ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra počítačů

Bakalářská práce

Systém pro predikci úspěšnosti a výpočtu skóre módu „mythic+“ ve hře World of Warcraft

Architektura systému, logika systému, perzistence a vizualizace dat.

Martin Lukeš
Otevřená informatika

Květen 2021
Vedoucí práce: Ing. Jiří Šebek

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Lukeš** Jméno: **Martin** Osobní číslo: **474746**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Software**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Systém pro predikci úspěšnosti a výpočtu skóre módu „mythic+“ ve hřeWorld of Warcraft. – Architektura systému, logika systému, perzistence avizualizace dat.

Název bakalářské práce anglicky:

System for predicting success and calculating the score of the "mythic+"mode in the game World of Warcraft. - System architecture, system logic,data persistence and visualization.

Pokyny pro vypracování:

1. Vytvořte Business Analýzu systému (cíle, požadavky, business domain model, business proces model, návrh implementace) a digram nasazení.
2. Vytvořte detailní analýzu systému (use-case diagram a scénáře, systémové požadavky, analytický doménový model)
3. Implementujte vrstvu, která zajistí datový tok a perzistentí vrstvu která zajistí uložení dat.
4. Analyzujte sesbíraná data, zvolte nejvhodnější atributy dat k predikci úspěchu a výpočtu skóre.
5. Implementujte business logiku systému a prezentační vrstvu, kde budou data vhodně vizualizována.
6. Otestujte, zjistěte relevanci daného rating systému. Otestujte systém vůči jiným sytémům, které poskytují takové skóre.

Funkce aplikace:

1) část aplikace je addon, který sbírá "combat data" přímo ze hry a odesílá je na zpracování na server
2) část aplikace je na serveru, kde probíhájí 2. výpočty, ze kterého jsou 2 výstupy. 1 výstup je live predikce úspěchu na základě sbíraných dat během módu "mythic+". 2 výstup je skóre, které se vypočítá na konci módu "mythic+" na základě všech nasbíraných dat.
3) část aplikace je webové rozhraní, kde budou výstupy serveru vizualizovány

Seznam doporučené literatury:

1. Cadle, James, Debra Paul, and Paul Turner. Business analysis techniques. British Informatics Society Limited, 2010.
2. Bittner, Kurt, and Ian Spence. Use case modeling. Addison-Wesley Professional, 2003.
3. Silberschatz, Abraham, Henry F. Korth, and Shashank Sudarshan. Database system concepts. Vol. 4. New York: McGraw-Hill, 1997.
4. Ludicrous Speed, LLC. “World of Warcraft Rankings for Mythic+ and Raid Progress.” Raider.IO, 2017, raider.io/.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jiří Šebek, kabinet výuky informatiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **21.02.2021** Termín odevzdání bakalářské práce: **21.05.2021**

Platnost zadání bakalářské práce: **19.02.2023**

Ing. Jiří Šebek podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Poděkování / Prohlášení

Na prvním místě bych chtěl poděkovat Davidu Kulovi, že se mnou pracoval na tomto systému a vedoucímu Jiřímu Šebkovi, že si nás vzal pod svá křídla. Dále děkuji všem testerům, kteří nám pomohli otestovat všechny možné extrémní i obvyklé případy. V neposlední řadě chci poděkovat své rodině a přátelům za podporu při studiu.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 21. 5. 2021

.....

Abstrakt / Abstract

Tato bakalářská práce se zabývá problémem dosavadního hodnocení módu mythic+ ve hře World of Warcraft a převážně pak vývojem nového systému `KeystonePro`, který by měl existující nedostatky pokrýt. Vývoj se skládá ze zvolení vhodných technologií, sběrů požadavků, návrhu diagramů, implementace a uživatelského testování. Stěžejní částí implementace je určení vhodných parametrů pro výpočet skóre a vytvoření predikce na základě historických úspěchů.

Klíčová slova: World of Warcraft doplněk; hodnocení, predikce úspěchu módu mythic+.

This bachelor's thesis focuses on the problem of the current score system of the mythic+ mode in the game World of Warcraft and mainly on the development of a new system `KeystonePro`, which should cover the existing shortcomings. The development consists of selecting suitable technologies, collecting requirements, designing diagrams, implementation, and user testing. A key part of the implementation is to determine the appropriate parameters for score calculation and the creation of prediction based on historical results.

Keywords: World of Warcraft addon; score, prediction of mythic+ success.

Title translation: System for predicting success and calculating the score of the `mythic+` mode in the game World of Warcraft (System architecture, system logic, data persistence and visualization.)

Obsah /

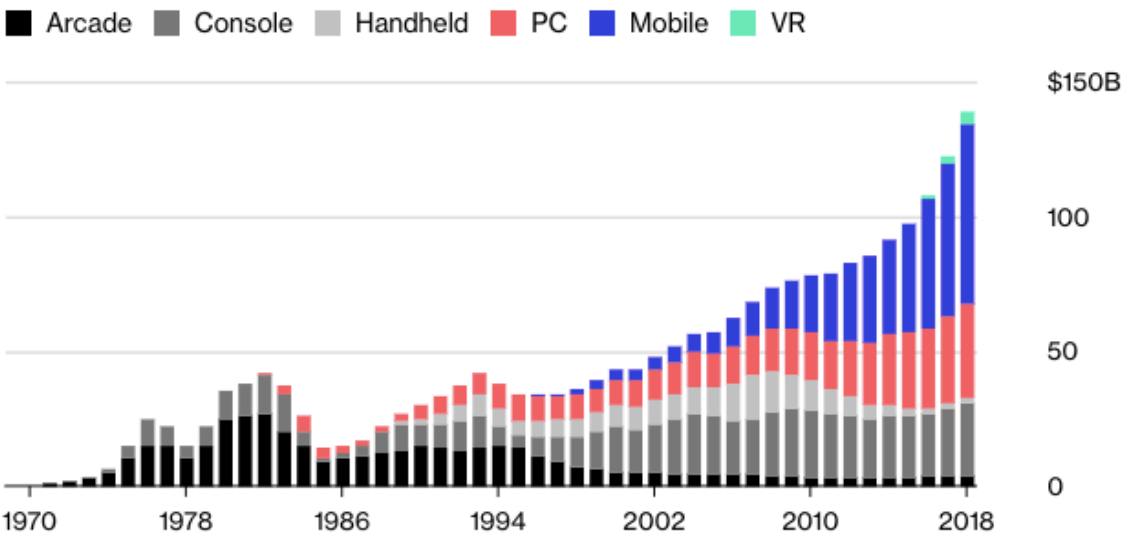
Tabulky /

Kapitola 1
Úvod

Tato bakalářská práce je zaměřená na vývoj systému, který slouží jako doplněk již existující počítačové hry World of Warcraft, a to konkrétně módu mythic+. Tento systém jsme s kolegou Davidem Kulou, který se také na vývoji systému podílel, pojmenovali KeyStonePro. Abych dokázal zasvětit do tématu všechny čtenáře, dovolím zde zanechat lehký vhled do herního průmyslu, MMORPGs a eSportu.

Herní průmysl je odvětvím průmyslu, které se zabývá vývojem, marketingem a prodejem videoher. Zahrnuje desítky pracovních profesí, jako třeba herní návrhář, vývojář, grafik, hardware specialista, nebo třeba i profese, které vyžadují společnosti zabývající se eSportem. V roce 2019 zaměstnával 143 045 zaměstnanců v USA a 1750 v České republice. Existuje mnoho zařízení, na kterých se dají videohry hrát, například konzole, osobní počítače, tablety a mobily či zařízení, které pracují s virtuální realitou. [1] [2]

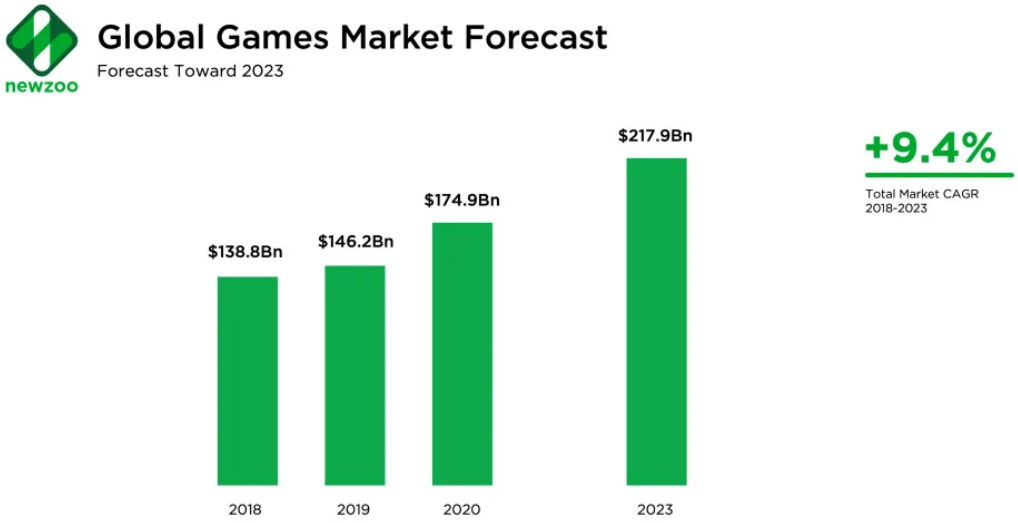
Na grafu 1.1 můžeme vidět odhad toho, jak se zvyšoval obrat herního průmyslu v průběhu historie a podíl herních zařízení v jednotlivých letech.



Obrázek 1.1. Globální odhady obratu z odvětví videoher od roku 1971 do roku 2018, podle průzkumu trhu společnosti Pelham Smithers. [3]

V České republice činil v roce 2018 obrat 3,79 miliard Kč. [2]

V roce 2020 začal herní průmysl také stoupat díky pandemii. Na grafu 1.2 můžeme vidět předpověď růstu do roku 2023. [4]



Obrázek 1.2. Předpověď růstu obratu v herním průmyslu provedená v listopadu 2020. [4]

MMORPGs (z angličtiny a massively multiplayer online role-playing games) je žánr počítačových her, který získal na popularitě v 21. století, a to hlavně s příchodem stabilního internetu. Jedná se vlastně o podskupinu žánru role-playing games (RPGs) hranou online přes internet v persistentním světě se stovkami či tisíci hráči připojených na stejném serveru. Persistentní svět znamená online herní svět, který funguje a pokračuje ve svém dění i když se hráč odpojí a zrovna nehraje. Hráč se může vždy zpět napojit a pokračovat kde přestal. Když je hráč odpojen, ostatní hráči mohou pokračovat ve svém hraní. [5]

Hráč převezme roli fiktivní postavy a ovládá ji, plní za ni úkoly a snaží se dosáhnout různých cílů. Oproti klasickým RPG, které jsou většinou pro jednoho hráče, hráči MMORPGs mohou dosáhnout svých cílů společně s ostatními hráči, nebo mezi sebou soutěžit. [5]

Právě ze soutěžení v online světě vznikl pojem eSportu (elektronický sport). Jedná se o formu sportu, kde primární znaky sportu zprostředkovávají elektronické systémy. Vstup hráčů a týmu jsou stejně jako výstup eSport systému zprostředkovány pomocí rozhraní člověk-počítač. Prakticky řečeno, eSport se týká kompetitivních (amatérsky i profesionálně) videoher, které obsahují různé ligy, žebříčky a turnaje, kde hráči obvykle patří pod různé týmy nebo jiné sportovní organizace, které bývají sponzorovány obchodními organizacemi. Během posledních let popularita eSports rapidně roste, a to díky velké přítomnosti online her a online přenosů. [6]

Z hlediska finanční stránky je na tom eSport velmi dobře. Historicky největší výhry dosáhl hráč Johan Sundstein ve hře Dota 2 a to 6 963 091 amerických dolarů. Ale nejsou to pouze jedno časové záležitosti. Například ve hře League of legends se průměrné platy profesionálních hráčů podobají platům profesionálních hráčů fotbalu. V americké soutěži si ročně vydělají průměrně 410 000 amerických dolarů, v Evropě kolem 250 000 euro. [7–8]

1.1 Charakteristika hry World of Warcraft

World of Warcraft je MMORPG hra od společnosti Blizzard Entertainment (dnes Activision Blizzard), která má své počátky již v roce 2004. [9] Děj se primárně odehrává ve

fiktivním světě `Azeroth`, který je plný mnoha různých stvoření a ras. Hráči si mohou vybrat z 24 ras a 12 tříd pro svého hrdinu. Rasy jako human, elf, dwarf či orc, nebo i rasy, které nemají v jiných fantasy světech obdoby, jako tauren či draenei. Třídy se liší podle svého způsobu boje, některé bojují na blízko, některé na dálku a některé dokážou oboje. Nejdůležitější rozdíly jsou však v tom, jestli umí efektivně léčit, vydržet spoustu poškození, nebo ho naopak hodně udělit. Těmto rolím se říká postupně `healer`, `tank` a `damage`. 12 ras se hlásí k frakci `Alliance`, dalších 12 k frakci `Horde`. Pro příklad moje postava je rasy troll, třídy shaman, moje bojová role je `healer` a jelikož je to troll, tak patří pod frakci `Horde`. [10–12]

Obrázek 1.3. Troll shaman. Screenshot z první verze hry World of Warcraft, která vyšla v roce 2004 a byla znovu oficiálně zpřístupněna v roce 2019.

Hra se od svých počátků hodně změnila. Z počátku šlo pouze o samostatné či skupinové plnění úkolů, kde se odměny získávaly za zabíjení různých stvoření umělého světa. Tomuto způsobu se říká `Player versus Enviroment`, zkráceně `PvE`. [13] Později se přidaly odměny i za zabíjení hráčů opačné frakce, tomuto způsobu se říká `Player versus Player`, zkráceně `PvP`. [14] Hráči se poměřovali pouze silou svého vybavení, které získávaly jako odměnu. Postupně v dalších expanzích přibyly, či se přirozeně vytvořily soutěžní módy. Nejpopulárnější přirozeně vytvořený kompetitivní mód je závod v pokoření momentálně nejsilnějšího bosse, kterému se obecně říká `World first race`. Přirozeně vytvořený je kvůli tomu, že již od počátku se ve hře takový bossové vyskytovali, jenže v té době ještě skoro nikdo neměřil pořadí zabití, až později v tomto poli začaly přibývat statistiky a hráči v tom přirozeně začali soutěžit. [15–16] Prvním uměle přidaným kompetitivním módem byly arény. Hráči si v nich tvoří týmy po dvou, třech, nebo dříve i po pěti hráčích a poměřují zde své síly s jinými hráči. V tomto módu se v roce 2008 začali organizovat i první oficiální turnaje od Blizzardu. [17] V roce 2016 se do hry přidal další kompetitivní mód `mythic+`. [18]

1.1.1 Mód mythic+

Konečně se dostáváme k módu, jehož přídavek je téma této práce. Mythic+ funguje následovně: Hráči si vytvoří skupinu pěti hráčů, se kterou se pokusí do daného časového

limitu zabít všechny nepřátele v určitém dungeonu. Čím rychleji, tím lépe. Dungeon je oblast, kde jsou nepřátelé obvykle silnější než jinde ve hře, a proto je na jejich zabití potřeba více hráčů. V dungeonech se spolupracuje s ostatními hráči a bojuje proti počítačem ovládaným protivníkům. Proto mythic+ patří pod způsob `PvE`. Časové limity bývají v rozsahu 30-45 minut v závislosti na velikosti dungeonu. Parta hráčů je obvykle složena z `tanka`, který slouží jako štít skupiny a který se snaží pokrýt co nejvíce poškození, neboť toho vydrží více než ostatní role. Dále je v týmu jeden `healer`, který se snaží léčit své spoluhráče od obdrženého poškození a tři `damage`, jejichž hlavním úkolem je poškození soupeřům udělovat. Možnost je mít i jiné složení, nicméně toto je nejčastější a hra je vyvíjena tak, aby tato varianta zůstala i nadále nejúčinnější. V nejnovější expanzi `Shadowlands` je momentálně 8 různých dungeonů pro tento mód. [19]

Co se týče eSportu, tak v něm má mód mythic+ významné zastoupení. MDI (Mythic dungeon international) je oficiální turnaj od společnosti Blizzard, který je turnajem s největším cenovým fondem v tomto módu od samého vydání. Týmy trénují a zaznamenají si své nejlepší časy ve třech dungeonech na turnajových serverech. Nejlepších 8 týmů se jednou za 2-3 týdny setká v takzvaném `MDI Cup`, kde proti sobě během tří dnů hrají ve formátu na dvě vyřazení. Podle pořadí v těchto turnajích získají peníze a také body do průběžné tabulky, z které se pak 8 nejlepších týmů kvalifikuje do finále, kde se hraje o 300 000 amerických dolarů. Blizzard tyto turnaje vždy zadarmo vysílá a diváci mohou sledovat, jaké strategie týmy volí a kde ztrácejí drahocenné vteřiny, které určují úspěch v tomto módu. [20]

1.2 Motivace

Skutečnost, že se v módu mythic+ spolupracuje s ostatními hráči, vede k problému podobnému jako v kolektivních sportech. Úspěch nezáleží jen na vás, ale na celém týmu. Proto je důležité být nejen dobrý sám jako jedinec, ale najít si i kvalitní spoluhráče. Jenže na základě čeho poznám, že jiný hráč má odpovídající schopnosti k pokoření dungeonu v co nejlepším čase? Hra sama o sobě nabízí o hráčích základní informace. Více se však můžeme dozvědět ze systémů třetí strany, které například ukazují, jaké má hráč nejlepší historické časy či zobrazují různé osobní statistiky. Nicméně buď se tyto systémy neujaly, nebo nezobrazují vše potřebné tak, aby se dalo poznat, zdali je dobrý samotný hráč, nebo převážně jeho spoluhráči. Tato fakta mě vedla k tomu, abych pracoval na systému, který tyto nedostatky zakryje a nabídne ještě lepší rozhraní, které dokáže popsat co nejlépe hráčovu dosavadní dovednost v tomto módu.

