspected URI");
    }
    array_shift($formatedUri);
    if ( count($formatedUri) == 0 ) {
        echo "Welcome to CarambolCounters API :)\n";
        exit (0);
```

```
        } else {
            $this->className = $formatedUri[0];
            array_shift($formatedUri);
            if ( count($formatedUri) != 0 ) {
                $this->args = $formatedUri;
            }
        }
    }
    protected function getDb()
    {
        $this->dbConn = DBSingleton::getDbConn();
    }
    protected function wrapToArr()
    {
        $this->coreArray['username'] = $this->username;
        $this->coreArray['userId'] = $this->userId;
        $this->coreArray['token'] = $this->token;
        $this->coreArray['password'] = $this->password;
        $this->coreArray['remoteIP'] = $this->remoteIP;
        $this->coreArray['args'] = $this->args;
        $this->coreArray['dbConn'] = $this->dbConn;
        $this->coreArray['requestType'] = $this->requestType;
        $this->coreArray['data'] = $this->data;
    }
}
```

3.2.2 Implementace klienta

```
package cz.cvut.fit.kotrlja1.carambolcounter;

import android.os.AsyncTask;
import android.util.Log;
import android.util.Pair;

import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.Reader;
import java.net.HttpURLConnection;
import java.net.SocketTimeoutException;
import java.net.URL;
```

```
/**
 * Created by jkotrlik on 02/05/16.
 */
public class NetworkHandler extends AsyncTask<NetworkEncap, Pair<Integer, String>, Pa

    private static final String DEBUG_TAG = "NETWORKING";
    private static final int TIMEOUT = 5000;
    private static final int MAXINPUTLENGTH = 5000;

    public NetworkHandler() {}

    /**
     * This is Android's API function meant to work in background. As it is not pret
     * the argument to be field of some type, which might be useful in some cases, bu
     * we are forced to pickup the first (and only) element from it and the need to
     * in try catch block
     * It needs to be the same type as our networking function, as it returns the res
     * of background task, which makes it pretty flexible and here it's customized e
     * for our needs
     */
    @Override
    protected Pair<Integer, String> doInBackground(NetworkEncap... networkEncap) {
        try {
            return sendHTTPRequestAndReceiveResponse(networkEncap[0]);
        } catch (IOException e) {
            Log.d(DEBUG_TAG, e.getMessage());
        }
        return null;
    }

    /**
     * Method sendHTTPRequestAndReceiveResponse is the final network class, here the
     * happens. I've typed as pair, because this way we get both result code and resu
     * it. Thanks to that this class takes care only for networking and logic can be
     * It also doesn't contain any context, so the users are protected from getting
     * toasting announcements
     */
    private Pair<Integer, String> sendHTTPRequestAndReceiveResponse(NetworkEncap netw
            throws IOException {
        InputStream is = null;
        String serverUrl = networkEncap.getServerUrl();
        String data = networkEncap.getData();
        String method;
        /**
```

```
 * It's safer to transform method type from enum to String here. It decrease
 * possibility of manipulation to minimum. And, a bit paranoid, different en
 * "the mighty four" ends up with RuntimeException
 */
switch (networkEncap.getConnMethod()) {
    case PUT:
        method = "PUT";
        break;
    case DETETE:
        method = "DELETE";
        break;
    case POST:
        method = "POST";
        break;
    case GET:
        method = "GET";
        break;
    default:
        throw new RuntimeException("Non existing method connection");
}

try {
    URL url = new URL(serverUrl);
    /**
     * It says Http, but HTTPS is implemented. Thank God.
     */
    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
    /**
     * Timeout's in milliseconds. If it takes server more than 5 seconds to
     * 200 or error or send JSON with no more than, say, 200 lines then some
     * is definitely wrong
     */
    conn.setReadTimeout(TIMEOUT );
    conn.setConnectTimeout(TIMEOUT );
    conn.setRequestMethod(method);

