FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Backend pro tvorbu Dungeons and Dragons postav s uživatelským obsahem
Student:	Filip Čihák
Vedoucí:	Ing. Jan Matoušek
Studijní program:	Informatika
Obor / specializace:	Softwarové inženýrství 2021
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

Dungeons & Dragons (D&D) je společenská hra na hrdiny (RPG - "role playing game"), kde se každý hráč vtěluje do role fantasy postavy, kterou si sám navrhne na základě stanovaných pravidel a možností. Pro D&D je specifické, že kromě základních pravidel existuje mnoho dalších rozšiřujících herních prvků, které si často pro vlastní potřeby vytvářejí i sami hráči (tzv. homebrew). Systém by měl zprostředkovávat tvorbu postavy právě s obsahem z různých zdrojů, které bude možné flexibilně definovat.
Cílem práce je návrh a implementace API a backend webového serveru, který bude umožňovat vedení pravidel a obsahu pro postavy ve hře D&D a zároveň bude poskytovat rozhraní pro jejich tvorbu. Spolupracujte s Žanetou Troškovou, která realizuje frontend část systému.

Pokyny k vypracování:

1. Analyzujte pravidla a zvyklosti hraní D&D se zaměřením na herní postavy a ohledem na to, které prvky hráči často upravují a tvoří k nim vlastní obsah.
2. Na základě provedené analýzy vytvořte řádný návrh potřebného rozhraní a systému.
3. Implementujte aplikaci s ohledem na předchozí analýzu a návrh.
4. Aplikaci podrobte vhodnému testování dle vlastního návrhu.
5. Shrňte dosažené výsledky, navrhněte budoucí rozšíření či směřování projektu.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 28. ledna 2024 v Praze.

Bakalářská práce

BACKEND PRO TVORBU DUNGEONS AND DRAGONS POSTAV S UŽIVATELSKÝM OBSAHEM

Filip Čihák

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Jan Matoušek
13. května 2024

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat především vedoucímu práce Ing. Janu Matouškovi za cenné rady a konzultace během celého vývoje aplikace a za jeho trpělivost a ochotu mi pomoci. Dále bych chtěl poděkovat autorce frontendové části aplikace Žanetě Troškové za spolupráci na tomto projektu. Děkuji také všem hráčům, kteří se podíleli na analýze, a všem, kteří mi poskytli zpětnou vazbu a podporu během vývoje. Nakonec bych chtěl poděkovat své rodině a přátelům za podporu a trpělivost během psaní této práce i celého mého studia.

Prohlášení

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.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 13. května 2024

Abstrakt

Tato bakalářská práce se zabývá vývojem backend serveru webové aplikace pro tvorbu postav do hry Dungeons & Dragons. Vytvořená aplikace umožňuje definovat vlastní herní obsah, jako například třídy postav, kouzla a zbraně. Tyto prvky mohou hráči přidat do svých deníků postav a hrát tak např. za danou třídu či nosit danou zbraň. Backend je implementován v jazyce Java pomocí frameworku Spring Boot a pro perzistenci dat používá databázový systém MongoDB. Poskytuje rozhraní REST API, ze kterého čerpá autorka Žaneta Trošková ve své bakalářské práci pro frontend část aplikace. Na závěr je aplikace otestována a jsou navrhnuty možnosti pro budoucí rozšíření systému.

Klíčová slova Dungeons & Dragons, webová aplikace, backend, MongoDB, Java, Spring Boot

Abstract

This bachelor thesis is concerned with the development of the backend server of a web application for creating characters for the game Dungeons & Dragons. The developed application allows to define custom game content such as character classes, spells and weapons. Players can add these elements to their character sheets to play as a given class or carry a given weapon, for example. The backend is implemented in Java using the Spring Boot framework and uses the MongoDB database system for data persistence. It provides a REST API which author Žaneta Trošková uses in her bachelor thesis for the frontend part of the application. Finally, the application is tested and possibilities for future extensions of the system are suggested.

Keywords Dungeons & Dragons, web application, backend, MongoDB, Java, Spring Boot

Seznam zkratek

AC	Armor Class
API	Application Programming Interface
BSON	Binary JSON
CRUD	Create, Read, Update, Delete
DAO	Data Access Object
DI	Depedency Injection
DBMS	Database Management System
DTO	Data Transfer Object
D&D/DnD	Dungeons & Dragons
EDN	Extensible Data Notation
HP	Hit Points
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IoC	Inversion of Control
JAR	Java Archive
JSON	JavaScript Object Notation
MVC	Model–View–Controller
NoSQL	Not only SQL/Non-SQL
NPC	Non-Player Characters
ODT	OpenDocument Text
PB	Proficiency Bonus
PDF	Portable Document Format
RDBMS	Relational Database Management System
REST	Representational State Transfer
RPG	Role-Playing Game
SPA	Single Page Application
SQL	Structured Query Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XFA	XML Forms Architecture
XML	Extensible Markup Language
WYSIWYG	What You See Is What You Get

Úvod

Dungeons and Dragons (často zkracováno jako *D&D* nebo *DnD*) je společenská hra, která spadá do kategorie tzv. *role-playing games*, nejčastěji překládáno jako hry na hrdiny. Tyto hry se vymezují od jiných tím, že se v nich hráči nesnaží vyhrát nad ostatními, ale vystupují jako postavy ve fantasy světě, kterým prochází a společně v něm tvoří rozhodnutí.

Jedná se o kombinaci deskové hry, vymýšlení příběhového děje a někdy i improvizovaného herectví. Pravidla hráčům staví mantinely například na to, jaké mají jejich postavy dovednosti a jaké mohou používat předměty, avšak neomezují, kam mohou postavy jít nebo jaká rozhodnutí mohou učinit. Tím se naopak zaobírá jeden z hráčů, který nemá jednu vlastní postavu, ale zastupuje roli tzv. *Dungeon Mastera* nebo *Game Mastera*, do češtiny obvykle překládáno jako pán jeskyně. Ten ve hře působí jako rozhodčí pravidel a „režisér“ nebo „moderátor“ příběhu. Je na něm tvorba herního světa, do kterého pak ostatní hráče uvede, podle svého nejlepšího vědomí rozhoduje, jak se v něm projeví akce hráčů. Zároveň hraje za všechny ostatní osoby, se kterými se hráči ve světě potkávají, tedy například různé jiné dobrodruhy, obchodníky, ale i záporné postavy všeho druhu. Ty se souhrnně označují jako *NPC*, zkratka pro *non-player characters*.

Ikonickou součástí hry jsou kostky, které se používají pro náhodné rozhodování o nepředvídatelných událostech a určování proměnlivých výsledků. Těch se v D&D pro tyto účely používá celá škála od čtyřstěnné přes klasickou šestistěnnou až po dvacetistěnnou a podle toho se označují jako např. *d4*, *d6* a *d20*.

Dungeons and Dragons jsou jednou z nejstarších a vůbec nejpopulárnějších her svého druhu, v oběhu je již páté vydání pravidel. V nedávné době hra zaznamenala rozmach zvlášť v období pandemie koronaviru, kdy se hra navíc často přesunula od fyzického stolu k online hraní. Tím přirozeně vznikl mnohem větší zájem o software nástroje, které usnadňují přípravu a hraní hry. Systém vznikající v rámci této bakalářské práce se zabývá automatizací nejdůležitějšího procesu v D&D, kterým je tvorba postavy hráče.

Hlavním cílem bakalářské práce je konkrétně navrhnout aplikační rozhraní a implementovat ho ve formě backend webového serveru, který bude sloužit pro správu obsahu (definic herních prvků) do hry Dungeons & Dragons. Tento obsah bude využit k tvorbě a ukládání herních postav.

Pro splnění tohoto cíle bude nejprve analyzováno, které prvky a pravidla hra obsahuje a jak je hráči využívají, zejména které z nich jsou hráči doplňovány a měněny, a je tedy potřeba s ohledem na ně aplikaci navrhnout. Na závěr budou implementované funkcionality otestovány, připraveny pro nasazení a bude navrhnuto, jak by bylo možné aplikaci dále rozvíjet.

Práce vzniká ve spolupráci s Žanetou Troškovou, která paralelně vytváří frontendovou část systému ve své bakalářské práci na FIT ČVUT s názvem *Webový frontend pro tvorbu postav do hry Dungeons and Dragons*.

Kapitola 1
Analýza

Tato kapitola nejprve analyzuje prvky, které tvoří postavy v Dungeons & Dragons, a následně se zaměřuje na existující software řešení pro tvorbu postav do této hry. S ohledem na tyto poznatky pak spolu s výsledky vlastního sběru informací rozebírá požadavky na systém, který bude v rámci této práce navržen a implementován.

1.1 Analýza herních pravidel

Tato sekce se zabývá rozborem pravidel páté edice Dungeons & Dragons, které se týkají tvorby herní postavy, zejména pak toho, které volby musí pro tvorbu hráč učinit a jak ovlivňují další parametry postavy. U herních prvků budou uvedeny zejména ty vlastnosti, které mají vliv na herní mechaniky a nejsou pouze „kosmetického“ charakteru. Vzhledem k tomu, že pravidla a veškeré další materiály ke hře jsou vydány v angličtině, spolu s originálním názvem (v závorce) bude pro každý z nich zaveden český překlad, který bude použit ve zbytku práce.

1.1.1 Volby

Každá postava v D&D je definována několika základními prvky, které hráči musí vybrat, a které mají vliv na vlastnosti postavy. Tyto volby se navzájem neomezují, je tedy možné kombinovat například různé rasy s různými třídami. Výsledná postava je pak kombinací všech těchto prvků a vlastností, které z nich vyplývají.

1.1.1.1 Rasa

Rasa (Race) určuje druh lidí, ke kterému postava patří. Jedná se tak o člověka, elfa nebo trpaslíka. Volba rasy určuje zejména fyzické a biologické vlastnosti postavy [1]:

- **Velikost** (Size) – kategorie velikosti, do které zástupci rasy spadají.
- **Rychlost** (Speed) – rychlost pohybu postav rasy, ve hře vždy uváděna v počtu stop za herní tah.
- **Jazyky** (Languages) – jazyky, které zástupci rasy vždy ovládají, případně počet dalších jazyků, které si mohou vybrat.
- **Zvýšení atributů** (Ability Score Increases) – bonusy k obecným atributům postavy, zpravidla +2 k jednomu a +1 k druhému.

- **Rasové vlastnosti** (Racial Features) – další vlastnosti, které postava získává na základě rasy, jako např. zvláštní smysly či možnost provádění drobné magie.

Některé rasy také mají poddruhy (Subraces), u nich je část z těchto vlastností společná a část specifická pro daný poddruh. V některých materiálech jsou poddruhy vnímány spíše jako samostatné rasy, jinde se naopak jedná spíše o např. kulturní variace jedné rasy. [1]

V posledních letech se objevuje trend, kdy se tvůrci snaží omezit vliv rasy a hlavně jejich stereotypních vlastností, a proto se často ponechává volba jazyků a zejména zvýšení atributů kompletně na vůli hráče, jak je vidět například v případě nového vydání některých ras v rozšíření *Mordenkainen Presents: Monsters of the Multiverse*. [2]

1.1.1.2 Herní třída a úroveň

Herní třída (Class) je hlavním určením toho, jak může postava interagovat s herním světem jako dobrodruh. Určuje její specifické vlastnosti, přičemž mnohé z nich rozšiřují bojové možnosti postavy a umožňují tak hráčům překonávat hrozby. V rámci třídy navíc postava postupem času získává vyšší úrovně, které zpřístupňují další vlastnosti či vylepšují ty stávající. Třída je hlavním fantasy prvkem hry, patří mezi ně např. čaroděj (Wizard), válečník (Fighter) nebo bard (Bard). Třída určuje zejména následující [1]:

- **Dovednosti** (Proficiencies)[1] – kategorie schopností (Skills), zbraní, zbroje, nástrojů a záchranných hodů (Saving Throws), se kterými je postava buď zdatná automaticky, nebo ze kterých si může vybrat několik, ve kterých bude zdatná.
- **Kostka bodů života** (Hit Dice) – počet stěn kostky, kterou se určuje počet bodů života postavy.
- **Počáteční vybavení** (Starting Equipment) – vybavení, které postava dostane na začátku hry.
- **Vlastnosti třídy** (Class Features) – zvláštní vlastnosti, které postava postupně získává s postupem na vyšší úrovně, typicky schopnosti pro boj, kouzlení či přežívání během dobrodružství.
- **Specializace** (Subclass) – doplňující volba mezi 1. a 3. úrovní, která postavě přidává další vlastnosti zaměřené na určitý aspekt třídy.

1.1.1.3 Skóre atributů

Atributy (Abilities) měří obecnou zdatnost postavy v různých oblastech. V D&D se používá následujících šest atributů [1]:

- **Síla** (Strength) – určuje tělesnou sílu a schopnost zvedat těžké předměty.
- **Obratnost** (Dexterity) – určuje zručnost, reflexy a pohybovou koordinaci.
- **Odolnost** (Constitution) – určuje zdraví a tělesnou energii.
- **Inteligence** (Intelligence) – určuje schopnost učení, paměť a logické myšlení.
- **Moudrost** (Wisdom) – určuje intuici a smyslové vnímání.
- **Charisma** (Charisma) – určuje sílu osobnosti a schopnost jednat s lidmi.

[1] Vzhledem k zaměnitelnosti českých slov „dovednost“ a „schopnost“ a skutečnosti, že se původní výrazy často používají i při hraní v češtině, se bude text v dalších sekcích držet termínů Proficiency, Skill a Saving Throw.

Postava má v každém z nich určené skóre (Ability Score) mezi 1 a 20, které poté ovlivňuje mnoho dalších herních statistik, jak bude popsáno dále. Metod pro určení těchto hodnot existuje řada. I pouze původní pravidla nabízejí tři možnosti: hod kostkami a přiřazení výsledků, výběr z předem daných hodnot (Standard Array), nebo bodový systém nákupu atributů (Point Buy). [1]

1.1.1.4 Zázemí

Zázemí (Background) popisuje, odkud postava pochází a jaké má předchozí zkušenosti. Jedná se např. o vojáka, šlechtice či akademika. Zázemí poskytují zejména následující [1]:

- **Schopnosti, jazyky, nástroje** (Skills, Languages, Tools) – kategorie schopností, jazyky a nástroje, se kterými je postava zdatná, případně počet dalších nástrojů nebo jazyků, které si může vybrat.
- **Vybavení** (Equipment) – další vybavení, které postava dostane na začátku hry.
- **Zvláštní vlastnost** (Feature) – herní vlastnost spjatá s povoláním, životním stylem nebo společenským postavením daného zázemí.

Jelikož zázemí má obecně malý vliv na herní mechaniky, často hráči nevybírají z těch uvedených v pravidlech, ale míchají tyto vlastnosti nebo určují vlastní.

1.1.1.5 Vybavení

Každá postava u sebe nese nějaké vybavení (Equipment). D&D popisuje mnoho předmětů různých druhů, ale pro herní mechaniky jsou důležité následující kategorie [1]:

- **Zbraně** (Weapons) – určují vlastnosti útoků, které postava může provádět, jsou vždy buď jednoduché (Simple), nebo vojenské (Martial), a buď na blízko (Melee), nebo na dálku (Ranged).
- **Brnění** (Armor) – určují, jak dobře je postava chráněna před útoky, jsou řazena na lehké, střední a těžké.
- **Nástroje** (Tools) – pomůcky, které postava může používat k zvláštním účelům, např. šperháky (Thieves' tools) nebo alchymistická sada (Alchemist's supplies).

Zbraní může postava nést několik najednou, ovšem brnění se v kontextu pravidel vždy chápe jako celá sada, a postava tedy může mít nasazené nejvýše jedno. Na rozdíl od předchozích prvků může hráč kdykoli ve hře měnit vybavení postavy, pokud má k němu přístup. [1]

1.1.1.6 Kouzla

V D&D se vyskytuje mnoho magie, ovšem centrálním prvkem jsou kouzla (Spells). Jsou dostupné zejména těm třídám, které mají vlastnost *Spellcasting*, která jim umožňuje udržovat seznam kouzel k seslání, a stanovuje limity pro počet jejich použití v podobě tzv. kouzelnických slotů (Spell Slots) různých úrovní. Kouzla (kromě těch 0. úrovně) lze seslat pouze spotřebováním slotu stejné nebo vyšší úrovně. Ke každému kouzlu jsou v pravidlech evidovány následující informace [1]:

- **Úroveň kouzla** (Spell Level) – nabývá hodnot 0 až 9, určuje náročnost na seslání i sílu kouzla.
- **Doba seslání, dosah, trvání** (Casting Time, Range, Duration) – vlastnosti důležité během hraní hry pro rozhodování o použití kouzla.

- **Komponenty, škola magie, popis** (Components, School, Description) – další informace o zařazení kouzla, podmínkách pro jeho seslání a účincích, nemá vliv při tvorbě postavy.

Vlastnost *Spellcasting* dané třídy také stanovuje, kdy může postava svůj seznam kouzel měnit, často se ovšem hráči dohodnou s pánem jeskyně na změnách i mimo tyto příležitosti. [1]

1.1.2 Vypočtené vlastnosti

Jakmile hráč zvolí výše uvedené prvky, zbytek vlastností postavy určí výpočtem z nich. V původních pravidlech jsou tyto vlastnosti často uvedeny tabulkami, ovšem pro účely této práce bude vhodnější je popsat pomocí vzorců, které lze snadno převést do kódu.

1.1.2.1 Modifikátory atributů

Z každého skóre atributu lze vypočíst modifikátor (Ability Modifier), který bude použit v dalších výpočtech, a to následovně [1]:

$$\text{Modifikátor} = \left\lfloor \frac{\text{Skóre} - 10}{2} \right\rfloor \tag{1.1}$$

Skutečně platí, že pokud je skóre nižší než 10, je modifikátor záporný, a tudíž snižuje výsledné hodnoty dalších vlastností.