1.3 Existující systémy

Již existují systémy, které nabízejí podobnou funkcionalitu jako náš systém a doplněk KeyStonePro. Pojďme se podívat na to, jak fungují, na jejich popularitu a nedostatky.

1.3.1 Raider.IO

Raider.IO je systém, který vyvíjí skupina Ludicrous Speed od roku 2017. Raider.IO získává data od Blizzard's API a přehledně je vykresluje ve webové aplikaci, desktopové aplikaci, i v doplňku přímo ve hře. Nejedná se o extra komplexní data, pro mód mythic+ se zobrazuje pouze skóre skupiny a jaké podmínky skupina měla, ku příkladu vybavení hráčů v daný moment, či úroveň mythic+ módu. Podle toho, v jakém čase a na jaké úrovni se hráčům podařilo dungeon dokončit, dostanou skóre. Raider.IO má přesah i

do `World first race` módu. Ve webové a desktopové aplikaci je možnost propojení s World of Warcraft účtem a také možnost si vizuálně upravit profil. [21]

Podle mého pozorování ze hry, se tímto World of Warcraft doplňkem řídí většina hráčů, kteří hrají mythic+ mód. Pokud máte špatné skóre, je velmi těžké se dostat do skupiny.

Raider.IO je nejrozšířenější doplněk pro výpočet skóre mythic+ módu. Skóre je vypočítáno velmi jednoduše, což je výhoda pro lehké pochopení systému, ale nevýhoda v nedostatku zachycených faktorů. Největším problémem je to, že skóre je vypočteno z úspěchu celé party, ačkoli se pak připočítává každému jedinci zvlášť.

1.3.2 Warcraft Logs

Warcraft Logs je systém, který vyvíjí anonymní skupina od roku 2013. Zaměřují se na módy mythic+, `World first race`, ale i na mód `Torghast`, který je nový od roku 2020. Ve hře World of Warcraft je možnost povolení sběru dat z boje, které se pak ukládají na disk. Je to vlastně takové logování[1] (od toho Warcraft Logs). Tato data využije systém Warcraft Logs, který je přes desktopovou aplikaci `Warcraft Logs Uploader` získá na server, kde se následně srovnávají hráči v různých odvětvích. Co se týče módu mythic+, tak hráče srovnávají v tzv. points, které fungují obdobně jako skóre v Raider.IO, tzn. nějaký poměr času a úrovně módu. Dále jsou statistiky v uděleném poškození, léčení, v uděleném poškození jenom do silných bossů v dungeonu a samostatně jen v čase dokončení. Tyto statistiky určují jednoduše pořadí mezi ostatními hráči, kteří mají logy v systému zaznamenány. [23] Z mého pozorování musím konstatovat, že Warcraft Logs je populární hlavně v módu `World first race` a v mythic+ módu nezískal takové popularity.

Warcraft Logs poskytuje daleko komplexnější data než Raider.IO, avšak do svých skóre nezohledňuje úplně všechny klíčové aspekty mythic+ módu. Líbí se mi, že se skóre tvoří v závislosti na ostatních hráčích.

1.4 Cíl

Cílem mé práce je vytvořit systém – doplněk hry World of Warcraft, který bude hráčům sloužit jako pomocník k výběru co nejlepších spoluhráčů pro vytvoření party v módu mythic+. Chci zohlednit faktory, s kterými konkurenční systémy nepracují, a hlavně vytvořit důraz na osobní výkon daného hráče. Jako novinku, co jiné doplňky neposkytují, chci vytvořit predikátor, který na základě historických výsledků hráčů v partě statisticky odhadne úspěch či neúspěch v módu. Data chci sbírat ve hře World of Warcraft, odtud se budou posílat na server, kde se zpracují do trénovacích modelů a vypočítá se z nich hráčovo skóre. Skóre a predikce budou vykreslovány pomocí webové aplikace.

[1] Označuje činnost, při níž systém vytváří logy. Při logování se zaznamenávají data za účelem jejich analýzy. [22]

Kapitola 2
Byznys analýza

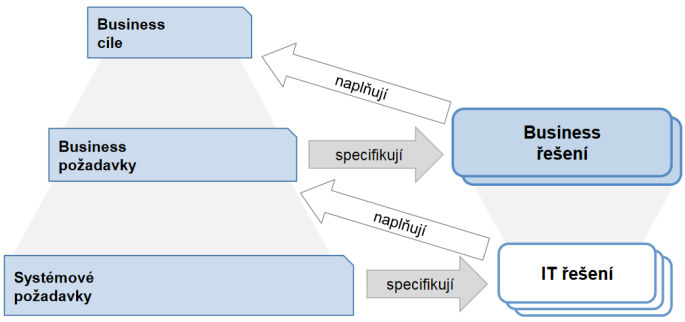
2.1 Sběr požadavků

Každý software vzniká na základě nějakých požadavků. Tyto požadavky se získávají od zainteresovaných osob (nebo institucí). Těmto osobám nebo institucím se také říká stakeholdeři. [24] Dle Kano modelu se požadavky dělí do tří kategorií.

- Nutné vlastnosti, které stakeholdeři ani nezmiňují, ale očekávají, že je systém bude mít. Jejich přítomnost je nenadchne, protože si ji nejspíš ani nevšimnou, za to nepřítomnost je znepokojí. Příklad: Aplikace bude obsahovat tlačítko na vypnutí.
- Uspokojivé/požadované vlastnosti stakeholdeři od systému očekávají, ale jen protože vyjádřili jejich požadavek. Jejich přítomnost je lineárně úměrná spokojenosti stakeholdera. Pokud jsou přítomny, stakeholder je spokojen, pokud nejsou, tak není. Příklad: Systém umožní uživateli registraci.
- Zbytné vlastnosti, které stakeholder neočekával, ale potěší ho. Pokud je systém nemá, stakeholder si toho ani nevšimne, pokud je systém má, považuje je za velké plus. Příklad: Přehazování pořadí chatovacích oken v chatovací aplikaci [25]

Stakeholdeři našeho systému jsme byli primárně já a kolega David Kula, dále ostatní spoluhráči ze hry a také konkurenční systémy jako Raider.IO nebo Warcraft Logs. Velká část požadavků se získala na začátku vývoje, nicméně některé požadavky se objevily až později a v budoucnu ještě budou iteračně přibývat.

Nejprve jsem vytvořil byznys cíle, které jsou naplněny řešeními, které jsou realizovány z byznysových požadavků. Po zvolených systémových technologiích se vytvořily systémové požadavky, z kterých bylo realizováno IT řešení, které naplňuje byznysové požadavky.



Obrázek 2.1. Vysvětlení úrovní požadavků a vztahů řešením [26]

2.2 Byznys cíle

Byznys cíle definují směřování organizace (strategii) a určují jakých cílů chce organizace dosáhnout. [26] Organizací jsme v tomto případě já a David Kula. Naše byznys cíle jsou tyto:

- Vytvoření komplexního hodnocení hráčů v módu “mythic+”, které zohlední více důležitých faktorů než konkurenční systémy.
- Možnost predikování úspěšnosti na základě historických výsledků hráčů.

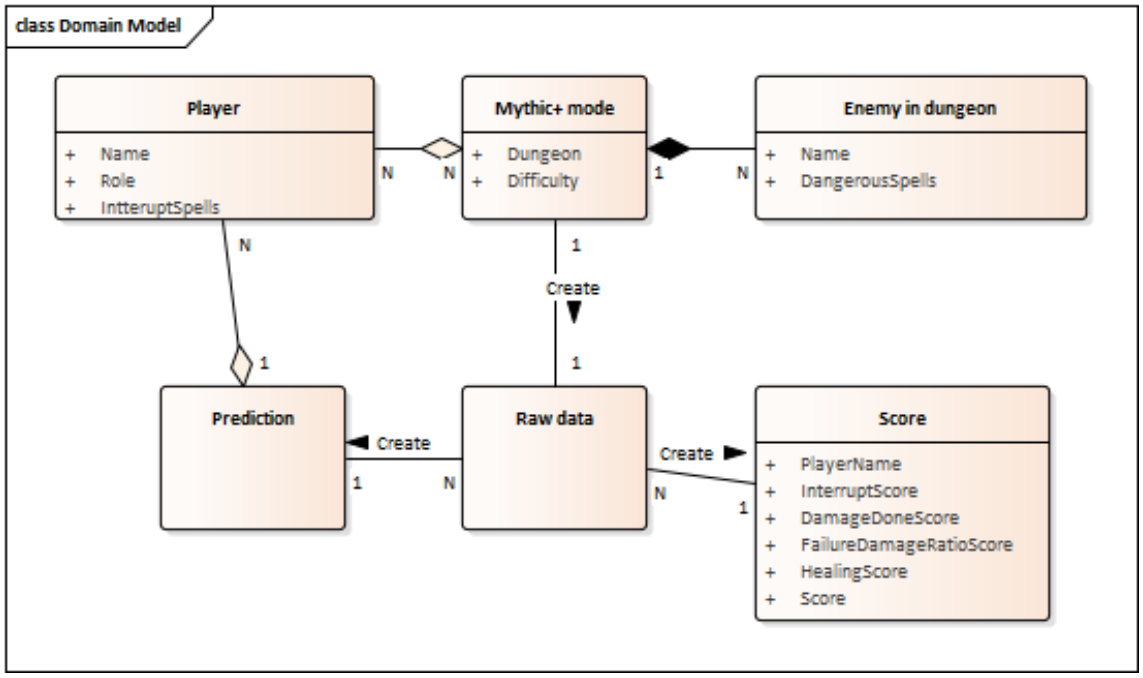
2.3 Byznys požadavky

Byznys požadavky reagují na potřeby organizace. Popisují důvody, proč má být změna realizována. Pro náš systém byly vytvořeny tyto požadavky:

- **BR1 – informace k použití:** Jako hráč chci vědět kde doplněk stáhnout, jak ho nainstalovat a jak s ním manipulovat.
- **BR2 – transparentnost:** Jako hráč chci zpětnou vazbu, že doplněk ve hře sbírá data.
- **BR3 – přehlednost:** Jako hráč chci vidět jednotlivá skóre přímo ve hře, bez navštívení aplikací třetí strany.
- **BR4 – odesílání dat:** Jako hráč potřebuji být schopen poslat sesbíraná data k analýze na server.
- **BR5 – zpětná vazba:** Jako uživatel chci mít zpětnou vazbu, že data nejsou poškozena, a že byla úspěšně odeslána.
- **BR6 – vizualizace dat:** Jako hráč potřebuji vidět vizualizaci vypočtených skóre všech hráčů.
- **BR7 – transparentnost:** Jako hráč bych rád znal detail výpočtu skóre.
- **BR8 – predikce úspěšnosti:** Jako hráč potřebuji znát predikci na úspěšnost v dungeonu konkrétní party hráčů.
- **BR9 – transparentnost:** Jako hráč bych chtěl vědět detail predikčního modelu.
- **BR10 – aktuálnost:** Jako predikční model chci být aktualizován v závislosti na nových datech.

2.4 Byznys doménový model

Byznys doménový model poskytuje seznam podnikových/obchodních entit a jejich atributů. Jsou to vlastně vstupy a výstupy byznys procesů. Jak můžete vidět na obrázku 2.2, je reprezentován pomocí koncepčního diagramu tříd. [27] Pod diagramem se vyskytuje slovní popis jeho entit.



Obrázek 2.2. Diagram byznys doménového modelu

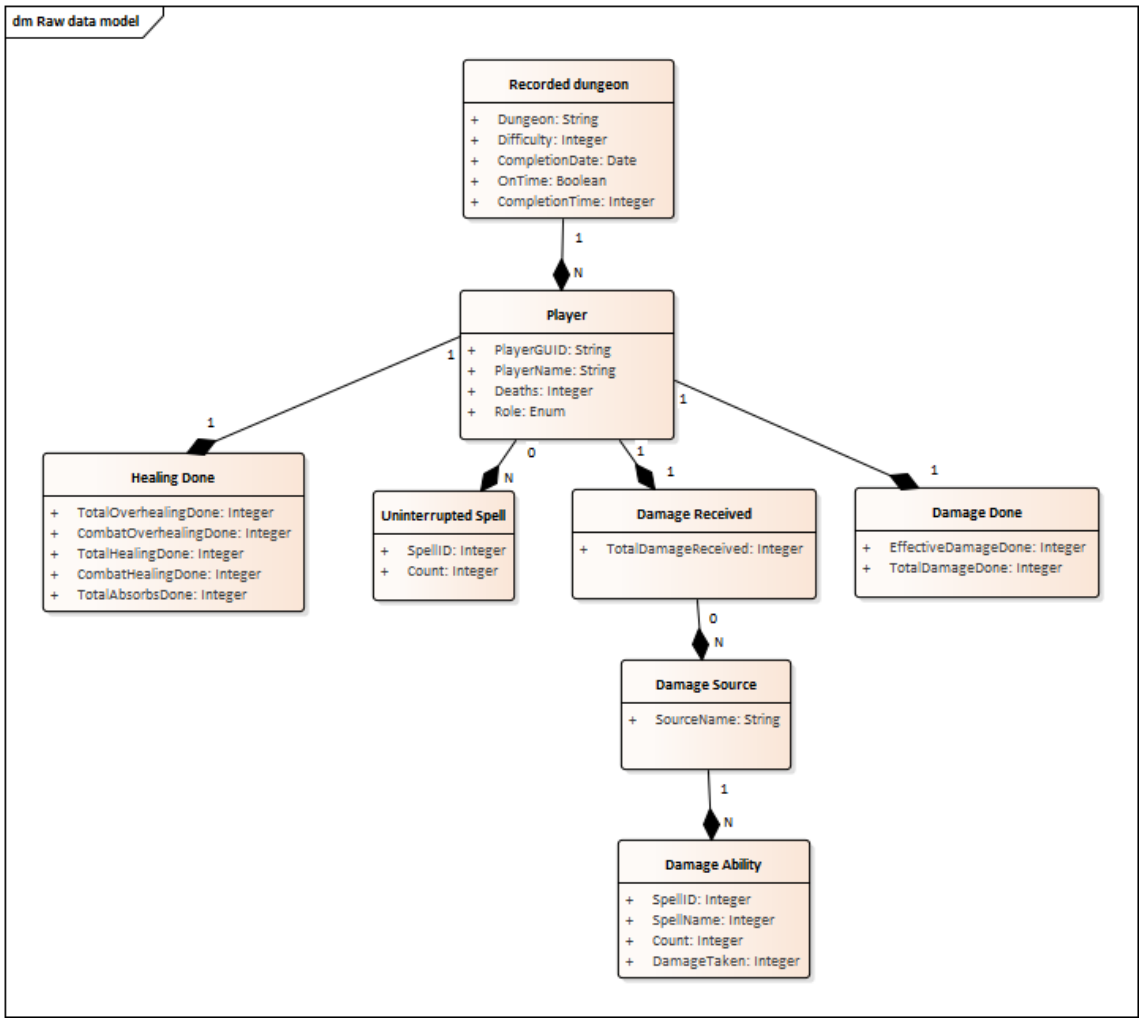
- **Player** – entita popisující hráče, který se účastní módu mythic+.
  - **Name** – jméno hráče ve hře.
  - **Role** – hráčova herní role, tedy buď `TANK`, `HEALER` nebo `DAMAGE`.
  - **InterruptSpells** – jsou kouzla, kterými hráč může přerušit přerušitelná kouzla nepřátel.
- **Mythic+ mode** – entita popisující danou instanci běhu tohoto módu. Je vázána na typicky 5 hráčů a N nepřátel v dungeonu.
  - **Dungeon** – jméno dungeonu pro daný běh.
  - **Difficulty** – obtížnost značí jak silní nepřátelé jsou v dungeonu. S větší obtížností přibývají v dungeonu takzvané `affixy`, což jsou různá vylepšení nepřátel, či nepříjemnosti, které mění styl hry. Příkladem takového `affixu` jsou explozivní koule, které když se nezničí do několika sekund, tak dají velké poškození všem hráčům. [28]
- **Enemy in dungeon** – entita nepřátel, kteří se vyskytují v dungeonech a kteří se musí zabíjet, aby hráči docílili úspěchu.
  - **Name** – jméno nepřítele.
  - **DangerousSpells** – tento seznam obsahuje kouzla, na která se oproti běžným kouzlům dá nějakým způsobem reagovat a předejde se tedy jejich nebezpečnosti.
- **Prediction** – obsahuje vypočtenou predikci pro konkrétních N (obvykle 5) hráčů.
- **Score** – tato entita obsahuje finální skóre a také jednotlivé části skóre konkrétního hráče.
  - **PlayerName** – jméno hráče, jemuž skóre náleží.
  - **InterruptScore** – skóre vypočtené v závislosti na přerušování kouzel neboli `interrupts`.
  - **DamageDoneScore** – skóre vypočtené v závislosti na udělování poškození neboli `damage done`.