    conn.setRequestProperty("Content-Type", "application/x-www-form-urlencod

    for (Pair<String, String> pairs: networkEncap.getHeaders()) {
        conn.setRequestProperty(pairs.first, pairs.second);
    }

    if ( !data.isEmpty() ) {
        conn.setDoOutput(true);
```

```
            DataOutputStream reqStream = new DataOutputStream(conn.getOutputStrea
            reqStream.writeBytes(data);
            reqStream.flush();
            reqStream.close();
        }

        conn.setDoInput(true);
        /**
         * App stars the connection. Here's where we'll wait for the timeout;
         */
        conn.connect();
        int response = conn.getResponseCode();
        Log.d(DEBUG_TAG, "The response is: " + response);
        is = conn.getInputStream();

        /**
         * static final int MAXINPUTLENGTH sets the number of characters to be r
         * processed. It would be nice to be able to set this to "UNLIMITED", bu
         * of receiving never ending streamof blobs to block the app completely
         * attractive. 5000 should be enough for whatever response our server ge
         */
        String contentAsString = readInputStream(is, MAXINPUTLENGTH);
        return Pair.create(response, contentAsString);

        /**
         * IOException would catch timeout, but then we (and user) wouldn't now
         * wrong. It's also polite to really close the input stream, whatever
         * happened -> finally{}.
         */
    } catch ( SocketTimeoutException e) {
        return Pair.create(0, "Request timeout");
    } catch (IOException io) {
        Log.d(DEBUG_TAG, io.getMessage());
        return Pair.create(0, "Unsuccessful connection");
    } finally {
        if (is != null) {
            is.close();
        }
    }
}

private String readInputStream(InputStream stream, int len) throws IOException {
    /**
     * Standard java stream -> String conversion.
```

```
         */
        Reader reader = new InputStreamReader(stream, "UTF-8");
        char[] buffer = new char[len];
        reader.read(buffer);
        return new String(buffer);
    }

}
```

3.2.3 Testování

Testování proběhlo na úrovni mých přátel a známých a během vývoje jsem z něj vyzjistil mnohé zajímavé a užitečné informace týkající se UI a UX. Například drtivá většina dotázaných prohlásila, že koule „klikat“ je mnohem pohodlnější než tahat je po display sem a tam. Také mne odradili od úprav UI pro landscape režim a nyní je zakázán pro každou aktivitu v aplikaci.

Závěr

Cílem práce bylo analyzovat, navhnout a implementovat distribuovanou aplikaci pro sledování kulečníkových her, implementovat serverovou část, Android aplikaci a OS Linux nebo webové rozhraní. Provedl jsem analýzu a návrh, implementoval jsem serverovou část a Android aplikaci. Chystám se doimplementovat i zbývající část. Zadání jsem splnil částečně.

Literatura

[1] 1. díl - Úvod do UML. *UML [online]*, duben 2013, [cit. 2016-06-29]. Dostupné z: http://www.itnetwork.cz/navrhove-vzory/uml/uml-uvod-historie-vyznam-a-diagramy

[2] *Českomoravský billiardový svaz*. [cit. 2016-06-30]. Dostupné z: https://www.cmbs.cz

[3] *Women and Men Rankings*. [cit. 2016-06-30]. Dostupné z: http://www.wpa-pool.com/web/rankings

[4] World eightball pool federation. [cit. 2016-06-29]. Dostupné z: http://www.wepf.org/menu.php?option=11

Příloha **A**

Seznam použitých zkratek

GUI Graphical user interface

API Application Protocol Interface

URI Uniform Based Resource

URL Unified Resource Locator

API Application Protocol Interface

REST Representational State Transfer

MVC Model View Controller

HTTP(S) Hypertext Transfer Protocol(Secure)

CRUD Create, Read, Update, Delete

UI User Interface

UX User Experience

Příloha B

Obsah přiloženého CD

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