1.1.2.2 Body života a kostky bodů života

Body života (Hit Points), dále označované jako HP, určují, kolik zranění je postava schopná přežít. Pro jejich určení pravidla stanovují následující postup zahrnující kostku bodů života a úroveň postavy [1]:

1. Na 1. úrovni získá postava počet HP rovný počtu stěn kostky a k tomu přičte modifikátor odolnosti.
2. Na každé další úrovni hráč touto kostkou hodí, k výsledku přičte modifikátor odolnosti a toto číslo přičte k celkovým HP.
3. Pokud se v průběhu hry změní modifikátor odolnosti, změní se retrospektivně i počet HP z každé úrovně.

Pravidla také připouští místo házení vždy použít průměr zaokrouhlený nahoru a tento způsob je pro svou konzistentnost a rychlost často používán. Celkové HP pak můžeme vyjádřit takto [1]:

$$\text{HP} = HD_s + (L-1) \cdot \left\lceil \frac{HD_s}{2} \right\rceil + L \cdot ConMod \tag{1.2}$$

kde HD_s je počet stěn kostky pro životy, L je úroveň postavy a $ConMod$ je modifikátor odolnosti.

Tyto kostky jsou navíc během hry k dispozici pro léčení během odpočinku, kdy jejich typ je opět dán třídou a počet je rovný úrovni postavy. [1]

1.1.2.3 Proficiency Bonus

Z hlediska dovedností postavy je potřeba určit Proficiency Bonus – bonus dovedností, dále označený jako PB, který je dán úrovní postavy (L) následovně [1]:

$$\text{PB} = \left\lceil \frac{L}{4} \right\rceil + 1 \tag{1.3}$$

1.1.2.4 Schopnosti a záchranné hody

V D&D se pro rozhodování o výsledku netriviálních akcí používají hody schopnosti (Skill Checks) a záchranné hody (Saving Throws). Hra používá 18 kategorií Skills a 6 kategorií Saving Throws. Mezi kategorie Skills patří např. plížení (ovlivněné obratností), přesvědčování (ovlivněné charismatem) nebo práce se zvířaty (ovlivněná moudrostí). Kategorie Saving Throws přímo odpovídají atributům. [1]

Každá z kategorií je ovlivněna jedním z atributů a postava má ke každé kategorii přiřazen modifikátor hodu, jehož hodnota se k výsledku hodu přičítá. Modifikátor je dán jako modifikátor ovlivňujícího atributu, ke kterému se přičte Proficiency Bonus, pokud má postava Proficiency na danou kategorii. [1]

Pokud se ve hře postava například snaží vyjednat u obchodníka lepší cenu za zboží, pán jeskyně zhodnotí chování a argumenty postavy a podle toho nastaví číselnou obtížnost akce. Hráč poté hodí kostkou *d20* a k výsledku přičte v tomto případě svůj modifikátor k přesvědčování, a pokud je výsledek vyšší nebo roven stanovené obtížnosti, akce se považuje za úspěšnou a hra v tomto duchu pokračuje. Na deníku postavy je obvykle zaznamenán pouze modifikátor každé kategorie, jelikož hod se provádí vždy stejným způsobem.

1.1.2.5 Bojové vlastnosti

V boji se uplatní několik obecných vlastností, a to iniciativa, rychlost a stupeň obrany.

Iniciativa (Initiative) určuje pořadí tahů v boji. U ní je opět určen modifikátor, který je roven modifikátoru obratnosti a přičítá se k hodům iniciativy na začátku boje. [1]

Rychlost (Speed) udává počet stop, který dokáže postava urazit za tah. Její základ je dán rychlostí rasy, ale je snížena o 10, pokud postava nosí těžké brnění a zároveň nemá v síle skóre požadované daným brněním (Strength Requirement). [1]

Stupeň obrany (Armor Class), dále pouze AC, je číslo stanovující, jak těžké je postavu zasáhnout. Jeho výpočet zjistíme takto [1]:

1. Pokud postava nosí brnění, základní AC i aplikované modifikátory atributů jsou dány brněním.

2. Jinak, pokud má třída postavy vlastnost *Unarmored Defense*, tato vlastnost určuje základ AC i aplikované modifikátory.

3. Jinak je AC rovno 10 + modifikátor obratnosti.

Pokud navíc postava drží štít a má s ním Proficiency, zíkává k AC bonus rovný 2. [1]

1.1.2.6 Útoky

Během svého tahu v boji může postava vést útoky (Attacks). Nejprve je určen úspěch útoku hodem kostkou *d20* a přičtením modifikátoru útoku (Attack Bonus) stanoveným jako modifikátor atributu + Proficiency Bonus. Zbraně na blízko používají modifikátor síly, zbraně na dálku modifikátor obratnosti a zbraně s vlastností *Finesse* vyšší z obou. Proficiency Bonus je možné přičíst pouze tehdy, pokud postava má Proficiency s danou zbraní. Ve hře pak útok uspěje, pokud je výsledek vyšší nebo rovný AC cíle útoku. [1] Na deníku postavy je obvykle opět zaznamenán pouze modifikátor, jelikož hod se provádí vždy stejným způsobem.

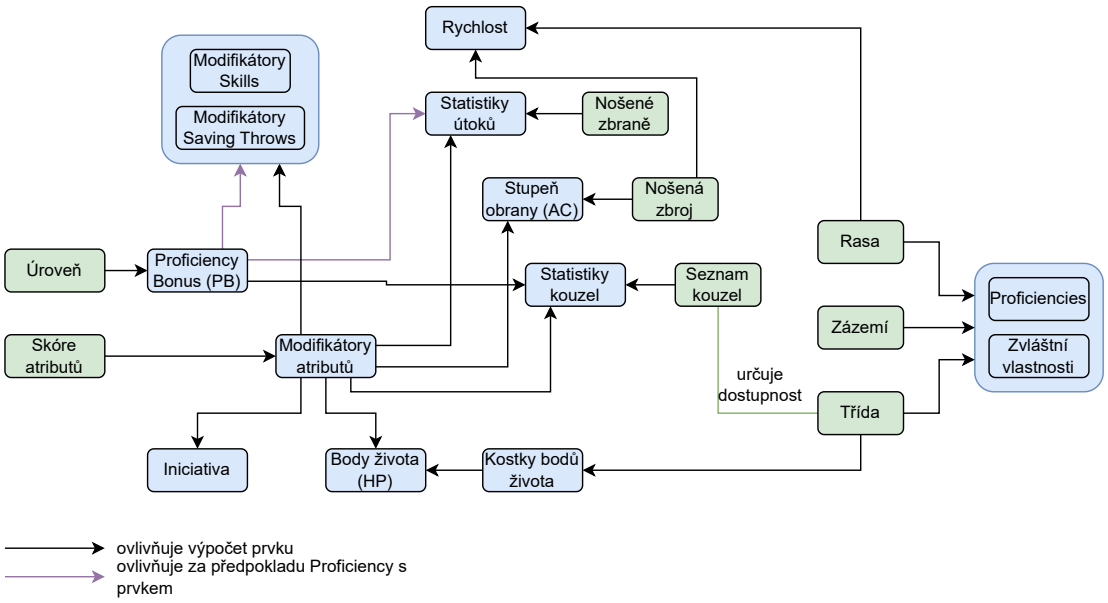
Pokud útok uspěje, je určeno hodem poškození (Damage). Kostka či kostky určené pro hod jsou dané zbraní a k výsledku je přičten stejný modifikátor atributu použitý v minulém kroku (vždy bez Proficiency Bonusu). Dále je důležité uvádět typ poškození, který opět závisí na použité zbrani. [1]

1.1.2.7 Kouzelnické statistiky

Postavy, které mohou sesílat kouzla, mají několik dalších statistik s tím spojených. Schopnost *Spellcasting* jejich třídy jim určuje, který atribut používají pro kouzla (Spellcasting Ability). Modifikátor útoku kouzla (Spell Attack Bonus) je dán stejně jako pro útoky zbraní, ale používá tento atribut. Obtížnost vyhnutí se kouzlu (Spell Save DC) je dána jako 8 + modifikátor atributu kouzelnictví + Proficiency Bonus. [1]

1.1.3 Vizualizace závislostí

Pro lepší konceptualizaci všech vlastností postavy a vztahů mezi nimi je vhodné zobrazit je graficky v diagramu 1.1. Tento diagram navíc poslouží jako základ pro návrh aplikace.



Obrázek 1.1 Diagram závislostí

1.2 Analýza existujících řešení

Pro úplnost je také dobré zanalyzovat, jakým způsobem přistupují k tvorbě postav a importu vlastního obsahu jiné dostupné aplikační nástroje pro D&D.

1.2.1 D&D Beyond

D&D Beyond (dále pouze Beyond) je oficiální online webový nástroj pro elektronický přístup k pravidlům hry a tvorbu postav.[3] Tato aplikace nabízí plnou tvorbu postavy, přičemž primárně používá herní prvky z knih, které má uživatel zakoupené v aplikaci[2]. Je schopna sledovat takřka všechny herní pravidla a výsledné postavy zobrazuje ve vlastním webovém rozhraní, které kromě informací z klasického deníku postavy zobrazuje i vysvětlivky a další informace dané pravidly.

[2] I pokud má uživatel fyzickou knihu, je nutné si ji zakoupit digitálně v Beyond, aby byla dostupná pro tvorbu postav.

1.2.1.1 Určení aplikace

Aplikace je především uzpůsobena pro online hraní, jelikož klikání na např. modifikátory hodů umožňuje virtuálně házet kostkami a obsahuje také komponenty pro hraní v reálném čase jako např. sledování aktuálních bodů života a spotřebovaných schopností postavy. Toto pojetí je použito i v případě exportu deníku do PDF, které obsahuje mnoho podrobností (např. popis všech možných akcí v boji, vlastnosti kouzel) a odkazy na strany pravidel v knize. Tím pádem je PDF poměrně nevhodné pro tisk kvůli většímu počtu stran, malé velikosti písma a přetékajícím okénkům PDF formuláře, které po tisku není možné posouvat.

1.2.1.2 Tvorba obsahu

Z hlediska tvorby a úprav herních prvků nabízí Beyond dvě možnosti. První je úprava některých prvků přímo v rozhraní deníku postavy pomocí rozklikávacího menu Customize, který umožňuje provést vybrané drobné změny jako např. změnu názvu či popisku předmětu nebo ruční úpravu bonusu či poškození útoku. Tyto změny se vždy ukládají pouze pro danou postavu, kde jsou zadány.

Druhou možností je sekce Homebrew, která umožňuje uživatelům přidávat vlastní obsah. Beyond dovoluje zadat vlastní rasy, zázemí, specializace tříd (nikoli však celé nové třídy), kouzla, magické předměty (nikoli nemagické), výjimečné schopnosti a cizí stvoření (ty ovšem nesouvisí s deníkem postavy). [4] Všechny tyto prvky jsou zadávány pouze vlastním webovým formulářem, není možné je psát či importovat ve strojově čitelném formátu. Beyond pak přirozeně neumožňuje tyto prvky exportovat do souboru. Je ale možné sdílet prvky se spoluhráči vytvořením „kampaně“, kde všichni připojení hráči mezi sebou vidí všechny homebrew prvky vytvořené ostatními členy.

Veřejné sdílení je omezenější – publikování je možné, pokud sdílený obsah splňuje pravidla Beyond, které zakazují mimo jiné obsah podobný tomu v oficiálních knihách nebo obsahující zmínky jiného copyrightovaného materiálu, či obsah, který není v anglickém jazyce. [5] Stažení veřejně publikovaného obsahu v rámci Beyond je pak možné pouze pro aktivní předplatitele. [6]

1.2.1.3 Zhodnocení

Celkově je tak Beyond vhodný zejména pro nové skupiny, které nevlastní knižní materiály fyzicky. Je pro ně tak nejjednodušší koupit si je digitálně a využívat Beyond jako jediný zdroj pravidel a tvorby postav. Zároveň je pro ně ideální rozhraní deníku postavy, které provádí tvorbou, připomíná základní pravidla a umožňuje házení kostkami, které tak nemusí shánět fyzicky.

Naopak pro zkušené hráče, kteří mají knihy, jsou zvyklí na klasické papírové deníky a chtějí do větší míry hrát vlastním způsobem, je Beyond méně vhodný, jelikož homebrew je omezené a vyžaduje ruční proklikávání mnoha formulářů, navíc ho není možné kategorizovat a snadno navzájem sdílet s jinými skupinami.

1.2.2 Dungeon Master's Vault

Dungeon Master's Vault je komunitní webová aplikace pro tvorbu postav a generování nápadů pro pány jeskyně (jako například náhodné popisy měst nebo lidí). [7] Aplikace je dostupná veřejně a zároveň má open-source verzi pro vlastní provoz na GitHubu. Aplikace má server napsaný v jazyce Clojure, který komunikuje s databází Datomic. [8] Aplikace je tedy postavena na server-rendered webových stránkách, které se načítají znovu při každé změně, a nemá žádné REST API, které by umožňovalo vytvoření SPA. Dungeon Master's Vault poskytuje tovrbu postavy z dynamicky definovaných zdrojů, podobně jako je cílem této práce. Vzhledem ke zvolené architektuře je ale frontend nepřehledný, obsahuje mnoho různých formulářů a procesy pro správu postav jsou tak zbytečně složité. Analýze uživatelského rozhraní této aplikace se více věnuje Žaneta v analytické části své bakalářské práce. Datovým formátem použitým pro import obsahu

je EDN, což je datový formát využívající syntaxi jazyka Clojure [9], a je tak vhodný především pro aplikace využívající tuto technologii. Všechny prvky jsou v tomto formátu definovány jako stromová hierarchie vlastností, které mají být na herní postavu aplikovány. [8] Existuje možnost exportu postavy do PDF, ovšem Dungeon Master's Vault do PDF vůbec neexportuje některé informace, jako například textové popisy získaných schopností. Tento export tedy obsahuje převážně pouze číselné informace o postavě. Výsledný dokument zároveň nepodporuje interaktivní formuláře a není tak možné výstup jakkoli upravit či další informace do PDF dopsat.

1.2.3 Aurora Builder

Aurora Builder je desktopová aplikace pro Windows podporující tvorbu postav. Je volně stažitelná, ovšem není open-source a její vývojář v roce 2020 ukončil vývoj. [10] Aplikace je nicméně stále dostupná a funkční. Obsah aplikace je také čerpán z importovaných souborů, v případě Aurora Builderu se jedná o XML soubory, které nesou informace o herních vlastnostech prvků i popisné texty pro zobrazení v aplikaci, které jsou již formátované pomocí HTML značek. [11] Vzhledem k tomu, že se jedná o desktopovou aplikaci, není zde možnost snadno sdílet postavy s ostatními hráči.

1.3 Katalog požadavků

Sestavení požadavků na systém je klíčovým krokem při vývoji softwarového produktu. Formalizují dohodu mezi zainteresovanými stranami na tom, co má produkt splňovat. Slouží jako základní reference pro všechny další kroky vývoje od návrhu až po testování a nasazení. [12] Proces sběru požadavků se obecně sestává z těchto kroků [13]:

1. identifikace zainteresovaných stran – rozhodnutí, kdo bude mít zájem na výsledku a měl by být zapojen do procesu,
2. sběr informací – získání informací o potřebách a očekáváních od zainteresovaných stran,
3. specifikace požadavků – zdokumentování získaných informací ve specifikovaném formátu, na základě kterého bude možné tvořit produkt.

Dále je vhodné v závislosti na projektu požadavky spravovat z hlediska jejich typu, důležitosti či náročnosti na splnění. Pro tento účel se často používají různé klasifikační metody, jako např. MoSCoW, která bude použita i v této práci.

1.3.1 Sběr požadavků

Cílem této bakalářské práce je dle zadání vytvořit backendovou část aplikace pro tvorbu D&D postav z flexibilně definovaného obsahu. Vzhledem k důrazu na tuto vlastnost by tak systém měl být cílený hlavně na pokročilejší skupiny, které často chtějí prvky z pravidel upravovat či tvořit vlastní. Proto jsem na podzim 2023 začal sbírat podklady od několika herních skupin, kterých jsem byl součástí či jsem s jejich členy byl v kontaktu. Dále byli do procesu zapojeni samozřejmě i vedoucí práce a Žaneta, která realizuje frontendovou část, a bylo tak nutné zajistit kompatibilitu obou částí aplikace a zohlednit nároky frontendu.

Stěžejní požadavky se zakládají na pravidlech hry, které byly rozebrány v předchozí sekci. Pozoroval jsem přímo během hry, které informace hráči využívají, a také s nimi přímo konzultoval jejich zkušenosti a názory na různé nástroje pro tvorbu postav. Hlavním podkladem od hráčů ovšem byly poskytnuté deníky postav, u kterých bylo analyzováno, jaké části jsou využívány. Bylo vyhodnoceno, která herní pravidla jsou plně dodržována a není tedy žádoucí je měnit, a naopak, která pravidla jsou potřeba v aplikaci pojmout flexibilně. Informace k zejména nefunkčním požadavkům byly získány konzultacemi s vedoucím práce a Žanetou.

1.3.2 Kategorizace požadavků

Pro lepší orientaci je vhodné požadavky na řešení systému rozdělit. Klasickým způsobem je v první řadě dělení na funkční a nefunkční požadavky systému.

Funkční požadavky popisují, co má systém dělat. Nejčastěji je definují uživatelé, kteří systém budou používat, a popisují potřebné funkce, které by měl systém poskytovat.

Nefunkční požadavky popisují, jak má systém fungovat. Jsou to zejména kvalitativní vlastnosti systému, týkající se např. jeho výkonu, dostupnosti, integrace s jinými systémy, nároků na zabezpečení, dodržení standardů a další. Typicky je definují spíše sami vývojáři, provozovatelé, správci a další experti než koncoví uživatelé. [12, 13]

Zdokumentovaným požadavkům bude dále přiřazena důležitost, která bude používat rozřazení dle metody MoSCoW. Tato metoda hodnotí požadavky dle toho, jak důležité je jejich splnění pro následující vydání produktu a do jaké míry je naopak možné je odložit či eventuálně úplně vyřadit. Pro účely bakalářské práce bude jako vydání verze produktu chápáno odevzdání práce.