- **FailureDamageRatioScore** – skóre vypočtené v závislosti na zlomku obdrženého poškození z kouzel, kterým se dá nějakým způsobem vyhnout či zabránit (zkráceně `failure damage taken`) oproti všemu obdrženému poškození.
- **HealingScore** – skóre vypočtené v závislosti na léčení neboli `healing`.
- **Score** – celkové skóre vypočtené v závislosti na všech předchozích kategoriích.

- **Raw data** – data sbíraná během mythic+ runu, je jim věnována speciální podkapitola.

2.4.1 Raw data model

Z raw dat sbíraných během dungeonu se vytvářejí skóre a predikce úspěšnosti, jsou to proto velmi klíčová data pro náš systém. Na obrázku 2.3 můžete vidět datový diagram sbíraných dat a jeho popis. Je dobré si všimnout, že mezi všemi třídy dat je kompoziční vazba. To znamená, že na diagramu je každá vertikální úroveň závislá na hodnotě nad sebou. Nemá bez ní žádnou výpovědní hodnotu.



Obrázek 2.3. Raw data diagram

- **Recorded dungeon**, obsahuje informace o konkrétním běhu módu mythic+. Dungeony se liší svojí velikostí a nepřáteli, proto je zde atribut jména dungeonu (`Dungeon`). Obtížnost (`Difficulty`) značí, jak silní nepřátelé jsou v dungeonu a kolik je v dungeonu affixů, o kterých jsem se již zmínil. Dalším atributem je čas dokončení

(`CompletionTime`), který v milisekundách indikuje přesný čas běhu od začátku dungeonu. Atribut `OnTime` značí, zdali čas dokončení byl menší než potřebný čas k dokončení a atribut `Completion Date`, indikuje real-time čas dokončení dungeonu, který je později využit k určení jedinečnosti běhu.

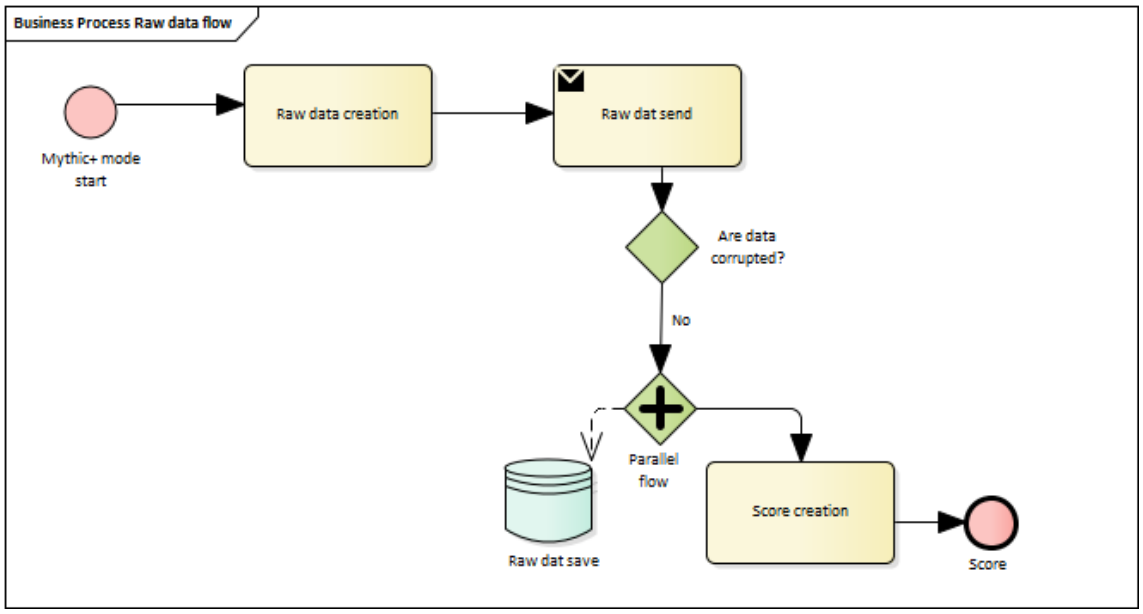
- **Player**, obsahuje data spjatá k jednotlivým hráčům. `Player GUID` je jedinečný identifikátor, který World of Warcraft přiřazuje každému hráči, `Player Name` je jméno hráče doplněné o server na kterém hraje, `Deaths` počítá počet smrtí za run a `Role` je hráčova role, tedy buď `TANK`, `HEALER` nebo `DAMAGE`.
- **Healing done**, obsahuje data spjatá s léčením. Přídavná jména `Combat` a `Total` značí, zda během léčení byl hráč v souboji s nepřáteli anebo byl v souboji s nepřáteli i mimo něj. `Overhealing` znamená, že hráč použil léčení, ačkoli hráč, kterého léčil, vyléčit nepotřeboval. `Absorbs` znamená, že hráč vykryje počet poškození podle toho, jak velký má na sobě absorb. Je to takové předpovídání, že hráč bude dostávat poškození. Pokud ho opravdu dostane, poškození se `vyabsorbuje` a do dat se připočte hodnota absorbu.
- **Uninterrupted spell** Existují kouzla protivníků, která jsou značena jako nebezpečná a která se mohou určitými kouzli hráčů přerušit. Pokud může hráč takové přerušovací kouzlo použít a neudělá to, započítá se to do tohoto pole. `SpellID` je identifikační číslo kouzla, které poskytuje World of Warcraft, `Count` je počet započítání daného kouzla.
- **Damage Received** obsahuje atribut `TotalDamageReceive` s hodnotou celkového obdrženého poškození za běh.
- **Damage Source** obsahuje atribut `SourceName`, který obsahuje název protivníka, od kterého hráč obržel poškození.
- **Damage Ability** obsahuje identifikátor protivníkova kouzla (`SpellID`), jeho název (`SpellName`), počet poškození, kterým tímto kouzlem bylo hráči uděleno (`DamageTaken`) a kolikrát bylo kouzlo vůči hráči použito (`Count`).
- **Damage Done** obsahuje počet uděleného poškození hráčem během dungeonu. Rozlišuje se poškození uděleno v soubojích, kde protivník umřel (`EffectiveDamageDone`) a všechno udělené poškození (`TotalDamageDone`).

2.5 Byznys proces model

Modelování byznys procesů je analytická reprezentace nebo jen jednoduchá ilustrace byznys procesů organizace. Procesový model je klíčovou součástí pro efektivní řízení byznysových procesů. Často se porovnávají tzv. `as-is` proces modely a `to-be` proces modely. `as-is` proces model je reprezentace stávajících procesů organizace a `to-be` obsahuje efektivnější řešení onoho procesu. [29]

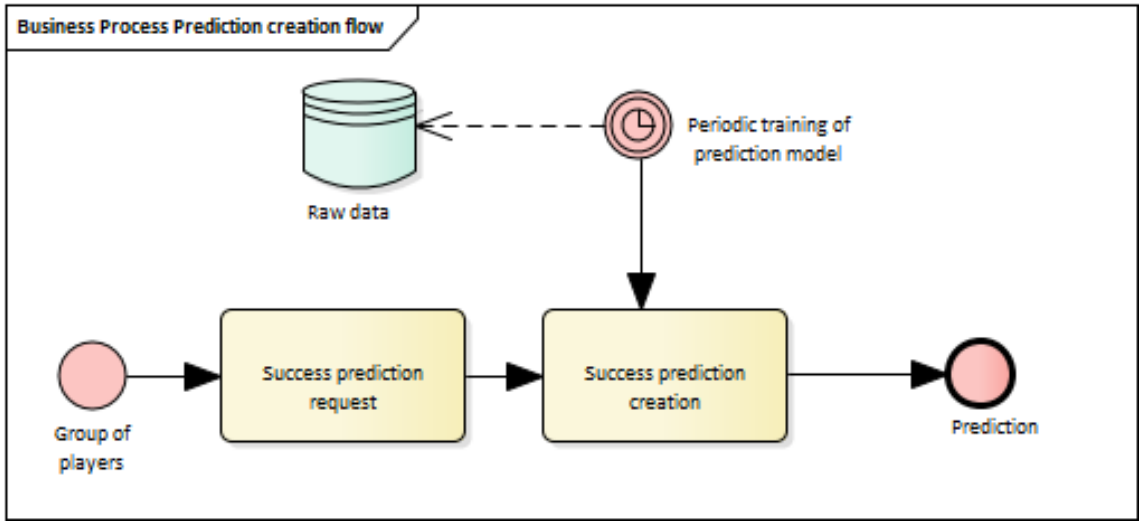
V našem systému, se objevují dva hlavní výstupy, a to skóre hráče a predikce skupiny v mythic+ runu. Tyto výstupy jsou i výstupy našich dvou byznys proces diagramů.

Na diagramu 2.4 můžeme vidět tok raw dat systémem až po vytvoření skóre. Můžete si povšimnout, že byznys proces model nepopisuje konkrétní technologie systému. Také si můžete povšimnout, že start i cíl proces modelu jsou vždy entity z doménového modelu 2.2.



Obrázek 2.4. Diagram byznys procesu toku raw dat

Na diagramu 2.5 vidíme proces žádosti o vytvoření predikce pro danou skupinu hráčů. Vyskytuje se zde časová entita, která zobrazuje pravidelné přetrénování predikčního modelu.



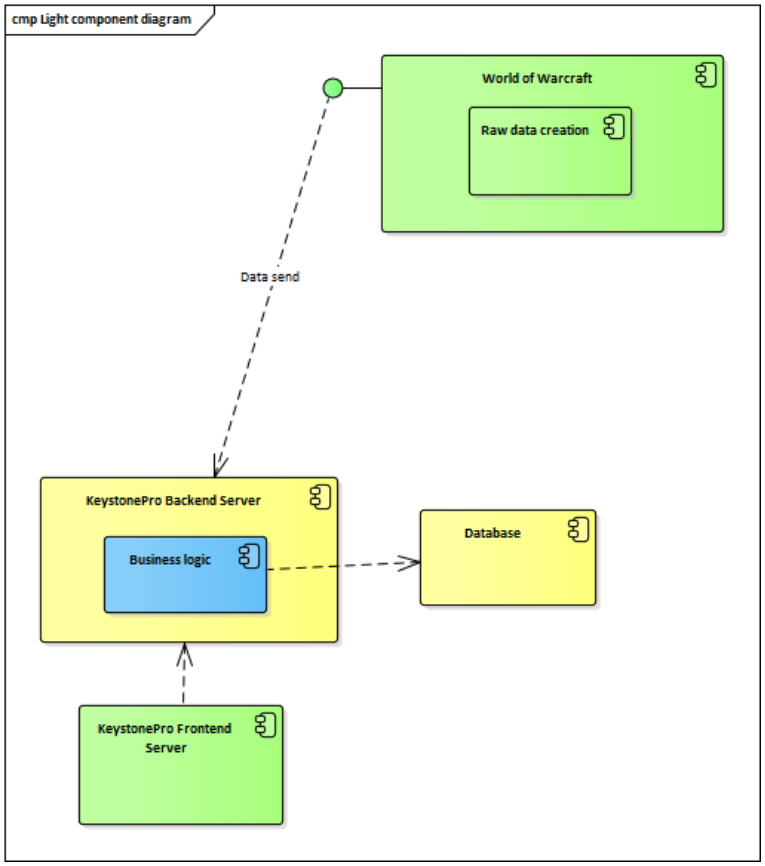
Obrázek 2.5. Diagram byznys procesu žádosti o predikci

Diagramy byly vytvořeny podle standardu `Business Process Modeling Notation` (BPMN) v softwaru Enterprise architect.

Kapitola 3
Analýza architektury systému

3.1 Obecný diagram komponent

V této kapitole se věnuji architektonické stránce systému, porovnáním možných technologií a zvolení konkrétních. Na diagramu 3.1 můžeme vidět první prototyp architektury systému, ještě bez konkrétních technologií a hlubších dekompozic systému.



Obrázek 3.1. První verze diagramu komponent

Z diagramu můžeme vyčíst, že systém má 3 hlavní komponenty. Aplikaci pro sběr dat, backendovou část, kde budou data ukládána a zpracovávána a frontendovou část, kde budou vizualizována finální skóre a výpočty. Backendovou částí je obecně myšlena ta část systému, která není pro uživatele vidět. Řeší práci nad daty, to znamená, že může obsahovat různé výpočty, konverze a ukládání dat. Oproti tomu frontend se říká části, kterou uživatel vidí. Jedná se o uživatelská rozhraní systému. Někdy se ji také říká prezentační vrstva. [30] Aplikaci pro sběr dat a frontend měl na starost kolega David Kula (značeno zeleně). Já měl na starost backendovou část a komunikaci mezi jednotlivými komponenty (značeno žlutě). Výpočty na backendu jsme měli rozděleny (značeno

modře). U částí, na kterých jsem nepracoval, nebudu porovnávat možné implementace, ale pouze zmíním finální implementaci.

3.2 Aplikace sběru dat ze hry

První částí systému je aplikace, která sbírá bojová data hráčů v průběhu mythic+ módu. Aplikace je implementována ve skriptovacím jazyce Lua, který je prakticky jedinou možnou volbou, protože je v něm naprogramováno uživatelské rozhraní hry World of Warcraft, které se kompiluje přímo za běhu hry v ní.

Aplikace měla v původním návrhu i odesílat nasbíraná data na server. Nicméně hra World of Warcraft tyto operace z bezpečnostních důvodů v rámci doplňku zablokovala. Proto vznikl nový požadavek na vytvoření aplikace, která bude sloužit jako nahrávač raw dat na server. Volbě technologie pro tuto komponentu se budu věnovat v další kapitole.

3.2.1 Specifikace Lua

Lua je mocný, efektivní a lehce zabudovatelný skriptovací jazyk. Podporuje procedurální programování, objektově orientované programování, funkcionální programování, data-driven programování a techniku datového popisu. Lua kombinuje jednoduchý procedurální syntax se silnými konstrukty datového popisu založenými na asociativních polích a rozšiřitelných sémantikách. Lua je dynamicky typovaná, běží interpretací bytekódu na virtuálním stroji a má automatickou správu paměti s inkrementální garbage collection, což ji dělá ideální pro konfiguraci, skriptování a rapidní prototypování.

Ve hře World of Warcraft se používá ke tvorbě uživatelského rozhraní. Lua skripty se dají spouštět přímo ve hře skrze herní chat, nebo se předem napsané skripty dají spustit jako addon (uživatelské custom rozšíření UI vehře, jakýsi uživatelem vyrobené plugin) přímo přidáním skriptu do adresářové struktury hry. Skrze Lua skripty se dají ve hře zachytávat herní eventy a k nim se dají přiřazovat vlastnoručně napsané funkce pro odběr těchto eventů. Takto můžeme získat přístup k logům hry v reálném čase, ze kterých lze zjistit co konkrétně který hráč v okolí provádí a co se mu stalo/děje.

Užití jazyka Lua ve hře World of Warcraft spočívá hlavně na event-driven programování. Programátor se přihlašuje k odběru různých eventů generovaných hrou a ty potom spouštějí logiku a běh addonu. [31]

3.3 Možnosti aplikace pro odesílání dat

Hra World of Warcraft je podporovaná dvěma operačními systémy. Operačním systémem Windows a macOS. [32] Proto bylo důležité, zvolit typ aplikace takový, aby vyhověl oběma operačním systémům, které hráči používají. V této kapitole se věnuji možnostem typů aplikace pro odesílání dat a zvolení konkrétního druhu.

3.3.1 Nativní aplikace

Nativní řešení patří mezi nejstarší podoby aplikace. Jeho základním znakem je to, že musí být vyvíjeno pro každou platformu zvlášť. V našem případě by se jednalo o dvě samostatná řešení. Vývoj takové aplikace je proto, při srovnání s ostatními formami aplikací, podstatně náročnější. Na druhou stranu ovšem nativní aplikace nabízejí nejlepší podmínky pro práci off-line a díky užší propojenosti s operačním systémem rychlejší odezvu na uživatelské příkazy. [33]

3.3.2 Multiplatformní desktopová aplikace

Oproti klasické nativní aplikaci, multiplatformní aplikace je kompatibilní na více operačních systémech. V kontextu mobilních aplikací se tomuto typu aplikace říká hybridní aplikace. Výhodou je méně náročná implementace pro více operačních systémů, nevýhodou mohou být problémy s kompatibilitou vůči operačnímu systému nebo horší výkon z důvodu nepřímé provázanosti s daným operačním systémem.

3.3.3 Browser aplikace

Nejjednodušší variantou, ať už z hlediska složitosti vývoje nebo nákladů, stále zůstává webová aplikace. Kód je tvořen za pomoci technologií HTML, CSS či JavaScript a dále prezentován prostřednictvím browseru (Google Chrome, Microsoft Edge, Opera, …). Výkon takové aplikace je proto přímo závislý na možnostech daného prohlížeče, jenž nemusí být zcela kompatibilní s JavaScriptem. Na druhou stranu ovšem webové technologie přinášejí výhodu v podobě multiplatformnosti. Výhodou je jistě snadná instalace, za předpokladu, že uživatel má nainstalovaný nějaký webový browser. Nevýhodou může být horší výkon aplikace. [33]

3.3.4 Volba technologie

Požadavky na naší aplikaci implikují funkčnost, která je podobná konkurenčnímu systému `Warcraft logs`, který obsahuje desktopovou aplikaci `Warcraft logs uploader`. Z důvodu stejné domény uživatelů těchto dvou aplikací, jsem se rozhodl také pro desktopovou aplikaci. Kvůli jednoduchosti a kvůli času, jsem se rozhodl pro multiplatformní vývoj.

3.4 Možnosti vývoje desktopové multiplatformní aplikace

Vybral jsem, že aplikace bude multiplatformní a desktopová. Po mé rešerši jsem zjistil, že existuje spousta frameworků, ve kterých jde taková aplikace vyvíjet. V této kapitole se podíváme na nejzajímavější kandidáty v tomto odvětví a jednoho si vybereme.

3.4.1 Qt

Qt je multiplatformní framework napsaný v programovacím jazyce c++ se zaměřením na operační systémy Windows, MacOs a Linux pro desktopový vývoj, iOS a Android pro mobilní vývoj a také na mnoho dalších méně populárních operačních systému. K dispozici jsou placené licence, ale i licence zdarma. Součástí projektu je vlastní integrované vývojové prostředí, ale existují i doplňky do populárnějších vývojových prostředí. Obvykle se Qt projekt píše v c++, ale existuje i například `Python bindings (PyQT)` pro psaní v programovacím jazyce Python. Obsahuje dvě grafické uživatelské rozhraní Qt Widgets a QtQuick. [34]

3.4.2 Electron

Electron je `open source` framework udržovaný společností GitHub, ve kterém se desktopové aplikace vyvíjí pomocí jazyků známých pro webový vývoj, tj. JavaScript, HTML, a CSS. Electron se skládá z tří hlavních pilířů: Chromium (báze kódu prohlížeče Google Chrome) pro zobrazování webového obsahu, Node.js pro práci s lokálním souborovým systémem a operačním systémem a vlastních aplikačních rozhraní pro práci s potřebnými nativními funkcemi operačního systému. [35]

3.4.3 JavaFX

JavaFX je `open source` platforma pro desktop, mobily a různé embedded systémy (vestavěné systémy) postavená na programovacím jazyku Java. Jazyk Java je specifický tím. že je platformě nezávislý, pokud tedy platforma obsahuje Java Virtual Machine. Kromě jazyku Java je podporován i jazyk Kotlin nebo Scala. Uživatelská rozhraní se v JavaFX tvoří pomocí XML notace (v JavaFX FXML) a mohou se stylovat pomocí CSS. [36–37]

3.4.4 Volba

Jako framework, ve kterém budu desktopovou aplikaci vyvíjet jsem si zvolil JavaFX. Z výše jmenovaných frameworků je jediný, s kterým už mám zkušenosti. Pro potřeby aplikace pro odesílání dat mně přišel nejvhodnější kandidát.

3.5 Možnosti backendového frameworku

Další komponentou systému KeyStonePro je backend, neboli serverová část, kde se bude pracovat s databází a kde budou probíhat výpočty skóre, tvorba modelu a určení predikce. Serverových frameworků je mnoho, Django, Flask, Express, ASP, Spring boot a ještě spousta dalších. Z důvodu požadavků na tvorbu predikčního modelu a určování predikce, jsem se rozhodl omezit na framework, který běží nad programovacím jazykem Python, jelikož má Python v odvětví strojového učení velmi dobré knihovny. Django a Flask patří do těchto frameworků.

3.5.1 Porovnání Django a Flask

Oba frameworky mají porovnatelně velké zastoupení na trhu, jsou `open-source`, mají dobrý výkon a bezpečnostní zabezpečení. Filosofie Flask je možnost tvořit projekty minimalisticky a modulárně. To znamená, že Flask neobsahuje skoro žádné vestavěné knihovny a vývojář má vše pod svojí kontrolou. Oproti tomu Django má spoustu vestavěných knihoven, což může vývoj zrychlit a učinit ho přímočařejší. Django má oproti Flask větší a aktivnější komunitu vývojářů. Django poskytuje svoje vlastní objektově relační mapovaní (ORM), zatímco Flask nemá žádné datové modely. Datové modely umožňují vývojářům propojit databázi přímo se třídami v projektu. [38]

3.5.2 Volba

Myslím si, že zvolení ani jednoho frameworku by nebyla velká chyba, ale kvůli větší přímočarosti vývoje, větší zastoupení komunity a přítomnému ORM, jsem se rozhodl zvolit Django framework.

3.6 Databáze

Další technologií, kterou bylo třeba zanalyzovat byl typ databáze. Z obrázku 2.3 můžeme vidět strukturu dat, která se bude vyskytovat v databázi. Jeden aktivní hráč World of Warcraftu odejde dle mého pozorování za týden kolem 10 běhů módu mythic+. V únoru 2021 bylo odhadem zaokrouhleně na statisíce 5 200 000 aktivních hráčů. Za předpokladu, že by náš doplněk nabyl popularity, se rozhodně jedná o `big data` a bylo by na místě zvolit NoSQL typ databáze. Jelikož ale Django přímo podporuje jen SQL databáze a hráčů bude ze začátku systém využívat málo, nechal jsem tento předpoklad do budoucna a zvolil jsem databázi SQL. [39–40]

3.7 Frontend

Technologie pro frontend, čili vizualizaci našich skóre a predikcí, byl vybrán Angular. Angular byl zvolen hlavně proto, že David v něm má největší zkušenosti, ale také proto, že patří na první příčky framework trendů v tomto odvětví na světě.

3.7.1 Specifikace Angularu

Angular byl poprvé vydán v roce 2010 Googlem, tehdy ještě pod názvem AngularJS. Dnes se jedná o javascriptový framework využívající jazyk TypeScript, který se konvertuje do Javascriptu. Jakýkoliv validní Javascript je přitom validní Typescript. Typescript je nadmnožinou Javascriptu.

Angular je vývojová platforma postavená na TypeScriptu. Jako platforma v sobě Angular zahrnuje:

- Framework postavený na komponentách pro stavění škálovatelných webových aplikací
- Kolekci dobře integrovaných knihoven, které pokrývají širokou škálu vlastností, zahrnujících přesměrovávání (routing), manipulaci s formuláři, klient-serverovou komunikaci a další
- Soubor vývojářských nástrojů pro pomoc s vývojem, vytvářením, testováním a aktualizací kódu

S Angularem využíváte platformy, která se dokáže škálovat od projektů pro jednoho vývojáře až na velkopodnikové aplikace. Angular je navržen tak, aby aktualizace kódu probíhaly co nejlehčeji to jde, abyste mohli využít novinek s minimálním úsilím. K tomu všemu se ekosystém Angularu skládá z různorodé skupiny developerů, autorů knihoven a tvůrců obsahu, kterých je dohromady přes 1,7 milionu. [41]

3.8 Komunikace mezi komponenty

Náš systém se skládá z mnoha komponent, které spolu potřebují komunikovat. Středobodem je Django server, kde se data shromažďují, probíhají nad nimi výpočty a dále jsou vystavována k dispozici pro vizualizaci na frontend. Proto je potřeba vytvořit aplikační rozhraní, které tyto požadavky pokryje.

Ačkoliv možností, jak takové rozhraní navrhnout existuje velké množství, v praxi nad všemi dominují dvě technologie, REST (Representational state transfer) a SOAP (Simple Object Access Protocol). Obě tyto technologie umožňují vytvořit plnohodnotné API, ale zatímco SOAP má jasně definované standardy, REST v tomto ohledu definuje pouze velmi obecné koncepty. [42]

Jedna z implementací formou restu je `Django REST framework`, který má dobrou dokumentaci a integraci do frameworku Django. Pro podporu protokolu SOAP jsem našel v Djangu také frameworky, jako například `spyne`, ale funkčnost a jednoduchost převládla pro zvolení implementace REST aplikačního rozhraní.

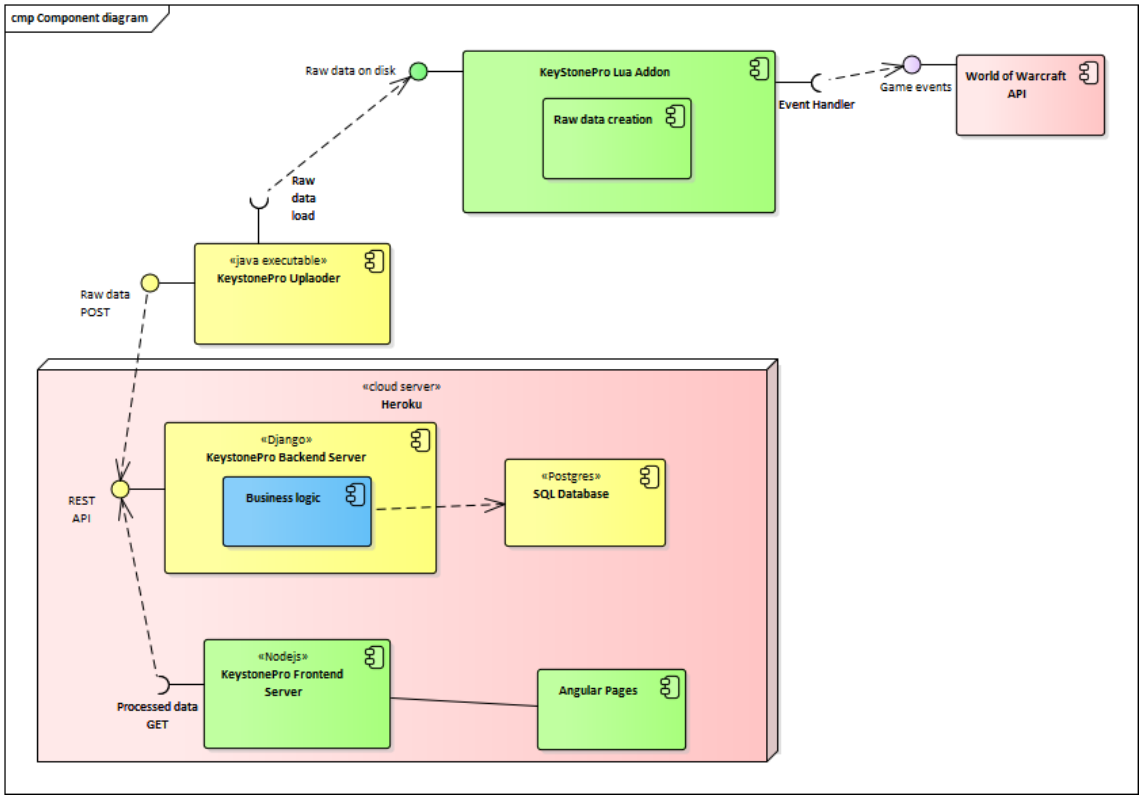
3.9 Nasazení komponent na fyzický server

Ačkoli WoW doplněk a data uploader budou fyzicky nainstalovány na strojích jednotlivých uživatelů, tak backend i frontend musí běžet na nějakém fyzickém serveru. Kvůli nevelkým zkušenostem v tomto odvětví a doporučení vedoucího jsme se rozhodli pro cloudový hosting Heroku, který je velmi dobře zdokumentovaný a na internetu se dají

najít konkrétní návody k nasazování služeb. Jak Python pro backend server, tak nodeJS pro frontend server je na Heroku k dispozici, stejně tak jako SQL databáze, a to konkrétně Postgres. Heroku nabízí různé cenové balíčky, v závislosti na velikosti požadavků, proto je do budoucna zajištěna škálovatelnost aplikace, jak co se týče uložiště, tak výkonu.

3.10 Finální diagram komponent

Na diagramu 3.2 můžete vidět rozkomponované původní komponenty doplněné o konkrétní technologie. Implementaci Lua addonu a Frontendu měl na starosti kolega David Kula (značeno zeleně), já měl na starost implementaci backendu, uploaderu a komunikaci mezi jednotlivými komponenty (značeno žlutě), výpočty na backendu jsme měli rozděleny (značeno modře).



Obrázek 3.2. Finální verze diagramu komponent

Kapitola 4
Detailní analýza systému

4.1 Systémové požadavky

Jelikož už máme navrženou architekturu systému, můžeme určit systémové požadavky. Rozdělíme je na funkční a nefunkční požadavky. Funkční požadavky vychází z byznysových požadavků, zatímco nefunkční požadavky jsou spíše odraz obecné kvality softwaru, která je pro konkrétní systém vyžadována.

4.1.1 Funkční požadavky

- **SRQ 101 Aplikační instrukce:** Systém bude obsahovat popis instalace a ovládání systému, požadavek vychází z BR1.
- **SRQ 102 Zransparentnost sběru dat:** KeyStonePro addon umožní uživateli sledovat, zdali se zapnulo a ukončilo sbírání dat, požadavek vychází z BR2.
- **SRQ 103 Odesílání dat na server:** KeyStonePro uploader nasměruje uživatele k souboru s daty a pomocí POST requestu je odešle na server, požadavek vychází z BR4.
- **SRQ 104 Zpětná vazba na data:** KeyStonePro uploader poskytne zpětnou vazbu o nepoškození nasbíraných dat, požadavek vychází z BR5.
- **SRQ 105 Zpětná vazba o nahrání dat:** KeyStonePro uploader poskytne zpětnou vazbu o úspěšnosti nahrání dat na server, požadavek vychází z BR5.
- **SRQ 106 Vizualizace skóre:** Frontend systému zobrazí skóre libovolného hráče, pomocí požadavku na backend systému ho získá a zobrazí ho na webovém rozhraní, požadavek vychází z BR6.
- **SRQ 108 Transparentnost výpočtu skóre:** Frontend systému poskytne detail výpočtu hráčova skóre, jaké atributy se zohlednily a podobně. Tento detail poskytne pomocí webového rozhraní, požadavek vychází z BR7.
- **SRQ 109 Vizualizace predikce:** Frontend systému zobrazí predikci úspěšnosti dokončení dungeonu v čase libovolné skupiny pěti hráčů, pomocí požadavku na backend systému ji získá a zobrazí ji na webovém rozhraní, požadavek vychází z BR8.
- **SRQ 111 Transparentnost AI modelu:** Frontend systému poskytne detail použitého AI modelu, jaké byli použity atributy, mohutnost trénovacích dat a podobně. Tento popis poskytne pomocí webového rozhraní, požadavek vychází z BR9.
- **SRQ 112 Přeučování AI modelu:** Backend systému bude periodicky přeučovat svůj AI model, kde zohlední nová příchozí data.

4.1.2 Nefunkční požadavky

- **SRQ 201 Škálovatelnost systému:** Počet uživatelů systému se může zvyšovat, v extrémním případě ho můžou používat všichni hráči World of Wacraftu, kterých v únoru 2021 bylo odhadem zaokrouhleně na statisíce 5 200 000. Proto je potřeba, aby systém byl škálovatelný, jak v kapacitě dat, tak ve výkonové zátěži.
- **SRQ 202 Zabezpečení dat:** Uživatelé by neměli mít možnost přepisovat data, aby nedošlo ke zkreslení pravdy. Systém musí zamezit uživatelům úpravy dat.

4.2 Případy užití systému

Modelování případů užití (z angličtiny use cases) vyjadřuje, kdo a jakým způsobem bude používat systém. Zachycuje všechny uživatele systému a jejich práva vůči systému, přičemž uživatel může být i například `čas` nebo jiný systém. Modelování případů užití zachycuje pouze funkční požadavky. Cílem modelování je najít hranice systému a omezit zvětšování řešení. Mimo jiné se podle případů užití může dělat dokumentace či scénáře testování.