Must have – požadavek, který je nezbytné splnit, protože by bez něj produkt nefungoval či postrádal smysl. V kontextu bakalářské práce se jedná především o požadavky, které vyplývají přímo ze zadání a cíle práce.

Should have – požadavek, jehož splnění přináší produktu významnou hodnotu, ale je možné jej odložit na další vydání.

Could have – požadavek, který neovlivňuje hlavní cíl produktu, jeho vynechání ve vydání má tedy menší dopad oproti should have požadavkům.

Will not have (this time) – požadavek, o kterém bylo rozhodnuto, že nebude zařazen do aktuálního vydání. Jsou to požadavky, které byly vyhodnoceny jako nevhodné pro aktuální podobu produktu nebo byly odloženy z důvodu nedostatku času či zdrojů. [14]

Každý požadavek bude pojmenován, označen kódem pro snadné odkazování, textově popsán a bude kategorizován dle výše popsaných kritérií včetně zdůvodnění těchto rozhodnutí.

Další části práce se zaměří na řádné splnění požadavků označených jako Must have a Should have. Požadavky označené jako Could have budou implementovány pouze v případě, že bude dostatek času.

1.3.3 Seznam funkčních požadavků

F1: Přístup k obsahu

- **Důležitost:** Must have
- Každá definovaná položka (herní prvek) v systému dostane unikátní identifikátor, podle kterého ji bude možné najít a zobrazit. Každá položka bude patřit do právě jednoho zdroje obsahu. Zdrojem může být například kniha pravidel, rozšíření nebo vlastní vymyšlený balíček herních prvků. Dále možné bude vyhledávat všechny položky daného typu a filtrovat je podle jména a zdroje obsahu. Kategorie flexibilně definovaných herních prvků systému budou: rasy, třídy, zázemí, zbraně, brnění, nástroje, jazyky a kouzla.
- **Zdůvodnění:** Tento požadavek je základní funkcionalitou potřebnou pro další práci s obsahem. Rasy, třídy, zázemí a kouzla patří mezi hlavní volby postavy. Byly k nim vydány nové možnosti v rozšiřujících knihách a i původní pravidla zmiňují doporučení pro tvorbu vlastních variant těchto prvků. Zbraně, brnění, nástroje a jazyky hráči převážně čerpali z původních pravidel, avšak někdy vlastnili i upravené varianty v podobě magických předmětů.

Jazyky a nástroje byly upravovány pro potřeby vlastního světa, jelikož pravidla prezentují hlavně možnosti spjaté s původním zasazením hry. Naopak kategorie atributů, Skills a Saving Throws byly vynechány a budou v systému řešeny pevně, jelikož byly dotázanými hráči vnímány jako pevná součást pravidel a nebyly zaznamenány jako předmět úprav. To bylo potvrzeno i s vedoucím práce.

F2: Import obsahu ze souboru

- **Důležitost:** Must have
- Systém bude načítat hromadně definice herních prvků z externích souborů ve strojově čitelném formátu. Schéma těchto souborů bude závislé na typu herního prvku a bude podrobně popsáno v dokumentaci. K importovanému obsahu bude možné přistoupit dle požadavku F1 a použít ho pro tvorbu postav.
- **Zdůvodnění:** Tento požadavek je základní funkcionalitou potřebnou k tomu, aby bylo možné do systému nahrát obsah z různých zdrojů a vytvářet z něj postavy.

F3: Vytvoření postavy

- **Důležitost:** Must have
- V systému bude možné vytvořit novou postavu zadáním základních voleb postavy, kterými jsou zvolená třída, rasa, zázemí, skóre atributů a volby na nich závislé. Také bude zvoleno, ze kterých zdrojů obsahu může postava čerpat herní prvky. Dále bude možné zadat jméno postavy a hráče, URL portrétu postavy pro lidskou identifikaci postavy. Systém zkontroluje validitu zadaných údajů a uloží postavu s unikátním identifikátorem, pomocí kterého ji bude možné vyhledat a pracovat s ní v rámci následujících požadavků.
- **Zdůvodnění:** Tento požadavek je základní funkcionalitou potřebnou pro další práci s postavami. Bylo vyhodnoceno jako nevhodné, aby se vždy zobrazovaly všechny dostupné prvky v systému, proto lze postavě zadat seznam povolených zdrojů.

F4: Editace základu postavy

- **Důležitost:** Should have
- Pro již vytvořenou postavu bude možné změnit základní volby postavy specifikované v požadavku F3. Systém zkontroluje správnost zadaných údajů a upraví odpovídajícím způsobem postavu tak, aby zůstala v souladu s omezeními specifikovanými v ostatních požadavcích.
- **Zdůvodnění:** Byť pravidla technicky neumožňují měnit základní volby postavy, je tato funkcionalita žádoucí, pokud hráč chce opravit chybu při zadávání údajů nebo upravit postavu po dohodě s pánem jeskyně. Všechny analyzované nástroje také poskytují podobnou funkci a byla označena dotázanými hráči jako užitečná.

F5: Odstranění postavy

- **Důležitost:** Must have
- Systém umožní odstranit postavu ze systému, tato akce bude nevratná.
- **Zdůvodnění:** Tento požadavek je základní funkcionalitou. Odstranění postavy je důležité pro udržení přehlednosti a čistoty systému.

F6: Správa Skills a Saving Throws postavy

- **Důležitost:** Must have
- Systém umožní zadat, se kterými kategoriemi Skills má postava Proficiency. Proficiency v Saving Throws bude nastavena automaticky podle třídy postavy.
- **Zdůvodnění:** Tento požadavek umožňuje práci s jednou z hlavních částí postavy. Saving Throws dle pravidel nelze volně upravovat a je tedy vhodné mít je nastavené konzistentně pro všechny postavy. Proficiency v Skills byly naopak ve zhlédnutých denících postavy často volně modifikovány, proto je bude možné měnit i v aplikaci.

F7: Správa bojového vybavení postavy

- **Důležitost:** Must have
- Do systému bude možné zadat, které zbraně a brnění (případně žádné) postava používá. Zbraní může mít postava dle pravidel libovolný počet, brnění však jen jedno. Volby zbraní i brnění musí vycházet z povolených zdrojů postavy (viz požadavek F3). Každou z těchto zbraní bude moci postava používat pro útoky.
- **Zdůvodnění:** Tento požadavek umožňuje práci s jednou z hlavních částí postavy. Pravidla zde nekladou mnoho omezení, proto nedává smysl je zde zavádět. Zbraně a brnění jsou také často měněny v průběhu hry a je tedy žádoucí, aby je bylo možné kdykoli měnit i v aplikaci.

F8: Správa seznamu kouzel postavy

- **Důležitost:** Must have
- Pokud postava patří k herní třídě, která má schopnost sesílat kouzla, systém umožní zadat, která kouzla postava zná a může je sesílat. Volby kouzel musí vycházet z povolených zdrojů postavy (viz požadavek F3), jinak může být libovolný.
- **Zdůvodnění:** Tento požadavek umožňuje práci s jednou z hlavních částí postavy. Systém zde nebude kontrolovat mnoho pravidel, jelikož bylo zjištěno, že hráči po dohodě často využívají i kouzla původně dostupná jiným herním třídám, případně získávají více kouzel než je standardní počet pro danou třídu a úroveň postavy. Pro vyhledávání kouzel jsou navíc zvyklí používat jiné online databáze, která tato omezení vypisují a obvykle si je kontrolují při výběru sami. Hráči navíc požadovali možnost měnit kouzla kdykoli, jelikož to umožňuje opravit chyby v zadání nebo změnit strategii postavy a shoduje se to s přístupem všech analyzovaných nástrojů.

F9: Vypočtení a zobrazení informací o postavě

- **Důležitost:** Must have
- Systém umožní zobrazit detail postavy, který bude obsahovat alespoň následující informace:
  - identifikátor postavy;
  - hodnoty voleb zadaných při vytváření/editaci postavy (viz požadavek F3) – jména, obrázek, třída, rasa, zázemí, skóre atributů, …;
  - úroveň postavy;
  - obecné vypočtené hodnoty postavy – modifikátory atributů, Proficiency Bonus, počet bodů života a kostek bodů života, rychlost, inticiativa, stupeň obrany – jejich výpočet bude v souladu s pravidly popsanými v sekci 1.1.2;

  - vypočtené modifikátory pro Skills a Saving Throws s ohledem na Proficiency (viz požadavek F6) dle sekce 1.1.2.4;
  - souhrnný seznam získaných Proficiencies s různými zbraněmi, brněními, nástroji a jazyky;
  - souhrnný seznam dalších zvláštních vlastností postavy, které byly získány z třídy, rasy a zázemí;
  - vypočtené údaje útoků zbraněmi postavy (viz požadavek F7) dle sekce 1.1.2.6;
  - obecné kouzelnické statistiky (dle 1.1.2.7), počet dostupných kouzelnických slotů dle úrovně postavy a seznam kouzel (viz požadavek F8), pokud postava umí sesílat kouzla (viz 1.1.1.6).
- **Zdůvodnění:** Tento požadavek je základní funkcionalitou potřebnou pro práci s postavami a umožňuje získat výsledek vytvořené postavy. Jako informace byly vybrané ty, které se nachází v oficiálním deníku postavy, kromě neherních popisů postavy (např. Alignment a Personality Traits) a informacích závislých na konkrétní herní situaci (např. počet dočasných bodů života, množství peněz).

F10: Inventář postavy

- **Důležitost:** Could have
- Systém umožní zapisovat a číst seznamy předmětů, která má postava na sobě nebo u sebe. Mezi tyto přeměty patří výše popsaných zbraní a brnění, ale i další, které přímo neovlivňují herní vlastnosti postavy.
- **Zdůvodnění:** Tento požadavek byl označen jako méně důležitý, jelikož jiné předměty než zbraně a brnění neovlivňují vlastnosti postavy a jejich správa v aplikaci není tolik důležitá. Po zhlédnutí deníků a konzultaci s hráči bylo zjištěno, že tyto předměty (např. zásoby jídla, šípy, komponenty zaklínadel) jsou často zapisovány do deníku postavy ručně, jelikož se často během hry mění. Inventář je tak spíše doplňkovou funkcionalitou, která by vyžadovala složitější implementaci z důvodu nutnosti zavedení nových prvků do systému.

F11: Kontejnery v inventáři postavy

- **Důležitost:** Will not have
- V rámci funkcionality inventáře bude možné navíc vytvářet a vkládat „kontejnery“ jako např. batohy a měšce, tedy předměty, které obsahují jiné předměty a umožnují je tak vnořovat a organizovat.
- **Zdůvodnění:** Tato funkcionalita se objevila jako možnost, která je popsaná v pravidlech a je často implementována v RPG hrách a na D&D Beyond. Po zhlédnutí deníků byla ovšem ihned zařazena jako Will not have, jelikož nikdo z hráčů takto podrobně své předměty neorganizoval. Zároveň by byla její implementace značně náročná a zbytečně by zvyšovala složitost aplikace.

F12: Postup postavy na vyšší úroveň

- **Důležitost:** Should have
- Systém umožní postavě zvýšit úroveň, což ovlivní výpočet některých vlastností nebo přidá nové schopnosti. Systém nebude sledovat počet zkušenostních bodů ani jiných kritérií pro postup.
- **Zdůvodnění:** Tento požadavek umožňuje práci s jednou z hlavních částí postavy, avšak vyšší prioritu má tvorba postav na 1. úrovni, proto je označen jako Should have. Po konzultaci s hráči bylo dohodnuto, že zvýšení úrovně vždy probíhá po dohodě s pánem jeskyně a není tedy nutné sledovat zkušenostní body v aplikaci.

F13: Přidání více herních tříd

- **Důležitost:** Will not have
- Systém bude podporovat tzv. „multiclassing“, tedy možnost kombinovat vlastnosti herních tříd při zvyšování úrovně postavy.
- **Zdůvodnění:** Funkcionalita tohoto požadavku je součástí rozšiřujících pravidel pro pokročilé hráče. Byla označena jako Will not have, jelikož by vyžadovala složitější návrh a implementaci herních tříd postavy tak, aby se poté korektně vypočítávaly kombinované vlastnosti. Z dotázaných hráčů vyjádřil zájem o tuto funkcionalitu pouze jeden, ostatní toto pravidlo neznali či považovali za nevhodné z důvodu vytváření zvláštních či nevyvážených postav.

F14: Export postavy do PDF

- **Důležitost:** Should have
- Systém bude poskytovat možnost informace o postavě převést do podoby stránkového deníku postavy ve formátu PDF. Výsledný PDF soubor bude mít informace vyplněné v interaktivních formulářových polích, které tak bude možné ručně upravit a doplnit. Výsledný PDF soubor se zobrazí čitelně v nejnovější verzi prohlížečů Google Chrome a Mozilla Firefox.
- **Zdůvodnění:** I když tento požadavek není nezbytný pro fungování aplikace, má vysokou prioritu, jelikož umožňuje hraní s postavou i bez přístupu k aplikaci. Oficiální deník postavy ve formátu PDF je velmi používaným mezi dotázanými hráči, a proto by bylo vhodné poskytnout podobný formát jako výstup z aplikace. Ostatní analyzované nástroje neposkytují ideální PDF výstupy, proto by tato funkcionalita mohla patřit mezi hlavní výhody aplikace. Prohlížeče byly do požadavku přidány, jelikož různé PDF nástroje zobrazují textová pole odlišně. Proto byla potřeba soustředit se na ty nejpoužívanější mezi hráčskou skupinou.

1.3.4 Seznam nefunkčních požadavků

N1: API

- **Důležitost:** Must have
- Aplikace bude dostupná přes API, ke kterému bude dostupná dokumentace jednotlivých koncových.

N2: Testování aplikace

- **Důležitost:** Must have
- Aplikace bude pokryta vhodnými testy, které budou navrhnuty podle architektury aplikace.

N3: Verzování zdrojového kódu

- **Důležitost:** Should have
- Zdrojový kód aplikace bude verzován pomocí systému Git a bude dostupný ve veřejném repozitáři.

N4: Kontejnerizace aplikace

- **Důležitost:** Should have
- Aplikace bude spustitelná ve formě kontejneru Docker, který bude možné spustit na jakémkoli systému s odpovídajícími prostředky, který podporuje Docker.

N5: Autentizace uživatelů

- **Důležitost:** Will not have
- Systém nejprve autentizuje uživatele a ten bude mít přístup pouze k postavám, které vytvořil.
- **Zdůvodnění:** Tento požadavek byl označen jako Will not have, jelikož je zatím aplikace určena pro menší a uzavřené skupiny hráčů, kteří mezi sebou sdílí herní prvky i přístup k postavám. Tento požadavek bude přehodnocen, pokud by bylo v plánu aplikaci nasadit veřejně.

N6: Flexibilita pravidel

- **Důležitost:** Should have
- Aplikace bude sledovat základní pravidla hry Dungeons & Dragons, ale bude zároveň dostatečně flexibilní, aby bylo možné např. Proficiencies a Skills přidávat a upravovat dle potřeby. Zvláštní vlastnosti postavy s složitějšími interakcemi nebudou v některých případech promítnuty do výpočtů.

1.4 Případy užití

Jelikož tato práce pojednává o backend aplikaci, která bude poskytovat API pro frontend a nebude tak interagovat s uživateli přímo, nebudou zde popsány případy užití. Případy užití by odpovídaly jednotlivým koncovým bodům API, jejichž návrh bude popsán v následující kapitole.

Kapitola 2
Návrh

V této kapitole bude popsán návrh aplikace, který vychází z analýzy požadavků a bude sloužit jako základ pro implementaci. Nejprve budou porovnány a zvoleny techonologie, které budou tvořit architekturu aplikace. Poté bude navrženo vhodné API rozhraní a datový model aplikace. V závěru bude zvláštní pozornost věnována návrhu funkcionality pro import dat a generování PDF souborů.

2.1 Výběr technologií

V této sekci bude rozebráno, která rozhodnutí z hlediska použití technologií bylo potřeba pro vývoj projektu učinit. U těch hlavních budou analyzovány alternativy a zdůvodněn finální výběr.

2.1.1 DBMS

Základem pro backend každého business webového serveru je použití databázové technologie. Rozhraní systému má především zjišťovat a upravovat stav určených entit a ten musí být zachycen v databázi. Chceme tedy zvolit tzv. Database Management System (DBMS), tedy systém řízení bází dat, který bude vyhovovat potřebám projektu.