Slovní popis případů užití obsahuje hlavní tok, alternativní tok, může obsahovat `pre conditions` a `post conditions`, které popisují stav systému před a po případu užití. [43]

1. USE CASE Poslání mythic+ záznamů
   - Popis: Umožňuje vybrat složku se hrou a odeslat uložená data z disku dále na server k zpracování.
   - Pre conditions: Uložená data z addonu KeyStonePro na disku.
   - Post conditions: Odeslaná data na serveru.
   - Hlavní Tok
     1. Uživatel otevře aplikaci KeyStonePro Uploader a zvolí složku s hrou World of Warcraft
     2. Aplikace najde data na disku a nabídne uživateli data odeslat
     3. Během odesílání dat uživatel vidí co se odesílá a progres odesílání.
     4. Aplikace oznámí úspěšné odeslání
   - Alternativní tok
     1. Uživatel otevře aplikaci KeyStonePro Uploader a zvolí složku s hrou World of Warcraft
     2. Aplikace najde data na disku a nabídne uživateli data odeslat
     3. Během odesílání dat uživatel vidí co se odesílá a progres odesílání.
     4. Aplikace oznámí neúspěšné navázání připojení se serverem nebo neúspěšné odeslání dat z důvodu jejich porušenosti nebo duplicitě.

2. USE CASE Uložení dat na serveru
   - Popis: Umožňuje uložit data do databáze a vhodně je roztřídit a zpracovat.
   - Pre conditions: Odeslaná data od klienta.
   - Post conditions: Uložená a zpracovaná data na serveru.
   - Hlavní tok
     1. Z aplikace na straně klienta přijdou data.
     2. Data jsou v pořádku a server pošle kladnou odpověď klientovi.
     3. Server data serializuje a vhodně uloží do databáze.
     4. Server data upraví do vektorů pro vytvoření modelu a skóre.
   - Alternativní tok
     1. Z aplikace na straně klienta přijdou data.
     2. Data nejsou v pořádku a server pošle informační odpověď klientovi.

3. USE CASE Zobrazení skóre
   - Popis: Umožňuje uživateli zobrazit skóre jakéhokoli hráče.

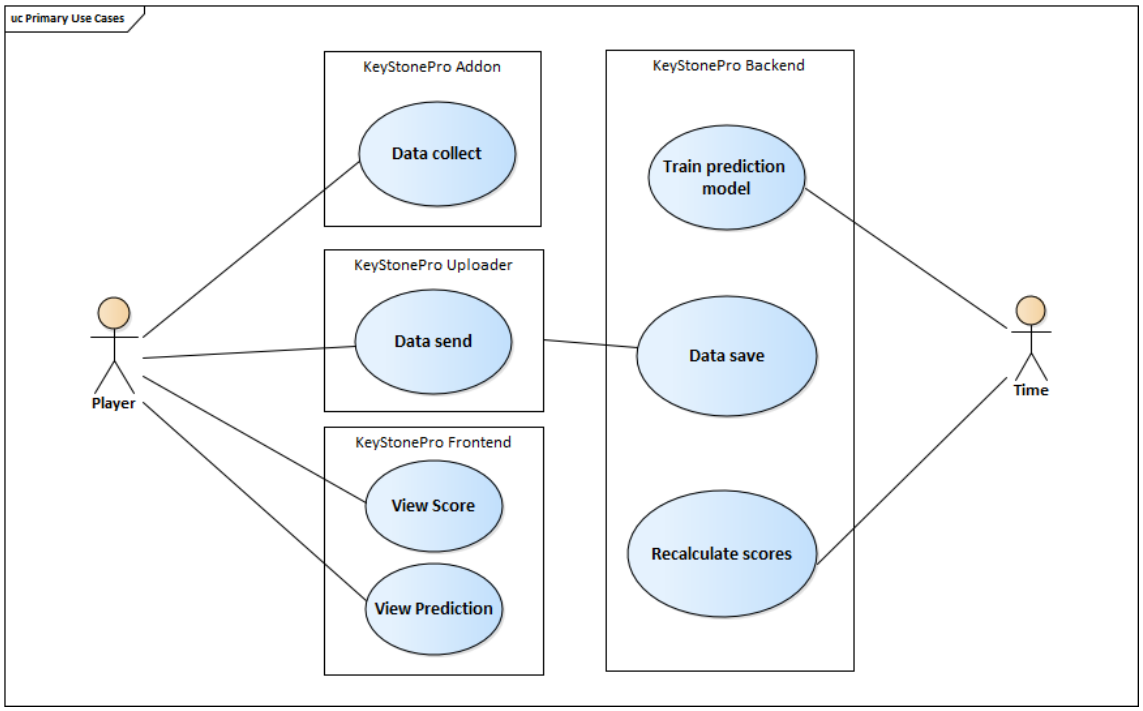
- Pre conditions: Vytvořené skóre daného hráče.
- Post conditions: Zobrazené skóre na frontendu.
- Hlavní tok

  1. Uživatel zažádá o vizualizaci skóre nějakého hráče.
  2. Frontend aplikace skóre i s jeho popisem vykreslí.

- Alternativní tok

  1. Uživatel zažádá o vizualizaci skóre nějakého hráče.
  2. Uživatel neexistuje nebo nemá zaznamenána žádná data na backendu.
  3. Frontend aplikace oznámí uživateli nenalezení skóre daného hráče.

Je zbytečné zde takto popisovat všechny případy užití, zbytek případů užití můžete vidět na diagramu 4.1.



Obrázek 4.1. Use case diagram systému KeyStonePro

Kapitola 5
Návrh a implementace aplikace pro odesílání záznamů

Jelikož nebylo možné, odesílat nasbíraná data z dungeonů přímo pomocí Lua addonu KeyStonePro, byl vytvořen požadavek pro vytvoření samostatné aplikace pro tento účel.

V kapitole 3.3 jsem se rozhodl aplikaci dělat desktopovou, a to pomocí frameworku, který zajistí aplikaci spustit jak na operačním systému Windows a MacOS. V kapitole 3.4 jsem si jako takový framework zvolil JavaFX a tedy programovací jazyk Java. Existují 2 možnosti pro vývoj aplikací v JavaFX, buď pomocí samostatného SDK nebo pomocí automatizačního nástroje pro vývoj jako je Maven nebo Gradle. Jelikož mám s prací s Mavenem zkušenosti, zvolil jsem tuto variantu. [36]

5.1 Parsování Lua do JSON

První problém, kterému jsem musel čelit, byl formát raw dat z addonu KeyStonePro. Formát dat je totiž ve formátu Lua, který není úplně rozšířený a existující frameworky si s ním nedovedou poradit. Nabízely se dvě možnosti, buď na backendu zpracovat data z Lua formátu, což by znamenalo vytvořit si úplně vlastní serializaci dat, nebo data z formátu Lua převést do nějakého běžného formátu, na který dokáže většina frameworků reagovat. Jako takový formát se nabízel například JSON nebo XML. Jelikož Lua a JSON mají velmi podobnou strukturu, zvolil jsem implementovat převod z Lua do JSON.

Pro většinu převodů jsem využil regulárních výrazů a na některé převody jsem si napsal vlastní funkce, které pomocí cyklů projížděly původní text a pomocí podmínek ho vhodně měnily. Pro příklad ve formátu Lua, se pro přiřazení klíče k hodnotě používá znak `=`, zatímco v JSON formátu znak `:`, nebo poslední element v objektu má v Lua za sebou čárku, zatímco v JSONu ne. Ukázka kódu 5.1 ukazuje, jak za pomocí regulárních výrazů docílím těchto převodů.

```
s.replaceAll("=", ":").replaceAll(",(\\r?\\n.+)\\}", " $1}")
```

Výpis kódu 5.1. Nahrazení textu za pomoci regulárních výrazů.

Funkce *replaceAll* nahradí všechny části textu, které odpovídají regulárnímu výrazu v prvním parametru parametrem druhým. První převod je tedy intuitivní, druhý převod popíšu. Všechny záznamy, které mají čárku, za kterou může být neviditelný znak `\r`, za kterým musí být znak pro odřádkování `\n` za kterým bude 1 a více libovolných znaků a za kterými bude znak `}` se nahradí mezerou, tím konkrétním, co je v závorce a znakem `}`. Neviditelný znak `\r` bývá obsažen ve Windows textových souborech, zatímco v UNIX ne, proto je volitelný.

Takových převodů bylo potřeba udělat opravdu spoustu, nebudu je zde zanechávat všechny. Na obrázcích 5.1 a 5.2 můžete vidět porovnání části původního Lua formátu a výsledného JSON formátu.

```
["Grasping Rift"] = {
    [323825] = {
        ["spellId"] = 323825,
        ["count"] = 12,
        ["damageTaken"] = 8882,
        ["spellName"] = "|cff71d5ff|Hspell:323825:0|h[Grasping Rift]|h|r",
    },
    ["sourceName"] = "Grasping Rift",
},
```

Obrázek 5.1. Ukázka obdrženého kouzla protivníkem ve formátu Lua.

```
{
  "sourceName": "Grasping Rift",
  "damageAbility": [
    {
      "spellId": 323825,
      "spellName": "|cff71d5ff|Hspell:323825:0|hGrasping Rift|h|r",
      "count": 12,
      "damageTaken": 8882
    }
  ]
},
```

Obrázek 5.2. Ukázka obdrženého kouzla protivníkem ve formátu JSON.

5.2 Práce s POST http requestem

V moment, kdy byla data v JSON formátu připravena, jsem začal implementovat kód s požadavky na vytvoření POST http requestu a zpracování odpovědi ze serveru. Za pomoci java knihovny `java.net` jsem vytvořil funkci 5.2, která těmto požadavkům vyhoví.

```
private String sendDungeonRecord(String jsonInputString) throws IOException{
    try{
            URL url =
             new URL("http://key-stone-pro.Herokuapp.com/recordeddungeons/");
            HttpURLConnection con = (HttpURLConnection)url.openConnection();
            con.setRequestMethod("POST");
            con.setRequestProperty("Content-Type", "application/json; utf-8");
            con.setRequestProperty("Accept", "application/json");
            con.setDoOutput(true);
            try(OutputStream os = con.getOutputStream()) {
                byte[] input =
                jsonInputString.getBytes("utf-8");
                os.write(input, 0, input.length);
            }
            try(BufferedReader br = new BufferedReader(
                new InputStreamReader(con.getInputStream(), "utf-8"))) {
                    StringBuilder response = new StringBuilder();
                    String responseLine = null;
                    while ((responseLine = br.readLine()) != null) {
                        response.append(responseLine.trim());
                    }
                } catch(java.io.IOException e){
                    return "Bad request";
            }
        }
        catch (MalformedURLException ex) {
            return "Can't reach the server";
    }
    return "OK";
}
```

Výpis kódu 5.2. Poslání dat přes POST request.

Funkce vytvoří POST request na http adresu backendové aplikace s vhodnými parametry requestu, dále použije svého parametru jsonInputString, který je naplněn raw

daty ve formátu JSON a vloží ho do výstupu requestu. Čeká, než přijde odpověď ze serveru a pomocí zachycení výjimek vyprodukuje vhodný výstup funkce.

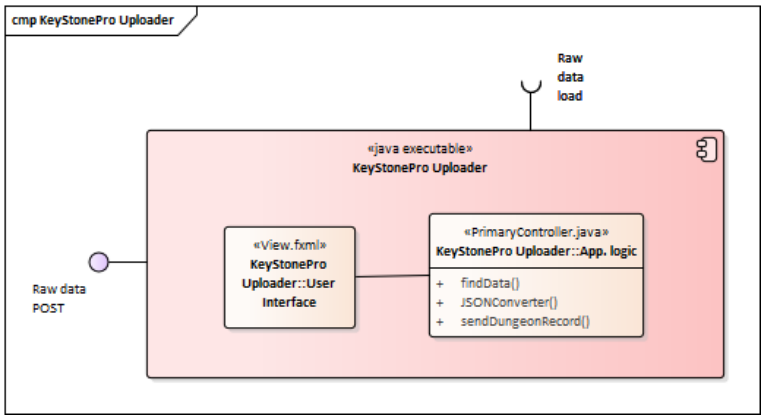
5.3 Uživatelské rozhraní

Z hlediska funkčnosti byla aplikace připravena, nicméně bylo ještě potřeba aplikaci doplnit uživatelským rozhraním, aby měl uživatel přehled, co se právě děje a měl i částečně funkce pod svojí kontrolou. Pro tvorbu uživatelského rozhraní byl použit framework JavaFX.

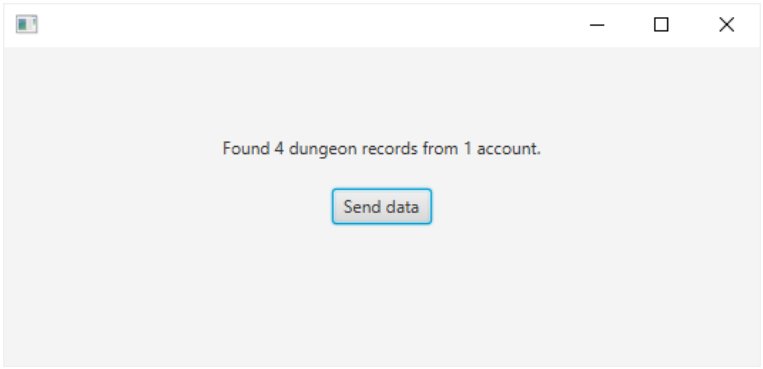
Jelikož aplikace měla pouze jeden hlavní požadavek, a to odeslání dat na server, přišlo mi zbytečné se zaobírat složitými architektonickými vzory uživatelského rozhraní, jako je MVC, MVP, či MVVM. V aplikaci tak je pouze odděleno popsání uživatelského rozhraní, a to za pomocí FXML souboru a logika systému, která se skládá z jedné java třídy, na kterou jsou prvky uživatelského rozhraní navázány.

Uživatelské rozhraní se skládá z popisků, tlačítek a progres baru. Popisky naznačují uživateli, co má v daný moment dělat a na tlačítka jsou navázány funkce načtení a odeslání dat. Progres bar zobrazuje uživateli, v jaké fázi je posílání dat na server, aby podle toho získal odhad, jak dlouho bude ještě akce trvat. Aby se současně mohla posílat data a vykreslovat stav na progres baru, využil jsem technik více vláknového programování, a to konkrétně implementací java třídy `Runnable`.

Na obrázcích 5.3 a 5.4 můžete vidět diagram, kde je zachycená struktura aplikace a ukázku uživatelského rozhraní aplikace.



Obrázek 5.3. Diagram KeyStonePro Uploaderu



Obrázek 5.4. Uživatelské rozhraní KeyStonePro Uploaderu

5.4 Výsledná podoba spustitelné aplikace

Po dokončené implementace zbývalo vyřešit, jak bude vypadat výsledná podoba spustitelné aplikace.

Java je známa svým formátem JAR, který ale pro spuštění vyžaduje na daném zařízení nainstalovaný JRE, tato možnost byla možná, ale v případě, že by uživatel neměl JRE nainstalovaný, mi přišlo nepřívětivé, aby si kvůli takovéto malé aplikaci, musel JRE instalovat.

Druhá možnost byla vytvořit aplikaci jako tzv. Jlink. Jlink je v podstatě složka, která v sobě obsahuje množinu modulů společně s jejich závislostmi, které jsou pro danou aplikaci potřeba. Není proto třeba mít globálně nainstalovaný JRE, protože Jlink má vše potřebné v sobě. [44]

Třetí možností bylo použití nástroje jpackage, který ze souboru JAR vytvoří instalátor pro aplikaci na daném operačním systému. Aplikační rozhraní toho nástroje obsahuje možnosti o koncovce spustitelného souboru jako třeba `exe` nebo `msi` pro Windows, či `pkg` nebo `dmg` pro MacOs, verze aplikace nebo jestli má vytvořit zástupce na ploše či v menu. [45]

V prozatímní fázi vývoje jsem využil možnosti Jlinku, nicméně do budoucna vypadá uživatelsky nejpřívětivější varianta za pomocí nástroje jpackage.

Kapitola 6
Návrh a implementace backendové části aplikace

Backendová část systému pokrývá mnoho požadavků. Perzistence dat, výpočty skóre a predikce a také poskytnutí REST API pro zprostředkování všech těchto klíčových výstupů a příjímání vstupů v podobě raw dat. V kapitole 3.5.2 jsem se rozhodl pro tento účel využít frameworku Django, který jsem si vybral mimo jiné díky přítomnému ORM, jehož implementací jsem začal.