2.1.1.1 Relační DBMS

Patrně nejznámější a nejrozšířenější kategorií DBMS jsou ty relační (zvané RDBMS). Všechna data jsou uložena v tabulkách, kde každá tabulka reprezentuje relaci. Sloupce tabulky reprezentují prvky relace (atributy) a řádky tabulky reprezentují jednotlivé instance relace (záznamy). [15] Tento koncept navrhl již v roce 1970 Edgar F. Codd ve své práci. [16]

Hlavní předností těchto systémů je jejich schopnost zachytit složité vztahy mezi daty a zároveň zaručit integritu dat díky transakčnímu zpracování s vlastnostmi ACID (Atomicity, Consistency, Isolation, Durability). Naopak ale vyžadují pečlivé navržení struktury tabulek a její striktní dodržování a vzhledem k způsobu uložení dat je obtížné škálovat tyto systémy horizontálně, tedy přidáním dalších serverů. [15]

Rozhraní pro práci s těmito systémy bylo postupně standardizováno ve formě jazyka SQL (Structured Query Language), tím pádem je práce s různými RDBMS velmi podobná. Mezi nejrozšířenější RDBMS patří Oracle Database, MySQL, Microsoft SQL Server a PostgreSQL. [17]

2.1.1.2 NoSQL DBMS

NoSQL databátové systémy, kde výraz NoSQL znamená „Not only SQL“ nebo „Non-SQL“, jsou skupinou, do které se typicky řadí všechny DBMS, které nedodržují relační model. Místo toho ukládají data v jiné podobě, na základě čehož se dělí na další podkategorie. [18, 15] Mezi nejpoužívanější patří tyto:

Dokumentové – ukládají data jako dokumenty ve strukturovaných formátech podobných těm pro aplikační kód či komunikaci, nejčastěji JSON. To jim umožňuje stále udržovat strukturovanou a hierarchickou podobu dat, ale schéma ukládaných dokumentů může být flexibilnější oproti relačním databázím. [19] Nejznámějším zástupcem je MongoDB. [17]

Klíč-hodnota – ukládají data jako dvojice klíč-hodnota, kde klíč je unikátní identifikátor a hodnotou jsou libovolná data. Tento model je velmi jednoduchý a rychlý, ale zároveň neumožňuje složitější dotazy a vztahy mezi daty. Často se tak používá pro cachování. [18] Mezi nejznámější patří Redis. [17]

Grafové – ukládají data jako grafy, kde vrcholy reprezentují entity a hrany reprezentují vztahy mezi nimi. Tento model je vhodný pro ukládání vysoce propojených datových struktur. [18] Nejznámějším zástupcem je Neo4j. [17]

NoSQL systémy typicky využívají distribuovanou architekturu, což jim umožňuje snadnější horizontální škálování. Jejich specifický přístup k datům má v případě správného použití rychlejší dotazování. Některé z nich na druhou stranu negarantují ACID vlastnosti a nerelační modely často vedou k duplikaci dat. Tím pádem mohou mít větší nároky na prostor a být vhodné pouze pro specifické typy dotazů. [20]

2.1.1.3 Rozhodnutí pro DBMS

Pro potřeby tohoto projektu jsem se v aktuální fázi rozhodl použít dokumentový DBMS MongoDB. Tento výběr byl motivován především tím, že data herní postavy mají přirozenou stromovou strukturu, kterou lze zachytit v podobě dokumentu, tedy se hodí pro ukládání v MongoDB. Naopak pro data postav není potřeba modelování vztahů M:N, které je v MongoDB náročnější realizovat. MongoDB navíc stejně jako relační databáze podporuje transakce s vlastnostmi ACID. [21]

Jelikož MongoDB ukládá data ve formátu BSON, lze do něj snadno importovat JSON dokumenty [22]. Tím pádem bude snadnější proces importu uživatelského obsahu, jednoho z hlavních požadavků projektu, oproti importu těchto dokumentů do relační databáze.

Dále jsem zvolil MongoDB kvůli jeho jednoduchosti a rychlosti, které jsou pro prototypování a vývoj v raných fázích projektu velmi důležité, nebude tak zpočátku nutné řešit přepisování schématu.

2.1.2 Architektura API

Jako API, tedy Application Programming Interface, se obecně označuje rozhraní, které umožňuje komunikaci mezi dvěma softwarovými komponentami. Jedním z nejčastějších výskytů API je právě ve webových aplikacích, kde se využívá pro komunikaci mezi frontendovou a backendovou částí. Klient posílá požadavky na server, který je zpracovává a odpovídá na ně. Tímto způsobem může frontend dynamicky získávat a upravovat data, která jsou uložena na serveru, a zobrazovat je uživateli. Všechny webová API typicky pracují s protokolem HTTP, který je základem pro komunikaci na internetu. Postupem času se vyvinulo několik používáných architektur, které definují, jak by mělo API vypadat a jak by mělo fungovat, z nichž nejznámější jsou REST a GraphQL. [23]

2.1.2.1 REST

REST, zkratka pro Representational State Transfer, je styl pro návrh webové architektury, který popsal v roce 2000 Roy Fielding ve své disertační práci. Stanovuje následující obecná pravidla, která by měl systém dodržovat:

- **Klient–server** – systém by měl být rozdělen na dvě části, které jsou nezávislé na sobě a mohou se vyvíjet nezávisle.
- **Bezestavovost** – každý požadavek od klienta by měl obsahovat všechny informace potřebné k jeho zpracování, server by neměl ukládat žádný stav o klientovi mezi požadavky.
- **Cache** – odpovědi na požadavky by měly být označeny, zda je možné je ukládat do mezipaměti, aby se snížila zátěž serveru.
- **Jednotné rozhraní** – rozhraní mezi komponentami by mělo být jednotné, aby bylo snadné ho používat a bylo odděleno od implementace.
- **Vrstvení** – systém by měl být rozdělen do vrstev, kde každá vrstva pracuje pouze se sousedními vrstvami.
- **Kód na požádání** – server by měl klientovi poskytovat kód, který mu umožní rozšířit jeho funkcionalitu. Toto pravidlo bylo ovšem už původně označeno za volitelné a dnes se v ním v návrhu API obvykle nepracuje. [24]

Pro splnění pravidla jednotného rozhraní pak REST specifikuje tyto architektonické prvky, kterými se rozhraní vytvořená ve stylu REST odlišují od jiných [25, 26]:

- **Zdroje a identifikace** – základním stavebním prvkem rozhraní je zdroj (resource), který je definuje jako jakákoli pojmenovatelná informace. Požadavek pak vždy směřuje na konkrétní zdroj, který identifikuje.
- **Reprezentace a manipulace** – každý zdroj může mít různé reprezentace, není vázaný na konkrétní formát. Reprezentace, kterou získá klient, obsahuje všechny informace potřebné k manipulaci s tímto zdrojem. Toho server obvykle dosahuje posíláním metadat v reprezentaci zdroje, kterou poskytuje klientovi.
- **Samopopisující zprávy a odkazy** – každá odpověď serveru obsahuje informace, jak má klient reprezentaci zpracovat a jaké další akce může provést.

API dodržující tento styl označujeme za RESTful nebo pouze REST API. V praxi se pak webová REST API skládají z různých koncových bodů (endpoints). Každý koncový bod se skládá z URL adresy, která identifikuje zdroj, a HTTP metody, která určuje, jaká operace se s tímto zdrojem má provést. Pro reprezentaci zdrojů se v webových REST API nejčastěji používá formát JSON nebo XML, potřebná metadata jsou pak posílána v HTTP hlavičkách, které jsou součástí odpovědí serveru, jako např. Content-Type a Accept. [25, 26]

2.1.2.2 GraphQL

GraphQL je jazyk pro dotazování a manipulaci dat určený pro API, který v roce 2012 začala vyvíjet společnost Facebook a v roce 2015 byl uvolněn jako open-source. [27] GraphQL v kontextu webových API také používá HTTP, ale typicky má pouze jediný koncový bod (`/graphql`), na který se posílají všechny požadavky. [28] O GraphQL se nejčastěji hovoří v souvislosti se získáváním dat, kdy klient na server posílá dotazy specifikující, jaká data požaduje, a server mu podobnou strukturou vrací odpověď. Na serveru je nad daty definováno tzv. schéma, které popisuje, jaká data jsou dostupná na dotazování, jaké typy a argumenty mají a zda je lze i upravovat

pomocí mutací. [29, 30] Klient navíc může zjistit strukturu schématu pomocí tzv. introspekce, kdy server poskytuje speciální dotazy, které vrací informace o schématu. [31] Příklad použití tohoto jazyka je vidět na výpisech 2.1 a 2.2, kde klient požaduje získání jména a výšky ve stopách postavy s identifikátorem 1000. Server odpovídá požadovanými daty ve formátu JSON. [30]

```
{
  human(id: "1000") {
    name
    height(unit: FOOT)
  }
}
```

Výpis kódu 2.1 Dotaz v GraphQL [30]

```
{
  "human": {
    "name": "Luke Skywalker",
    "height": 5.6430448
  }
}
```

Výpis kódu 2.2 Odpověď v GraphQL [30]

2.1.2.3 Porovnání REST a GraphQL

GraphQL bylo vytvořeno jako moderní alternativa pro tvorbu API. Jeho hlavní výhodou je možnost získávat v jednom dotazu pouze ta data, která klient potřebuje, a tím snižovat zátěž serveru. REST API naopak poskytuje pevně definované koncové body, které vrací pevně definované reprezentace zdrojů, což může být v některých případech neefektivní. Existence schémat navíc znamená silnou typovou kontrolu a možnost introspekce, což nelze v REST API dosáhnout, jelikož se jedná pouze o sadu pravidel. Naopak mezi hlavní přednosti REST API patří jednoduchost implementace zvlášť na straně serveru, kdy není potřeba udržovat funkční schéma a logika je rozdělena do jednotlivých koncových bodů s jasně definovanými parametry. REST API může být navíc snadno cachováno a je stále mnohem více rozšířeno než GraphQL. [32, 33]

2.1.2.4 Rozhodnutí pro API

Po zvážení těchto faktorů a jejich konzultaci s Žanetou jsem se rozhodl pro použití REST API v tomto projektu. Důvodem bylo zejména to, že aplikace nebude v současné formě vyžadovat složité dotazy, které by mohlo GraphQL usnadnit. Oba máme zkušenosti s nástroji a knihovnami pro práci s REST API, zatímco GraphQL by pro nás oba bylo novým konceptem, který by vyžadoval další učení a testování. Jako formát pro reprezentaci dat jsem zvolil JSON, jelikož je v současné době nejrozšířenější a existuje tak pro něj největší podpora v rámci knihoven. Zároveň je snadno čitelný a psaný člověkem, což je výbornou vlastností pro testování a ladění API.

2.1.3 Framework pro backend server

Po zvolení DBMS a typu API je třeba rozhodnout, jak bude implementován samotný server. Ten musí přes API přijímat požadavky, vyhodnocovat je, na základě nich provádět operace nad databází a výsledky posílat zpět klientovi. Jelikož se jedná o častou úlohu a je přesně dáno

rozhraní komunikace s klientem i databází, existuje mnoho frameworků, které vývojáře od těchto detailů abstrahují a umožňují jim se soustředit zejména na implementaci business logiky serveru.

Některé frameworky podporují kromě API přístupu i tvorbu HTML šablon a renderování stránek na serveru, což ale pro tento projekt není potřeba, jelikož frontend ve své práci implementuje Žaneta v podobě tzv. Single Page Application (více podrobností popisuje Žaneta ve své práci).

V následujících sekcích budou postupně rozebrány faktory, které vedly k výběru konkrétního frameworku.

2.1.3.1 Rozšířenost

Rozšířenost technologie je důležitá, jelikož větší popularita znamená větší množství dostupné dokumentace, řešených problémů či podpůrných modulů. Do výběru byly proto v první řadě zařazeny frameworky, které byly v průzkumu Stack Overflow v roce 2023 označeny profesionálními vývojáři za nejpoužívanější. Otázka se týkala všech webových frameworků a technologií, ale do výsledné tabulky byly zařazeny pouze ty, které jsou určeny pro tvorbu API a serverové logiky. Výsledky lze vidět v tabulce 2.1.

Název	Jazyk	Používanost
Express.js	JavaScript	19.5%
ASP.NET Core	C#	18.9%
Next.js	JavaScript	17.3%
Spring Boot	Java	13.5%
Flask	Python	11.2%
Django	Python	10.9%

Tabulka 2.1 Nejpoužívanější backendové frameworky v roce 2023 [34]

2.1.3.2 Technické aspekty

Dalším aspektem je znalost programovacího jazyka, na kterém je framework postavený. Tímto způsobem jsem do užšího výběru nezařadil Python frameworky, jelikož nemám s tímto jazykem téměř žádné zkušenosti a vývoj by tak pro mě byl těžký. Dále jsem vyřadil Express.js i Next.js kvůli jejich architektuře. Express je minimalistický framework poskytující pouze základní funkce pro tvorbu koncových bodů API, což by znamenalo nutnost hledání dalších knihoven pro splnění požadavků. [35, 36] Next.js je framework zaměřený především na tzv. server-side rendering webových stránek, což, jak bylo zmíněno, není součástí této práce. [37]

Oba zbývající frameworky, tedy ASP.NET Core a Spring Boot, splňují všechna kritéria, která jsem pro výběr stanovil. Jsou kompatibilní s knihovnou pro tvorbu PDF formulářů v podobě knihoven Apache PDFBox (Java) [38] a PDFSharp (C#) [39] potřebnou ke splnění požadavku F14. Byť to v této fázi projektu není klíčové, oba frameworky se řadí mezi nejvýkonnější a jsou schopny rychle zpracovávat větší množství požadavků, jak dokládá benchmarking provedený společností TechEmpower v roce 2023 [40]. Od použití objektově orientovaného jazyka, jako je Java i C#, navíc očekávám dobrou možnost implementace společných částí pro obsluhu různých typů obsahu díky generickým třídám [41, 42] a polymorfismu [43, 44].

2.1.3.3 Rozhodnutí pro framework

Pro implementaci jsem nakonec zvolil framework Spring včetně jeho nadstavby Spring Boot z důvodu větší předchozí zkušenosti s Springem i obecně s ekosystémem Javy. Spring Boot navíc poskytuje velmi rychlý způsob vytvoření nového projektu s přednastavenými závislostmi, konfigurací a strukturou projektu [45].

2.2 API

Tato sekce se bude věnovat návrhu koncových bodů API, které budou vytvořeny v rámci této práce. Koncové body budou navrženy tak, aby splňovaly požadavky definované v analýze a byly co nejvíce v souladu s principy REST API.

2.2.1 HTTP metody

Pro správný návrh koncových bodů REST API je potřeba představit hlavní HTTP metody a jejich vlastnosti, aby při návrhu nebyl některý z těchto principů porušen. Tyto metody pro práci se zdroji jsou uvedeny v tabulce 2.2.

Metoda	Popis	Safe	Idempotent
GET	Získání reprezentace zdroje	Ano	Ano
POST	Odeslání dat na zdroj pro specifické zpracování[1]	Ne	Ne
PUT	Nahrazení reprezentace zdroje	Ne	Ano
DELETE	Odstranění reprezentací zdroje	Ne	Ano

Tabulka 2.2 HTTP metody v REST API [46]

Tyto vlastnosti mají následující význam:

Safe – metoda je Safe, pokud nemění stav serveru, tedy provádí pouze čtení dat. Každá Safe metoda by měla být zároveň Idempotent. [47]

Idempotent – metoda je Idempotent, pokud je výsledný stav serveru stejný, ať je stejný požadavek klientem proveden jednou nebo vícekrát. Na každý z těchto požadavků může server odpovědět jinak, ale jeho stav by neměl být ovlivněn tím, kolikrát byl požadavek zadán. Například první požadavek pomocí DELETE odstraní zdroj ze serveru a server odpoví kódem 200 OK, na další požadavky odpoví kódem 404 Not Found a nezmění svůj stav, neboť zdroj již na serveru neexistuje. [48]

Pro některé operace se zdroji se používají i další metody, jako např. PATCH pro částečné nahrazení zdroje, ale tyto metody nebudou v tomto návrhu použity. Standard také specifikuje pomocné metody jako např. OPTIONS a HEAD [46], ale jejich funkčnost v API framework Spring Boot zajišťuje automaticky [49] a pro návrh tedy není třeba se jimi zabývat.

2.2.2 Předání parametrů a dat

V poslední řadě je potřeba zmínit, jakými způsoby mohou být na koncový bod předány parametry a další data potřebná k zpracování požadavku. Spring Boot, podobně jako jiné frameworky pro tvorbu webových aplikací, podporuje tyto způsoby:

Parametry cesty umožňují parametrizovat přímo URL adresu koncového bodu, kde se pak parametrizovaná část URL uvádí ve složených závorkách (např. `/api/people/{id}`). Framework pak požadavky s různými hodnotami parametrů směřuje na stejný koncový bod, kde jsou hodnoty parametrů dostupné. [49]

Parametry dotazu umožňují předat data na konci URL adresy za otazníkem, kde mají parametry formát *klíč=hodnota* a jsou odděleny znakem & (např. `/api/people?page=0&size=10`). Tyto parametry se nejčastěji používají pro filtrování, třídění a stránkování dat, která jsou vracena z koncového bodu. [50]

[1] používáno pro vytvoření nového zdroje

Tělo požadavku umožňuje předat libovolná další data, v případě této aplikace ve formátu JSON. Tímto způsobem lze předávat složitější strukturovaná data, jako například nové nebo upravené reprezentace zdrojů, které má server zpracovat. [51]

2.2.3 Návrh koncových bodů

Na základě těchto principů a požadavků z analýzy jsem navrhl následující koncové body pro API, kde je vždy uvedena použitá HTTP metoda, URL adresa, popis a další informace o koncovém bodu. Všechny koncové body budou začínat předponou `/api`, což slouží k jasnému oddělení API od dalších potenciálních částí aplikace.

API obsahu

- `GET /api/`*`contentType`*`/{id}`, kde *contentType* je jeden z typů obsahu (armor, backgrounds, classes, languages, races, spells, tools, weapons). Vrátí položku obsahu daného typu s daným ID, pokud existuje.
  - Parametry: `id` – identifikátor položky obsahu.
  - Odpověď: 200 OK – položka obsahu vrácena, 404 Not Found – položka obsahu s daným ID neexistuje.
- `GET /api/`*`contentType`*, kde *contentType* je jeden z typů obsahu. Vrátí seznam všech položek obsahu daného typu splňující zadané filtry, pokud jsou specifikovány. Volitelně používá stránkované rozhraní pomocí parametrů `page` a `size`, jinak jsou všechny položky vráceny na jedné stránce.
  - Parametry: `name` – hledané názvy položek, vyhledává podřetězec, tedy položky obsahující zadaný řetězec v názvu, `source` – seznam ID (zkratek) zdrojů, ze kterých mají být položky vráceny, `page` – nulou indexovaná stránka výsledků, `size` – počet položek na stránku.
  - Odpověď: 200 OK – seznam položek obsahu vrácen (může být prázdný v případě žádné shody).

API zdrojů obsahu

- `GET /api/sources/{id}`. Vrátí zdroj obsahu s daným ID, které je zkratkou názvu zdroje, pokud existuje.
  - Parametry: `id` – identifikátor zdroje obsahu.
  - Odpověď: 200 OK – zdroj obsahu vrácen, 404 Not Found – zdroj obsahu s daným ID neexistuje.
- `GET /api/sources`. Vrátí seznam všech zdrojů obsahu, které jsou dostupné v systému.
  - Odpověď: 200 OK – seznam zdrojů obsahu vrácen.

API postav

- `GET /api/characters/{id}`. Vrátí kompletní informace o postavě s daným ID, pokud existuje.
  - Parametry: `id` – identifikátor postavy.
  - Odpověď: 200 OK – kompletní informace o postavě vráceny, 404 Not Found – postava s daným ID neexistuje.

- `GET /api/characters`. Vrátí seznam všech postav, které jsou dostupné v systému.
  - Odpověď: 200 OK – seznam postav vrácen.
- `POST /api/characters`. Vytvoří novou postavu s poskytnutými daty. Postava je uložena a vrácena.
  - Tělo požadavku: `CharacterInput` – vstupní data pro vytvoření postavy.
  - Odpověď: 200 OK – postava vytvořena a vrácena, 400 Bad Request – neplatná vstupní data.
- `PUT /api/characters/{id}`. Přepíše existující postavu novými daty, ale zachovává ID postavy. Postava je uložena a vrácena.
  - Parametry: `id` – identifikátor postavy.
  - Tělo požadavku: `CharacterInput` – nová data pro postavu.
  - Odpověď: 200 OK – postava upravena a vrácena, 400 Bad Request – neplatná vstupní data, 404 Not Found – postava s daným ID neexistuje.
- `DELETE /api/characters/{id}`. Odstraní postavu s daným ID. Postava je trvale odstraněna ze systému.
  - Parametry: `id` – identifikátor postavy.
  - Odpověď: 200 OK – postava odstraněna, 404 Not Found – postava s daným ID neexistuje.
- `PUT /api/characters/{id}/skills`. Upraví Skills postavy poskytnutým seznamem Skills, ve kterých má postava zkušenosti. Nový seznam Skills je uložen a vrácen.
  - Parametry: `id` – identifikátor postavy.
  - Tělo požadavku: `List<SkillInput>` – seznam Skills a jejich zkušeností.
  - Odpověď: 200 OK – Skills upraveny a vráceny, 400 Bad Request – neplatná vstupní data.
- `PUT /api/characters/{id}/weapons`. Upraví zbraně postavy poskytnutým seznamem ID zbraní, kterými je postava vybavena. Nový seznam je uložen a vrácen.
  - Parametry: `id` – identifikátor postavy.
  - Tělo požadavku: `List<String>` – seznam ID zbraní.
  - Odpověď: 200 OK – zbraně upraveny a vráceny, 400 Bad Request – neplatná vstupní data.
- `PUT /api/characters/{id}/armor`. Upraví zbroj postavy poskytnutým ID zbroje, kterou postava nosí. Nová zbroj je uložena a vrácena.
  - Parametry: `id` – identifikátor postavy.
  - Tělo požadavku: `IdWrapper` – objekt s jedním ID zbroje.
  - Odpověď: 200 OK – zbroj upravena a vrácena, 400 Bad Request – neplatná vstupní data.
- `PUT /api/characters/{id}/spells`. Upraví kouzla postavy poskytnutým seznamem ID kouzel, která postava zná. Nový seznam je uložen a vrácen.
  - Parametry: `id` – identifikátor postavy.
  - Tělo požadavku: `List<String>` – seznam ID kouzel.
  - Odpověď: 200 OK – kouzla upravena a vrácena, 400 Bad Request – neplatná vstupní data nebo postava s daným ID nemůže kouzlit.

- `POST /api/characters/{id}/levelup`. Zvýší úroveň postavy o jedna. Postava je upravena a vrácena.
  - Parametry: `id` – identifikátor postavy.
  - Odpověď: 200 OK – postava zvýšena a vrácena, 400 Bad Request – postava dosáhla maximální úrovně, 404 Not Found – postava s daným ID neexistuje.
- `GET /api/characters/{id}/pdf`. Vygeneruje PDF s informacemi o postavě. PDF je vráceno jako stažitelný soubor.
  - Parametry: `id` – identifikátor postavy.
  - Odpověď: 200 OK – PDF vygenerováno a vráceno, 404 Not Found – postava s daným ID neexistuje.

2.3 Návrh datových objektů

Dalším klíčovým krokem v návrhu aplikace bylo vytvořit strukturu datových objektů, které budou reprezentovat jednotlivé druhy obsahu a postavy v systému. Tuto strukturu budou mít dokumenty v MongoDB, JSON soubory pro import dat a datové třídy v aplikaci, jelikož bude nutné tyto formáty na sebe mapovat. V rámci požadavku F1 bylo stanoveno, že aplikace bude obsahovat následující typy obsahu: zdroje, třídy, rasy, zázemí, jazyky, nástroje, kouzla, zbraně, zbroje a také postavy, které budou využívat prvky všech těchto typů.

2.3.1 Struktura objektů

Tvorba datových objektů byla založena na pravidlech a požadavcích z analýzy. V první řadě bude mít dle požadavku F1 každý herní prvek pole identifikátoru (`_id`), jména (`name`) a zdroje (`source`), přičemž záznam zdroje bude sám tvořen unikátní identifikační zkratkou[2]. Další pole budou záviset na typu obsahu a budou odpovídat vlastnostem stanoveným v sekci 1.1.1. Za zmínku stojí následující dva druhy polí, které se opakují u více druhů obsahu a bylo pro ně tak zvoleno jednotné rozhraní:

Zisk Proficiency – možnost zisku Proficiency v kategoriích Skills, nástrojů a jazyků se objevuje u tříd, ras a zázemí v různých podobách, jako např. seznam Proficiencies, které postava získá automaticky, výběr z určité množiny či volný výběr do určitého počtu. Jelikož hráči v tomto ohledu požadovali flexibilitu, budou tato pole vždy obsahovat záznam o maximálním počtu, které lze získat (`amount`), a případně seznam výchozích Proficiencies (`defaults`). Systém bude dohlížet pouze na tento limit, ale bude umožňovat libovolně měnit zvolené Proficiencies, jak bylo požadováno.

Zvláštní vlastnosti – třídy, rasy a zázemí poskytují postavě vlastnosti různého charakteru. Ty je složité reprezentovat je jednotně. Proto bude seznam zvláštních vlastností ponechán pouze jako textový. Každá položka bude obsahovat název (`title`), popis (`text`) a úroveň postavy, od které je platná (`levelMinimum`). Stěžejní vlastnosti s velkým dopadem na tvorbu postavy, jako např. možnost kouzlení, budou reprezentovány samostatně. Velká část těchto vlastností poskytuje specifické bonusy, které by bylo obtížné reprezentovat v deníku postavy, proto zůstanou pouze v textové podobě.

Příklad struktury objektu pro rasu, na němž je patrné zohlednění formátu výše zmíněných polí, je vidět v ukázce 2.3.

[2]Pro oficiální knihy je již zavedená konvence zkratek tvořená písmeny z názvu knihy, jako např. PHB pro *Player's Handbook*.

```
{
    "_id": {"$oid": "66381c3cb1208b620221ec4d"},
    "name": "Half-Elf",
    "source": {"_id": "srd", "name": "System Reference Document"},
    "description": "Your half-elf character has some qualities ...",
    "features": [
        {
            "title": "Darkvision",
            "text": "Thanks to your elf blood, you have superior vision...",
            "levelMinimum": 1
        },
        //{ ... }
    ],
    "speed": 30,
    "sizes": ["medium"],
    "languages": {
        "amount": 3,
        "defaults": [
            {
                "_id": {"$oid": "66381c3ab1208b620221ec08"},
                "name": "Common",
                "exotic": false,
                "source": {"_id": "srd", "name": "System Reference Document"}
            },
            //{ ... }
        ]
    },
    "abilities_plus_2": {
        "amount": 1,
        "defaults": ["charisma"]
    },
    ...,
    "skills": {
        "amount": 2,
        "defaults": []
    }
}
```

Výpis kódu 2.3 Ukázka struktury objektu pro rasu

2.3.2 Vazby mezi objekty

Dalším důležitým aspektem návrhu bylo rozhodnutí, jak budou realizovány odkazy mezi objekty. Prvním krokem byla identifikace vazeb dat, v rámci níž se objevuje následující:

- Všechny ostatní typy obsahu se odkazují na zdroj.
- Třídy, rasy a zázemí se navíc odkazují na nástroje a jazyky.
- Postava se může odkazovat na všechny ostatní typy. Vždy obsahuje seznam použitých zdrojů a zvolenou třídu, rasu a zázemí. Může obsahovat seznamy zbraní, zbroje, kouzel, jazyků a nástrojů.

Z hlediska kardinality se u všech těchto vazeb jedná o vztahy 1:1 nebo 1:N. Pro jejich realizaci MongoDB nabízí způsoby popsané v následujících sekcích.

2.3.2.1 Vkládání (embedding)

Vztažené dokumenty jsou vnořeny přímo do rodičovského dokumentu. To je možné díky použití formátu BSON, který umožňuje jiný dokument vložit jako objekt nebo pole objektů. Tento způsob je vhodný v případě, kdy jsou vložené dokumenty menší a nemění se příliš často, jelikož změna vnořeného dokumentu vyžaduje úpravu všech rodičovských dokumentů, které s ním mají tuto vazbu. Pokud je v rámci vztahu 1:N vloženo pole dokumentů, musí být věnována pozornost 16 MB limitu velikosti dokumentu v MongoDB. Jeho hlavní výhodou je rychlý přístup ke všem informacím v rámci jednoho dotazu. [52]

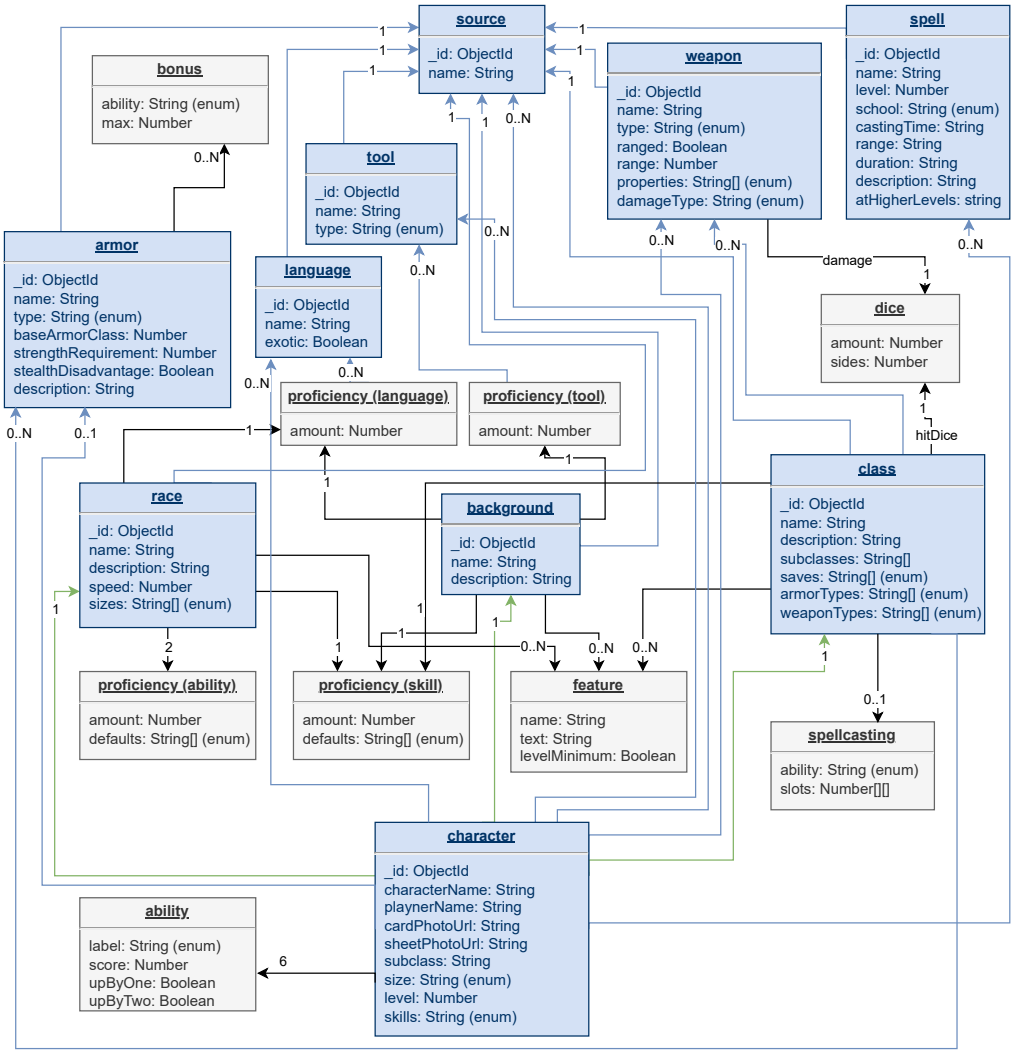
2.3.2.2 Odkazování (referencing)

Vztažené dokumenty jsou v rodičovském dokumentu reprezentovány pouze pomocí identifikátoru a samotné záznamy jsou uloženy v jiné kolekci. Jedná se tak o obdobu cizího klíče v relačních databázích. Tento způsob je vhodný pro objemnější dokumenty, které by bylo nevhodné kopírovat do každého rodičovského dokumentu, a pro záznamy, které se často mění, jelikož není nutné upravovat dokumenty, které na ně odkazují. [52] Při čtení je ovšem nutné pro získání odkázaného dokumentu použít operátor `$lookup`, který provede „join“ operaci mezi dvěma kolekcemi [53].

2.3.2.3 Použití v aplikaci

V aplikaci byly využity oba způsoby. Vazby na třídy, rasy a zázemí byly realizovány pomocí odkazování, jelikož se jedná o dokumenty, kde mohou být rozsáhlé textové popisy, které by bylo neefektivní kopírovat do dokumentu každé postavy, která je využívá. Ostatní vazby byly realizovány pomocí vkládání, jelikož se jedná o menší dokumenty, u kterých je známa přibližná velikost.

Celý datový model, včetně navržené struktury a znázornění vazeb, je možné vidět na diagramu 2.1. Modře podbarvené objekty tvoří samostatné kolekce dat, zatímco šedě podbarvené objekty ne (jedná se o vnořené objekty v rámci jiných objektů). Modře vyznačené vazby na objekty kolekcí jsou realizovány vkládáním, zatímco zeleně vyznačené vazby odkazováním. Černě vyznačené vztahy vedou k objektům nemající vlastní kolekci a jsou tak vždy vložené.



Obrázek 2.1 Datový model

2.3.3 Import dat

S problematikou datového modelu je také úzce spjat návrh řešení importu obsahu do aplikace (viz požadavek F2). Zpočátku byly zvažovány dva odlišné přístupy k importu dat, které byly vypozorovány i v existujících aplikacích: import vyplněním formuláře s potřebnými informacemi a import přímo nahráním datového souboru. Tento formulář by v aplikaci dávalo smysl umístit na webový frontend. Ten řeší ve své práci Žaneta, která již měla požadavků na implementaci dostatek, tudíž bylo rozhodnuto se zaměřit na import pomocí datových souborů.

Tento způsob je také výhodný v tom, že umožňuje importovat velké množství dat najednou, což je užitečné pro rychlé zpracování např. obsahu z celé knihy. Jelikož aplikace cílí na pokročilé hráče, kteří znají informační strukturu herních prvků, neměli by uživatelé mít problém s přípravou datových souborů za předpokladu, že jim bude formát popsán a demonstrován na příkladech položek, které znají z oficiálních zdrojů.

Dále bylo nutné navrhnout, jaký formát má být pro soubory použit. Vzhledem k použití MongoDB pro perzistenci dat bude nejvhodnější použít formát JSON. Ten navíc v porovnání s jinými strukturovanými formáty, jako např. XML, neobsahuje tolik speciálních znaků a práce s ním je tak jednodušší i pro uživatele, kteří budou datové soubory připravovat ručně.

Tím pádem bude struktura souborů k importu odpovídat struktuře dokumentů v MongoDB, která byla popsána v předchozí sekci. Je ovšem potřeba vyřešit problém, jak v souborech pro import odkazovat na jiné položky, aby byli uživatelé dostatečně abstrahováni od interních identifikátorů a realizace vazeb v MongoDB.

Bylo tak využito skutečnosti, že každá položka obsahu má název a zdroj, který spolu s jejím typem slouží k identifikaci (nepředpokládá se existence dvou stejně pojmenovaných prvků stejného typu v jednom zdroji pravidel). Uživatel tak vytvoří odkaz definováním zvláštního objektu v místě souboru, kde by měl být vložen či referencován jiný dokument. Tento objekt pro referenci bude obsahovat pole `ref` s hodnotou typu obsahu (cílové kolekce) a pole `name` s názvem cílové položky. Při importu bude tento objekt nahrazen odpovídající položkou z databáze.

Celé prostředí importu bude připraveno tak, že budou vytvořeny složky pro jednotlivé typy obsahu, do kterých budou uživatelé vkládat soubory. Název souboru bude odpovídat názvu zdroje, ze kterého pochází obsažené položky, a bude obsahovat pole objektů, kde každý objekt reprezentuje jednu položku. Uživatel u objektů zadá stejná pole, jako jsou obsažena v MongoDB dokumentech, kromě `_id`, které databáze vygeneruje, a `source`, které bude nahrazeno dle pojmenování souboru. Tento formát zajistí, že se uživatelé nebudou muset zabývat identifikátory ani metadaty položky jako typ a zdroj. Zároveň bude tato struktura složek a souborů dobře strojově zpracovatelná a třídění bude přirozené pro uživatele.