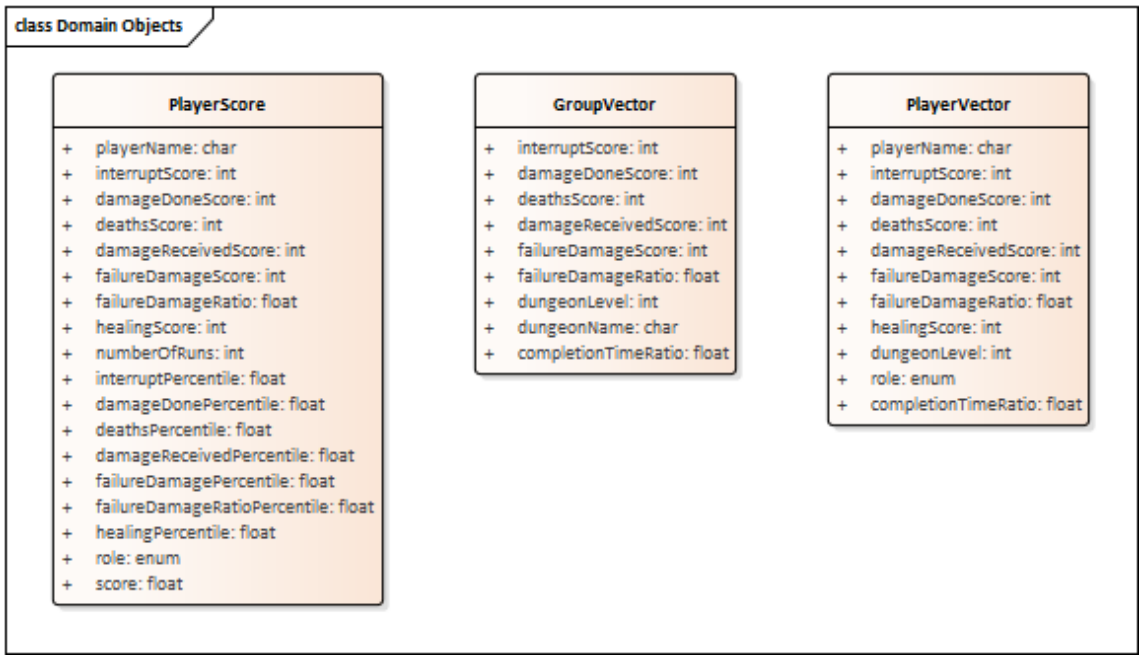
6.1 ORM v Django frameworku

ORM (objektově relační mapování) je technika, která zajišťuje automatickou konverzi dat mezi objektově orientovaným programovacím jazykem a relační databází. V django frameworku k tomuto převodu slouží používání tzv. modelu, kde jsou tato data obsažena. Jeden model většinou odpovídá relační tabulce v databázi a atributy tohoto modelu databázovým polím. Vytvoření jednoho modelu spočívá ve vytvoření třídy, která dědí z rodičovské třídy *django.db.models.Model* a sepsání atributů modelu. [46] Databáze se o změnách v ORM dozví přes takzvané migrace, které obsahují strukturu modelů a jejich změny v průběhu času. Výhodou migrací je možnost jednoduchého nahrání databázového schématu přes jediný příkaz na různé databáze. [47] Této výhody jsem často využil ve vývoji, když jsem přepínal mezi testovací a `produkční` verzí databáze.

První část modelů byla vytvořena 1:1 dle diagramu, který je zachycen na obrázku 2.3, také však byly vytvořeny nové modely GroupVector a PlayerVector, které slouží pro uchování menšího počtu atributů, než mají raw data. Takový převod jsem nazval preprocesing raw dat. Poslední modelem je PlayerScore, který byl rozšířen o několik atributů oproti obrázku 2.2. Na obrázku 6.1 je zachycena entita DamageAbility z obrázku 2.3 převedena do ORM formátu. Na obrázku 6.2 můžete vidět diagram PlayerScore modelu a nových vektor modelů.

```
class DamageAbility(models.Model):
    damageSource = models.ForeignKey(DamageSource, on_delete=models.CASCADE,
                                     related_name="damageAbility")
    spellId = models.IntegerField()
    spellName = models.CharField(max_length=200)
    count = models.IntegerField()
    damageTaken = models.IntegerField()
```

Obrázek 6.1. Ukázka ORM na entitě DamageAbility



Obrázek 6.2. Diagram nových modelů

6.2 REST API v Django frameworku

S připravenými entitami v modelu byl čas na implementaci jejich čtení i vytváření. Pro tento účel bylo v kapitole 3.8 zvoleno aplikační rozhraní REST implementované pomocí frameworku s všeříkajícím názvem `Django REST Framework`. Django REST Framework nabízí hlavní 3 funkcionality: Serializers, Views, Routing.

6.2.1 Serializers

Serializers, je sada tříd a funkcí zajišťující serializaci dat přijatých z requestů do jak ORM, tak no-ORM objektů. Je možnost využijí jak tříd, která fungují jako rozhraní poskytující funkce `create` a `update`, jejichž funkcionalitu musí uživatel doplnit sám, tak tříd, kde jsou typické scénáře již implementovány. [48] Potřeboval jsem uložit raw data z JSON formátu, která budou přicházet od klientů využívající aplikaci KeyStonePro Uploader. Možná jste si všimli, že podle obrázku 2.3 jsou všechny objekty do sebe vnořeny, proto jsem využil tzv. vnořené serializace, kdy jsem ve funkci `create` objektu RecordedDungeon vytvářel objekty v něm vnořené.

```
from rest_framework import serializers
from .models import *

class RecordedDungeonSerializer(serializers.ModelSerializer):
    playerInfo = PlayerRecordSerializer(many=True)

    class Meta:
        model = RecordedDungeon
        fields = ('id', 'dungeon', 'difficulty', 'completionDate', 'onTime',
          'completionTime', 'playerInfo')

    def create(self, validated_data):
        player_records_data = validated_data.pop("playerInfo")
        recorded_dungeon = RecordedDungeon.objects.create(**validated_data)
        for player_record_data in player_records_data:
            healing_done_data = player_record_data.pop("healingDone")
```

```
            missed_interrupts_data = player_record_data.pop("missedInterrupts")
            damage_received_data = player_record_data.pop("damageReceived")
            damage_done_data = player_record_data.pop("damageDone")

            player_record = PlayerRecord.objects.create(
             recordedDungeon=recorded_dungeon,
              **player_record_data)

            HealingDone.objects.create(playerRecord=player_record,
              **healing_done_data)
            for missed_interrupt_data in missed_interrupts_data:
                UninterruptedSpell.objects.create(playerRecord=player_record,
                  **missed_interrupt_data)
            damage_receive_create(player_record, damage_received_data)
            DamageDone.objects.create(playerRecord=player_record, **damage_done_data)
        return recorded_dungeon
```

Výpis kódu 6.1. Serializér entity RecordedDungeon.

Na prvním řádku třídy RecordedDungeonSerializer můžeme vidět odkaz na `serializér` vnořeného objektu PlayerRecord, dále odkaz na model, který je naimportován z modulu .models a jména příslušných atributů. Dále vidíme přepsanou metodu `create`, kde jsou postupně vytvářeny všechny vnořené objekty.

```
class UninterruptedSpellSerializer(serializers.ModelSerializer):

    class Meta:
        model = UninterruptedSpell
        fields = ('spellId', 'count')
```

Výpis kódu 6.2. Serializér entity UninterruptedSpell.

Tento serializér nemá přepsanou třídu `create`, využívá proto výchozí implementaci.

6.2.2 Views

Další sadou tříd a funkcí jsou Views. Ve Views se obsluhují metody requestu, jako je GET, POST, UPDATE a DELETE jednotlivých entit. Podobně jako u Serializers, je možnost využít tříd, kde si chování uživatel nastavuje do určité hloubky sám, nebo využít tříd, které mají typické využití předimplementované. [49]

U metody POST entity RecordedDungeon, jsem nejen chtěl uložit raw data do databáze pomocí ORM, ale také provést preproces dat do vektorů. V případě, že data byla v pořádku, poslal jsem odpověď uživateli a až potom jsem vytvořil preprocesing nových dat. Bylo to z důvodu, abych uživatele nezdržoval. Jelikož odpovědí na požadavek ve výchozím nastavení funkce končí, musel jsem chování přepsat.

```
class ResponseThen(Response):
    def __init__(self, data, then_callback, **kwargs):
        super().__init__(data, **kwargs)
        self.then_callback = then_callback

    def close(self):
        super().close()
        self.then_callback()
```

Výpis kódu 6.3. Přepsaná třída Response třídou ResponseThen, která jako argument bere funkci, která se vykoná po odeslání odpovědi klientovi.

U entit PlayerScore, PlayerVector a GroupVector jsem využil implementace pomocí tzv. ModelViewSet, které potřebují jako atributy uplné minimum, a to množinu objektů entit daného typu a jeho `serializér`. ModelViewSet implementuje všechny metody

POST pro vytvoření nového, UPDATE pro aktualizaci existujícího, DELETE pro smazání existujícího a GET jak pro čtení jednoho, tak všech entit daného typu. Pro účely vývoje byly povoleny všechny metody, ale do budoucna je v u těchto entit plánována jen metoda GET. [50]

```
class GroupVectorViewSet(viewsets.ModelViewSet):
    queryset = GroupVector.objects.all()
    serializer_class = GroupVectorSerializer
```

Výpis kódu 6.4. Implementace view pomocí ModelViewSet

Speciální implementaci view měla také entita Prediction, kde se ověřilo, zda parametry GET požadavku jsou platné a v případě že ano, tak se vytvořila predikce, k jejíž vytváření se v textu dostanu později. Zajímavostí také je, že jako výsledek se u jako jediné entity nevrací objekt ORM, ale pouze dočasný python objekt. Není totiž důvod, predikci nijak uchovávat.

6.2.3 Routers

Poslední zajímavou funkcí jsou Routers, které slouží k provázání url k určitému views.

```
router = routers.DefaultRouter()
router.register(r'playerscore', views.PlayerScoreViewSet)
router.register(r'playervector', views.PlayerVectorViewSet)
router.register(r'groupvector', views.GroupVectorViewSet)
router.register(r'prediction', views.PredictionViewSet, basename='Prediction')
```

Výpis kódu 6.5. Registrace Views systému KeyStonePro za pomoci Routers, u predikce je doplněn atribut `basename`, z důvodu, že predikce nemá množinu objektů ORM.

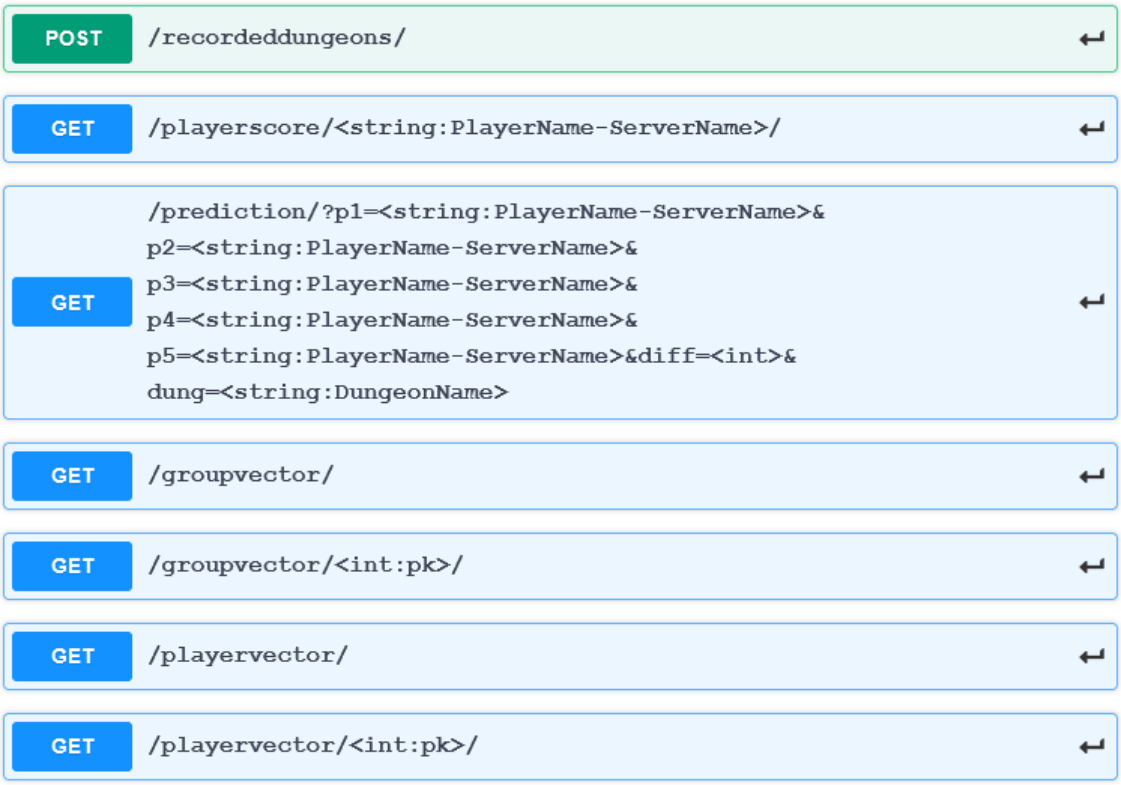
I bez použití Routers lze provázat url k určítému view, jak můžete vidět v ukázce 6.6, kde je provázána poslední zbylá entita RecorderDungeon

```
urlpatterns = [
    path('', include(router.urls)),
    path('recordeddungeons/', views.RecordedDungeonList.as_view()),
    path('recordeddungeons/⟨int:pk⟩/', views.RecordedDungeonDetail.as_view()),
    path('api-auth/', include('rest_framework.urls', namespace='rest_framework'))
]
```

Výpis kódu 6.6. Registrace Views systému KeyStonePro bez pomoci Routers.

6.2.4 Dokumentace API

Základem každého API je dobrá dokumentace, po vzoru `code-first` jsem nejdřív vytvořil API a potom pomocí služby swagger.io vytvořil dokumentaci. Služba swagger.io obsahuje nástroj Inspector, ve kterém když provedete API požadavky dle potřeby, tak z nich dokáže vygenerovat dokumentaci. Na obrázku 6.3 můžete vidět grafické znázornění KeyStonePro API pomocí služby swagger. [51] Musím podotknout, že takto znázorněná dokumentace atributů u požadavku o predikci není validní, nicméně pro ušetření místa, mi to nepřišlo nevhodné a myslím si, že je z toho použití velice intuitivní.



Obrázek 6.3. KeyStonePro API ve swagger.io

- POST RecordedDungeons používá aplikace KeyStonePro Uploader pro nahrání raw dat, metoda přijímá JSON ve specifickém formátu.
- GET PlayerScore využívá frontend KeyStonePro pro zobrazení složek i celkového skóre uživateli. Parametr, podle kterého se skóre vrací je hráčovo jméno se serverem.
- GET Prediction využívá frontend KeyStonePro a vrací vypočtenou predikci pro určitou skupinu hráčů (p1-p5), v určité obtížnosti (diff) a určitém dungeonu (dung).
- GET groupvector, playervector používá kolega David Kula pro testovací účely svých predikčních algoritmů, než je nasadí přímo na backend. K dispozici je čtení všech vektorů, či určitého pomocí jeho ID.

6.2.5 CORS

Důležitou součástí veřejných API je CORS (Cross-Origin Resource Sharing) HTTP-hlavička, která říká, jakým doménám povolí přístup k odpovědím daného API. [52] Pro implementaci CORS v django slouží framework `django-cors-headers`, který se velmi jednoduše postará o specifická omezení. GET requesty KeyStonePro API byly vytvořeny k použití na frontendu KeyStonePro a pro testování vývojáře Davida Kuly, proto jsem povolil přístup pouze pro tyto domény.

```
CORS_ALLOWED_ORIGINS = [
    "http://localhost:4200",
    "https://key-stone-pro-stats.Herokuapp.com",
]
```

Výpis kódu 6.7. Povolené domény k využívání API.

Na první doméně se nachází testovací aplikace Davida a na druhé frontend KeyStonePro systému.

6.3 Preprocesing raw dat

Pro potřeby výpočtu skóre i predikce nebyly raw data vždy úplně vhodná, proto jsem se rozhodl vytvořit vektory, kde budou jednotlivé složky vytvořené z určitého výpočtu nad raw daty. Jak již bylo zmíněno, výpočty těchto vektorů se vykonají bezprostředně po vložení raw dat do databáze, a to v jiném vlákně, aby se nezdržoval požadavek kladený uživatelem. V této sekci chci popsat složky těchto vektorů i s případnými výpočty.

- **completionTimeRatio** - začnu poslední složkou ve vektoru, protože se objevuje v několika dalších výpočtech. Každý dungeon, má jiný časový limit, proto je hodnota z raw dat o dokončení dungeonu v milisekundách sama o sobě nic neříkající. Aby se dala data porovnávat všechna mezi sebou, musí být v poměru vůči časovému limitu konkrétního dungeonu.

```
def completion_time_ratio_calc(self):
    dungeon_timers = {  # in milliseconds
        "De Other Side": 2580000,
        "Halls of Atonement": 1860000,
        "Mists of Tirna Scithe": 1800000,
        "The Necrotic Wake": 2160000,
        "Plaguefall": 2280000,
        "Sanguine Depths": 2460000,
        "Spires of Ascension": 2340000,
        "Theater of Pain": 2220000
    }
    # in milliseconds
    completion_time = self.request_data["completionTime"]
    dungeon = self.request_data["dungeon"]
    return completion_time / dungeon_timers[dungeon]
```

Výpis kódu 6.8. Výpočet složky completionTimeRatio.