2.4 Návrh modulu pro generování PDF

Generování PDF deníku postavy specifikované v požadavku F14 je další funkcionalitou, u které je potřeba vytvořit podrobný návrh. V rámci požadavku bylo stanoveno, aby aplikace negenerovala PDF jako statický dokument, ale aby předvyplnila formulářová pole v PDF šabloně představující deník postavy. Interaktivní formuláře lze v rámci PDF vytvářet pomocí techonologií AcroForms nebo XFA (XML Forms Architecture). Pro tyto účely bude vhodné použít AcroForms, jelikož jsou podporovány požadovanými prohlížeči a lze v nich snadno vytvářet pole typu text field, check box, radio button a list, což je dostačující pro deník postavy. XFA je určeno k tvorbě složitějších formulářů se skriptováním pomocí XML, ale není podporováno ve webových prohlížečích a jeho funkcionalita není potřebná pro tento projekt. [54]

Z pohledu zpracování šablony hráči označili jako preferovanou možnost vyplňování oficiálního PDF deníku, který také používá AcroForms, ovšem kvůli jeho nejasné licenci ho není možné distribuovat v rámci bakalářské práce. Proto bylo rozhodnuto vytvořit vlastní PDF šablonu deníku, která bude také vytvořena pomocí AcroForms a bude obsahovat potřebná pole. Pro tvorbu šablony byl zvolen open-source textový procesor LibreOffice Writer. [55] Ten poskytuje WYSIWYG tvorbu dokumentů včetně vkládání statických textových a interaktivních formulářových polí. Výsledek poté lze exportovat do PDF, přičemž je nutné použít možnost „Create PDF form“, která do výsledku vloží AcroForms, a zároveň je vhodné soubor exportovat jako standardizované archivní PDF verze 3 (PDF/A-3), které zajistí vložení použitých fontů do souboru a zvýšení kompatibility mezi prohlížeči. [56]

Bylo zvoleno konzistentní pojmenování polí odkazující na jejich obsah, které umožní jejich snadné strojové zpracování. To se uplatní zejména u textových polí, které společně reprezentují jeden seznam, kdy budou pojmenovány jednotně s přidaným indexem či identifikátorem, což umožní snadné iterování přes celý seznam v kódu. Celé schéma jmen AcroForms polí je uvedeno v příloze A. U delších textových polí nastaveno zalamování textu a u kratších polí naopak jednořádkový režim s vhodným zarovnáním textu uvnitř pole.

Na přiloženém médiu je k dispozici zdrojový soubor LibreOffice Writer ve formátu ODT, který lze použít pro úpravu šablony, i výsledný PDF soubor šablony, který je použit pro gene-

rování deníku postavy. Při dodržení výše zmíněného pojmenování polí a formátu PDF tak může potenciální zájemce nahradit soubor vlastní šablonou či adaptovanou oficiální šablonou deníku, která je pro osobní použití k dispozici na oficiálních stránkách Wizards of the Coast. [57]

Kapitola 3
Implementace

Tato kapitola popisuje implementaci navržených částí aplikace. V první části je popsán postup vytvoření projektu ve frameworku Spring Boot a použité moduly a závislosti. V další části je popsána implementace jednotlivých částí aplikace. Na závěr je popsána kontejnerizace aplikace a dosažené výsledky implementace.

3.1 Vytvoření projektu

Pro vytvoření nového projektu ve frameworku Spring Boot je možné použít webový nástroj Spring Initializr, který umožňuje rychle nakonfigurovat potřebné závislosti a vygenerovat celou základní strukturu projektu. Tu je poté možné stáhnout, rozbalit do vývojového prostředí a začít s implementací. Nástroj jsem využil a pro projekt jsem ponechal standardní předvybrané nastavení projektu: implementaci v jazyce Java verze 17, binarní soubory ve formátu JAR, nástroj pro sestavení a správu závislostí Gradle s konfiguračními soubory v jazyce Groovy a nejnovější stabilní verzi frameworku Spring Boot. [58]

Jelikož se verze Spring Bootu několikrát v průběhu vývoje změnila o minoritní verze (které mohou obsahovat opravy chyb a nenarušují zpětnou kompatibilitu), průběžně jsem verzi aktualizoval. Díky tomu, že Spring Boot má pro Gradle plugin, který automaticky spravuje všechny moduly a interní závislosti frameworku, pro změnu verze stačilo upravit řádku s verzí v konfiguračním souboru `build.gradle`. Při dalším sestavení projektu Gradle automaticky stáhne a začne používat novou verzi frameworku. Na dostupnost nové verze pak upozorní rozšíření pro Spring Boot ve Visual Studio Code v podobě varování.

Pro pokrytí potřebné funkcionality a usnadnění vývoje jsem přidal ve Spring Initializr tyto závislosti [58]:

- **Spring Web** pro vytvoření REST API a obsluhu HTTP požadavků,
- **Spring Data MongoDB** pro práci s MongoDB databází a mapování dokumentů na Java objekty,
- **Spring Boot DevTools** pro rychlejší vývoj a ladění aplikace,
- **Lombok** pro snížení ručně psaného kódu.

Tyto závislosti budou popsány v následující sekci, spolu s dalšími závislostmi, které jsem přidal během implementace.

3.2 Použité moduly a závislosti

V této sekci budou rozebrány jednotlivé části frameworku Spring a další závislosti, které byly použity.

3.2.1 Spring Framework a jádro aplikace

Spring Framework je projekt, který slouží pro vývoj podnikových (enterprise) aplikací v Javě[1]. Je rozdělen do mnoha modulů, přičemž vždy jsou přítomny moduly jádra frameworku. [59]

V jádru frameworku stojí tzv. IoC kontejner, který pracuje na principu inverze řízení (Inversion of Control – IoC) pomocí techniky Dependency Injection (DI). Tato technika funguje tak, že objekty aplikace nevytváří jiné objekty na kterých závisí, ale definují je jako závislosti typicky pomocí parametrů konstruktoru nebo označených členských proměnných. Kontejner, který řídí aplikaci, pak zajišťuje, že při vytváření objektu jsou mu závislé objekty předány. Ve Springu se spravované objekty označují jako tzv. beany. Tento návrhový vzor se snaží oddělit vytváření objektů od jejich použití a snížit vazby mezi částmi aplikace. [60, 61]

Konfigurační metadata říkající, které beany má kontejner spravovat a jak, mohou být kontejneru dodány pomocí XML souborů, ovšem dnes se obvykle preferuje použití konfigurace Java kódem se specifickými anotacemi (třída označená anotací `@Configuration` a metody označené anotací `@Bean`). Tyto beany pak tvoří páteř celé aplikace a mohou být využity, kdekoliv je potřeba. [62]

3.2.2 Spring Boot

Spring Boot je nadstavba nad Spring Frameworkem, která má za cíl, jak sami popisují její tvůrci, usnadnit tvorbu aplikací produkční kvality, které lze „prostě spustit“. [63] Jedná se o tzv. opinionated framework, což znamená, že zavádí konvenční způsoby, jak psát a strukturovat aplikační kód [64], a na základě těchto konvencí automaticky konfiguruje aplikaci. Mnoho konfigurací lze upravit, obvykle je k tomu použit soubor `application.properties`, kde lze nastavit vlastnosti aplikace, které se mají lišit od výchozích hodnot. [65]

Spring Boot také poskytuje modulu Spring Web vestavěný webový server, což umožňuje spustit aplikaci jako samostatný JAR soubor obsahující vše potřebné. [63] V praxi to znamená, že k napsání funčního REST API se zvolenými moduly stačí vytvořit odpovídající třídy s pracovním kódem, označit je anotacemi komponent jako např. `@RestController`, třídu obsahující metodu `main` anotovat jako `@SpringBootApplication` a vložit do ní jediné volání metody Spring Bootu, které inicializuje celou aplikaci. [66]

3.2.3 Spring Web MVC

Spring Web MVC je modul frameworku Spring, který poskytuje nástroje pro tvorbu webových aplikací. [67] Jak název napovídá, implementuje návrhový vzor MVC (Model-View-Controller). Ten rozděluje aplikaci na tři části: model, který obsahuje data a logiku aplikace, view, který zobrazuje data uživateli, a controller, který zpracovává požadavky uživatele a volá metody modelu pro získání dat. [68] Část view však není součástí této práce, jelikož ji zajišťuje frontendová část aplikace. V případě této backend aplikace budou tak z Spring Web MVC využity především funkcionality pro tvorbu kontrolerů a REST API.

[1] a v novějších verzích i dalších JVM jazyků Kotlin a Groovy

3.2.4 Spring Data MongoDB

Spring Data MongoDB je modul frameworku Spring pro integraci s MongoDB databází. [69] S databází umožňuje pracovat přes třídu `MongoTemplate`, která nabízí metody podobné příkazům jako při práci s MongoDB z příkazové řádky, ale také rozšířené rozhraní pro tvorbu složitějších dotazů. [70] Další možností je pak použití tzv. repository rozhraní, které jsou hlavní abstrakcí Spring Data pro celou vrstvu přístupu k datům (tzv. DAO – Data Access Object). Ty už přímo definují metody pro základní operace nad daty, jako je vytvoření, čtení, aktualizace a odstranění (CRUD operace), ale podporují i rozšíření o vlastní metody pro některé složitější operace. Spring Data MongoDB pak k těmto rozhraním vytvoří implementace, které je možné použít v aplikaci. [71]

Pro oba přístupy modul poskytuje vrstvu objektového mapování, která převádí MongoDB dokumenty na Java objekty a naopak. Ve veškerém aplikačním kódu tak lze pracovat pouze s doménovými objekty. [72] Při použití Spring Bootu opět není potřeba žádnou z těchto tříd konfigurovat, stačí pouze předat URI databáze do aplikační konfigurace a Spring Boot vytvoří a nastaví potřebné beany.

3.2.5 Spring Boot DevTools

Spring Boot DevTools je pomocný modul Spring Bootu určený pro rychlejší vývoj aplikace. Jeho hlavní funkcí je automatické restartování aplikace při detekci změn v kódu. Dále upravuje některé výchozí konfigurace Spring Bootu, zejména týkající se cachování a podrobnějšího výpisu chyb v API. Jelikož tyto nastavení naopak nejsou žádoucí v produkčním prostředí, DevTools jsou automaticky zakázány, pokud je aplikace spuštěna ze sbaleného JAR souboru. [73]

3.2.6 Lombok

Project Lombok je knihovna pro jazyk Java, která pomocí anotací generuje kód s cílem redukce psaní tzv. boilerplate kódu. Umožňuje tak implementovat často používané konstrukce a vzory bez nutnosti je ručně psát a udržovat. [74] Konkrétní využití Lomboku v této aplikaci bude popsáno v následujících sekcích.

3.3 Postup implementace

V následující sekci jsou popsány kroky implementace jednotlivých částí aplikace a techniky, které byly využity.

3.3.1 Modely

Nejprve bylo potřeba implementovat třídy modelů, tedy datové třídy, které reprezentují jednotlivé prvky hry a samotné postavy. Pro jednotlivé importovatelné prvky byla vytvořena abstraktní třída `SourcableEntity` reprezentující herní prvek s názvem a zdrojem (z jaké sady pravidel pochází). Tato abstraktní třída později umožnila jednotné zpracování všech prvků pomocí těchto atributů v rámci společného rozhraní. Konkrétní použité techniky při implementaci modelů lze ukázat na příkladech 3.1 a 3.2 (pro stručnost výpisů jsou vynechány importy a formátování je upraveno).

Všechny modely byly anotovány jako `@Value`, anotací Lomboku pro neměnné datové třídy. To modelům označí všechny členské proměnné jako `final`, vytvoří jim getter metodu a třídě implementuje konstruktor a správné přetížení metod `equals`, `hashCode` a `toString`. [75] Je

to doporučený přístup Spring Data MongoDB pro mapované objekty, jelikož je umožňuje vytvářet přímo přes konstruktor se všemi parametry (to je až o 30% rychlejší než vytváření pomocí reflexe a setter metod) a zároveň zaručuje, že aplikační kód nebude manipulovat se stavem objektu, který má odpovídat dokumentu v databázi. [72] Pro snadné vytváření nových či upravených instancí postav byla přidána anotace Lomboku `@Builder` s parametrem `toBuilder = true` vytvářející doplňující statickou třídu návrhového vzoru Builder. Tu lze použít např. jako `Character.builder().name(„frodo").level(1).build()` pro vytvoření objektu s názvem „frodo“ a úrovní 1, nebo `bilbo.toBuilder().level(2).build()` pro vytvoření kopie objektu `bilbo` s úrovní změněnou na 2. [76] U postavy byly dále vytvořeny další getter metody pro výpočet odvozených atributů.

Objekty vytvořené z těchto tříd jsou zároveň mapovány na dokumenty z databáze MongoDB, proto bylo potřeba je vytvořit tak, aby mapování Spring Data MongoDB fungovalo správně. Názvy členských proměnných tak byly voleny stejně jako názvy polí v dokumentech. Tam, kde to nebylo vhodné, byla použita anotace `@Field` pro manuální mapování jmen. Typy v BSON dokumentech lze mapovat na odpovídající typy v Javě, tedy primitivní typy na Java ekvivalenty, pole na kolekci `List<>` či `Set<>` a vložené objekty na jinou třídu či kolekci `Map<>`. Pro práci s atributem, který je v MongoDB realizován referencí na jiný dokument (viz 2.3.2), byla použita anotace `@DocumentReference`, která při načítaní dokumentu z databáze načte i referencovaný dokument a při ukládání objektu do databáze uloží jeho ID. [72]

3.3.2 Přístup k datům

Pro přístup k datům v MongoDB bylo využito výše popsané rozhraní repository modulu Spring Data MongoDB. Pro správu každého typu entity je vytvořeno rozhraní rozšiřující rozhraní `CrudRepository`[2] s parametry typu entity a typu ID. Tímto způsobem lze používat CRUD operace nad entitou bez další implementace. [77] Pro repository spravující herní prvky byla potřeba vytvořit vlastní vyhledávání podle jména a filtrování podle zdroje, včetně kombinací těchto parametrů. Do rozhraní lze přidat vlastní dotazovací metody, pro které framework vytvoří implementaci na základě názvu metody a parametrů (např. `findByLastnameIgnoreCase`) nebo lze požadovaný dotaz definovat ručně pomocí anotace `@Query` a dotazovacího jazyka MongoDB. [78]

Tento způsob ovšem nemá dobrou podporu pro nepovinné parametry (pro kombinaci filtrů bylo potřeba vytvořit více metod s různými kombinacemi parametrů), proto bylo ve finální verzi přistoupeno k implementaci vlastní obecné repository třídy pro herní prvky. Ta byla pojmenována `SourcableBaseRepository` a je využívána místo rozhraní `CrudRepository` pro správu herních prvků. Výsledné řešení je vidět na ukázce 3.3. Třída je definována jako abstraktní a generická s využitím tzv. upper bounded type, tedy generickým typem musí být třída rozšiřující `SourceableEntity` [79]. Pro vykonání dotazů využívá `MongoTemplate`, základní třídu pro práci s MongoDB ve Spring Data, která umožňuje vytvářet dotazy Java kódem pomocí objektů `Query` a `Criteria`. [70, 80]

3.3.3 Business logika a DTO

Pro implementaci business logiky byly vytvořeny třídy služeb, které obsahují metody pro zpracování požadavků z API včetně ošetření vstupních dat a vyvolání výjimek v případě chyb. Služby jsou anotovány jako `@Service` a využívají repository pro přístup k datům.

Pro reprezentaci dat předávaných mezi vrstvou business logiky a kontrolerů, včetně požadavků a odpovědí API, byly vytvořeny DTO třídy (Data Transfer Object). [81] Ty navíc obsahují validační anotace specifikace Jakarta Bean Validation jako např. `@NotNull` a `@Size`, které definují pravidla pro platnost dat. [82]

[2] nebo některé z odvozených rozhraní

Pro složitější celky, kterými jsou export postavy do PDF a převod mezi DTO týkající se postav, byly vytvořeny vlastní třídy anotované jako `@Component`. Ty jsou pak využívány v metodách služeb pro zpracování těchto částí.

Jak již bylo krátce zmíněno v sekci 2.1.3.2, pro programové zpracování PDF souborů i s AcroForms je patrně nejrobustnější volně dostupná knihovna Apache PDFBox, proto bude v aplikaci použita. Metody knihovny umožňují načíst PDF soubor s AcroForms, nastavit hodnoty do polí a uložit vyplněnou kopii souboru. [83] PDFBox umožňuje načíst pole z PDF pomocí názvů polí (v knihovně označených jako `fullyQualifiedName`) [84], jejichž schéma bylo vytvořeno v rámci návrhu v sekci 2.4.

3.3.4 Kontrolery

Pro zpracování HTTP požadavků byly vytvořeny třídy kontrolerů, jejichž metody se váží na jednotlivé koncové body API. Jejich implementace odpovídá návrhu v sekci 2.2 a využívají modul Spring Web MVC popsaný v sekci 3.2. Pro vstup i výstup z těchto metod byly použity DTO třídy, jak bylo popsáno v předchozí sekci. Na závěr byly přidány dokumentační komentáře tak, aby se zobrazovaly v nástroji Swagger UI pro dokumentaci API.

V případě Spring Web MVC je kontroler reprezentován třídou označenou anotací `@Controller` nebo `@RestController`. Ta obsahuje metody s anotací `@RequestMapping` nebo jejími odvozeninami, které definují URL adresy, na které reaguje a jaké metody mají být volány. Dále jsou využívány anotace, jako např. `@PathVariable`, definující, jaké parametry mají být z požadavku získány. [49] Druhy parametrů požadavků byly vysvětleny v sekci 2.2.2.

3.3.5 Import dat ze souborů

Implementace funkcionality importu dat ze souborů v podobě, jak byla navrhnuta v rámci sekce 2.3.3, byla jednou ze specifik vývoje této aplikace. První zvažovanou možností bylo vytvořit koncové body přijímající data k importu, které by server pomocí business logiky aplikace zpracoval a uložil. Tento způsob měl však několik úskalí: jelikož bylo dohodnuto, že frontend nebude implementovat uživatelské rozhraní pro import, bylo by nutné ještě vytvořit další rozhraní, které by s koncovými body pracovalo. Tento postup by byl navíc vhodný spíše pro přijímání jednotlivých herních prvků, než pro hromadné zpracování souborů, které bylo navrženo dle požadavku F2.

Proto bylo rozhodnuto vytvořit import skript nezávislý na serveru, který vykoná potřebné operace na souborech a vloží data do databáze přímo. Během vývoje byly využívány pro vkládání dat do databáze řádkové programy `mongosh` a `mongoimport`, ovšem ani jeden z nich nebyl vhodný pro obsluhu importu v navrženém rozsahu.