- **interruptScore** - ve vektoru PlayerVector se jedná o sečtení všech `missedInterrupts` daného hráče, u GroupVector sečtení `missedInterrupts` všech hráčů.
- **damageDoneScore** - ve vektoru PlayerVector se `effectiveDamageDone` daného hráče dělí složkou `completionTimeRatio`, u GroupVectoru se sečtou všechny takto vypočtené PlayerVectory. Výsledek vyjde jako desetinné číslo, ale zaokrouhlí se na celé.
- **damageReceivedScore** - jedná se o výraz, kde se `totalDamageReceived` daného hráče dělí složkou `completionTimeRatio`. Tento výraz platí pro PlayerVector, u GroupVectoru se je to zase součet PlayerVectorů v jednom dungeon záznamu. Výsledek vyjde jako desetinné číslo, ale zaokrouhlí se na celé.
- **failureDamageScore** - nepřátelé v módu mythic+ používají spousta kouzel, avšak ne všechna jsou nebezpečná, nebo se jim dá nějakým stylem vyhnout. Pokud je však kouzlo nebezpečné a dá se mu vyhnout, přidal jsem ho do tabulky tzv. `failure_spells`. Dále jsem ještě vytvořit tabulku `failure_spells_no_tank`, kde se vyskytují vyhnutelná nebezpečná kouzla, která ale ohrozí hráče jedině pokud nemají roli `TANK`. Prošel jsem všechny záznamy o obdrženém poškození a podíval jsem se, zda je jejich zdroj stejný jako v tabulce, pokud ano, poškození jsem přičetl do složky tohoto vektoru. V PlayerVectoru je hodnota jednoho hráče v rámci jednoho běhu, v GroupVectoru všech hráčů v rámci jednoho běhu.
- **failureDamageRatio** - je pouze poměr složek failureDamageScore a damageReceivedScore vyjádřený jako desetinné číslo.
- **healingScore** - pro PlayerVector je sečteno `totalHealingDone` s `totalAbsorbsDone` a odečteno `totalOverhealingDone` jednotlivého hráče. Celý tento výraz je pak vydělen složkou completionTimeRatio. U GroupVectoru je všech N záznamů posčítáno.

Výsledek vyjde jako desetinné číslo, ale zaokrouhlí se na celé. Výpočet pro PlayerVector je znázorněn pomocí kódu 6.9.

```
def healing_score_calc(self, player_name) -> int:
    healing_done = self.players_data[player_name]["healingDone"]
    return int((healing_done["totalHealingDone"] +
     healing_done["totalAbsorbsDone"] -
     healing_done["totalOverhealingDone"]) / self.completion_time_ratio)
```

Výpis kódu 6.9. Výpočet složky healingScore pro PlayerVector.

- **deathsScore, dungeonLevel, dungeonName, role, playerName** - pouze intuitivní přiřazení hodnot z původních raw dat. DeathsScore je individuální v PlayerVector, v GroupVector součet všech.

Implementace vektorových výpočtů byla vytvořena jako implementace programovatelného rozhraní, může se proto takových druhů výpočtu udělat mnoho. Formát výstupu ale zůstane stejný.

6.4 Výpočet skóre

Již první myšlenky vytvoření systému KeyStonePro byly spjaty s výpočtem skóre, které bude dbát více na individuální výkon hráče než celého týmu, jako protiklad hlavního konkurenčního systému `Raider.IO`. Této filisofie jsem se také snažil celou dobu držet.

V rámci vývoje David Kula používal algoritmus Perceptron pro určení predikce. Líbilo se mi, že Perceptron určí váhu jednotlivým složkám vektorů a chtěl jsem tyto váhy i použít ke tvorbě skóre. Později se však ukázalo, že Perceptron je nevhodný z důvodu toho, že složky vektoru nebyly lineárně separabilní.

Z tohoto důvodu jsem musel najít jiný způsob výpočtu. Rozhodl jsem se porovnávat zvlášť úspěšnost v určitých oblastech hráčů mezi sebou a vytvořit tím percentily. To znamená, že pokud má hráč percentil 93.0, tak 7 procent hráčů je lepších než on a 93 procent hráčů ne. V některých oblastech je nesmyslné porovnávat skóre vůči jiným rolím. Například `HEALER` nedá nikdy takové poškození jako `DAMAGE` a naopak `DAMAGE` nikdy nevyléčí tolik jako `HEALER`. Proto by se výpočet skóre dal shrnout jako percentilní počet v různých oblastech s ohledem na herní role. Celkové skóre je průměr těchto percentilů.

Prozatím je skóre přepočítáno po vložení nového záznamu raw dat do databáze, nicméně do budoucna budou skóre přepočítávána vždy periodicky např. 1x denně a to tak, že se zprůměrují všechny PlayerVectory daného hráče a vůči ostatním průměrům se vytvoří percentily.

Výpočet skóre není uzavřená kapitola našeho systému, po boku predikce je to funkce, která náš systém od konkurenčních odlišuje, proto je v plánu ještě mnoho vylepšení, které budou brát v potaz další faktory v průběhu módu mythic+.

6.5 Výpočet predikce

Výpočet predikce měl na starost kolega David Kula. David se ve své práci zmiňuje, jaké algoritmy zkoušel a jak je implementoval. Finálně zvolený algoritmus byl Adaboost.

Cílem v této části bylo natrénovat klasifikátor tak, aby rozpoznal úspěšně dokončený dungeon včas od neúspěšného dokončení po vypršení času - depletu. Pro trénování David rozdělil datovou množinu v poměru 48:16 na trénovací sadu a testovací sadu, kde

jeden prvek datové množiny představuje jeden běh dungeonem v módu mythic+. Obměňováním složení trénovací a testovací sady se chybovost pohybovala od 14% po 25%. Tyto veliké změny v chybovosti měl na svědomí nedostatek dat, a proto natrénování klasifikátoru záleželo na jednotlivém vzorku dat více, než by mělo. Na finální testovací sadě byla dosažena celková chybovost 18,75%, z toho 0% na třídě úspěch a 37,5% na třídě neúspěch/deplete. Toto rozdělení bylo zvoleno proto, protože poměr úspěchů a neúspěchů v reálném prostředí zhruba odpovídal poměru v testovací sadě. [53]

6.6 Nasazení na Heroku

Poslední důležitou částí spjatou s backendem bylo jeho nasazení na server a internet. V kapitole 3.9 jsem se k tomuto účelu rozhodl použít cloudový server Heroku. K nasazení aplikace na Heroku je potřeba učinit několik kroků.

Jedním z nich, je propojení Heroku s repositářem aplikace na portálu Github, nebo vytvoření nového Heroku git repositáře. Rozhodl jsem se pro druhou variantu, která byla velmi přímočará. Dalším krokem byla instalace app-serveru, jelikož při nasazení již není možné použít vývojový server, který je součástí frameworku Django. Jako takový server jsem se rozhodl využít Waitress, hlavně kvůli podpoře operačního systému Windows, na kterém jsem vyvíjel. Tento Waitress server jsem pouštěl při každém nasazení aplikace na Heroku. K tomuto účelu na Heroku slouží tzv. Procfile jehož součástí jsou příkazy, které jsou puštěny při každém nasazení aplikace. Můj Procfile obsahuje následující řádek.

```
web: waitress-serve --listen "*:$PORT" --threads 30 djangoProject.wsgi:application
```

Výpis kódu 6.10. Procfile backendové aplikace.

Příkaz pouští Waitress server na portu proměnné PORT, kterou spravuje Heroku, jako další parametr mám počet vláken a zdrojový soubor pro start aplikace.

Kapitola 7
Vizualizace skóre a predikce

Společnou částí zadání s Davidem Kulou bylo vytvoření prezentační vrstvy neboli frontendu a tedy vizualizace skóre a predikce. Během vývoje a po domluvě s vedoucím naší práce jsme se rozhodli, že implementaci této části bude dělat kolega David. Nicméně i já jsem přispěl lehkou vizualizací backendových dat a nasazením frontendu na Heroku.

7.1 Vizualizace backendových dat

V minulé kapitole jsem psal o `Django REST framework` a jeho hlavních částech. Součástí tohoto frameworku je i HTML výstup všech zdrojů API, který je přívětivější pro člověka než pouhý JSON. Tento HTML výstup je dobrý pro snadné procházení API zdrojů, stejně jako provedení podporovaných requestů. Toho jsme ostatně i patřičně využívali při testování našeho API. Tato API vizualizace je postavena na frameworku `Bootstrap` a je snadné jejich zdrojové soubory přepsat. Pro naše účely stačilo užití výchozí verze. [54]

7.2 Nasazení prezentační vrstvy na Heroku

Taktéž jako backend, i frontend jsem nasadil na cloudový server Heroku a opět jsem využil Heroku git repozitáře. Konkrétně jsem nasazoval Angular CLI aplikaci za pomoci Nodejs frameworku Express. Kód 7.1 je kód souboru server.js, který se pouští při každém nahrání aplikace. Nejprve se načte framework express, dále se použije funkce pro přesměrování http protokolů na https, definuje se cesta pro statické soubory, definuje se přesměrování jakýchkoli subdomén na kořenový index.html a aplikace se spustí na správném portu.

```
const express = require('express');
const app = express();

function requireHTTPS(req, res, next) {
  // The 'x-forwarded-proto' check is for Heroku
  if (!req.secure && req.get('x-forwarded-proto') !== 'https') {
    return res.redirect('https://' + req.get('host') + req.url);
  }
  next();
}

app.use(requireHTTPS)
app.use(express.static('./dist/KeystonePro'));

app.get('/*', function (req, res){
  res.sendFile('index.html', {root: 'dist/KeystonePro/'});
});

app.listen(process.env.PORT || 8080);
```

Výpis kódu 7.1. server.js - startovní kód express frameworku. [55]

Kapitola 8
Instalace KeyStonePro

8.1 Instalace addonu

Addon nemá žádné instalační skripty a ani jej není třeba nějak kompilovat či instalovat. Zdrojový kód addonu stačí umístit do správné složky v adresáři s hrou World of Warcraft a hra si potom sama kód najde, přeloží a spustí.

Nacházíme-li se v kořenovém adresáři hry, pak extrahujeme složku s kódy do složky ./_retail_/Interface/AddOns/. Po příštím přihlášení do hry by měl být addon aktivní a připravený ke sběru dat. Můžeme se o tom přesvědčit v sekci výběru postav před nalogováním do hry. Vlevo dole je tlačítko AddOns a po kliknutí na něj se objeví seznam, kde by měl být addon zobrazen jako KeystonePro a jeho políčko by mělo být zaškrtnuto. Dále je doporučeno mít zaškrtnuto také políčko v pravém horním rohu okna se seznamem addonů `Load out of date AddOns`.

Obrázek 8.1. AddOn interface přímo ve hře

Na obrázku můžeme vidět AddOn interface přímo ve hře. Lze v něm zapínat a vypínat načtené addony z adresáře s hrou. [53]

8.2 Instalace uploaderu

Uploader nemusí být instalován, stačí si rozbalit složku KeyStonePro Uploader ve kterém se nachází skript pro OS Windows, nebo pro MacOS.

V kořenovém adresáři složky KeyStonePro Uploader najdeme skripty KeyStonePro_Uploader.bat pro Windows nebo ve složce bin/KeyStonePro_Uploader pro MacOS. Pro správné fungování aplikace, musíme být odhlášeni ze hry.

Kapitola 9
Testování

9.1 Testování uživateli

Náš systém jsme po prvních osobních testech vystavili k otestování členům World of Warcraft guildy[1] `Monkeys`.

Co se týče částí, na kterých jsem pracoval, tak jako první nedostatek vyšlo neošetření ukládání duplicitních běhů módu mythic+. Jde o to, že mód mythic+ absolvuje parta pěti hráčů, ale stačí, aby pouze jeden hráč měl systém KeyStonePro. Pokud ho má více členů party ničemu to nevadí, ale všichni sbírají duplicitní data. Systém nechce obsahovat stejný záznam vícekrát, proto se tento problém vyřešil databázovým omezením o unikátnosti.

Další nedokonalost, která vyšla z pozorování od uživatelů, bylo neintuitivní zdůraznění, že komponenta KeyStonePro Uploader lze použít pouze pokud má uživatel vypnutou hru. Tento nedostatek se zatím vyřešil upozorněním v READ.ME souboru k této komponentě, nicméně do budoucna by bylo dobré implementovat nějaké výraznější upozornění, nebo přímo neumožnění toto udělat.

9.1.1 Testování důležitosti jednotlivých složek skóre

Naše skóre je složeno z vícero faktorů: neúspěšnost přerušování kouzel (interruptScore), udělené poškození za vteřinu (damageDoneScore), obdržené poškození, kterému se dá vyhnout (failureDamageRatio) a léčení za vteřinu (healingScore). Chtěli jsme otestovat, zdali je každá z těchto složek důležitá pro čas dokončení dungeonu. Vydali jsme se tedy se stejnou skupinou pěti hráčů otestovat různé bizarní případy. Aby bylo vůči čemu srovnávat, musím zmínit, že skupina v našem složení většinou dojde mód mythic+ na úrovni 14 v rozmezí 0.7x - 0.9x časového limitu dungeonu.

První případ byl takový, že hráči nebudou vůbec používat svá přerušovací kouzla, tedy hodnota neúspěšnosti přerušování bude velmi vysoká. Jako dungeon byl zvolen `Mists of Tirna Scithe` úrovně 15. První část dungeonu šla velmi hladce a skoro to vypadalo, že dungeon dokončíme jen s mírným zpožděním oproti normálu. Nicméně na posledním bossovi jsme se zdrželi velice dlouho a výsledek byl 1.7x časového limitu dungeonu. Z tohoto testu tedy lze konstatovat, že složka interruptScore je pro tvorbu skóre relevantní.

Druhý případ měl scénář, že nikdo ze skupiny nebude mít roli `TANK`, to znamená, že ve skupině nebude nikdo, kdo vydrží hodně obdrženého poškození, a tedy neušetří práci hráči který léčí. Zároveň ale bude přítomen jeden `DAMAGE` navíc, což může urychlit rychlost zabíjení nepřátel. Jako dungeon byl zvolen `The Necrotic Wake` úrovně 14. Po prvních minutách v dungeonu, jsme se rozhodli vzdát a snížit úroveň na 13. A překvapivě se nám velmi zadařilo, dungeon jsme došli 0.7x časového limitu. Z toho by mělo vyplynout, že role `TANKA` není důležitá, pravda je ale spíše taková, že na úroveň

[1] Guilda je ve hře World of Warcraft skupina hráčů, která spolu často překonává herní výzvy, na které je potřeba více hráčů.

dungeonu 13 už naše parta má velmi dobré vybavení a proto `TANK` nebyl potřeba. Rozdíl byl vidět jen u úrovně 14, kde už byly problémy.

Poslední dva případy měly otestovat závislost vybavení hráčů vůči obtížnosti dungeonu. Na vybavení hráče jsou závislé faktory udělení poškození za vteřinu, tak léčení za vteřinu. Nejprve jsme s naší partou šli dungeon `Sanguine Depths` úrovně 2, který jsme dokončili v 0.3x časového limitu. Poté, jsme si sundali některé části vybavení a šli jsme na dungeon `Theater of Pain` úrovně 14. Byl to opravdu špatný výsledek, dungeon jsme dokončili v 2.3x časového limitu. Udělené poškození a léčení za vteřinu tedy hraje velkou roli.

Složku failureDamageRatio bylo těžké otestovat, jelikož hráči v naší skupině už mají tak zaryté reflexy, že by pro ně bylo těžké se nevyhýbat kouzlům, kterým se už snaží půl roku vyhýbat. A také už morálka hráčů po posledním téměř dvouhodinovém běhu velmi klesla.

9.2 Porovnání relevantnosti skóre s existujícími systémy

Populární existující systémy měřící skóre v módu mythic+ jsou `Raider.IO` a `Warcraft Logs`, viz. kapitola 1.3. V tabulce 9.1 je 10 hráčů, kteří mají v našem systému skóre od 60 do 100 a v tabulce 9.2 je 10 hráčů, kteří mají skóre v rozmezí 0-40. Tito hráči jsou porovnáni vůči jejich skóre z konkurenčních systémů. Poznamenám, že Raider.IO skóre se pohybuje od 0 do cca 3000, Nicméně skóre větší než 2000 už dosahuje velmi nízké procento hráčů. Warcraft Logs skóre je průměr percentilů za všechny hráčovy běhy pro jeho specializaci, tedy pohybuje se mezi 0 a 100 stejně jako náš addon KeyStonePro.

Player name	KeyStonePro	Raider.IO	Warcraft Logs
Sneakyball-Thrall	91.0	1,536.4	94.8
Älde-Uldum	90.6	1,403.6	97.8
Nivasboy-Zenedar	84.1	546.1	47.9
Rastafüry-Archimonde	80.1	1,244.1	86.8
Snuwut-Draenor	78.4	1,211.4	84.3
Flappydab-Blackrock	78.1	758.1	56.1
Rykerros-Alonsus	76.2	1,365.1	98.2
Sickyo-Thrall	72.4	1,360.3	94.1
Dudris-Thrall	69.5	974.9	74.2
Snövit-Blackhand	65.2	1,374.7	97.2

Tabulka 9.1. Porovnání skóre hráčů s nadprůměrným KeyStonePro hodnocením s ostatními systémy.