`mongoimport` slouží k automatizovanému importu dat např. z JSON souboru, ale neumožňuje vykonávat transformace dat a logiku, kterou je potřeba provést před vložením dat do databáze. [85] `mongosh` je primárně interaktivní shell pro MongoDB, který umožňuje načíst uložený JavaScript soubor a vykonat ho, ovšem oficiálně nepodporuje přidání modulů pro načítání dalších souborů ze souborového systému. [86]

Implementace skriptu v požadovaném rozsahu zůstala v JavaScriptu, kód je ovšem spouštěn v běhovém prostředí Node.js. V něm je využito modulu `mongodb` pro připojení k databázi a modulů `fs` a `path` pro práci se soubory a cestami. Skript tak může vytvořit potřebné kolekce, načíst uživatelské JSON soubory, provést potřebné transformace dat (zejména vyhodnocení odkazů na jiné prvky) a vložit je do databáze. Během vývoje se vyskytlo několik výzev, kterým bylo potřeba čelit:

- **Udržování konzistence dat** – bylo potřeba rozhodnout, co se stane s aktuálními daty v databázi při spuštění importu. Import byl navržen tak, že uživatelé mají ve složkách všechny své datové soubory. Proto bylo rozhodnuto, že při importu se vždy vymažou všechna data her-

ních prvků z databáze a nahradí novými. Tímto způsobem se zamezí duplikaci dat a zůstane konzistence mezi daty v databázi a souborech.

- **Změny ID a zachování postav** – data postav nejsou zadány uživatelem v podobě souboru. Jelikož skript při každém spuštění vytvoří nové ID pro každý prvek, bylo potřeba zachovat správnost referencí v dokumentech postav. Bylo rozhodnuto, že skript nejprve zazálohuje data postav a po importu je znovu vloží s novými ID referencovaných prvků. Postavy tak používají podobný formát odkazů jako ostatní soubory s prvky, ale skript vytvoří soubor s postavami automaticky. Tento soubor zároveň uživatelům umožňuje zálohovat a sdílet postavy mezi instancemi aplikace.

- **Pořadí vložení prvků** – bylo potřeba zvolit pořadí, ve kterém budou prvky vloženy do databáze. Musí být zajištěno, že při vyhodnocování odkazů jsou již odkazované prvky v databázi. Bylo vytvořeno pevné pořadí vložení typů prvků na základě diagramu závislostí (viz 2.1).

Tímto způsobem byl vytvořen skript, který umožňuje importovat data z uživatelských souborů do databáze MongoDB a následně je využívat v aplikaci. Při běhu skriptu je zobrazena informace o průběhu importu a případných chybách. Uživatelská dokumentace k importu je k dispozici v souboru `README.md` v repozitáři projektu.

3.4 Kontejnerizace

Pro splnění požadavku N4 na kontejnerizaci aplikace byly připraveny tři kontejnery pro jednotlivé části backend aplikace. Pro MongoDB byl použit přímo oficiální obraz `mongo`, jelikož při vytváření instance databáze nebylo potřeba žádné speciální konfigurace. Pro Spring Boot server a import skript byly vytvořeny Dockerfile soubory pojmenované `Dockerfile_app` a `Dockerfile_script`. Kontejner pro Spring Boot využívá obraz s vývojovým prostředím Javy 17, kontejner pro import skript pak obraz s prostředím Node.js. Oba specifikují další kroky k sestavení kontejneru, které jsou náležitě zdokumentovány v komentářích v Dockerfile souborech.

Následně byl vytvořen soubor `docker-compose.yml`, který obsahuje konfiguraci pro spuštění a propojení všech kontejnerů. Nejprve spustí na obvyklém portu 27017 MongoDB databázi a přiřadí ji tzv. volume za účelem zachování dat mezi běhy. Poté spustí skript pro import dat a backend server (na obvyklém portu 8080), které jsou závislé na připojení k MongoDB. Informace jsou jim předány pomocí proměnných prostředí nastavených v souboru `.env`. Po vložení dat se kontejner s importem ukončí a nespotřebovává tak zbytečně zdroje. Celé řešení je tak možné spustit na pozadí příkazem `docker compose up -d` v kořenové složce repozitáře a zastavit příkazem `docker compose down`.

3.5 Výsledek implementace

V implementační části byla vytvořena kompletní backend část aplikace a samostatný skript pro import herních dat ze souborů. Výsledek splňuje všechny požadavky z analýzy, které byly označeny jako Must have nebo Should have, a je připraven k použití. Repozitář projektu se zdrojovými kódy, konfiguračními soubory, dokumentací a vzorovými daty pro import je dostupný na přiloženém médiu a na veřejném úložišti GitHub[3] pod open-source licencí MIT. Soubor `README.md` obsahuje uživatelskou dokumentaci k spuštění aplikace a importu dat. Dokumentace API je dostupná na URL adrese `http://localhost:8080/swagger-ui.html` po spuštění backend serveru.

[3] `https://github.com/ph0enix56/anteater`

```java
@Value
@Builder(toBuilder = true)
public class Character {

    @Id
    private String id;

    private String characterName;
    private List<Source> sources;

    @Field("classId")
    @DocumentReference
    private DndClass dndClass;

    private Map<Ability, AbilityInput> abilities;
    private Set<Skill> skills;
    private List<Proficiency<Tool>> tools;
    private Armor armor;

    // ... more fields of same types omitted

    public Armor getArmor() {
        if (armor != null) return armor;
        if (dndClass != null && dndClass.getDefaultArmor() != null)
            return dndClass.getDefaultArmor();
        return Constants.NO_ARMOR_DEFAULT;
    }
}
```

Výpis kódu 3.1 Model postavy

```java
@Getter
@ToString
@EqualsAndHashCode
@AllArgsConstructor
public abstract class SourceableEntity {
    @Id
    private String id;
    private String name;
    private Source source;
}

@Value
@EqualsAndHashCode(callSuper = true)
public class Tool extends SourceableEntity {
    private ToolType type;
    public Tool(String id, String name, Source source, ToolType type) {
        super(id, name, source);
        this.type = type;
    }
}
```

Výpis kódu 3.2 Model obsahu se zdrojem

```
public abstract class SourcableBaseRepository<T extends SourceableEntity> {
    private final MongoTemplate mongoTemplate;
    @NonNull private final Class<T> entityClass;

    public SourcableBaseRepository(MongoTemplate mongoTemplate,
            Class<T> entityClass) {
        this.mongoTemplate = mongoTemplate;
        this.entityClass = entityClass;
    }

    public Page<T> search(String name, List<String> sourceIds,
            @NonNull Pageable pageable) {
        Query query = new Query();
        if (name != null) query.addCriteria(Criteria.where("name")
            .regex(name, "i"));
        if (sourceIds != null && !sourceIds.isEmpty()) {
            query.addCriteria(Criteria.where("source.id").in(sourceIds));
        }
        long total = mongoTemplate.count(query, entityClass);
        query.with(pageable);

        List<T> content = mongoTemplate.find(query, entityClass);
        return new PageImpl<>(content, pageable, total);
    }

    // ... other methods omitted
}
```

Výpis kódu 3.3 Obecné repository pro herní prvky

Kapitola 4
Testování

V této kapitole bude popsán proces testování implementované aplikace. Testování bylo důležité provádět již během vývoje, jelikož na funkcionalitu backend serveru navazovala frontendová část. Pro hladký průběh integrace bylo nutné, aby byly průběžně hledány a odstraňovány chyby a frontend nebyl zdržován ve vývoji.

4.1 Testovací strategie

Pro otestování všech aspektů aplikace byla zvolena kombinace několika druhů testů tak, aby efektivně ověřily danou funkčnost a bylo snadné je provádět. Kde to bylo možné, byly využity automatizované testy. Celkově byly použity následující typy testů pro tyto části aplikace:

- Jednotkové testy – pro testování správnosti metod a tříd zajišťujících výpočetní logiku.
- Integrační testy API – pro ověření funkčnosti celého API aplikace proti návrhu.
- Akceptační testy – pro ověření použitelnosti API pro vývoj frontendu.
- Manuální testy PDF výstupů – pro ověření správnosti exportu postavy do PDF.

4.2 Jednotkové testy

Jednotkové testy testují malé části kódu, například funkce nebo metody. Zvyšují kvalitu kódu a jsou běžnou součástí vývoje software. Při každé změně určité části kódu je dobré spustit jednotkové testy, aby se ověřilo, zda změna neovlivnila ostatní části kódu. [87]

Pro psaní jednotkových testů byl použit testovací framework JUnit, který je součástí konfigurace projektu pomocí Spring Initializr. Jednotkovými testy jsou pokryty třídy obsahující vlastní psanou logiku, která se z velké části stává z výpočtů různých atributů postavy (viz příklad 4.1). Kontrola tříd služeb a přístupu k datům bude řešena v rámci integračních a akceptačních testů, jelikož tyto třídy jsou úzce spjaty s databází a pro jejich samostatné otestování by bylo potřeba simulovat velké množství dat.

Při testování jednotlivých metod tříd byla využita metoda označená anotací `@Before` pro instancování testovacích dat. Data byla následně využita v jednotlivých testech. Pokud test potřeboval modifikovat data, byla vytvořena jejich kopie. Poté je zavolána metoda k otestování a následně je pomocí `assert` funkcí frameworku kontrolováno, zda výsledek odpovídá očekávanému výstupu.

```
@Test
public void getArmorByClassTest() {
    var testCharacter = mockCharacter.toBuilder().armor(null)
        .dndClass(dndClass).build();
    Armor returnedArmor = testCharacter.getArmor();
    assertEquals(returnedArmor, armorByClass);
}
```

Výpis kódu 4.1 Test na ověření správného doplnění brnění ze třídy

4.3 Integrační testy API

API testy umožňují zjistit, zda API odpovídá specifikacím a zda vrací správné odpovědi. Existuje několik typů API testů, mezi které se řadí následující [88]:

- Jednotkové API testy – testují jednotlivé koncové body, zda vrací správné odpovědi/status kódy.
- End-to-end API testy – testují sekvenci koncových bodů, které na sebe navazují.

Během vývoje byly jednotlivé koncové body testovány manuálně pomocí nástroje Insomnia. Manuální testování pomohlo při ladění API během vývoje, avšak bylo časově náročné. Po větších zásazích v implementaci aplikace bylo potřeba mít jistotu, že API stále funguje správně. Proto byl vybrán nástroj k automatizaci testování API.

Jako testovací nástroj jsem zvolil Postman, protože jsem s ním již pracoval v rámci předmětu BI-TJV, tudíž jsem s ním měl zkušenosti, což psaní testů podstatně urychlilo. Testy se v nástroji Postman píší v jazyce JavaScript a jejich psaní usnadňuje zabudovaná knihovna Chai.js. [89]

V Postman byla vytvořena kolekce testů, která testuje jednotlivé koncové body. Pro každý koncový bod byl vytvořen jednotkový test, který ověřuje, zda koncový bod vrací správný status kód při dodání validních dat. Následně byly přidány i testy, které ověřují, zda API korektně odpovídá při zavolání koncového bodu s nevalidními daty. Tím byly otestovány třídy služeb aplikace.

Zároveň byla kolekce koncipována tak, aby simulovala uživatelské chování a prováděla end-to-end průchod aplikací. V rámci běhu kolekce je nasimulován uživatelský tok v aplikaci. Nejprve je vytvořena nová postava a jsou získány všechny postavy (kontrola, zda se nově vytvořená postava zobrazí v seznamu). Postavě jsou dále např. přidány předměty a je změněna třída postavy tak, aby se změnil přístup ke kouzlení. U toho je testována platnost operací. Nová postava je v testu tvořena dynamicky, protože identifikátory jednotlivých tříd, ras a zázemí se mění při každém importu dat.

Celá kolekce testů se nachází v repozitáři projektu ve složce `tests/api`. Je uložena v JSON formátu v souboru určenému pro import zpět do nástroje Postman.

4.4 Akceptační testy

Akceptační testy jsou testy, které ověřují, zda aplikace splňuje požadavky uživatelů. [90] V rámci backend části to znamenalo ověřit, zda je API vhodné pro použití frontend částí aplikace. Tato část testování byla provedena manuálně v rámci spolupráce s Žanetou během napojování frontendu na backend. Byla ověřena kompatibilita návrhů obou částí a srozumitelnost dokumentace API. Při integraci byla následně ověřena end-to-end funkčnost aplikace jako celku.

Výsledkem bylo provedení drobných úprav v API tak, aby bylo jednodušší s ním pracovat. Zároveň bylo nahlášeno několik defektů, které byly následně opraveny. Bylo tak zajištěno, že API je možné využívat frontendem a celá aplikace je pro koncové uživatele funkční.

4.5 Manuální testy PDF výstupu

Jelikož se projevilo jako složité otestovat správnost generování PDF deníků, byla tato funkcionalita testována zvlášť. K tomu byli přizváni přímo hráči D&D z dotázaných skupin. Ti na podporovaných prohlížečích vyzkoušeli použití vyexportovaného PDF deníku. Bylo potvrzeno, že deník generovaný aplikací je vhodný ke hře.

Kapitola 5
Budoucí rozvoj

Tato kapitola shrne možnosti, jakými lze do budoucna aplikaci rozšířit. Jedná se o podněty, které vyplynuly během analýzy nebo implementace aplikace, ale z důvodu nízké priority nebo odklonu od aktuálních cílů práce nebyly implementovány.

5.1 Rozšíření o další funkcionalitu

Do budoucna by bylo možné dodat i funkční požadavky, které byly označeny jako Could have a Will not have z důvodu časové náročnosti a nižšího přínosu v prvotní verzi aplikace. Jedná se zejména o požadavek F10 na plnohodnotný inventář postavy (a potenciálně i F11 na kontejnery, pokud by o něj byl dodatečný zájem). Na implementaci těchto požadavků by bylo potřeba upravit datový model tak, aby byl integrován s již existujícími zbraněmi a zbrojí. Dalším tímto požadavkem je F13 na podporu „multiclassing“. Z uživatelského testování provedeného Žanetou v její práci také vyplynuly další možné funkcionality včetně těch, které by vyžadovaly změny v backendu. Ty ale byly pro cíle práce vyhodnoceny jako nedůležité.

5.2 Správa obsahu přes frontend

Jak již bylo zmíněno v sekcích 2.3.3 a 3.3.5, v současné době byl z důvodu vytíženosti frontendového projektu a preferencí hráčů upřednostněn hromadný import herního obsahu do aplikace ze souborů pomocí skriptu. Do budoucna by bylo možné vytvořit uživatelské rozhraní pro správu obsahu umožňující přidávat, upravovat a mazat jednotlivé prvky herního obsahu. To by přirozeně vyžadovalo i rozšíření API backendu. Oba tyto způsoby by mohly být využívány zároveň, jelikož skript je vhodný pro hromadné změny a sdílení datových souborů mezi uživateli, zatímco formulářové rozhraní by bylo lepší pro drobné úpravy a rychlé přidání jednotlivých prvků.

5.3 Autentizace a veřejný provoz

V rámci požadavku N4 byla diskutována možnost autentizace uživatelů a přístupu pouze k vlastním postavám. Z hlediska backendu by pak byla nutná implementace autentizačního mechanismu a následného zabezpečení koncových bodů API tak, aby neumožňovaly manipulaci s cizími daty. Paralelně s tím by bylo potřeba vytvořit odpovídající rozhraní pro frontend. Poté by bylo možné přesunout aplikaci na veřejně dostupný server, kde by mohla být využívána širším okruhem hráčů. Otázkou zůstává, jak nahlížet na nahraný obsah, jelikož veřejný obsah nesmí porušovat

autorská práva. Tím pádem by již nebylo možné mít v databázi nahrané oficiální herní prvky, jako tomu teď může být díky soukromému nasazení aplikace.

Závěr

Cílem této bakalářské práce bylo navrhnout a vytvořit backendovou část webové aplikace, která umožní hráčům hry Dungeons & Dragons vytvářet a spravovat postavy využívající vlastní definovaný herní obsah. V rámci práce byly analyzovány pravidla hry a jiné online nástroje řešící podobný problém. Zároveň byly zhlédnuty existující deníky postav hráčů několika herních skupin a hráči byli dotázáni na jejich požadavky na takový nástroj. Na základě těchto informací byly stanoveny požadavky na aplikaci, byl proveden návrh aplikace se zaměřením na architekturu a datový model. Zvláštní pozornost byla věnována také návrhu funkcionality pro import herního obsahu do aplikace a export vytvořených postav do PDF souboru.

Aplikace byla poté implementována a otestována. Ze stanovených požadavků byly splněny zejména ty s vyšším hodnocením důležitosti, jak bylo určeno dle zadání práce a informací od hráčů. Poslední část práce se věnuje možným směrům dalšího rozvoje aplikace, které zahrnují dodání méně prioritní funkcionality nebo rozšíření způsobů vkládání herního obsahu.

Výsledkem je funkční backend server vytvořený v jazyce Java pomocí frameworku Spring Boot využívající databázi MongoDB pro perzistenci dat. Server využívá flexibilní datový model pro jednotlivé herní prvky, jako např. třídy, rasy, zbraně a kouzla, které načítá z databáze. Poskytuje REST API pro tvorbu a správu postav, které využívá autorka Žaneta Trošková ve své bakalářské práci, v rámci které byla vytvořena frontendová část aplikace. Prvky je možné importovat do aplikace z JSON souborů hromadně pomocí skriptu. Aplikace byla otestována několika způsoby, včetně jednotkových testů v Javě a API testů pomocí nástroje Postman. Celé řešení bylo nasazené do Docker kontejnerů pro snadné spuštění aplikace. Je zdokumentováno, jak spustit aplikaci i jak využít API rozhraní. Výsledek je dostupný na veřejném repozitáři na platformě GitHub pod open-source licencí MIT. Všechny stanovené cíle práce tak byly splněny.

Příloha A

Jména polí PDF šablony deníku postavy

V této příloze se nachází seznam identifikačních jmen AcroForm polí, které bude aplikace využívat pro vyplnění informací. Při vytváření jiné šablony pro export postav do PDF je tedy potřeba, aby tyto pole existovala a odpovídal jejich typ. Kde není zmíněno jinak, pole by měla být textová. Poslední pole uvedená jako „seznam“ musí být víceřádková, ostatní nemusí podporovat odřádkování a mohou tak využívat různé zarovnání textu.