Player name	KeyStonePro	Raider.IO	Warcraft Logs
Najaha-TwistingNether	37.4	1,176.9	71.7
Elsje-Doomhammer	35.3	1,298.7	93.2
Slagschaduw-Sunstrider	32.5	1,480.0	96.7
Hollyragnar-Kazzak	32.5	640.3	27,2
Warcore-TwistingNether	29.9	1,182.7	78.9
Øneshøt-Kazzak	29.4	908.4	70.7
Grönflicka-Defias Brotherhood	26.4	585.0	74.6
Skhizm-BurningSteppes	25.9	715.9	42.5
Keriggen-Drak'thul	18.9	332.0	39.1
Bigbadbull-Saurfang	17.7	674.1	48.1

Tabulka 9.2. Porovnání skóre hráčů s podprůměrným KeyStonePro hodnocením s ostatními systémy.

Z tabulek můžete vidět, že u většiny hráčů odpovídá vysoké skóre našeho addonu i vysokému skóre z addonů jiných a naopak. Jsou zde ale výjimky, a to hlavně z důvodu malého vzorku nasbíraných dat. Většina hráčů v tabulce má totiž v našem systému jen jeden zaznamenaný běh, za to v jiných systémech desítky i stovky. Proto se některému hráči nemusel povést jeden běh a hned je u nás hodnocen negativně zatímco podle jiných systémů se zdá, že je dobrý, a naopak.

Další faktor, který může hrát v odlišných skóre roli, je ten, že systém KeyStonePro úplně nelze porovnat 1:1 s ostatními systémy. Skóre je totiž hodnoceno z komplexnějších dat než u ostatních systémů. Ostatní systémy například vůbec nezohledňují počet neúspěšných přerušení nebo počet obdrženého poškození z kouzel, kterým se dá vyhnout. Raider.IO má skóre z nejlepších časů a nejlepších úrovní mythic+ módu daného hráče a Warcraft Logs porovnává udělené poškození a léčení mezi ostatními hráči. Tedy u ostatních systémů hráč, který dostane vyboostěno[1] může dosáhnout stejně vysokého skóre, jako hráč, který je ve všech aspektech hry dobrý.

[1] Tento termín se používá v kontextu, kdy je skupina složena z většiny dobrých a menšiny špatných hráčů, tito špatní hráči jsou vyboostěni těmi dobrými.

Kapitola 10
Závěr

Cílem mé bakalářské práce bylo vytvořit doplněk hry World of Warcraft, který pokryje nedostatky konkurenčních produktů a poskytne i nové funkce v odvětví módu mythic+. Úvod jsem věnoval zasvěcení do herního průmyslu a hry World of Warcraft a prozkoumal jsem konkurenční systémy, od čehož se odrazilo tvoření a sběr požadavků. Provedl jsem analýzu systému, pomocí diagramů znázornil entity, procesy a komponenty systému, zmapoval možnosti implementace a zvolil nejvhodnější. Věnoval jsem se návrhu a implementaci aplikace pro odesílání dat, aplikačního rozhraní serveru a s tím spjatou komunikaci mezi komponenty, výpočtu skóre a úpravě a perzistenci dat. Zajistil jsem nasazení všech komponent na internet a rozdistribuoval mezi užší komunitu k otestování. Společně s ostatními jsem systém otestoval a postupně přicházel na možná vylepšení, která jsou vyjmenována v následujících bodech.

- **Zobrazování skóre a predikce přímo ve hře** - Ačkoli by se mohlo zdát, že zobrazení skóre na webu stačí, nemusí tomu tak být. Pro hráče je pohodlné mít ve hře všechno při ruce bez používání jiných aplikací. Nicméně uživatel bude stejně muset do hry data aktualizovat.
- **Vylepšení KeyStonePro Uploaderu** - KeyStonePro Uploader je v současnosti složka obsahující spouštěcí skript, což rozhodně není uživatelsky nejpřívětivější. Do budoucna bude vhodný upgrade na instalační aplikaci. Dále by bylo dobré, aby uploader výrazněji upozornil na zavření World of Warcraft, nebo se vůbec nemohl pustit současně se hrou z důvodu možné neaktuálnosti souboru s raw daty.
- **Vytvoření účtů** - Bylo by dobré, aby si hráči v rámci systému mohli vytvořit účet, a to z hlediska více aspektů. Za prvé to může pomoci při zabezpečení dat, jelikož se potenciální pachatel bude dát odhalit a zablokovat. Za druhé to dává možnost vytvoření více druhů účtů, například VIP a klasický. VIP účet by za příplatek dovoloval využití prioritní fronty při aktualizaci skóre, zatímco normální účet by si musel počkat na pravidelnou aktualizaci všech.
- **Sběr více druhů dat** - Mód mythic+ je velmi komplexní disciplína a ačkoli náš systém pokrývá více odvětví než konkurenční, je pořád spousta prostoru na vylepšení. Náš systém zohledňuje pouze přerušitelná kouzla, ale existují i nepřerušitelná kouzla, která jdou přerušit jen speciálními kouzly, takzvanými `stuny`. Měření vhodného použití `stunu` by mohl být další atribut do našeho skóre.
- **Nezapočítávání extrémů** - Jak bylo zmíněno v kapitole 9.2 mohou se stát případy, kdy se hráči extrémně run povede, nebo extrémně nepovede. Takové případy mohou jeho skóre až nežádaně ovlivnit. Proto by bylo dobré, takovéto extrémní případy nezapočítávat.
- **Vylepšení výpočtu skóre** - Se sběrem nových druhů dat, se bude moct stát, že už mezi sebou hráči budou hůře porovnatelní, a tedy percentilní vyhodnocování skóre bude muset být nahrazeno jiným.
- **Nová webová doména** - Momentální doména *key-stone-pro-stats.herokuapp.com* rozhodně není vhodná pro systém s cílovou skupinou tisíců lidí. Proto bude dobré, minimálně odstranit subdoménu a nahradit ji za nynější doménu.

- **Nasazení addonu na curseforge** - Všechny addony jsou ke stažení na portálu curseforge. Proto je důležité tam nahrát i náš addon.

Vytvoření analýzy celého systému, aplikace v podobě Jlink a hlavně práce s REST Api a ORM technologiemi, které pro mě obě byly novinkou jsou mi velkým přínosem.

V následujících měsících Blizzard chystá novou aktualizaci do hry World of Warcraft, ve které by měl přidat in-game systém, který bude fungovat podobně jako nynější `Raider.IO`. Pokud by tomu tak bylo a tedy `Raider.IO` by kvůli tomu šlo do ústraní, tak je to velmi dobrá příležitost pro systém KeyStonePro.

Literatura

[1] Simon Tripp, Martin Grueber, Joseph Simkins a Dylan Yetter. *"Video games in the 21st century: The 2020 Economic Impact Report". TEConomy Partners, LLC.* 2019.
https://www.theesa.com/wp-content/uploads/2019/02/Video-Games-in-the-21st-Century-2020-Economic-Impact-Report-Final.pdf.

[2] Lubor Kopecký. *"Český herní průmysl 2020*. 2020.
https://herniprumysl.cz/HerniPrumyslCR2020.pdf.

[3] Yuki Nakamura. *"Peak Video Game? Top Analyst Sees Industry Slumping in 2019"*. 2019.
https://www.bloomberg.com/news/articles/2019-01-23/peak-video-game-top-analyst-sees-industry-slumping-in-2019.

[4] Tom Wijman. *"Global Game Revenues Up an Extra $15 Billion This Year as Engagement Skyrockets"*. 2020.
https://newzoo.com/insights/articles/game-engagement-during-covid-pandemic-adds-15-billion-to-global-games-market-revenue-forecast.

[5] Leigh Achterbosch, Robyn Pierce a Gregory Simmons. *"Massively multiplayer online role-playing games: the past, present, and future". Computers in Entertainment (CIE).* 2008.
https://dl.acm.org/doi/10.1145/1324198.1324207.

[6] Juho Hamari a Max Sjöblom. *"What is eSports and why do people watch it?", Internet Research, Vol. 27 No. 2, pp. 211-232.* 2017.
https://doi.org/10.1108/IntR-04-2016-0085.

[7] Aaron Alford. *"The average 2020 LCS player salary is reportedly $410,000"*. 2020.
https://dotesports.com/league-of-legends/news/the-average-2020-lcs-player-salary-is-reportedly-410000.

[8] Daniel Rigon. *"H2K Gaming founder: LEC players make around 250,000 euros per year"*. 2019.
https://www.espn.com/esports/story/_/id/27668371/h2k-gaming-founder-lec-players-make-250000-euros-per-year.

[9] *World of Warcraft official site.*
https://worldofwarcraft.com/en-us/. Accessed: 2021-05-01.

[10] *Race, Wowpedia.*
https://wowpedia.fandom.com/wiki/Race. Accessed: 2021-05-01.

[11] *Class, Wowpedia.*
https://wowpedia.fandom.com/wiki/Class. Accessed: 2021-05-01.

[12] *Faction, Wowpedia.*
https://wowpedia.fandom.com/wiki/Faction. Accessed: 2021-05-01.

[13] *Player vs. Enviroment, Wowwiki*.
https://wowwiki-archive.fandom.com/wiki/Player_vs._Environment. Accessed: 2021-05-01.

[14] *Player vs. Player, Wowwiki*.
https://wowwiki-archive.fandom.com/wiki/Player_vs._Player. Accessed: 2021-05-01.

[15] *World first, Wowwiki*.
https://wowwiki-archive.fandom.com/wiki/World_first. Accessed: 2021-05-01.

[16] *Vanilla Raid History of World First Kills - World of Warcraft - Method*.
https://www.method.gg/raid-history. Accessed: 2021-05-01.

[17] *BlizzCon 2008 Arena Tournament*. 2019.
https://liquipedia.net/worldofwarcraft/Arena_World_Championship/2008. Accessed: 2021-05-01.

[18] *Mythic plus, Wowwiki*.
https://wowwiki-archive.fandom.com/wiki/Mythic_plus. Accessed: 2021-05-01.

[19] *Everything About Mythic+ Keystones and Dungeons*. 2021.
https://www.wowhead.com/mythic-keystones-and-dungeons-guide. Accessed: 2021-05-03.

[20] *Mythic Dungeon International - WoW*.
https://worldofwarcraft.com/en-us/esports/mythic. Accessed: 2021-05-03.

[21] *World of Warcraft Rankings for Mythic+ and Raid Progress*.
hhttps://raider.io/. Accessed: 2021-05-06.

[22] *Co je to Logování?, IT Slovník*.
https://it-slovnik.cz/pojem/logovani. Accessed: 2021-05-19.

[23] *Warcraft Logs - Combat Analysis for Warcraft*.
https://www.warcraftlogs.com/. Accessed: 2021-05-06.

[24] *Learn What Stakeholders Are and the Roles That They Play*.
https://www.investopedia.com/terms/s/stakeholder.asp. Accessed: 2021-05-09.

[25] *The Kano Model – A tool to prioritize the users' wants and desires*.
https://www.interaction-design.org/literature/article/the-kano-model-a-tool-to-prioritize-the-users-wants-and-desires. Accessed: 2021-05-09.

[26] Martin Komárek. *Textově formulované požadavky a jejich atributy. Předmět Softwarové inženýrství 2019/2020*. 2019.
https://moodle.fel.cvut.cz/pluginfile.php/213891/course/section/40891/SpecifikacePozadavkuprednaska2a.pdf.

[27] *Create a Business Domain Model, Sparx Systems*.
https://sparxsystems.com/enterprise_architect_user_guide/14.0/model_domains/create_the_fact_model.html. Accessed: 2021-05-09.

[28] *Mythic+ Dungeon Affixes, Wowhead*.
https://www.wowhead.com/affixes. Accessed: 2021-05-07.

[29] *Business Process Modeling: Definition, Why, Technique and Benefits*. 2019.
https://kissflow.com/workflow/bpm/business-process-modeling/. Accessed: 2021-05-10.

[30] *Backend*.
https://techterms.com/definition/backend. Accessed: 2021-05-07.

[31] *About, Lua.*
https://www.lua.org/about.html. Accessed: 2021-05-19.

[32] *World of Warcraft System Requirements, Blizzard Support.*
https://us.battle.net/support/en/article/76459. Accessed: 2021-05-10.

[33] *Webová, nativní a hybridní aplikace: srovnáváme pro a proti.* 2021.
https://www.rascasone.com/cs/blog/webova-nativni-hybridni-aplikace-klady-zapory. Accessed: 2021-05-10.

[34] *About Qt.*
https://wiki.qt.io/About_Qt. Accessed: 2021-05-10.

[35] *Quick Start Guide, Electron.*
https://www.electronjs.org/docs/tutorial/quick-start. Accessed: 2021-05-10.

[36] *JavaFX.*
https://openjfx.io/. Accessed: 2021-05-10.

[37] *The Java Language Environment.*
https://www.oracle.com/java/technologies/introduction-to-java.html. Accessed: 2021-05-10.

[38] *Flask vs Django: How to Understand Whether You Need a Hammer or a Toolbox.* 2019.
https://steelkiwi.medium.com/flask-vs-django-how-to-understand-whether-you-need-a-hammer-or-a-toolbox-39b8b3a2e4a5. Accessed: 2021-05-11.

[39] *World of Warcraft, The Game Statistics Authority.* 2021.
https://activeplayer.io/world-of-warcraft/. Accessed: 2021-05-11.

[40] *Documentation, Django.*
https://docs.djangoproject.com/en/3.2/ref/databases/. Accessed: 2021-05-11.

[41] *Angular.*
https://angular.io/guide/what-is-angular. Accessed: 2021-05-19.

[42] Lukáš Melena. *Metodika návrhu RESTful API. Diplomová práce. Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky. Vedoucí práce Ing. Libor Gála.* 2017.

[43] Martin Komárek. *Modelování požadavků pomocí UML - digram případů užití a scénáře případů užití . Předmět Softwarové inženýrství 2019/2020.* 2019.
https://moodle.fel.cvut.cz/pluginfile.php/213891/course/section/40891/P%C5%99edn%C3%A1%C5%A1ka4a5_PripadyUzitiKompletV2.pdf.

[44] *Java Platform, Standard Edition Tools Reference.* 2017.
https://docs.oracle.com/javase/9/tools/jlink.htm. Accessed: 2021-05-14.

[45] *Java Platform, Standard Edition Java API Reference.* 2020.
https://docs.oracle.com/en/java/javase/15/docs/specs/man/jpackage.html. Accessed: 2021-05-14.

[46] *Models, Documentation. Django.*
https://docs.djangoproject.com/en/3.2/topics/db/models/. Accessed: 2021-05-16.

[47] *Migrations, Documentation. Django.*
https://docs.djangoproject.com/en/3.2/topics/migrations/. Accessed: 2021-05-16.

[48] *Serializers - Django REST framework.*
https://www.django-rest-framework.org/api-guide/serializers/. Accessed: 2021-05-16.

[49] *Views - Django REST framework*.
https://www.django-rest-framework.org/api-guide/views/. Accessed: 2021-05-16.

[50] *Viewsets - Django REST framework*.
https://www.django-rest-framework.org/api-guide/viewsets/. Accessed: 2021-05-16.

[51] *Documenting Your Existing APIs: API Documentation Made Easy with OpenAPI & Swagger*.
https://swagger.io/resources/articles/documenting-apis-with-swagger/. Accessed: 2021-05-16.

[52] *Cross-Origin Resource Sharing (CORS) - HTTP: MDN*.
https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS. Accessed: 2021-05-16.

[53] David Kula. *Systém pro predikci úspěšnosti a výpočtu skóre módu „mythic+“ ve hře World of Warcraft. – Datový model, sběr dat, logika systému a vizualizace dat. Bakalářská práce. ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE, Fakulta elektrotechnická. Vedoucí práce Ing. Jiří Šebek.* 2021.

[54] *The Browsable API - Django REST framework*.
https://www.django-rest-framework.org/topics/browsable-api/. Accessed: 2021-05-19.

[55] *How to Deploy Your Angular 9 App to Heroku in Minutes*. 2021.
https://betterprogramming.pub/how-to-deploy-your-angular-9-app-to-heroku-in-minutes-51d171c2f0d. Accessed: 2021-05-19.

Příloha A
Zkratky

API	Application Interface
CLI	Command line interface
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IT	Information technology
JAR	Java Archive
JRE	Java Runtime Environment
JSON	JavaScript Object Notation
MVC	Model-View-Controller
MVP	Model-View-Presenter
MVVM	Model-View-ViewModel
OS	Operating system
PC	Personal Computer
SDK	Software development kit
SQL	Structured Query Language
UI	User Interface
URL	Uniform Resource Locator
VR	Virtual Reality
WoW	World of Warcraft
XML	Extensible Markup Language