Zkratky atributů jsou malými písmeny a obsahují první 3 písmena původního názvu, tedy např. *str* a *int*.

Zkratky Skills jsou malými písmeny a obsahují první 4 písmena původního názvu, tedy např. *athl* a *pers*.

`characterName`	jméno postavy
`playerName`	jméno hráče
`classAndLevel`	název herní třídy a úroveň
`race`	název rasy
`abi``Score`	hodnota skóre atributu, kde *`abi`* je zkratka atributu
`abi``Mod`	hodnota modifikátoru atributu, kde *`abi`* je zkratka atributu
`skil``Prof`	check box Proficiency v Skills, *`skil`* je zkratka Skillu
`skil``Mod`	hodnota modifikátoru v Skills, *`skil`* je zkratka Skillu
`abi``SaveProf`	check box Proficiency v Saving Throws, *`abi`* je zkratka atributu
`abi``SaveMod`	hodnota modifikátoru v Saving Throws, *`abi`* je zkratka atributu
`proficiencyBonus`	hodnota Proficiency Bonusu
`armorClass`	hodnota AC
`initiative`	hodnota iniciativy
`speed`	hodnota rychlosti
`hpMax`	maximální počet bodů života
`hdMax`	maximální počet kostek bodů života
`atk`*`N`*`name`	název útoku, *`N`* je číslo útoku 0–3
`atk`*`N`*`bonus`	hodnota modifikátoru útoku, *`N`* je číslo útoku 0–3
`atk`*`N`*`damage`	poškození útoku, kde *`N`* je číslo útoku 0–3
`armor`	název nošené zbroje a informace, zda zhoršuje plížení
`proficiencies`	seznam zbraní, zbroje, nástrojů a jazyků ovládaných postavou
`features`	seznam zvláštních vlastností

Bibliografie

1. MEARLS, Mike; CRAWFORD, Jeremy. *Dungeons & Dragons: Player's Handbook*. 5th. Renton, WA: Wizards of the Coast, 2014. ISBN 9780786965601.
2. *Mordenkainen Presents: Monsters of the Multiverse*. Renton, WA: Wizards of the Coast, 2022. ISBN 9780786967872.
3. WIZARDS OF THE COAST. *Homebrew* [online]. 2024. [cit. 2024-04-15]. Dostupné z: `https://www.dndbeyond.com/`.
4. WIZARDS OF THE COAST. *Homebrew* [online]. 2024. [cit. 2024-04-15]. Dostupné z: `https://www.dndbeyond.com/homebrew`.
5. WIZARDS OF THE COAST. *Rules & Guidelines* [online]. 2024. [cit. 2024-04-15]. Dostupné z: `https://www.dndbeyond.com/homebrew-rules-guidelines#PublicHomebrewGuidelines`.
6. WIZARDS OF THE COAST. *Explore D&D Beyond* [online]. 2024. [cit. 2024-04-15]. Dostupné z: `https://www.dndbeyond.com/store/subscribe`.
7. DUNGEON MASTER'S VAULT. *Dungeon Master's Vault* [online]. 2024. [cit. 2024-04-18]. Dostupné z: `https://www.dungeonmastersvault.com/`.
8. CODEGLAZE. *orcpub* [online]. 2024. [cit. 2024-04-18]. Dostupné z: `https://github.com/Orcpub/orcpub/blob/develop/README.md`.
9. HICKEY, Rich. *edn* [online]. 2014. [cit. 2024-04-18]. Dostupné z: `https://github.com/edn-format/edn/blob/master/README.md`.
10. DRIESSEN, Bas. *Aurora Development Postponed Indefinitely* [online]. 2020. [cit. 2024-04-18]. Dostupné z: `https://aurorabuilder.com/posts/1360/aurora-project-postponed-indefinitely/`.
11. DRIESSEN, Bas. *Anatomy of the Element* [online]. 2020. [cit. 2024-04-18]. Dostupné z: `https://aurorabuilder.com/documentation/anatomy-of-the-element/`.
12. IEEE COMPUTER SOCIETY. *Software Requirements Specifications* [online]. [cit. 2024-04-08]. Dostupné z: `https://www.computer.org/resources/software-requirements-specifications`.
13. ALTEXSOFT. *Functional and Nonfunctional Requirements: Specification and Types* [online]. 2023. [cit. 2024-04-08]. Dostupné z: `https://www.altexsoft.com/blog/functional-and-non-functional-requirements-specification-and-types/`.
14. PRODUCTPLAN. *MoSCoW Prioritization* [online]. 2024. [cit. 2024-04-08]. Dostupné z: `https://www.productplan.com/glossary/moscow-prioritization/`.

15. ALTEXSOFT. *Comparing Database Management Systems* [online]. 2023. [cit. 2024-04-09]. Dostupné z: https://www.altexsoft.com/blog/comparing-database-management-systems-mysql-postgresql-mssql-server-mongodb-elasticsearch-and-others/.
16. CODD, Edgar Frank. *A Relational Model of Data for Large Shared Data Banks* [online]. 1970. [cit. 2024-04-09]. Dostupné z: https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf.
17. DB-ENGINES. *DB-Engines Ranking* [online]. 2024. [cit. 2024-04-09]. Dostupné z: https://db-engines.com/en/ranking.
18. IBM. *What is a NoSQL database?* [Online]. [cit. 2024-04-10]. Dostupné z: https://www.ibm.com/topics/nosql-databases.
19. AMAZON WEB SERVICES. *What is NoSQL?* [Online]. 2024. [cit. 2024-04-10]. Dostupné z: https://aws.amazon.com/nosql/.
20. MONGODB. *NoSQL vs. SQL Databases* [online]. 2024. [cit. 2024-04-10]. Dostupné z: https://www.mongodb.com/nosql-explained/nosql-vs-sql.
21. MONGODB. *What are ACID Properties in Database Management Systems?* [Online]. 2024. [cit. 2024-04-10]. Dostupné z: https://www.mongodb.com/basics/acid-transactions.
22. MONGODB. *JSON and BSON* [online]. 2024. [cit. 2024-04-10]. Dostupné z: https://www.mongodb.com/json-and-bson.
23. AMAZON WEB SERVICES. *What is an API (Application Programming Interface)?* [Online]. 2024. [cit. 2024-04-04]. Dostupné z: https://aws.amazon.com/what-is/api/.
24. FIELDING, Roy Thomas. *Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST)* [online]. 2000. [cit. 2024-04-04]. Dostupné z: https://ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.
25. RED HAT. *What is a REST API?* [Online]. 2020. [cit. 2024-04-04]. Dostupné z: https://www.redhat.com/en/topics/api/what-is-a-rest-api.
26. AMAZON WEB SERVICES. *What is a RESTful API?* [Online]. 2024. [cit. 2024-04-04]. Dostupné z: https://aws.amazon.com/what-is/restful-api/.
27. BYRON, Lee. *GraphQL: A data query language* [online]. 2015. [cit. 2024-04-04]. Dostupné z: https://engineering.fb.com/2015/09/14/core-infra/graphql-a-data-query-language/.
28. THE GRAPHQL FOUNDATION. *Serving over HTTP* [online]. 2024. [cit. 2024-04-04]. Dostupné z: https://graphql.org/learn/serving-over-http/.
29. THE GRAPHQL FOUNDATION. *Schemas and Types* [online]. 2024. [cit. 2024-04-04]. Dostupné z: https://graphql.org/learn/schema/.
30. THE GRAPHQL FOUNDATION. *Queries and Mutations* [online]. 2024. [cit. 2024-04-04]. Dostupné z: https://graphql.org/learn/queries/.
31. THE GRAPHQL FOUNDATION. *Introspection* [online]. 2024. [cit. 2024-04-04]. Dostupné z: https://graphql.org/learn/introspection/.
32. RED HAT. *What is GraphQL?* [Online]. 2019. [cit. 2024-04-09]. Dostupné z: https://www.redhat.com/en/topics/api/what-is-graphql.
33. AMAZON WEB SERVICES. *What's the Difference Between GraphQL and REST?* [Online]. 2024. [cit. 2024-04-09]. Dostupné z: https://aws.amazon.com/compare/the-difference-between-graphql-and-rest/.
34. STACK OVERFLOW. *Stack Overflow 2023 Developer Survey: Web frameworks and technologies* [online]. 2023. [cit. 2024-04-12]. Dostupné z: https://survey.stackoverflow.co/2023/#most-popular-technologies-webframe-prof.

35. STRONGLOOP. *Express* [online]. 2017. [cit. 2024-04-12]. Dostupné z: https://expressjs.com/.
36. STRONGLOOP. *4.x API* [online]. 2017. [cit. 2024-04-12]. Dostupné z: https://expressjs.com/en/4x/api.html.
37. VERCEL. *Rendering* [online]. 2024. [cit. 2024-04-12]. Dostupné z: https://nextjs.org/docs/app/building-your-application/rendering.
38. APACHE. *Apache PDFBox® - A Java PDF Library* [online]. 2024. [cit. 2024-04-12]. Dostupné z: https://pdfbox.apache.org/.
39. *Features of PDFsharp Library* [online]. 2024. [cit. 2024-04-12]. Dostupné z: https://docs.pdfsharp.net/PDFsharp/Overview/Features.html.
40. TECHEMPOWER. *Web Framework Benchmarks: Round 22: Composite Scores* [online]. 2023. [cit. 2024-04-12]. Dostupné z: https://www.techempower.com/benchmarks/#hw=ph&test=composite&section=data-r22&f=zhavon-zik073-zik0zj-zik0zj-zik0zj-zik0zj-hra0hr-zik0zj-zik0zj-zhxjwf-zik0zj-zik0zj-zik0zj-zijocf-1ekf.
41. ORACLE. *Generic Types* [online]. 2022. [cit. 2024-04-13]. Dostupné z: https://docs.oracle.com/javase/tutorial/java/generics/types.html.
42. MICROSOFT. *Generic classes and methods* [online]. 2024. [cit. 2024-04-13]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/types/generics.
43. ORACLE. *Polymorphism* [online]. 2022. [cit. 2024-04-13]. Dostupné z: https://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html.
44. MICROSOFT. *Polymorphism* [online]. 2023. [cit. 2024-04-13]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/object-oriented/polymorphism.
45. BROADCOM. *Building an Application with Spring Boot* [online]. 2024. [cit. 2024-04-13]. Dostupné z: https://spring.io/guides/gs/spring-boot.
46. MOZILLA.ORG CONTRIBUTORS. *HTTP request methods* [online]. 2023. [cit. 2024-04-09]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods.
47. MOZILLA.ORG CONTRIBUTORS. *Safe* [online]. 2023. [cit. 2024-04-09]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/Safe/HTTP.
48. MOZILLA.ORG CONTRIBUTORS. *Idempotent* [online]. 2023. [cit. 2024-04-09]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/Idempotent.
49. BROADCOM. *Mapping Requests* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://docs.spring.io/spring-framework/reference/web/webmvc/mvc-controller/ann-requestmapping.html.
50. BROADCOM. *RequestParam* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://docs.spring.io/spring-framework/reference/web/webmvc/mvc-controller/ann-methods/requestparam.html.
51. BROADCOM. *RequestBody* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://docs.spring.io/spring-framework/reference/web/webmvc/mvc-controller/ann-methods/requestbody.html.
52. KARLSSON, Joe. *MongoDB Schema Design Best Practices* [online]. 2022. [cit. 2024-05-08]. Dostupné z: https://www.mongodb.com/developer/products/mongodb/mongodb-schema-design-best-practices/.
53. MONGODB. *$lookup (aggregation)* [online]. 2024. [cit. 2024-05-08]. Dostupné z: https://www.mongodb.com/docs/manual/reference/operator/aggregation/lookup/.
54. DATALOGICS. *How to Work With AcroForms & XFA* [online]. 2023. [cit. 2024-04-25]. Dostupné z: https://www.datalogics.com/how-to-work-with-acroforms-xfa.

55. LIBREOFFICE. *What is LibreOffice?* [Online]. [cit. 2024-04-25]. Dostupné z: https://www.libreoffice.org/discover/libreoffice/.
56. LIBREOFFICE. *LibreOffice 24.2 Help* [online]. [cit. 2024-04-25]. Dostupné z: https://help.libreoffice.org/24.2/en-US/text/shared/01/ref_pdf_export_general.html.
57. WIZARDS OF THE COAST. *Character Sheets* [online]. 2024. [cit. 2024-04-25]. Dostupné z: https://dnd.wizards.com/resources/character-sheets.
58. BROADCOM. *Spring Initializr* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://start.spring.io/#!type=gradle-project&language=java&platformVersion=3.2.5&packaging=jar&jvmVersion=17&groupId=com.example&artifactId=demo&name=demo&description=Demo%20project%20for%20Spring%20Boot&packageName=com.example.demo&dependencies=web,data-mongodb,devtools,lombok.
59. BROADCOM. *Spring Framework* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://spring.io/projects/spring-framework.
60. BROADCOM. *Introduction to the Spring IoC Container and Beans* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://docs.spring.io/spring-framework/reference/core/beans/introduction.html.
61. SEEMANN, Mark. *Dependency Injection is Loose Coupling* [online]. 2010. [cit. 2024-04-30]. Dostupné z: https://blog.ploeh.dk/2010/04/07/DependencyInjectionisLooseCoupling/.
62. BROADCOM. *Configuration Metadata* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://docs.spring.io/spring-framework/reference/core/beans/basics.html#beans-factory-metadata.
63. BROADCOM. *Spring Boot* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://spring.io/projects/spring-boot.
64. KALKAR, Mehmet Baris. *Opinionated Frameworks* [online]. 2022. [cit. 2024-04-30]. Dostupné z: https://baris.io/blog/opinionated-frameworks.
65. BROADCOM. *Common Application Properties* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html.
66. BROADCOM. *Building a RESTful Web Service* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://spring.io/guides/gs/rest-service.
67. BROADCOM. *Spring Web MVC* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://docs.spring.io/spring-framework/reference/web/webmvc.html#mvc.
68. CHRIS, Kolade. *MVC in Computer Science – The MVC Model* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://www.freecodecamp.org/news/what-does-mvc-mean-in-computer-science/.
69. BROADCOM. *Spring Data MongoDB* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://docs.spring.io/spring-data/data-mongodb/reference/index.html.
70. BROADCOM. *Template API* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://docs.spring.io/spring-data/data-mongodb/reference/mongodb/template-api.html.
71. BROADCOM. *Repositories: Core concepts* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://docs.spring.io/spring-data/data-mongodb/reference/repositories/core-concepts.html.
72. BROADCOM. *Object Mapping* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://docs.spring.io/spring-data/data-mongodb/reference/mongodb/mapping/mapping.html.

73. BROADCOM. *Spring Boot Reference Documentation: Developer Tools* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#using.devtools.
74. PROJECT LOMBOK AUTHORS. *Project Lombok* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://projectlombok.org/.
75. PROJECT LOMBOK AUTHORS. *Value* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://projectlombok.org/features/Value.
76. PROJECT LOMBOK AUTHORS. *Builder* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://projectlombok.org/features/Builder.
77. BROADCOM. *Defining Repository Interfaces* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://docs.spring.io/spring-data/mongodb/reference/repositories/definition.html.
78. BROADCOM. *Querying Documents* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://docs.spring.io/spring-data/mongodb/reference/repositories/query-methods-details.html.
79. ORACLE. *Bounded Type Parameters* [online]. 2022. [cit. 2024-04-30]. Dostupné z: https://docs.oracle.com/javase/tutorial/java/generics/bounded.html.
80. BROADCOM. *Querying Documents* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://docs.spring.io/spring-data/mongodb/reference/mongodb/template-query-operations.html.
81. BAELDUNG. *The DTO Pattern (Data Transfer Object)* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://www.baeldung.com/java-dto-pattern.
82. MORLING, Gunnar. *Jakarta Bean Validation specification: Built-in Constraint definitions* [online]. 2020. [cit. 2024-04-30]. Dostupné z: https://jakarta.ee/specifications/bean-validation/3.0/jakarta-bean-validation-spec-3.0.html#builtinconstraints.
83. APACHE SOFTWARE FOUNDATION. *Apache PDFBox® - A Java PDF Library* [online]. 2024. [cit. 2024-04-25]. Dostupné z: https://pdfbox.apache.org/.
84. APACHE SOFTWARE FOUNDATION. *LibreOffice 24.2 Help* [online]. 2018. [cit. 2024-04-25]. Dostupné z: https://pdfbox.apache.org/docs/2.0.13/javadocs/org/apache/pdfbox/pdmodel/interactive/form/PDAcroForm.html.
85. MONGODB. *mongoimport* [online]. 2024. [cit. 2024-05-09]. Dostupné z: https://www.mongodb.com/docs/database-tools/mongoimport/.
86. MONGODB. *Welcome to MongoDB Shell (mongosh)* [online]. 2024. [cit. 2024-05-09]. Dostupné z: https://www.mongodb.com/docs/mongodb-shell/.
87. AMAZON WEB SERVICES. *What is API testing?* [Online]. 2024. [cit. 2024-05-10]. Dostupné z: https://aws.amazon.com/what-is/unit-testing/.
88. POSTMAN. *What is API testing?* [Online]. 2024. [cit. 2024-05-06]. Dostupné z: https://www.postman.com/api-platform/api-testing/.
89. POSTMAN. *Write API test scripts in Postman* [online]. 2024. [cit. 2024-05-06]. Dostupné z: https://learning.postman.com/docs/writing-scripts/test-scripts/.
90. GILLIS, Alexander S. *acceptance testing* [online]. 2024. [cit. 2024-05-11]. Dostupné z: https://www.techtarget.com/searchsoftwarequality/definition/acceptance-test.

Obsah příloh

- `readme.txt` ..... stručný popis obsahu média
- `assets` ..... vytvořená šablona deníků postavy
  - `anteater_character_sheet.odt` ..... šablona deníku postavy ve formátu ODT
  - `anteater_character_sheet.pdf` ..... šablona deníku postavy ve formátu PDF
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF