Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Tvorba backendu pro online RPG hru

Matěj Němec

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Matěj Němec

Systémové inženýrství a informatika
Informatika

Název práce

Backend aplikace prohlížečové hry v ASP.NET Core (C#)

Název anglicky

Backend of a browser game application in ASP.NET Core

Cíle práce

Cílem práce je návrh a realizace backendu prohlížečové RPG hry v ASP.NET Core (C#). Dílčím cílem je návrh aplikace za využití jazyka UML, jak pro samotnou architekturu aplikace, tak i její databázi. Dále bude cílem práce hotovu aplikaci nasadit na server a provést testování s reálnými uživateli.

Metodika

Metodika teoretické části práce stojí na tradičním analytickém přístupu k získávání poznatků. Tyto poznatky budou čerpány z literatury, článků a dalších odborných zdrojů a po jejich analýze budou evaluovány a vzhledem k cílům práce vhodně shrnuty.

Získané teoretické poznatky budou využity při návrhu datového modelu a vývoji prototypu hry s prvky RPG. Tato hra bude realizována ve formě webové aplikaci využívající architektury MVC (model-view-controller) a frameworku ASP.NET Core. Při návrhu a vývoji hry bude využito standardních metod a postupů softwarového inženýrství. Poznatky z vývoje a testování budou popsány a zhodnoceny.

Doporučený rozsah práce

60-80 stránek

Klíčová slova

C#, ASP.NET Core, RPG, Webová aplikace

Doporučené zdroje informací

ALBAHARI, Joseph and Ben ALBAHARI, 2020. C# 8.0 in a Nutshell: The Definitive Reference. O'Reilly Media. ISBN 9781492051138.

CONNOLLY, Thomas M., Carolyn E. BEGG a Jennifer WIDOM. Database systems: a practical approach to design, implementation, and management. 5th ed. London: Addison-Wesley, c2010. ISBN 978-0-32-152306-8.

Předběžný termín obhajoby

2021/22 LS – PEF

Vedoucí práce

Ing. Petr Hanzlík, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 1. 11. 2021

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 23. 11. 2021

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 09. 11. 2022

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Tvorba backendu online RPG hry" jsem vypracoval(a) samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor(ka) uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 30.11.2022 _____

Poděkování

Rád bych touto cestou poděkoval vedoucímu práce Ing. Petru Hanzlíkovi Ph.D. za jeho ochotu a odbornou pomoc. Dále bych rád poděkoval Davidu Kohoutovi a Ing. Alžbětě Pokorné, za jejich odborné konzultace a ochotu při testování práce.

Tvorba backendu pro online RPG hru

Abstrakt

Cílem této diplomové práce je tvorba Backendové aplikace online prohlížečové RPG hry. Dále je cílem práce konečnou aplikaci nasadit na server a otestovat na reálných uživatelích. A to uživatelích z řad odborných vývojářů i běžných uživatelů, kteří by takovou hru mohli hrát. Pro tvorbu aplikace je zvolen framework ASP.NET Core a použitá architektura MVC, s primárně užitým jazykem C#. Jako metodika pro vývoj software byl zvolen postup vývoje prototyp. Při tvorbě byly užity metodiky principů SOLID a softwarového inženýrství. Pro formální dokumentaci aplikace byl využit jazyk UML. Pro závěrečné nasazení pak bylo využito cloudového uložiště ve službě Azure. Testování aplikace probíhalo v souladu se zvolenou metodikou průběžně při vývoji a pomáhalo zahlazovat průběžné nedostatky. Po dokončení vývoje, hlavní testování vývojáři odhalilo několik nedostatků, a to zejména z hlediska funkčnosti vytyčených funkčních požadavků. Tyto nedostatky byly úspěšně odstraněny. Uživatelské testování již žádné nedostatky neodhalilo, ale pomohlo nastínit další možný vývoj aplikace.

Klíčová slova: C#, ASP.NET Core, RPG, Webová aplikace

Designing a backend application for an online RPG game

Abstract

The goal of this thesis is to design and implement a backend application for an online browser RPG game. Furthermore, the goal of the thesis is a deployment of the application on a server and a testing with real users. Such users shall come from the ranks of both developers and potential players of such a game. For the programming work of the thesis ASP.NET Core 6 framework is being chosen with a use of MVC pattern. The main language used is the language of C#. Prototype development model is chosen as the main development approach for the thesis. Programming works were done by using methodology of SOLID principles and software engineering. The formal documentation was designed in UML language. For the final deployment the cloud service of Azure was chosen. The application has been tested during its development with use of the chosen methodology which helped to uncover some of its flaws. After the main programing works were done, real-life developers tested the app which brought couple of drawbacks unto the surface. These defects were regarding functional requirements and as such they were fixed and made no more. The user testing did not uncover any other shortcomings in the code, but it helped shape the future of the application's development.

Keywords: C#, ASP.NET Core, RPG, Web application

Obsah

1 Úvod

Videohry jsou v dnešní době nedílnou součástí informatiky a světa počítačů. Jejich vývojem a hraním jak volnočasovým, tak profesionálním se zabývá velké množství vývojářů i členů široké veřejnosti. Jedná se o odvětví, které pojme a vydělá miliardy dolarů ročně. Jakožto takové odvětví tak vyžaduje velkou časovou investici jak na poli vývojářů, tak i hráčů a všech dalších zúčastněných stran. Dále bývají hry velmi náročné na výpočetní kapacitu.

Ovšem každý potenciální hráč si právě takovou časovou investici vždy dovolit nemůže a ne vždy má po ruce dostatečně výkonný hardware, který by některé herní tituly podporoval. Proto vývojáři her dali vzniknout prohlížečovým hrám. I takové hry však pro svoji hratelnosti a chytlavost mohou stát hráče velké množství času. Proto vývoj prohlížečových her pokračoval a vznikly „Point and click“ online RPG hry. Tyto hry sice vyžadují denní účast, ale ta nebývá delší než řádově několik desítek, nebo i jednotek minut denně.

Všechny takové aplikace pak mají v základu minimálně dvě hlavní komponenty své architektury, které umožní jejich existenci. Frontend, kterým je uživatelské rozhraní a backend, který obstarává logický a funkční aspekt aplikace.

Vývoj backendu takové hry se právě zabírá tato práce. Hlavním cílem je navrhnout a naprogramovat backend pro online klikací RPG hru. K tomu však náleží i přípravné práce, které jsou součástí cílů této práce. Takové práce je tvorba analýzy jejíž součástí jsou diagramy a promyšlení všech prací na aplikaci. Dalším cílem práce je nasazení aplikace na existující server a otestování jak vývojáři, kteří vývoji rozumí a tudíž jsou schopni aplikaci lépe otestovat a pak i uživateli, kteří se vývojem nezaobírají, ale poskytnou pro vývoj každého softwaru důležitý uživatelský vhled.

Hlavním tématem hry je divoký západ. Výsledná aplikace však slouží jako podklad pro jakoukoliv aplikaci tohoto rázu a tudíž lze její tématiku z hlediska backendu upravit bez větší časové náročnosti.

Práce je rozdělena do dvou částí. První, teoretická část seznámí čtenáře se základy teoretických poznatků, nutných pro pochopení praktické částí. Čtenář bude seznámen s tématem Softwarového inženýrství, kde jsou zahrnuty některé metodiky užité při tvorbě práce. Dále s technologiemi, a jejich specifikacemi, užité při této práci. Mezi ty patří hlavně

vysvětlení náležitostí architektury MVC a technologie frameworku .ASP .NET Core a relační databáze s jazykem SQL. Nadále pak teoretické části práce náleží uvedení čtenáře do světa online RPG her a některých jejich základních mechanik.

V praktické části práce pak autor vysvětlí svůj postup při návrhu, vývoji, nasazení a testování aplikace. První bylo třeba provést analýzu s diagramy sloužícími coby základ logického rozložení aplikace. Dále samotný vývoj, který byl rozdělen do mnoha samostatných bloků a v poslední části pak testování aplikace.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem práce je návrh a realizace backendu prohlížečové RPG hry v ASP.NET Core (C#). Dílčím cílem je návrh aplikace za využití jazyka UML, jak pro samotnou architekturu aplikace, tak i její databázi. Dále bude cílem práce hotovu aplikaci nasadit na server a provést testování s reálnými uživateli.

2.2 Metodika

Metodika teoretické části práce stojí na tradičním analytickém přístupu k získávání poznatků. Tyto poznatky budou čerpány z literatury, článků a dalších odborných zdrojů a po jejich analýze budou evaluovány a vzhledem k cílům práce vhodně shrnuty.

Získané teoretické poznatky budou využity při návrhu datového modelu a vývoji prototypu hry s prvky RPG. Tato hra bude realizována ve formě webové aplikace využívající architektury MVC (model-view-controller) a frameworku ASP.NET Core. Při návrhu a vývoji hry bude využito standardních metod a postupů softwarového inženýrství. Zejména pak principů SOLID. Aplikace bude vyvíjena metodou prototyp. Poznatky z vývoje a testování budou popsány a zhodnoceny.

3 Teoretická východiska

Následující část podává čtenáři ucelené teoretické podklady potřebné pro vypracování praktické části práce.

3.1 Softwarové inženýrství

Kde informatika je teoretickou částí počítačové vědy, tam softwarové inženýrství pokrývá praktickou rovinu. Nejedná se o ustálený, či striktně zavedený pojem, avšak za všeobecně přijímaný náhled se považuje následující. Jedná se o disciplínu zabývající se vývojem, nasazením a údržbou softwaru. Disciplína tak leží v konjunkci informatiky, vývoje a řízení. Nejedná se pouze o samotný vývoj, který je v otázce, ale i o finanční a zákaznickou stránku věci.

Značná součást metodik a postupů jsou zaměřený na stranu cílového uživatele softwaru. Při jeho vývoji je tak mnohdy netřeba přemýšlet nad produktem jako nad programátorskou úlohou, byť korektních programátorských přístupů je třeba se držet, ale jako nad aplikací určenou pro uživatele. Od toho se pak odráží přístup k umísťování jednotlivých prvků, provázanost funkcionalit atd.

Mezi části Softwarového inženýrství náleží např. příprava podkladových diagramů, person čili vzorových uživatelů, užívání návrhových vzorů, dodržování metodik postupů vývoje, testování atd (Vondrák, 2002).

3.1.1 Analýza

Prvním krokem vývoje softwaru v dnešním pojetí je analýza. Navzdory faktu, že informační technologie jsou moderním trendem, říká se, že práce vývojáře začíná s tužkou a papírem. Před začátkem jakéhokoliv programování je třeba si práci i výsledný produkt důsledně promyslet. Připravit si jednotlivé komponenty a funkcionality.

Analýza pak zachází do důsledku a je třeba mít vymyšleno, jak konkrétně se například určitá funkce aplikace bude programovat. Důvodem k takto zevrubné přípravě je například předejití zjištění v půlce vývoje, že nějaká, již dávno naprogramovaná a samostatně funkční, komponenta aplikace nemůže fungovat zároveň s jinou, nebo její fungování omezuje. Takovýmto případům předchází i různé postupy vývoje.

Nadále nám analýza usnadňuje samotnou práci vývoje. Pakliže vývojář postupuje tak, že bez přípravy a předchozího promyšlení programuje jednotlivé části aplikace, může tak často psát redundantní či zcela zbytečný kód, nebo kód, který bude muset v budoucnu z velké části refaktorovat, čili přepisovat, nebo mazat pro jeho nadbytečnost.

Taková analýza pak začíná přípravou různých diagramů.

3.1.2 Diagramy

Diagramy jsou základním prvkem analýzy a vývoje jakéhokoliv softwaru. Jsou třeba pro vývojáře a někdy pro zákazníky. Práce na softwaru často probíhá v týmech, kde je třeba aby docházelo k plynulému a efektivnímu toku informací. Diagramy jsou pak velmi efektivním způsobem jak nejen položit praktický základ výsledné aplikace, ale také jak předat značné množství informací ve velmi komprimované podobě.

Diagramy jsou grafického charakteru a skládají z prvků, procesů a vazeb mezi nimi. Diagramy se dělí na analytické a návrhové. Ty první ukazují co konkrétně bude daný systém či software dělat a návrhové pak jak takové chování bude provedeno (Čermák; Martinů, 2018).

3.1.2.1 Jazyk UML

Pro tvorbu diagramů se v softwarovém inženýrství mimo jiné využívá jazyk UML. Zkratka znamená Unified Modeling Language. UML je soubor notací určených pro tvorbu diagramů a je uznáván jako mezinárodní standard (Čápka).

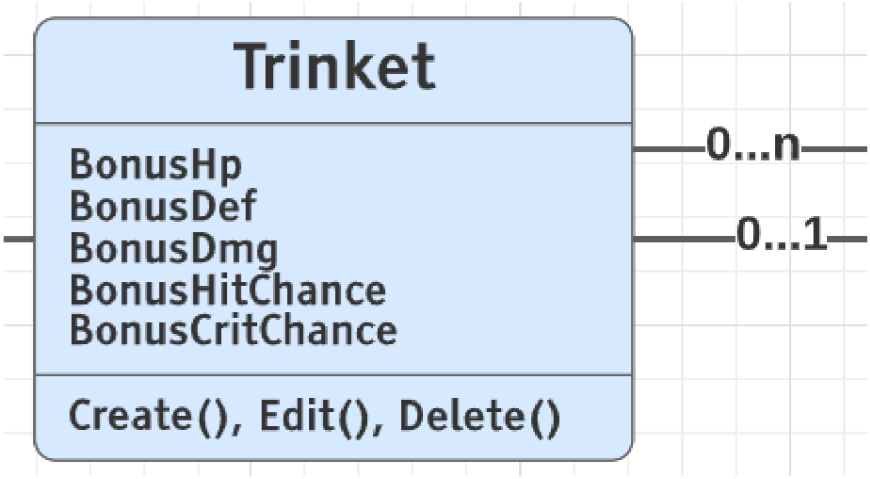
3.1.2.2 Diagram tříd

Diagram tříd je základním grafickým nástrojem pro analýzu objektově modelovaného softwaru. Tento diagram dává vyobrazit jednotlivé objekty výsledného programu. Tyto objekty zastupují takzvané modely, které budou v kódu tvořit třídy. Tématice objektového paradigmatu bude věnována kapitola později v této práci.

Jednotlivé modely mají v diagramu zaznamenán název, jejich atributy a metody. Dále jsou v diagramu tyto modely mezi sebou spojeny. Spojují je lomené čáry, které vždy ve svých počátečních a koncových bodech mají buď číselně, jinak, či za pomocí non-alfanumerických

symbolů, vyznačenou mohutnost. Mohutnost určuje kolik instancí může být instancí třídy jedné přiřazeno k jedné instanci třídy druhé.

Následující obrázek je příklad takového modelu. V horní části nese svůj název, v prostřední své atributy a ve spodní pak metody. Dále si lze všimnout v pravé části, kde je daný model spojen s jiným modelem vyznačené mohutnosti (Čermák; Martinů, 2018).



Obrázek 1 - Příklad modelu v diagramu tříd

Diagram tříd je často ve zjednodušené formě používán jako základní návrhový diagram pro daný software. V takové podobě se pak většinou nepíší metody, nebo všechny atributy ani mohutnosti. Nejedná se o v souladu s jazykem UML správně využitý diagram, avšak v praxi většinou ke svému prvotnímu účelu je dostačujícím.

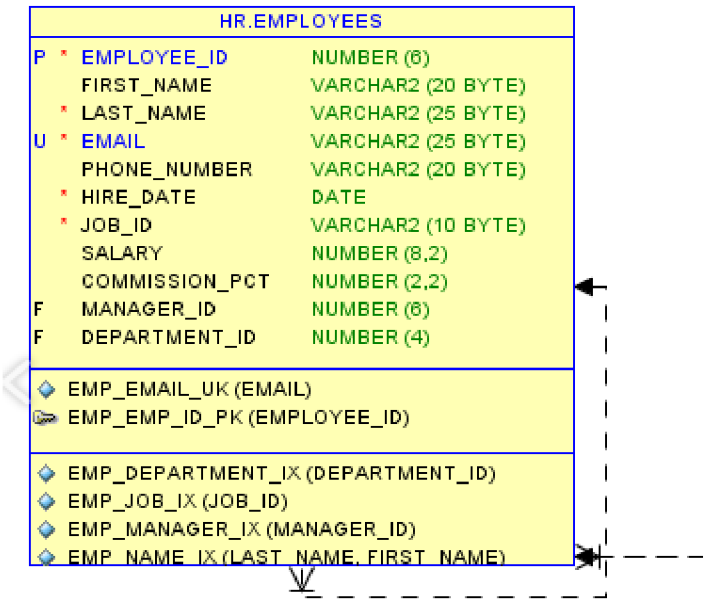
Následuje příklad jednoho ze způsobů značení mohutnosti:

- 0. Připojit lze nula instancí. Tento stav nastává velmi ojediněle a v jeho případě je na místě zvážit správnost návrhu.
- 0…1. K instanci lze připojit žádnou nebo pouze jednu instanci.
- 1. Připojuje se právě jedna instance.
- 0…*. Toto značení znamená možnost připojení nula až n instancí, kde n je kladné celé číslo.
- 1…*. Jedná se podobnou situaci jako u předchozího značení vyjma pravidla, že musí být připojena alespoň jedna instance.

3.1.2.3 Relační (databázový) diagram

Webové aplikace a s nimi značné množství softwaru ke svému správnému fungování využívají data. Taková data se pak ukládají do databází, které jsou oddělené od logiky aplikace. V takovém případě je někdy nutné navrhnout i databázi a rozdělení tabulek či objektů v ní. K tomu v počátku slouží databázový diagram. Ten se ve svém provedení velmi podobá diagramu tříd, ovšem často se nejedná o jeho 1 ku 1 přepis, jelikož mnoho objektově navržených produktů nemá každou třídu určenou k propisu do databáze. Diagramové tabulky či objekty pak nesou svůj název, atributy s datovými typy a propojení s ostatními prvky s jejich kardinalitou.

Dále se tento diagram někdy vyhotoví až po vzniku databáze, jelikož za jejím vznikem stojí mapování. Mapování je automatický proces převedení objektových modelů do databázové podoby, nebo obráceně. Tomuto tématu bude věnována kapitola v následujících částech práce.



Obrázek 2 - Příklad tabulky v relačním diagramu (Thatjeffsmith, 2011)

V objektovém pojetí se tabulky v databázi spojují tak jak již bylo vysvětleno u diagramu tříd. V relačním je to velmi podobné, ale názvosloví se liší.

Kardinalita se značí následovně:

- 1 ku 1. K jedné tabulce lze připojit pouze jednu další tabulku. Takový stav, který bychom označili za pravé 1 ku 1, ve skutečnosti nemůže při psaní databáze na straně serveru v některých jazycích nastat pro potřebu existence cizího klíče.
- 1 ku n. Jeden záznam může znát více záznamů jiné tabulky.
- m ku n. Záznam jedné tabulky může znát více záznamů tabulky druhé a ta zase naopak může znát víc záznamů tabulky první. Takového stavu se docílí za pomoc mezi vazební tabulky. Ta nese cizí klíče obou tabulek.

3.1.2.4 Workflow diagram

Workflow diagram, kterým by mohla analýza softwaru začít. Jedná se o přípravný digram, který stanovuje přesný postup jednotlivých akcí a operací na daném projektu. K pracím pak lze přiřadit kdo je vykonává, kdo za ně má zodpovědnost, v jaké termínu, prerekvizity atd.

Tento diagram pak usnadní plánování jednotlivých činností a odhad jejich příslušní časové dotace.

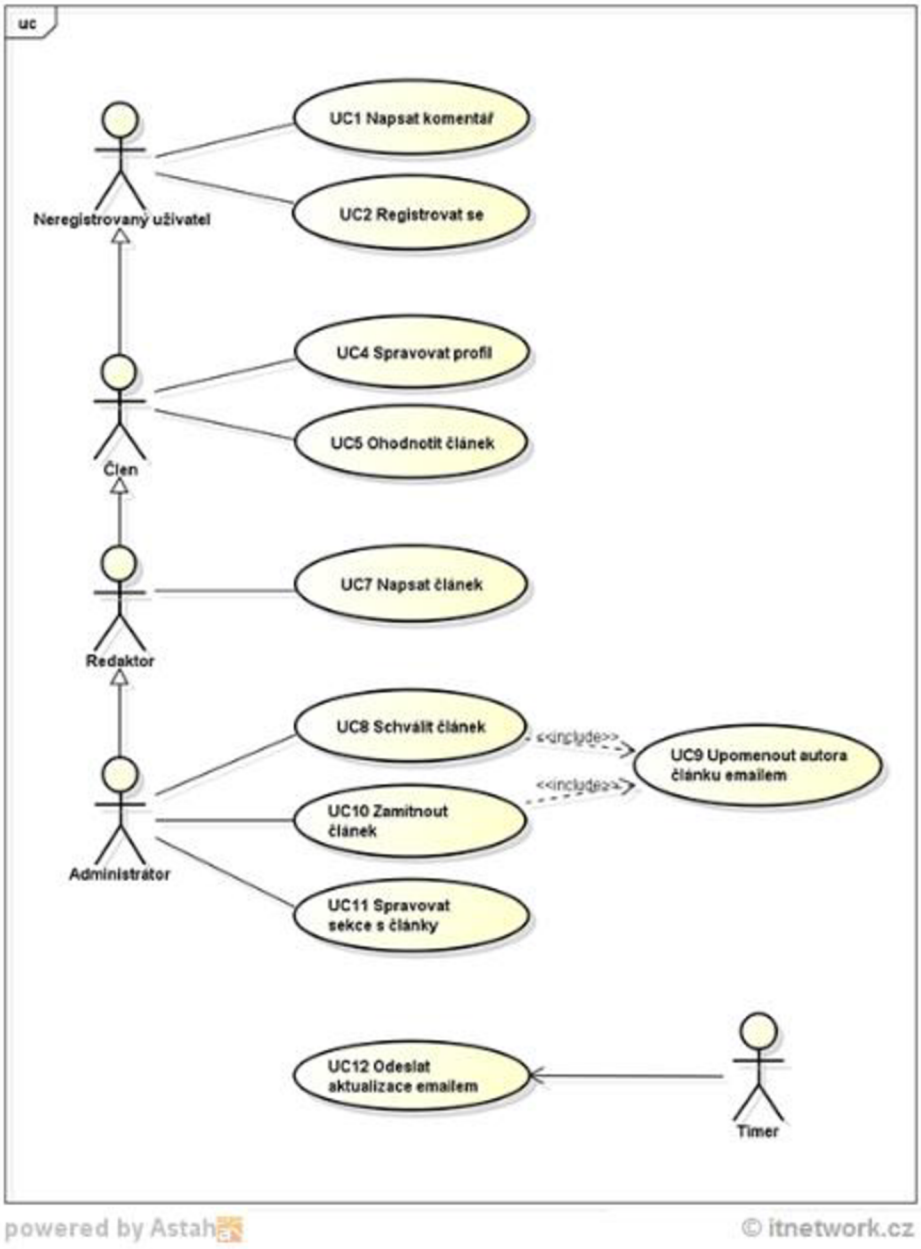
Jeho podoba se odvíjí od jazyku v kterém je psán, ale přístupy k němu bývají poměrně liberální. Lze ho pojmout formou tabulky, či diagramu, ve kterém jsou jednotlivé akce zastoupeny obdélníky s názvem či stručným popiskem dané činnosti uvnitř a vzájemně propojeny. Čísly se pak vyjádří jejich posloupnost. Množství přidaných informací náleží zvážení s přihlédnutím k povaze projektu, vykonavatelského týmu a dalších náležitostí projektu (IBM, 2021).

3.1.2.5 Use case diagram

Use diagram patří k základním diagramům a k jedním z prvních, kterých je třeba pro daný softwarový produkt zhotovit. Česky se dají překládat, jako diagramy užití. Takových diagramů se pak vytvoří mnoho, nebo jednotlivých případů užití konkrétního produktu bývá většinou mnoho. Jedná se také často o předmět debaty se zadavatelem či cílovým uživatelem.

Diagram případu užití je z výše zmíněné klasifikace zařaditelný mezi analytický diagramy. Čili diagramy, které dávají ve známost co aplikace dělá, nikoliv však jak to dělá. V mechaniky fungování use case diagramů hovoříme o 2 zásadních prvcích či částech. První je aktér. Jedná se nějakého člena systému, čili administrátor, uživatel, nebo samotný systém aplikace. Druhým prvkem je samotný případ užití, kde se jedná o sled aktivit, které v něm

probíhají. Účele není vysvětlení jak dané aktivity probíhají, ale pouze informovat o tom že a v jakém pořadí v reakci na čí vstup či výstup tak konají (Čápka, 2013).



Obrázek 3 - Příklad Use case diagramu (Čápka, 2013)

3.1.3 Vývoj software a jeho metodiky

Tvorba software v moderním pojetí je proces trvající v rozsahu řádů dní až let. Pro každý časově rozdílný formát je vhodný jiný postup jeho vývoje. V případě kratších projektů je třeba dbát metodik správného psaní kódu například, ale komplikované způsoby vývoje není třeba užívat, neb by se jednalo o zbytečně silný nástroj na neveliký problém. U rozsáhlejších

projektů a zejména projektů řešených v týmech či celých podnicích se pak takové metodiky stávají nutností.

Zmíněné metodiky svým přístupem k programování, rozvržení práce, stylu dokončování jednotlivých komponent, funkcionalit a částí, přispívají k maximalizaci efektivity využívání zdrojů, financí i časové. Je nutné se zvoleným postupem seznámit nejen řešitelský tým, ale i management, pomocné pracovníky i zákazníka.

3.1.3.1 Životní cyklus vývoje software

V literatuře často označován za SDLC, čili Software Development Life Cycle. Jako vše v živé i neživé přírodě i vývoj software a jeho životní cyklus podléhá jistým pravidelně se opakujícím pravidlům.

Životní cyklus vývoje, nebo někdy pouze softwaru, je série po sobě jdoucích vzájemně propojených činností, které dohromady ve svých jednotlivých stádiích určují stav, využití a podobu daného softwaru od jeho prvotní analýzy až po vyřazení z provozu.

Jednotlivé fáze se vztahují k vývoji větších projektů. V případě, že samotný programátor vytváří menší software například pro vlastní užití, není vždy třeba všech kroků v jejich plném rozsahu, nebo aby vůbec existovaly (Čermák; Martinů, 2018).

3.1.3.1.1 Sběr požadavků

První taková část je sbírání uživatelských požadavků. Po přijetí zakázky si zákazník lépe specifikuje svoje požadavky na daný produkt. Ty si vyhotovitel musí dobře zaznamenat a se zadavatelem zkonzultovat, aby v budoucnu nedošlo k nedorozumění. U těchto požadavků je třeba obeznámit zadavatele s jejich realističností zejména z hlediska časového horizontu vyhotovení. Mnohdy se stává, že management i zákazník nemají porozumění v oblastí vývoje a jejich představy o časové náročnosti na některé úkoly jsou pak zcela mylné (Jevtic, 2019).

3.1.3.1.2 Plánování a analýza

Na základě těchto podkladů a dalších skutečností se začne vypracovávat analýza. Taková analýza však není pouze jedna. Jistě se musí dát vzniknout jedné hlavní analýze celého projektu, kde je v předmětu řešení rozsah prací, alokace zdrojů, časová náročnost

jednotlivých úkonů, finanční hlediska, metodika vývoje a další. S touto analýzou by pak měl být obeznámen každý na projektu se podílející zaměstnanec včetně zadavatele.

Dále však následují jednotlivé menší analýzy řešitelských týmů jak pro celý projekt, například z hlediska programátorského, tak pro jednotlivé funkcionality či kroky.

Některé zdroje uvádějí analýzu před sběrem požadavků, některé zas po. Sběr požadavků je dokonce někdy brán jako součást plánování (Jevtic, 2019).

3.1.3.1.3 Prototypování a designe

Než započne samotné programování, je u větších projektů třeba vytvořit jakýsi prototyp jak pro zákazníka tak pro vyhotovitele. Prototyp nemá být 1:1 verzí výsledného produktu, pouze jaká si kostra se základními prvky. Nemusí se dokonce ani jednat o funkční program, ale o jakýsi drátový model tlačítek a textových okýnek pro účely předvedení možného designu aplikace (Jevtic, 2019).

3.1.3.1.4 Samotný vývoj

V této části dochází k významným programátorským a vývojářským činnostem. Jedná se o fázi kde programátoři programují jednotlivé funkce v zadané aplikace či softwaru. Programování je rozděleno často na několik částí dle vrstvy, které je programována. Jeden tým, nebo jeho část, by měl řešit backend softwaru. Čili jeho databázi, práci s daty, logiku jednotlivých funkcionalit. Další tým může řešit uživatelské rozhraní a frontend. Někdo jiný má zas na starosti testování.

Programátoři si dělají vlastní menší analýzy pro své vlastní potřeby.

V průběhu programování se často odevzdávají, nebo prezentují funkční prototypy. Popřípadě se mění zadání a různé požadavky.

3.1.3.1.5 Testování

Testování aplikace provádějí již při programování programátoři, avšak po dokončení významní části, nebo celého softwaru, je třeba v testování pokračovat. Existuje celá škála testů dle zaměření, provedení či účelu.

Například pro bezpečnost dat se provádí penetrační testy, kdy se tester snaží najít slabé místo v systému a dostat k citlivým datům, nebo vyřadit produkt z provozu.

Další testy jsou běžné uživatelské, kdy se provádí předpřipravené scénáře, kterými tester prochází, hledá a reportuje jednotlivé chyby.

U všech variant testování je třeba každou chybu vždy zevrubně zaznamenat a popsat, aby programátor přesně věděl jak se chyba chová a jak ji lze replikovat.

K testování patří i třeba testování náporu velkého množství uživatelů na server, nebo testování funkčnosti či hratelnosti nových herních prvků či mechanik. Toto testování je pak velmi populární u e-sportů a videoher. U testování často platí, že je třeba testovat pravidelně a ve více částech životního cyklu. Zejména pak při programování, nebo nasazení (Geeks for geeks).

3.1.3.1.6 Nasazení

Po dokončení programátorských částí je třeba výsledný produkt z firemních zdrojů nasadit pro zamýšlení užití zákazníkem. To může představovat nasazení na server a zaregistrování domény pro webovou aplikaci, nebo nasazení na firemní server aj. Nasazení zkrátka představuje převedení softwaru na místo určení pro jeho užívání (Jevtic, 2019).

3.1.3.1.7 Údržba

Tato část přichází po doručení finálního produktu zákazníkovi. Smlouvá, která je podepisována při předávání zadání vyhotoviteli je zde základem. Zákazník může v rámci zadání požadovat po vývojářích, aby po předem určenou dobu užívání přidávali prvky a nové funkcionality, nebo udržovali produkt v jeho správném a zamýšleném chodu. V jiných případech, kde například studio vydává produkt k jeho širokému užité veřejností, je v nejvyšším zájmu vývojáře, aby produkt udržoval a popř. vylepšoval a upravoval dne nutnosti.

3.1.3.1.8 Zánik a vyřazení z provozu

Poslední částí, někdy neoficiální, je vyřazení produktu z provozu. Pakliže již není jeho užívání třeba, nebo je například nahrazen jiným produktem, software se přestane používat, smaže se ze serverů, nebo se mu odstaví určité funkcionality.

3.1.3.2 Metodiky vývoje software

Následující kapitola je výčtem vybraných metodik vývoj software.

3.1.3.2.1 Vodopádový model

Nejjednodušší a nejběžnější metoda vývoje je tzv. vodopád. Jedná se o přístup, kdy vývoj jde z pomyslného bodu A do bodu B. Čili začne se např. analýzou, poté se programuje prvek po prvku až je software hotov.

V praxi se tento model často těší neblahé pověsti, avšak u malých projektů, neboj jednočlenných týmů nemusí tento model vykazovat obtíže. Naopak se může jednat o ideální přístup. Zejména pak u jednoduchých, např. školních úloh by bylo zcela zbytečné užívat složitějšího přístupu.

U větších projektů pak je již třeba si na vodopádové sklony dávat pozor, neb ani nejlepší vývojář nevidí na konec projektu a některé části kódu, které jsou v danou situaci správné, mohou podtrhnout nohy budoucím funkcionalitám, nebo být redundantní. Obtíže by se pak mohly dostat do stavu, kdy je třeba refaktorizace celého kódu.

3.1.3.2.2 Výzkumník

Jedná se o model, kde se vývojář učí při vývoji. Vývoj je tedy jakýmsi pokusem při kterém například zjišťuje co technologie umí, nebo nutné informace zjišťuje v průběhu vývoje. Výzkumník přichází, když zpracovatel neovládá danou problematiku. Projekt pak z jeho pohledu lze pojmout jako učební zkušenost.

Tento model lze považovat spíše za anti-model. Nemusí nutně být špatný, ve školním prostředí může být velmi benefiční, ale u profesionálních a rozsáhlých projektu by své využití nejít neměl (Čermák; Martinů, 2018).

3.1.3.2.3 Iterativní modely

Iterativní modely, nebo spíše přístup. V tomto případě se k vývoji přistupuje iterativně, tedy se jisté úkony opakují. Významnou složkou iterativního přístupu je aktivní účast zákazníka na vývoji. Ten se nutně neúčastní samotného vývoje, ale pravidelně se mu dostává například

prototypních verzí produktu a on je aktivně testuje. Na základě jeho podnětů se pak lépe pracuje k výsledku, který uspokojí obě strany.

Výhodou je ranné a snadné odhalení chyb, nedorozumění či nedostatků.

3.1.3.2.4 Prototyp

Model prototyp následuje první 3 fáze životního cyklu. Ve fázi čtvrté, a to ve fázi programování, zpracovávající tým naprogramuje prototypní verzi výsledného produktu. Ta nemusí mít všechny funkcionality, ani se vzhledově přibližovat finální verzi.

Tento prototyp je pak předložen zákazníkovi k otestování. Ten obratem doloží své připomínky a požadavky. Dle těch je prototyp upraven a jsou přidány další funkcionality. Tento iterativní postup se opakuje do bodu, kdy je zákazník spokojen a může nastat další část životního cyklu software.

Tímto přístupem, lze přistupovat i k jednotlivým částem a subsekcím systému či aplikace. Většina testovacích prací probíhá zároveň s vývojem.

3.1.3.2.5 Inkrementální model

V případě inkrementálního modelu se jedná o jistou kombinaci iterativního přístupu a vodopádového. Vodopádový model je brán za nepříliš vhodný pro velké projekty, ale zde je užit efektivně a vhodně.

Projekt či produkt je rozdělen na menší celky a ty jsou následně vyvíjeny, testovány a jeli to tak nasmlouváno, předkládány zákazníkovy vždy ve vodopádovém stylu. Po dokončení dané části je přestoupeno k části další a ta je opět vyvíjena stejným způsobem. Toto opakování je pak iterativní složkou tohoto vývojového přístupu. Tyto drobné části programu jsou v tomto přístupu považovány za ony přírůstky (Čermák; Martinů, 2018).

3.1.4 Persony

Součástí vývoje softwaru je tvorba person. Persona je tradičním pojetí jakési kolektivní nevědomí. Čili neexistující bytost představující nějakou existující. Dalo by se říct, že se jedná o jakéhosi slaměného panáka, kterému dáme kýžené vlastnosti pro jistou situaci. Tradičně se persony tvoří pro marketing účely (Řezníček, 2016).

Takovou situací v aktuálním pojetí je tvorba softwaru. Pro jeho správný a hlavně zákaznicky orientovaný přístup je třeba vždy nějaké persony navrhnout.

Tvorba person je metoda tvorby těchto neexistujících lidí. Jako persona se navrhne uživatel konkrétního produktu. Přiznají se mu vlastnosti, jméno a atributy. Tyto informace ani nemusí přímo souviset s daným produktem. Ale z velké části by k nějakému přesahu dojít mělo. U person je například v otázce jejich přístup k technologiím a zejména technologií souvisejícími daným produktem. Dále je u tvorby persony třeba stanovit jak bude software užívat a proč by k tomu mělo dojít. Tyto a další dodatečné informace se u person uvádějí s ohledem na potřeby, typ a účel vyhotovovaného produktu. Na získání informací o cílovém uživateli, by měl podnik zodpovědný za vývoj, vynaložit své zdroje i finance a pravidelně se v této otázce aktualizovat. Složitější práci v tomto ohledu mohu mít nováčci v poli, kteří třeba nemají ještě vytvořenou zákaznickou základnu. Ovšem lze předpokládat, že mají jisté znalosti o svém odvětví a dokážou se dopustit erudovaného odhadu. Na jeho základě pak mohou spustit například dotazníkové šetření. Zajeté podniky pak již mohou částečně staré persony recyklovat.

Pro případ každého většího projektu by se person mělo vyvinout mnoho a různého druhu. Lze vymyslet například pozitivní personu, to je ideální uživatel produktu, nebo negativní personu, což je ten nejméně pravděpodobný, nebo zcela nevhodný uživatel (Billestrup; Stage; Bruun; Nielsen; Nielsen, 2014).

3.2 Programování

Programování je cílená a vědomá činnost při které programátor píše množinu, v daném prostředí srozumitelných instrukcí, které říkají počítači, co a jak má vykonat. Výsledkem je pak počítačový program (Codeacademy).

Napsanému kódu se říká zdrojový kód. Tomu nemusí počítač nutně rozumět, a proto je třeba tento kód přeložit na strojový kód, kterému již počítač rozumí a může ho provést. Tento proces se nazývá kompilování, nebo kompilace kódu.

Příkladem jazyků, které takto kompilují jsou C# a C++.

Existují pak jazyky, které tzv. interpretují a nekompilují. Takový je například Javascript.

Programovací jazyky se klasifikují do několika skupin dle způsobu, jakým se kód píše, nebo co se v něm vyskytuje za konstrukty. Mezi ty patří funkcionální, procedurální, strojové, skriptovací, nebo objektové jazyky (Wilkins, 2021).

3.2.1 Objektově orientované programování

Objektově orientované programování, nebo zkráceně pouze objektové programování (dále jen OOP), je styl zaměřený na objekty, kterým přisuzujeme vlastnosti a schopnosti. Na rozdíl od funkcionálního programování, které je ryze založené an logice a funkcích a objekty nepoužívá. To ovšem neznamená, že by OOP nevyužívalo praktik funkcionálního programování.

Základním stavebním kamene OOP je třída. Třídě lze někdy říkat model. Třída je konstrukt uložený v programu, který má svůj název, atributy a schopnosti. Se samotnou třídou toho lze provádět více, ale o tom později v textu.

3.2.1.1 Struktura objektově orientovaného programování

Zá základní stavební bloky OOP lze považovat následující.

- Třída. Kolekce vlastností a metod, která slouží pro tvorbu jednotlivých objektů dále v programu.
- Objekt. Vytvořená instance třídy s reálnými daty, které mohou být unikátní. Jejich podoba je odražena od naprogramování. Některé objekty mají předdefinované vlastnosti, některé čekají na vstupní data a jiné jsou kombinací obojího. Objekt může odrážet prvek našeho světa.
- Atribut. Vlastnost různého typu přisouzená dané třídě. Může se jednat o číselný atribut, atribut textového řetězce, objekt jiné třídy aj.
- Metoda. Funkce uvnitř třídy, která definuje její chování a interakci s okolním programem. Metody třídy se volají na instance čili na vytvořené objekty a nelze je volat v absenci kterékoliv instance. Metody vždy pevně náleží konkrétní třídě. Metody mohou mít vstupy i výstupy (Gillis, 2021).
- Další odnoží tříd jsou rozhraní a abstraktní třída. Rozhraní je třída, která v sobě nese pouze metody a nemá ani atributy, ani k ní nelze vytvořit instanci. Abstraktní třída se jeví jako plnohodnotná třída, ale nemůže dát vzniknout objektům. Abstraktní třídu

volíme například v případě, že je třeba vytvořit třídu od které budou dědit třídy jiné, avšak není třeba její inicializace.

Příklad třídy zaměstnance:

Atributy:

- ID, unikátní identifikační číselný klíč.
- Jméno
- Datum narození
- Mail
- Pozice

Metody:

- Pošli mail
- Potvrď mail

Objekt by pak měl reálnou podobu a vypadal by např. následovně:

- 7734
- Lucie
- 13.2.1998
- lucka@genericka_firma.cz
- HR manager

Metody by se pak volaly na Lucii s konkrétně upravenými vstupy a následně i výstupy. Ale metoda pošli mail by nejspíše žádný výstup neměla, což je také možné.

3.2.1.2 Principy objektově orientovaného programování

Objektově orientované programování stojí na několika základních principech, které se v menší či větší míře většinou v praxi užívají.

3.2.1.2.1 Zapouzdření

Jedná se o princip, který znemožňuje komukoliv a čemukoliv zasahovat do atributů daného objektu vyjma jeho vlastní třídy. S objektem lze komunikovat pomocí jeho metod, ale nikoliv mu přímo měnit vlastnosti.

Pakliže je objekt namapován do databáze a tam uložen, lze jeho vlastnosti měnit v místě databáze, ale to přesahuje rámec objektového programování. V samotném programu by k vlastnostem třídy neměl být, vyjma přes specifické rozhraní, přístup nijak umožněn.

Nechť existuje třída pes. Pes má atribut míra hladu a metodu nakrmit. V kódu by mělo být umožněno psa nakrmit, ale nikoliv přímo zasahovat do jeho úrovně hladu (Petkov, 2018).

3.2.1.2.2 Abstrakce

Objekty a třídy vykazují při své komunikaci s jinými částmi kódu pouze takové části svých mechanismů, které jsou relevantní v dané komunikaci. Ukrývají tak nadbytečnou logiku, která není v daný moment třeba.

Příkladem by mohlo být zaslání zásilky na poště. Nám, jakožto zákazníkům pošty, stačí na poštu přijít, zásilku předat, službu zvolit a zaplatit. Veškeré vnitřní práce pošty s předáváním, vážením, sledováním a doručováním zásilky nás nemusí nikterak trápit, jelikož se naší komunikace netýkají (Gillis, 2021).

3.2.1.2.3 Dědičnost

Dědění je mechanismus, při kterém je jedna třída oddděděna od třídy druhé. V tomto pak získá její atributy a metody. Výsledný kód pak šetří na redundanci a někdy i složitosti. V praxi je snaha dědičnost vždy nepoužívat, kdy je to možné, ale naopak ji v některých případech nahradit skládáním například. Při skládání se jedna třída poskládá z tříd dalších.

Vhodné užití dědičnosti je v případě, kdy se v programu vyskytuje velké množství tříd se společnými vlastnostmi i metodami. Taková situaci vznikne v modelové ukázce, která následuje.

Uvažujme program zoologické zahrady. Namísto tvorby třídy pro každé zvíře, vytvoříme jednu třídu zvíře. Ta bude mít všechny, pro ostatní zvířata, společné atributy a metody. Vypadat bude následovně.

Zvíře

Atributy:

- Jméno
- Hmotnost
- Pohlaví
- Označení výběhu

Metody:

- Udělej zvuk
- Nakrm

Od této třídy se pak oddědí například třída lev, sklípkan brazilský a vlaštovka obecná. Tyto tři třídy budou automaticky mít výše vypsané atributy a obě metody. Další, pro ně již specifické, metody i atributy jim dopsat lze.

Třídě, od které se dědí se říká rodič a třídě, které je odděděna pak dítě, nebo potomek.

3.2.1.2.4 Polymorfismus

Slovo polymorfismus je odvozeno z řeckého slova a znamená mnoho tvarů.

Polymorfismus umožňuje užití jedné metody mnoha objekty. Výsledek užití konkrétní metody pak bude záležet na objektu na který je metoda volána a na vstupních parametrech.

Například metoda „udělej zvuk“ z metody zvíře bude volatelná i na potomky zvířete. Každý potomek pak může mít nastaveno programátorem, jak na tuto metodu bude reagovat. Každý potomek potom může mít reakci unikátní.

Bude-li metoda „udělej zvuk“ zavolána na objekt třídy pes, vrátí se textový řetězec „štěk“, proběhne-li volání na třídu kočka, výsledek pak bude „mňau“. Jedná se o stejnou metodu, ovšem vždy volanou na jiný objekt.

Dalším příkladem je pak volání metody Spočítej obsah. Tato metoda bude počítat jiné vstupní parametry u trojúhelníku a jiné u elipsy (Petkov, 2018).

3.2.2 Návrhové vzory

Návrhový vzor je obecné řešení a jeho postup k známému a často se opakujícímu problému v programování softwaru. Kořeny pojmu lze nalézt mimo softwarové inženýrství, například v architektuře.

Nejedná se o konkrétní kus kódu či knihovnu, kterou lze implementovat přímo do řešení. Samotné napsání a implementace je čistě na programátorovi. Úprava daného vzoru je závislá na daném programu a vždy unikátní, neb se vždy týká jiného softwaru, jiných tříd, metody, funkcí, proměnných atd . (Refactoring.guru).

Občas se návrhový vzor zaměňuje s algoritmem. Algoritmus je konečný, konkrétní postup činností vedoucí k jistému výsledku (Cambridge dictionary). Jako typický příklad se uvádí kuchařka. Jeden recept na jeden pokrm je jistý druh algoritmu. Návrhový vzor je pouze jakýmsi plánem či „blueprintem“ pro daný problém (Refactoring.guru).

Návrhových vzorů existuje celá řada, některé jsou anti-vzory, některé jsou velmi rozšířené a používané skoro v každém projektu.

3.2.2.1 Singleton

Jedináček je návrhový vzor, který se stará o to, aby konkrétní třída mohla dát vzniknout pouze jediné instanci. Takové chování je žádoucí v případě, že programátor chce zavést restrikci přístupu k určitým datům. Ovšem v praxi se jedináček často označuje za anti-vzor a jeho využití je u větších projektů poměrné vzácné.

Obecně se považuje jeho využití vhodné pouze v případě logování.

Příkladem z reálného světa je vláda. Každý stát má pouze jednu vládu.

3.2.2.2 Factory method

Factory, neboli továrna je vytvářecí návrhový vzor, který má pomoci dát vzniknout prostředí pro tvorbu instance třídy. Který objekt konkrétně bude vytvářen je řešeno právě touto factory metodou. Ergo není voláno vytváření objektu přímo, ale přes jakousi podtřídu, která sama zvolí jaká třída má být vytvořena. Tento návrhový vzor šetří velké množství kódu. Lze díky němu ošetřit neporušení O části principů SOLID. Volané třídy netřeba mutovat, ale lze

potřebné modifikace provést v místě rozhraní factory. Dále lze díky factory vzoru podpořit, nebo naopak porušit S princip. Za nevýhodu by šlo považovat zvýšení obtížnosti a složitosti struktury programovaného softwaru.

Příkladem buď již aplikace, která posílá virtuální vyučující do virtuálních tříd. Každý vyučující učí určitý předmět. Při iniciaci nové hodiny je třeba vyučujícího volat. Takové volání by proběhlo pomocí factory vzoru, který by si přebral o jaký vyučovaný předmět se jedná a na jeho základě pak inicioval objekt správného vyučujícího (Refactoring.Guru).

3.2.2.3 Strategy

Strategy pattern je návrhový vzor ovlivňující a zejména ošetřující chování kódu. V případě tohoto vzoru programátor vytvoří několik rozhraní s algoritmy a ty jsou pak volány při běhu aplikace v závislosti na vstupních požadavcích klienta.

Tato rozhraní se nazývají strategie. Při užití toho vzoru je třeba třídu, které se funkcionalita na kterou tento vzor aplikuje programátor, upravit tak, aby přijímala libovolné rozhraní a to pak rozhodovalo o chování aplikace. Jaké rozhraní pak bude třídě doručeno záleží čistě na chování uživatele na straně klienta.

Hlavní výhodou je odlehčení kódu a rozdělení zodpovědností. Mít všechny algoritmy přidružené k jedné třídě může časem vytvořit tzv. špagetový kód, který se stane noční můrou pro jakoukoliv opravu. Při vzniku sebemenší chyby se pak náprava stane velice náročnou. Rozdělit algoritmy dle jejich logiky a pouze volat potřebné rozhraní nesoucí správnou sadu těchto algoritmů tento problém značně ulehčí. Dále může zachránit kód o dědičnosti a nahradit jí skládáním.

Strategie je ovšem velmi silným nástrojem a proto je třeba zvážit jeho užití. V případě na algoritmy lehčí aplikaci jej nemusí být vůbec třeba (Kumar, 2021).

3.2.2.4 Prototype

Prototype pattern umožňuje vytvářet klona existující instance určité třídy. To samo o sobě v případě jednoduššího kódu nezní jako problém. Ovšem má-li daná třída některé své vlastnosti nastavené jako privátní, může vzniknout při získávání jejích hodnot obtíž. Další háček je v případě velkých tříd, které mají velké množství atributů.

Pro tento problém existuje řešení v podobě prototypu. Programátor vytvoří rozhraní, které nese metodu, nebo metody na klonování existujících tříd. Ty musí mít v sobě klonování umožněno. V kódu se pak volá příslušná klonovací metoda a ta daný prototyp vytvoří.

Výhodou tak je šetření kódu o redundantní volání iniciace tříd a také práce s třídami bez přímého kontaktu s nimi. To může sloužit jako pojistka proti nesprávným zásahů méně zkušených programátorů do kódu (Source Making).

3.3 SOLID principy

SOLID principy jsou sada 5 principů správného přístupu k psaní kódu pro objektově orientované programování. Účelem existence těchto principů je vytváření čistého, dobře čitelného, funkčního a snadno testovatelného kódu. Jejich dodržování může vést k existenci většího množství adresářů a složek v souboru projektu a tím vytvořit zdánlivě větší složitost, ale z hlediska samotného kódu dají vzniknout přehlednějšímu a čistšímu kódu. Vzniknout jim dal počítačový vědec Robert J. Martin v roce 2000. Akronym však vznikl později rukou Michaela Featherse (Erinç, 2020).

3.3.1 Single responsibility principle

V češtině princip jedné zodpovědnosti, zde S princip, diktuje programátorovi, aby své třídy psal tak, aby každá třída měla pouze jeden důvod k existenci a jediný důvod ke změně. Ergo aby třída dělala pouze jednu věc.

Dodržení tohoto principu vede k ulehčení testování. Třída, která má na starosti pouze jednu věc bude mít méně testovacích scénářů. Dále je přehlednější a méně provázaná s jinými částmi kódu. Tedy se v kódu lépe a snáz nacházejí a odstraňují chyby.

3.3.2 Open for extension, closed for modification principle

O princip říká otevřeno pro rozšíření, uzavřeno pro úpravu. Tento princip říká, že by se žádná třída neměla po svém vzniku již upravovat co se již existujících atributů týče. Takovým úpravám se říká mutace. Ovšem naopak by třída a kód okolo ní jako takový měli být napsáni tak, aby šla třída bez obtíží rozšířit. Vyvarování se takovému chování pomáhá předcházet chybám v aplikaci a v případě aplikace s databází a jejím mapováním i poškození databáze aplikace.

Příkladem může být případ s mapováním, kde se převádí modely aplikace na tabulky v databázi. Takový proces je automatický a jako takový může snadno dát vzniknout chybám. Při mutaci třídy pak takové chyby mohou vzniknout snáz. Největší problém pak je mutace atributů odkazujících se na jiné třídy. Databáze se tak může dostat do podoby, kdy se aplikace stane zcela nefunkční a je pak třeba celou databázi i s daty vymazat a namapovat znovu.

3.3.3 Liskov substitution principle

Princip Liskovi substituce je nejkomplexnějším z 5 SOLID principů a jedná se o princip L. Zjednodušeně říká, že jeli v nějaké části kódu volána třída B, která je oděděna od třídy A, kód by měl fungovat i když bude volána třída A, a naopak. Dále by volání mělo fungovat i na třídu C, také oděděnou od třídy A.

Například mějme třídu garáž, která opravuje vozidla. Mějme třídu vozidlo od kterého je oděděna třída automobil a dodávka. Třída garáž je napsána tak, aby opravy v ní probíhali na automobil. Při dodržení tohoto principu by měla oprava fungovat i třídu dodávka (Dupire, 2020).

3.3.4 Interface segregation principle

I princip představuje princip segregace rozhraní. Tento princip zadává velkým rozhraním, aby byla rozdělena na rozhraní menší a úzce specializována. Třídy, která tato rozhraní užívají potom nemusí zbytečně implementovat metody, které nepotřebují. Ergo třída by měla implementovat pouze to co používá. Jedná se o princip zlepšující přehlednost a funkčnost kódu. Dále je to princip, který dodržet je z těchto 5 principů nejjednodušší a často poměrně intuitivní (Millington, 2022).

3.3.5 Dependency inversion principle

Princip D říká, že třídy kódu by měly být závislé na abstrakci, tedy na rozhraních a nikoliv na jiných existujících implementacích. Implementace a části kódu řešící nastavení politik a pravidel by neměly implementovat třídy a moduly kódu zabývající se nižšími potřebami softwaru a naopak. Tento princip pomáhá snižovat složitost, vzájemnou provázanost a závislost kódu.

3.4 Webová aplikace

Webová aplikace je software, který je zpřístupňován prostřednictvím Internetu, nebo počítačové sítě. Jedná se o program, který je uložen na vzdáleném serveru a uživatel se k němu tradičně dostává pomocí internetového prohlížeče. Takové aplikace není nutno stahovat ani instalovat i když v nedávné minulosti a někdy i dnes je nutné pro jejich plný provoz stáhnout přídavný software do uživatelova internetového prohlížeče. Mezi takové aplikace se tradičně řadí e-mailový klient, e-shopy, sociální sítě či platformy jako YouTube. Výhodou je jejich okamžitá přístupnost a nenáročnost na užívání na straně uživatele. Nutné je však neustálé připojení k síti a není takto možné vést každý software. Mnoho aplikací je příliš velkých nebo na výpočetní kapacitu náročných, aby mohly být spuštěny přes internetový prohlížeč (TechTarget, 2019).

3.4.1 Webový server

Webový server lze chápat z dvou hledisek. Jakožto hardware a jako software.

Hardwarově se jedná o počítač, na kterém jsou uložena data o webových stránkách, jejich skripty, příslušné komponenty a přístupy.

Ze softwarového hlediska je webový server několik komponent, které spravují server jako hardware. Má na starosti práci s požadavky, URL adresami, http protokoly atd.

Tyto dvě komponenty musí spolupracovat a být provázány, aby jimi spravované webové aplikace mohly fungovat. Internetový prohlížeč, tedy klient, pošle žádost na webový server. Ten požadovanou stránku vygeneruje a pošle zpět, nebo pošle z databáze vytažená data a stránka se poskládá an straně klienta (MDN contributors, 2022).

3.4.2 Aplikační server

Aplikační server je softwarová komponenta, která slouží k běhu obvykle větších aplikací. Často se využívá u velkého podnikového softwaru. Aplikační server je jedna ze dvou částí větší aplikace, která je nainstalována na sdíleném serveru. Druhá část je pak v počítačích uživatelů a té se říká klient. Aplikační server zodpovídá za část funkcionalit daného softwaru. Například předává a zpracovává požadavky na databázi a pomáhá je zobrazovat na straně klienta (Management Mania, 2019).

3.4.3 Web hosting

Hostování webové aplikace spočívá v jejím uložení na fyzickém serveru, tedy na vzdáleném počítači. Provozovatel takového serveru pak zodpovídá za jeho funkčnost a správný běh. Programátor si tak pronajímá výpočetní kapacitu a paměť pro svoji aplikaci. Aplikace je pak spouštěna přes internetový prohlížeč, který komunikuje se serverem kde je uložena.

První věci v otázce hostingu je rychlost spouštěné aplikace. Ta závisí na vzdálenosti serveru od klienta, kvalitě hardwaru a správně nastavené architektuře. Například kde je databáze, na které jsou uložena data potřebná pro běh aplikace.

Druhá důležitá otázka je zabezpečení. Nejvyšší prioritou by měla být osobní a citlivá data uložena v databázovém prostoru přidruženém aplikaci. Například jsou-li aplikací ukládány platební údaje, bezpečnost musí být na nejvyšší úrovni.

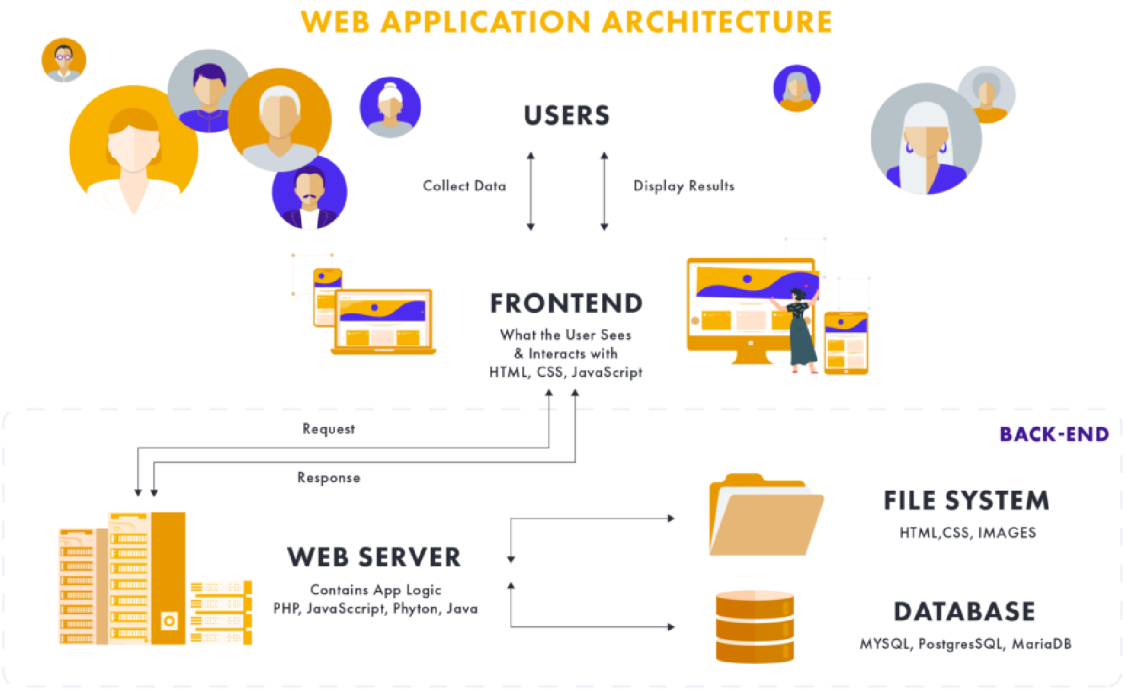
Otázka výpočetní kapacity, bezpečnosti i ceny je předmětem nastavení smlouvy mezi provozovatelem serveru a majitelem aplikace.

3.5 Backend

V jazyce počítačové vědy je backend část počítačového softwaru, kterou uživatel nevidí byť s ní nepřímo či přímo pracuje. Součástí backendu je i samotná databáze a její správa, server na kterém vše běží a část kódu aplikace. Jedná se o tu část, která je zodpovědná za logiku dané aplikace a za komunikaci s databází a za práci s daty uloženými v ní. Backend kód pracuje s uživateli zadanými příkazy a v reakci na ně upravuje uživateli vyobrazený obsah.

Mezi činnosti, které má backend na starosti patří spouštění skriptů které vyobrazují uživatelský obsah. Dále komunikaci s databází, ergo výběr, ukládání a šifrování dat. A pak především odpovídání na uživatelské žádosti zaslané prostřednictvím klientské části aplikace.

Souběžným pojmem k backendu je pak frontend. To je ta část aplikace, kterou již uživatel vidí a přímo s ní spolupracuje. Jsou to jemu zobrazované formuláře např. na webové stránce. Uživatel tak sice pracuje s frontend rozhraním, ale nepřímo tak ovládá i část backend rozhraní, které s tím frontendu neustále spolupracuje.



Obrázek 4 - Backend vs. Frontend rozdělení (Legievski, 2021)

3.5.1 N-vrstvá architektura

Počítačové systémy od jejich počátku existují v jistých výpočetních modelech nebo architekturách. Tyto architektury diktují uspořádání a organizaci dat, aplikací a veškerého hardwaru ne němž běží.

Jedna taková architektura je klient-server. Ta oddělila počítač s klientskou, čili obslužnou, částí od serveru, na kterém byla uložena data se kterými uživatel pracoval. To vedlo k odlehčení nároků na obsluhující koncový terminál a k centralizaci dat.

Jelikož se tento model ukázal jako funkční, architektura se vyvinula a přesunula také na úroveň softwaru v podobě třívrstvé architektury. Ta dělí software na tři vrstvy. Aplikační, prezenční a datovou.

Prezenční vrstva je část aplikace, která je viditelná pro uživatele čili frontend. Jedná se tak například o rozhraní e-mailového klienta, nebo o rozhraní e-shopu.

Aplikační vrstva (také funkční vrstva) je ta část softwaru, která zpracovává požadavky zadané klientskou částí a posílá je do databáze. Říká si tak o data, která poté pošle zpět na prezenční vrstvu k jejich zobrazení. Na této vrstvě operuje middleware.

Datová a nejnižší vrstva má pak na starosti práci s daty. Tato vrstva je kde pracuje systém řízeni báze dat. Ten dostává prezenční vrstvou zpracované, vypočítané a pro databázi přeložené požadavky z aplikační vrstvy. Dále pracuje s daty a ta pak pošle prezenční vrstvě k jejich dalšímu zpracování a výpočtům (Management Mania, 2015).

Vícevrstvá architektura je v softwarovém inženýrství označení pro aplikace, kde jejich zdrojové kódy jsou rozděleny mezi více částí a v případě nasazení i počítačů. Jedná se o obdobu třívrstvé architektury, akorát zde je možné mít aplikaci jako dvou, tří, čtyř, nebo vícevrstvou. Proto je někdy na tuto architekturu odkazováno jako na n-vrstvou.

Základní rozdělení je totožné s třívrstvou architekturou. A to na prezenční (frontend) část, aplikační (někdy též business) část a datovou část. Poslední dvě části dohromady tvoří backend. Jejich role jsou stejné jako v případě třívrstvé architektury. Rozdělení je možné i na pouze frontend a backend část. Frontend má na starosti stejné náležitosti jako prezenční část aplikace v případě třívrstvé, backend pak pojí obě složky aplikační a datové části. Architekturu lze rozvést i ve složitější a mnohavrstvý systém (Martin, 2022).

3.5.2 Databáze

Databáze v pojetí informatiky je organizovaná množina strukturovaných dat, nebo informací. Tato data či informace jsou elektronicky uloženy na nějakém serveru.

Pro obsluhu databázi je třeba softwaru, kterému se říká Systém řízení báze dat. Takový systém většinou užívá nějaký strukturovaný jazyk pro obsluhu dat (Oracle).

Způsob ukládání a práce s daty potom diktuje o jaký typ databáze se jedná. Nejčastější v dnešní době bývají relační a objektové databáze. V relační databázi jsou data uspořádána pro co možná nejvyšší přehlednost v tabulkách. Sloupce představují jednotlivé atributy a řádky záznamy. Tabulkám se říká relace. V objektových databázích se k datům přistupuje jako v objektovém paradigmatu programování. Data jsou uspořádána jako modely. Přístup k nim je velmi podobný jako u relačních databází, avšak příkazy a jejich logika nad nimi jsou rozdílné.

Mezi největší výzvy z hlediska databází dneška je bezpečnost dat a jejich množství. Se zpřísňujícími se politikami na uchovávání a práci s osobními daty se jejich úniky stávají více problematické. Nadále jakýkoliv únik dat z aplikace je nežádoucím jevem, který může daný podnik stát velkou finanční částku, nebo v krajních případech přivést k ukončení činnosti. Databáze se tímto stává nejzranitelnějším místem celé struktury většiny aplikací a je třeba, aby data byla zabezpečena. Pro jejich zabezpečení je třeba udržovat bezpečnostní protokoly první na úrovni fyzické, tedy aby k serverům s databázemi měli přístup jen povolené osoby. Dále je třeba, aby aplikace k těmto databázím přistupující byly bezpečně naprogramovány. Únik dat není jediná obtíž, která může databázi potkat. Poškození jejích dat v důsledku nesprávného naprogramování obslužného softwaru může nést podobné, nebo horší následky (Begg, Connolly, Widom, 2010).

3.5.2.1 Relační databáze

Relační databáze je databáze založena a odvozena z relačního modelu uspořádání dat. Relační model představuje způsob ukládání dat do tabulek. Každá tabulka představuje jistou entitu, neboj objekt. Ten se stává jakousi reprezentací objektu reálného světa. Sloupce v tabulce jsou jednotlivé popisující atributy tohoto objektu a řádky pak záznamy.

Například mějme tabulku Psů. Atributy, tedy sloupce budou Primární klíč, jméno, plemeno, hmotnost, pohlaví a datum narození. Příklad jednoho záznamu (řádku) by mohl vypadat následovně.

- 1
- Alík
- Labrador
- 29kg
- Fena
- 21.11.2019

Relace, po kterých je model pojmenován jsou jednotlivé tabulky.

Tabulky však nejsou tím jediným, co databáze v relačním modelu tvoří. Dalším základním stavebním kamenem jsou vztahy mezi těmito relacemi. Tabulky se propojují z hlediska kardinality, čili mohutnosti vazeb. Mohutnost vazby přisuzuje kolik záznamů tabulky jedné,

může náležet záznamu tabulky druhé. Propojení mezi tabulkami probíhá pomocí primárních a cizích klíčů. Varianty kardinality byly popsány výše, v předchozí kapitole teoretické části této práce (Begg, Connolly, Widom, 2010).

3.5.2.2 Základní terminologie

Následuje několik základních pojmů relační databáze potřebných pro pochopení širšího kontextu této práce.

3.5.2.2.1 Atribut

Atribut je sloupec v relaci. Představuje vlastnost objektu reálného světa jako je například věk. Atributy jsou specifikovány mimo jiné jejich datovým typem. Mohou být číselné, textové, znakové, pravdivostní, data, čas atd. Dále se u tabulek specifikuje, jestli jejich vyplnění při tvorbě záznamu je mandatorní. Například primární klíč by mandatorní měl být vždy (Begg, Connolly, Widom, 2010).

3.5.2.2.2 Primání klíč

Primární klíč je unikátní identifikátor jednoho řádku v tabulce relační databáze. Může se jednat o jeden, nebo více řádků. Obvyklý způsob řešení je v podobě jednoho řádku s číselnou hodnotou, nebo s klíčem v podobě kombinace alfanumerických symbolů. Relační databáze je nastavena tak, aby vynucovala unikátnost každého řádku primárního klíče (IBM, 2021).

3.5.2.2.3 Cizí klíč

Cizí klíč je záznam, nebo sada záznamů v tabulce, který nese hodnotu primárního klíče jiného záznamu. Relační databáze vyžaduje, aby jeho hodnota byla shodná s hodnotou již existujícího primárního klíče. Tímto se propojují záznamy v relační databázi a určuje se kardinalita. (IBM, 2021).

Příkladem takového propojení může být osoba v registru vozidel a její vozidlo. Vozidlo bude mít jako jeden ze svých atributů cizí klíč fyzické osoby. Jelikož fyzická osoba může vlastnit více vozidel, může se stát, že více vozů bude mít jako hodnotu svého cizího klíče stejný primární klíč tatáž osoby.

3.5.2.3 Integrita dat databáze

Integrita dat je jejich celková bezpečnost, přesnost, kompletnost a ochrana v databázi. Spadá pod ní i následování legislativních nařízení jako GDPR. Integrity dat se dosáhne následováním předepsaných protokolů, standardů, nařízení a praktik při přípravě a tvorbě databáze. Důležitost dodržování těchto praktik spočívá v ochraně databáze proti ztrátě, odcizení či poškození jejích dat. Z hlediska integrity uvažujeme několik jejich druhů (Naeem, 2020).

V základu se integrita dělí na fyzickou a logickou. Ta fyzická chrání databázi proti přírodním pohromám, fyzickému poškozené, krádeži serveru či hackerům. Nejedná se o integritu implementovatelnou z hlediska relační či databázové teorie. Logická integrita pak již takto definována je a dělí se na doménovou, entitní a relační.

3.5.2.3.1 Doménová integrita

Doménová integrita omezuje a stanovuje přijatelné hodnoty atributů relací jednotlivých záznamů. Například aby v záznamu plat nebyla nečíselná hodnota.

Tomu dostává pomocí datových typů, kontrolou povolení NULL hodnot a pak výčtem či omezením povolených hodnot, nebo jejich rozsahu. Dále lze nastavit defaultní hodnotu.

3.5.2.3.2 Entitní integrita

Entitní integrita dohlíží tomu, aby každý záznam v relaci byl unikátní. Tomu dopomáhá unikátnost primárního klíče. Entitní integrita by měla svými nástroji obstarat unikátnost tohoto klíče. Dále by primární klíč v žádném záznamu neměl být nulový ergo nabývat hodnoty NULL.

3.5.2.3.3 Referenční integrita

Referenční integrita je soubor pravidel dohlížejících na správnost záznamů z hlediska jejich vztahů. Vztahy záznamů se vytváří pomocí cizích klíčů. Díky mazání v databázi tak může dojít ke sporům či chybějícím datům.

Tato integrita se snaží pomocí různých pravidel ošetřit chování databáze v případě změny či smazání záznamů, nebo celých tabulek.

Například mějme příklad kde vedeme domácí mazlíčky a jejich páníčky. Mazlíček má cizí klíč svého páníčka. Páníček si však přeje být z databáze vymazán a tak je jeho záznam odstraněn. V tuto chvíli máme minimálně jednoho mazlíčka s cizím klíčem neexistujícího záznamu což je situace vyvolávající chybu. Referenční integrita tuto situaci nastavenými pravidly ošetří. Nejzákladnější postup je nastavit defaultní hodnotu, záznam smazat, nebo povolit hodnotu cizího klíče NULL a vložit ji.

3.5.2.4 Normalizace databáze

Normalizace dat je sada postupů a pravidel pro změnu logického uspořádání dat a tabulek v relační databázi. Normalizace má několik forem, které jsou číselně odstupňovány od 0 s tím, že dodržování každé následující předpokládá dodržení i těch předchozích.

Dodržování těchto forem vede k udržitelnosti a přehlednosti dat v databázi. Snižuje, nebo přímo vylučuje jejich redundanci a to z hlediska záznamů i atributů. S databází lze po jejich správném užití lépe pracovat díky vyšší výkonnosti a rychlosti zpracování, vyhledávání či mazáni dat. Dále pak dochází k vyvarování se chybových scénářů.

Normalizaci definoval matematik Edgar Frank Codd po kterém jsou pojmenováno 13 Coddových pravidel.

3.5.2.4.1 Normálové formy

Normálové formy zaručují výše zmíněný stav databáze. Za normalizovanou databázi lze považovat databázi v 3. normálový formě, avšak i v praxi se vyskytují případy kdy se využijí vyšší formy.

3.5.2.4.2 0 Normálová forma

Nultá normálová forma je prakticky vždy automaticky vycházející ze samotné existence databáze a vyžaduje přítomnost alespoň jednoho atributu s jednou hodnotou.

3.5.2.4.3 1 Normálová forma

První normálová forma vyžaduje splnění té nulté a proto je občas nultá považována za její součást.

První forma vyžaduje, aby všechny atributy všech relací byly atomické, ergo dále nedělitelné. V případě existence takového atributu je atributu dále rozdělen na další atributy, nebo na další relaci.

Například uvažujme následující relaci. Osoba a její druhý atribut je celé jméno. Toto je nepraktické řešení a proto je třeba rozdělit atribut na atributy dva. Jméno a příjmení popř. třetí a to titul. Dále je klasickým příkladem relace zvíře evidované u veterináře. Z hlediska první normálové formy by bylo nesprávné u relace zvíře vést atribut majitel. Místo toho by měl majitel být samostatná tabulka spojena cizím klíčem.

3.5.2.4.4 2 Normálová forma

Druhá normálová forma vyžaduje, aby všechny neklíčové atributy byly plně závislé na celém primárním klíči. Forma říká na celém, jelikož primární klíč se může skládat z více atributů. Dále vyžaduje splnění 1NF.

Příkladem buď již následující tabulka. Zůstaňme u příkladu s veterinární klinikou. Představme si, že tabulka zvířete má následující atributy. Jméno majitele, příjmení majitele, číslo OP majitele. I přes atomičnost atributů jsou to atributy, které patří do oddělené tabulky pro majitele. Pro splnění 2NF je tedy správným řešením založení takové relace a naplnění jí příslušnými atributy, které budou v tabulce zvířete smazány (Upadhaya, 2019).

3.5.2.4.5 3 Normálová forma

Třetí normálová forma tak jako ty předešlé vyžaduje splnění všech předešlých normálových forem. Dále vyžaduje aby všechny atributy v tabulce byly vzájemně nezávislé. To znamená, aby jeden atribut nešel plně odvodit z atributu, nebo kombinace atributů jiných. Zde se jedná o příklad formy, která ulehčuje databázi na množství atributů a tedy i dat a zamezuje redundanci.

Například tabulka zaměstnanec mimo jiné bude mít atribut hodinová mzda. Dále bude mít atribut velikost úvazku v hodinách a měsíční mzdu. Měsíční mzda však jde vypočítat z velikosti úvazku a hodinové mzdy.

Dalším příkladem je u zboží uvádění ceny bez daně, velikost daně a pak celkové ceny.

V obou příkladech je řešením smazání přebytečného atributu.

3.5.2.4.6 Ostatní normálové formy

Po 3. normálové formě následuje Boyceho-Coddova normálová forma, 4, 5, 6 forma a další. Jejich detailnější rozbor již však není pro předmět této práce potřeba.

3.5.3 SQL

Structured Query Language, tedy jazyk strukturovaných dotazů je jazyk pro tvorbu dotazů nad sadou dat v relačních databázích. Vychází z jazyka SEQUEL, které mu dala vzniknout firma IBM v 70. letech. Účel vzniku SEQUEL by stejný jako je dnešní účel jazyka SQL.

Jazyk SQL má na práci vybírání, úpravu a vkládání dat v databázi. Forma jeho příkazů je velmi podobná mluvení angličtině, což bylo záměrem při jeho vzniku pro co možná nejsnazší učební křivku. Mnoho jeho příkazů jsou pojmenovány přímo podle činnosti či operace, kterou vykonávají a jejich sled se snaží napodobit logický sled v mluvené angličtině či logickém proudu myšlenek.

3.5.3.1 SQL jazyky

Jazyk SQL se skládá z několik sad pod-jazyků. Každý z těchto jazyků je jeho plnou součástí a má na starosti jiný díl práce nad daty uloženými v databázi. Základní 4 takové jazyky uvažujeme následující:

- DML, Data Manipulation Language, jazyk pro manipulaci s daty.
- DCL, Data Control Language, jazyk pro kontrolu/ovládání dat.
- DDL, Data Definition Language, jazyk pro tvorbu/definici dat
- TCL, Transaction Control Language, jazky pro kontrolu/ovládání transakcí (JavaPoint).

3.5.3.1.1 Data Manipulation Language

Tento jazyk slouží pro manipulaci a vkládání dat do jednotlivých relací. Výčtem slouží k vyhledání a výpisu dat, smazání, vložení a úpravě dat. Nutné je podotknout, že slouží k práci s daty a nikoliv samotnými tabulkami.

Jeho nejběžnější příkazy jsou následující:

- Select, vyhledává data

- Delete, maže data
- Insert, vkládá data
- Update, upravuje data

3.5.3.1.2 Data Control Language

Tento jazyk je zaměřen na práci s právy přístupu k datům v databázi. Má především dávat a odebírat práva.

- Grant, přidá práva
- Revoke, seber práva (JavaPoint).

3.5.3.1.3 Data Definition Language

Jazyk pro definici dat pracuje s relacemi a objekty databáze. Hlavním úkolem je tvorba nových a úprava již existujících relací a objektů. Dále existující tabulky a objekty upravuje. DDL lze například použít k nastavení doménové integrity pomocí omezení na jednotlivé atributy.

- Create, užívá se pro tvorbu objektů.
- Drop, smaže jmenovaný objekt.
- Alter, upraví určitý objekt.

3.5.3.1.4 Transaction Control Language

Transakce je množina jednoho nebo více příkazů v databázi. Transakce buď proběhne celá, nebo neproběhne žádná její část. Transakce musí splňovat 4 následující pravidla. V rámci transakcí můžeme používat běžné příkazy ostatních jazyků jako Create, Select, atd.

- Atomicita, transakce proběhne celá, nebo vůbec.
- Konzistence, data musí být po jejím proběhnutí v celé databázi konzistentní.
- Izolovanost, transakce a operace v ní proběhnuvší musí být od ostatních transakcí zcela nezávislé.
- Trvalost, efekt transakce musí být trvalý.

Pro práci s transakcemi v rámci jazyka TCL používáme především 3 následující příkazy.

- Commit, spustí příkazy v transakci.

- Savepoint, vytvoří jakýsi záchytný bod ke kterému se lze vrátit.
- Rollback, příkaz pro návrat k nějakému bodu stavu dat v databázi (w3resource, 2022).

3.6 MVC architektura

Je již záležitostí minulou kdy webové stránky byly pouhé statické sbírky obrázků, tabulek a tlačítek na jejichž tvorbu stačila znalost několika jazyků a HTML a CSS. Tyto jazyky jsou samozřejmě stále hojně využívány pro tvorbu frontendu, ale pro samotné stránky jako celek již z moderního hlediska samy nestačí. To neznamená, že by se takové stránky dnes netvořily, nebo pro ně nebyl prostor. Na mnoho účelů jako pouze informativní stránky, například základních škol, jsou tyto nástroje většinou dostačující. Avšak jeli třeba, aby stránka měl nějaké funkce, je již třeba hovořit o webových aplikacích.

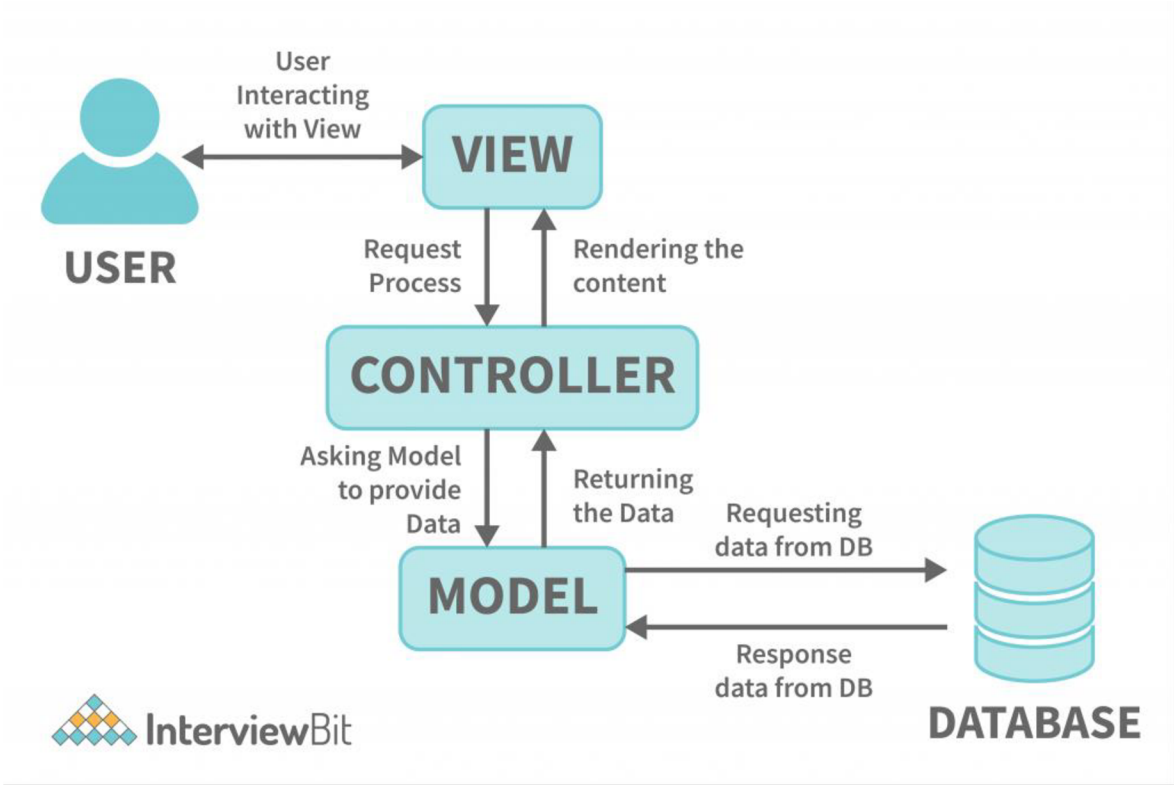
Pro tvorbu webových aplikací existují pravidla, standardy a postupy jako pro tvorbu každého softwaru. Mezi ty patří i různé architektury a návrhové vzory. Jedním takovým návrhovým vzorem je pak MVC architektura.

MVC je návrhový vzor architektského typu, teda vzor, který pomáhá řešit problém z hlediska architektury aplikace. Proto se jedná o jeden z komplexnějších a složitějších vzorů, které se neaplikují na jednu malou část kódu aplikace, ale na celou aplikaci jako logický celek.

MVC architektura není nijak omezena no konkrétní technologii a využívá se hojně například ve frameworcích jazyka PHP a ve frameworku ASP .NET a ASP .NET CORE.

3.6.1 Principy fungování MVC architektury

Zkratka MVC znamená Model-View-Controller. Rozděluje kód aplikace napříč backendem i frontendem na tři samotné logické celky. MVC architektura se především a většinou aplikuje v prostředí objektově orientovaného programování.



Obrázek 5 - Schéma MVC Architektury (InterviewBit, 2022)

3.6.1.1 Model

Model představuje abstrakci objektu reálného světa. V kódu je tvořen třídou. Tato třída pak nese své vlastnosti v podobě atributů. Tyto atributy mohou být jednoduché údaje jako věk a jméno, nebo listy, pole a jiné objekty dané aplikace.

Tyto třídy jsou pak pomocí mapování mapovány databáze, kde se ukládají data.

Aplikace užívající MVC architekturu nemusí mít všechny své modely jako součást tohoto vzoru. Běžnou součástí mohou být abstraktní třídy, třídy, které nejsou určeny pro mapování do databáze, rozhraní či třídy užité například jako součást Factory návrhového vzoru.

V případě objektového programování jsou součástí tříd i jejich metody. V případě MVC architektury se logika aplikace tvořená těmito metodami deleguje jinam.

3.6.1.2 Controller

Tím místem, kam se logika aplikace deleguje jsou z části kontrolery. Kontroler je třída která zpracovává požadavky uživatele. Obsluhuje data a ty předává dál na View. Každé View by

mělo mít pouze svůj jeden konkrétní kontroler přiřazený prostřednictví modelu. Tato třída obsahuje metody, které by běžně byly součástí třídy, s kterou tento kontroler pracuje. Kontroler dále využívá služeb a ostatních tříd, které nejsou modely v dané aplikaci. Dobrým pravidlem je, že kontroler by nikdy neměl využívat kontroler jiný ani přímo, ani nepřímo. Takový kód napsat lze, ale tvoří kód, který není čistý, bude nakloněný ke špatně opravitelným chybám a bude velice těžké ho testovat a správně odladit.

Důležitou součástí aplikace užívající MVC architekturu je router. Router umí rozpoznat jaký kontroler a jakou jeho metodu má zavolat a užít. Kontroler si pak pro svůj model získá data z databáze a ta dle dalšího kódu vykreslí uživateli v prohlížeči.

Kontrolery mívají pro některá View dvě metody náležící stejné funkcionalitě. Ty se dělí na Get a Post. Get metoda bere data z databáze, například pro zobrazení, a Post metoda zasílá data do databáze.

3.6.1.3 View

View je zobrazovaná část uživateli tohoto návrhového vzoru a lze ji přeložit jako zobrazení. Může se jednat o jednotlivé zobrazené stránky webové aplikace, nebo jejich částečná zobrazení. Každé zobrazení má přidělen jeden model a díky tomu aplikace ví, jaký kontroler má používat.

3.6.1.4 Viewmodel

Důležitým prvkem MVC architektury je existence viewmodelu. Pro potřeby vyobrazení na zobrazení je vždy třeba danému zobrazení říct, jaký model má užívat. Ovšem pak může nastat situace, kdy je třeba využití či zobrazení dat z více modelů. V praxi se jedná o velmi běžnou situaci.

Pro takový případ existuje speciální druh modelu, Viewmodel. Jedná se opět o jednoduchou třídu, do které se jako atributy nastaví potřebné atributy jiných tříd. Jaká data pak naplní model odeslaný na formulář zobrazení bude již mít na starosti konkrétní kontroler daného zobrazení.

3.7 ASP .NET

ASP.NET je firmou Microsoft vytvořený framework pro tvorbu webových a stránek a webových aplikací. Jedná se o framework, který je zdarma přístupný a funkční napříč platformami (Microsoft, 2022).

ASP.NET pro tvorbu webových aplikací užívá zejména jazyky C# a LINQ pro backend a HTML, JavaScript a CSS pro frontend.

Je postaven na vývojářské platformě .NET, která poskytuje knihovny a nástroje pro vývoj široké škály softwaru. .NET podporuje pro psaní svých aplikací jazyky C#, F# a Visual Basic (Microsoft, 2022).

3.7.1 ASP.NET Core

ASP .NET Core, nebo pouze .NET Core je vývojem frameworku ASP.NET. Je téže vyvinut firmou Microsoft a slouží k tvorbě online softwaru. Užívá z většiny stejných nástrojů a jazyků jako ASP .NET. Jedná se také o open-source zdarma použitelný framework, který je Microsoftem doporučován a v dnešní době nejvíce podporován. Většina vývoje na jejich straně je směřována právě k .NET Core, který díky tomu má již mnoho verzí a Microsoft tvoří stále nové (Roth, 2022).

3.7.1.1 ASP .NET Core Identity

Je API postavené pro správu a tvorbu uživatelských účtů v .NET aplikaci. Spravuje základní uživatelské rozhraní pro přihlášení, registraci a práci s účtem. Dále samotné přihlašování a registrace umožňuje. Má přednastavené kontrolery a třídy, které je posléze nutné namapovat do databáze, kam se účty a související objekty budou ukládat. Toto API také umožňuje přihlašování pomocí již existujících účtů na platformách jako Google a Facebook či Twitter.

Přednastavená rozhraní je možné upravit či kompletně přepracovat. Účtům lze přiznávat uživatelské role v kardinalitě m:n, čili jeden účet může mít víc rolí a jedna role může náležet více účtům.

Díky tomuto rozhraní lze nastavovat v aplikacích autorizaci, tedy přístup k jednotlivým kontrolerům a zobrazením (Anderson, 2022).

3.7.2 Scaffolding

Scaffolding je framework, který umožňuje generování kódu a základních funkcionalit v ASP.NET frameworku. Jedná se o běžně používaný nástroj pro ulehčení velkého množství práce. Lze pomocí něj vygenerovat třídy, kontrolery aj. a s nimi dále pracovat a upravovat je. Umí nadále vytvořit projekt se základními funkcionalitami, který lze dále využívat jako šablonu pro novou práci ve frameworku. Scaffolding je využitelný například při tvorbě rozhraní pro přihlašování a správu uživatelských účtů (Anderson, 2022).

3.7.3 NuGet

NuGet je nástrojem Microsoftu pro .NET framework určený pro správu balíčků. NuGet je velmi dobře ošetřený z hlediska správy jednotlivých balíčků napříč celým projektem. Stará se o nastavení jejich správné verze a dále o nastavení závislostí. Dalším z nástrojů správce NuGet je seznam referencí na užívané balíčky v projektu. Aby nebylo třeba na nových zařízeních pro stejný projekt instalovat balíčky znova, NuGet si v každém projektu umí držet seznam referncí pro nainstalované balíčky Stahované balíčky mohou být knihovny nástrojů, rozšíření, nebo vlastních modulů pro potřeby například týmu pracujícího na témže projektu (Microsoft, 2022).

3.7.4 C#

Programovací jazyk C# je objektově orientovaným jazykem. Jedná se o jazyk z rodiny jazyků C kde dalším příbuzným by byl jazyk C++. C# je přes svoji komplexnost a širokost poměrně snadno naučitelným jazykem i pro začátečníky. Jeho hluboké porozumění již tak snadné není, ale základy v něm se lze naučit i bez předchozí zkušenosti s vývojem. Z tohoto a dalších důvodů se tento jazyk stává často první volbou pro výuku nových programátorů v mnoha školních institucích (Albahari, 2020).

Jazyk je v dnešní době multiplatformní, což znamená, že aplikace psané v něm lze užívat na zařízeních různých operačních systémů a jejich užití není limitováno pouze pro Windows jako tomu bylo dříve. C# nachází své největší využití v tvorbě webových aplikací díky .NET frameworku, který v posledních letech nabírá na popularitě a jazyk tím oživil.

Následující kód je příkladem vytvoření dvou proměnných v jazyce C#. Int je číselná proměnná a string proměnná pro textové řetězce.

```
4   int cislo;
5   cislo = 10;
6
7   string slovo;
8   slovo = "Ahoj";
```

Obrázek 6 - Příklad založení a naplnění proměnných

Příklad vytvoření funkce, která přijme dva číselné argumenty a vrátí jejich součet.

```
    0 references
22  public static int Secti(int a, int b)
23  {
24      return a + b;
25  }
```

Obrázek 7 - Příklad funkce

Následný kód je ukázkou cyklu. Jedná se o tzv. for cyklus. Ten v tomto případě vypíše čísla od 0 do 9.

```
9
10      for (int i = 0; i < 10; i++)
11      {
12          Console.WriteLine(i);
13      }
```

Obrázek 8 - Příklad for cyklu

Další ukázka je příklad založení pole čísel s maximálním počtem prvků 10. Pole je následovně naplněno sudými čísli od 0 do 18 a vypsáno.

```
15      int[] poleCisel = new int[10];
16
17      for (int i = 0; i < poleCisel.Length; i++)
18      {
19          poleCisel[i] = i*2;
20      }
21
22      foreach (int cislo in poleCisel)
23      {
24          Console.WriteLine(cislo);
25      }
```

Obrázek 9 - Příklad práce s polem

Poslední ukázka je příklad vytvoření třídy Hat. Tato třída je oděděna od třídy Item a má tak její atributy. Dále má pak vlastní atributy BonusHitChance a BonusDefense.

```
namespace GhostTown.Models
{

    public class Hat : Item
    {

        public int BonusHitChance { get; set; }


        public int BonusDefense { get; set; }
    }
}
```

Obrázek 10 - Příklad tvorby třídy

Tato část byla vypracována podle (Albahari, 2020).

3.7.5 LINQ

LINQ je jazyk, nebo spíše jak sám Microsoft píše, *sadou technologií* (Microsoft, 2022), které dávají možnost integrace dotazovacích příkazů do běžného kódu jazyka C# v prostředí frameworku ASP.NET Core. LINQ využívá příkazů postavených na Lambda kalkulu (Microsoft, 2022).

Tyto příkazy jsou jednoduché krátké kusy kódu určené pro práci s daty. Většinou slouží pro hledání dat. Co se s daty pak následovně děje je již otázkou příkazu jehož součástí daný příkaz LINQ je. Jejich podoba se různí dle technologie v které jsou data uložena. Například data v relační databázi spravovaná SQL jazykem, nebo data v dokumentech XML (Microsoft, 2022).

3.7.6 Entity Framework

Entity Framework je open-source ORM framework. ORM znamená object-relational mapping, tedy objektově-relační mapování. Entity Framework je určený pro .NET frameworky (Entity Framework Tutorial).

V prostředí ASP.NET Core frameworku se dnes užívá Entity Framework Core. Jeho hlavní funkcí jsou migrace, které jsou samotným nástrojem mapování.

3.7.6.1 ORM

ORM je soubor technik pro přenos objektových tříd do podoby relací v relační databázi přidružené konkrétnímu projektu. Výhodou existence těchto technik je zaměření se programátora na modely své objektové aplikace bez nutnosti přemýšlení nad tabulkami v databázi. ORM tyto objekty přenese, tedy namapuje, včetně atributů a vazeb (Entity Framework Tutorial).

3.7.6.2 Migrace

Migrace je činnost EF Core, která mapuje objektové třídy aplikace na relační tabulky v databázi pro ukládání a práci s daty aplikace. EF Core aktivně sleduje stav modelů v aplikaci a při vytvoření migrace vytvoří sadu příkazů pro danou databázi a dá ji programátorovi k náhledu. Ten může migraci smazat, upravit, opravit, nebo spustit. Spuštění migrace pak změní schéma databáze.

Zde je třeba dbát Open/Closed principu SOLID metodik. Pakliže dojde k mutaci dat, EN Core si s vybranou mutací nemusí poradit a může ji namapovat špatně, což pak způsobí kompletní nefunkčnost databáze.

3.7.7 DbContext

Aby EF Core věděl jaké třídy má mapovat do databáze, existuje komunikační třída DbContext. V této třídě jsou uloženy záznamy jednotlivých tříd, které si programátor přeje uložit a namapovat do databáze.

Nadále tato třída slouží k nastavení některých pravidel databáze a například referenční integrity. Pomocí příkazů se v této části kódu nastaví jak pracovat s objekty v případě manipulace s objekty s nimiž mají vazbu.

Nadále zde nastavujeme například vlastního upraveného uživatele naší aplikace a tím i v databázi přepisujeme pro .NET framework nativního IdentityUsera vlastním.

Díky užívání ORM si programátor může myslet, že se obejde bez hlubší znalosti databází, což není pravda a databázi je stále nutné správně nastavit, jen tak lze učinit bez znalosti jazyka SQL.

3.8 Razor

Jako nástroj pro zobrazování zobrazení (View) uživateli ASP.NET Core používá nástroj zvaný Razor. Tím je značkovací syntaxe, která umožňuje užívat backendový jazyk jako C# v kódu webové stránky. Ergo je díky němu možné psát html značky a uprostřed nich napsat blok kódu v C#, nebo obráceně tak, aby tento blok pomáhal vykreslovat výslednou stránku.

Při spuštění požadované stránky server spustí kód stránky a přeloží ho pro .NET. Díky tomu může stránka operovat například s databází. Výsledný kód pak již umí aplikace zobrazit (W3Schools).

3.8.1 HTML

HTML, tedy HyperText Markup Language je značkovací jazyk říkající prohlížeči jak má vyobrazit zobrazované stránky uživateli. HTML má konečnou množinu značek díky kterým obsah zobrazuje. Velmi často je užíván v kombinaci s CSS, nebo JavaScriptem.

3.8.2 CSS

CSS, Cascading Styles Sheet je jazyk určený pro nastavení vzhledu vyobrazené webové stránky. Primárně se v něm pomocí konečné množiny příkazů nastavuje velikost a barva obsahu webové stránky.

3.9 Git

Jedno z velkých témat softwarového inženýrství a programování obecně je záloha a verzování kódu a aplikace. Při psaní jakéhokoliv kódu je důležité si práci ukládat a zálohovat a nikoliv pouze na lokálním disku.

V praxi se an vývoji většinou podílí celý tým, nebo větší počet týmů. Dohromady tak mnoho lidí pracuje současně na jednom projektu většinou i ve stejném zdrojovém kódu. Kód je tak třeba dělit na verze, zálohovat, kopírovat a vytvářet zkušební verze. Těmto výzvám a problémům odpovídá Git.

Git je zálohovací open-source systém pro správu verzí. Git má svojí vlastní sadu příkazů, které pro svoje funkce užívá.

3.9.1 Princip a funkce Gitu

Z jeho mnoha funkcí je jedna z nejdůležitějších právě ta na správu verzí a vytváření větví. Dále jsou hlavní výhody Gitu zálohování, usnadnění spolupráce, snadné sdílení kódu a možnost pracování z více koncových zařízení. V neposlední řadě pak není problém přidat do týmu nového vývojáře, nebo takové osobě stačí nasdílet repozitář s uloženým kódem v Gitu.

Verzování umožňuje ukládání aktuálního stavu zdrojového kódu do uložiště Gitu. Programátor se pak může k jednotlivým verzím vracet, nebo vyvolávat jejich podobu a tu upravovat. Tato funkcionalita předchází obtížím způsobeným napsáním špatného či aplikaci rozbíjejícího kódu. Pakliže programátor naprogramuje kus kódu, který není funkční, nebo uvádí aplikaci do nežádoucího stavu, pomocí verzování může kód vrátit na předchozí verzi a bez obtíží může pokračovat v práci.

Druhá důležitá funkce je větvení kódu. Uvažujme, že hlavní zdrojový kód je jakousi hlavní větví a do té by se nikdy nemělo přímo psát. Tato hlavní část se okopíruje do nové větve a v té se vyvíjí. Pak, když je nová větev a její účel splněn, otestován a je funkční se tato větev spojí s tou hlavní a ta pojme nový kód. Větve lze vytvářet z libovolných verzí kódu. V praxi pak každý programátor v týmu má svoji vlastní větev kódu, v které píše svůj kód. Po jeho dopsání je kód otestován a zkontrolován nadřízeným. Tvorba větví je praktická například z hlediska testování nových funkcionalit. Pakliže programátorská práce selže a kód s celou aplikací by byly nefunkční, větev lze smazat a vyvolat existenci nové a začít práci nanovo.

3.9.2 GitHub

GitHub je jeden z nejrozšířenějších poskytovatelů služeb Gitu. Jelikož se jedná o systém užívající Git, jeho funkce jsou totožné. Samotný GitHub přikládá některé své vlastní funkcionality.

Primárně tedy slouží ke sdílení, verzování a ukládání kódu.

3.10 RPG online hry

Zkratka RPG znamená role-playing game. Do češtiny se toto volně překládá jako hra na hrdiny a doslovný překlad by byl hra na role. Základ těchto her leží ve hře DnD (Dungeons and Dragons), což je hra na hrdiny určená k hraní skupinami skutečných lidí. Tato hra má celé svazky knih pravidel, ale ty si hráči mohou upravit dle svých hracích preferencí.

RPG hry jakožto software mají vždy několik prvků podobných. Hráč si vytvoří svoji vlastní postavu, jejíž mír customizace se odvíjí od dané hry. S danou postavou se pak pohybuje v herním světě, plní úkoly a nakupuje výbavu pro svoji postavu. Většina těchto her je postavená na nekonečné znovuhratelnosti.

Jendou z největších her tohoto žánru, která jej významně proslavila je hra World of Warcraft od studia Blizzard.

Obrázek 11 - Herní prostředí RPG hry World of Warcraft (Ketchuna, 2019)

3.10.1 Online prohlížečové klikací RPG hry

Specifickým sub-žánrem těchto her jsou pak online prohlížečové „klikačky“ jak se těmto hrám často přezdívá. Jedná se o hry, kde samotný herní požitek nespočívá v přímém pohybu po herním světe, jako u klasických her, nýbrž v ovládání postavy a jejího vývoje skrze zjednodušené textové a obrázkové rozhraní.

Hráč se tak s postavou nepohybuje fyzicky v herním světě a postavu neovládá klávesnicí. Většina činností se tak v těchto hrát odvíjí pasivně.

Tyto hry se pak hrají v drtivé většině v internetovém prohlížeči. Hráč si zde vytvoří účet a k němu přidruženou postavu. Většinou se hra hraje na více serverech, jelikož je určena pro velké množství hráčů. Hráč po založení postavy chodí na úkoly, které trvají předepsanou dobu reálného času, sbírá herní měnu a nakupuje za ní předměty, aby je ho postava byla silnější. Dále získává zkušenosti od kterých se odvíjí i jeho úroveň a tím i jeho síla. Dalším zásadním prvkem je utkávání se z ostatními hráči. Většina her tohoto žánru je si velmi podobná a poroto lze říct, že tyto mechaniky jsou pro naprostou většinu z nich společné.

Často tyto hry nabízí další herní prvky jako specifické lokace se silnějšími nepřáteli, různé pevnosti a další pro jednotlivé hry specifické herní prvky.

V neposlední řadě jednou společnou proměnnou všech těchto her je pak speciální měna. Ta se většinou dá získat v průběhu hry, ale v malém množství. Její funkce je pak užívání různých hráče zvýhodňujících činností a mechanik. Pointa je pak taková, že tuto měnu lze nakoupit za reálné peníze.

Tyto hry jsou prakticky vždy hratelné zdarma, avšak hráči kupující si tyto speciální měny jsou ve hře často znatelně zvýhodnění oproti hráčům, kteří si tak nepočínají. Úroveň tohoto zvýhodnění pak určuje jak moc je tato hratelná pro běžné hráče, kteří za ní peníze neutrácejí.

Trend těchto her začal mezi lety 2007 a 2010 a některé tyto hry mají velmi aktivní a rozsáhlou hráčskou bázi stále ještě dnes v roce 2022.

3.10.2 Významné hry v žánru

Následující hry patří k průkopníkům, nebo svou hráčskou bází stále významným členům tohoto žánru.

3.10.2.1 Gladiatus

Gladiatus byla jedna z prvních her tohoto žánru a patří k těm hrám, které slouží jako předlohy pro ostatní hry v žánru.

Hru vydala společnost Gameforge v roce 2007. Tato společnost má na svědomí mnoho her v žánru online adventur a RPG her a mnoho z těchto her je stále aktivně hráno dodnes.

Hráč se ocitá v roli gladiátora. Denně se může vydat na několik výprav, které mu vydělávají zlato a zkušenosti. Dále se může utkat s ostatními hráči. O svoji postavu se musí starat nákupem jídla pro její uzdravení a výbavou pro lepší statistiky.

Hra nabízí další herní prvky jako cestování do nových provincií, boj v podzemí se silnějšími nepřáteli atd.

Obrázek 12 - Menu postavy ve hře Gladiatus (Hernimag, 2010)

3.10.2.2 BiteFight

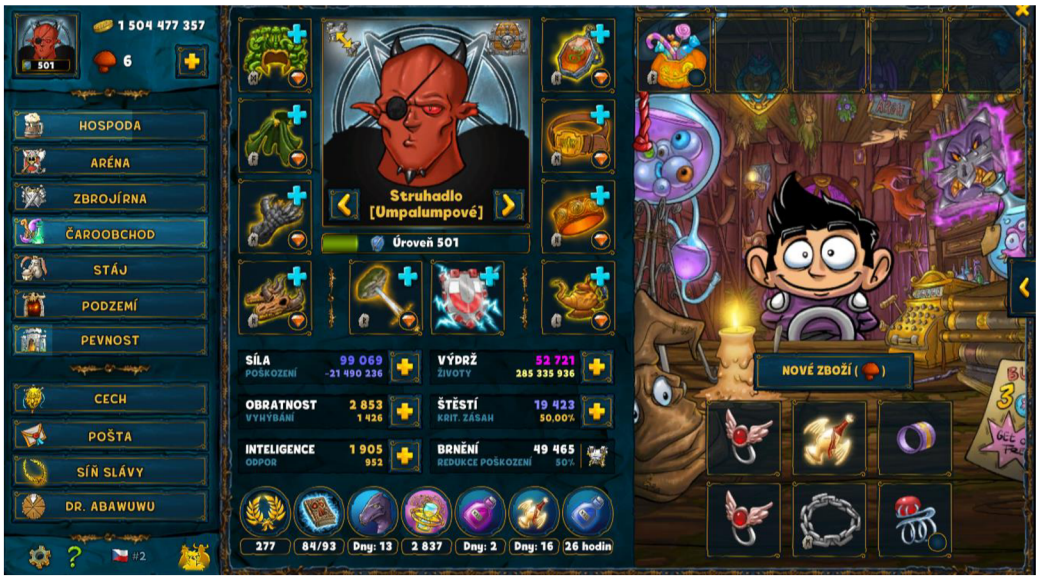
Ve hře BiteFight od společnosti Gameforge se hráč staví do role upíra nebo vlkodlaka. Tímto se hra liší od ostatních her, neb v soubojích mezi hráči se postavy napadají dle jejich rasy. Čili kromě tradičních prvků jako chození na úkoly, nákupu vybavení a budování hráčského cechu je zde hlavní hnací mechanika rivalita s opačnou rasou.

Obrázek 13 - Hlavní menu hry BiteFight (F2P.com)

3.10.2.3 Shakes and Fidget

Hra mechanicky okopírovala starší hru Tanoth napříč čemuž se stala mnohem populárnější a je hojně hraná do dnes. Jedná se o satirickou RPG online hru založenou na komiksech Shakes and Fidget, které parodovali monumentálně populární MMORPG hru World of Warcraft. Hra tak má svůj osobitý kreslený styl.

Ve své parodii však nekončí pouze u zmíněné hry. Její satiře neuniká ani herní řada Diablo, nebo filmová a knižní univerza Pána Prstenů, Harryho Pottera či Písně Ohně a Ledu v televizním podání známé jako Hra o Trůny.



Obrázek 14 - Obchod a přehled postavy ve hře Shakes and Fidget

4 Vlastní práce

Cílem této práce bylo primárně navrhnout a naprogramovat backend pro online prohlížečovou RPG hru s westernovou tématikou. Prototyp hry tak získala jméno Ghost Town dle písničky amerického country zpěváka Boba Wayna.

Ve skutečnosti však výsledný projekt může sloužit jako šablona pro jakoukoliv hru tohoto žánru. Stačí změnit sadu předmětů, změnit některé mechaniky soubojů a přejmenovat několik prvků a backend může dále sloužit i pro podobný projekt.

Tato část práce představuje průběh návrhu a tvorby prototypu backendu aplikace s implementací a testováním a představuje podobu výsledného projektu.

4.1 Mechaniky a princip hry

Pro hraní hry je třeba, aby se uživatel zaregistroval. Po registraci se může přihlásit a při prvním přihlášení si musí vytvořit postavu. S tou potom hraje. Hráč má postavu pouze jednu a té se v herním světě říká Gunslinger, čili pistolník. Se svojí postavou vydělává peníze chozením na úkoly a peníze pak utrácí za výbavu. S každým úspěšným úkolem také získává zkušenosti a tím i zvyšuje svoji úroveň a sílu.

Různé herní prvky mají jako jeden ze svých atributů obrázek. Ten je zde pouze textovým řetězcem. Jeho nastavení by pak bylo na frontend vývojáři. Důvod pro textový řetězec je takový, že ten v praxi bude odkazem na místo uložení obrázku. Jeho načtení a vyobrazení je pak záležitost frontendového vývoje.

4.1.1 Vybavení

Ve hře jsou 4 druhy různých předmětů. Každý pistolník může nosit přesně jeden kus od každého druhu předmětu.

Tyto předměty pak zvyšují statistiky hráčovi postavy.

Statistiky, které předměty přidávají jsou následující:

- Poškození
- Obrana

- Počet životů
- Šance na kritický zásah
- Šance na zásah

Následující atributy jsou pro všechny předměty společné.

Předměty mají svoji nákupní a prodejní cenu. Dále obrázek a požadovaná úroveň. Předměty zvyšují progresivně svoji sílu dle hráčovi úrovně. Hráči se v obchodě předměty s vyšší požadovanou úrovní než je ta jeho neobjeví. V poslední řadě pak mají název a typ.

4.1.1.1 Gun

Gun (střelná zbraň), je hlavním zdrojem poškození. Zbraň navyšuje poškození, šanci na kritický zásah a šanci na zásah. Zbraň kromě názvu má také typ, obrázek a prodejní a nákupní cenu.

Statistiky se pohybují rámcově dle typů zbraní.

Revolvery budou všeobecně nejlevnější, ale také budou mít průměrné statistiky. Pušky sice budou mít nejvyšší statistky vyjma poškození, ale jejich cena bude nevyšší. Brokovnice pak budou mít nejnižší šanci na zásah, ale nejvyšší poškození.

4.1.1.2 Hat

Klobouk zvedá hráčovi šanci na zásah a obranu. Obrana je spíše doplňková statistika, hlavním důvodem nákupu je zvýšení šance na zásah. Klobouky jsou všeobecně nejlevnějším kusem vybavením ve hře.

4.1.1.3 Clothing

Oblečení je hlavním obranným předmětem. Zvedá počet životů a obranu postavy. Je o poznání dražší než klobouk, ale cenou by neměl většinou na každé úrovní přesáhnout cenu zbraně.

4.1.1.4 Trinket

Cetka je divoká karta vybavení hráče. Jedná se o předmět který mění všech 5 bojových statistik hráče. Ovlivňuje tedy úroveň poškození, obrany, životů, šance na zásah a šance na

kritický zásah. Cetky mohou být skoro zadarmo, nebo velmi drahé. Jejich statistiky mohou být i záporné. Jejich role je přinést hře trošku náhodnosti co se vybavení týče. Ve hře mohou existovat cetky, které například zvednou poškození hráče o několik set, ale sníží jeho obranu na 0. Jejich cena se pak různí od síly jejich statistik.

4.1.2 Gunslinger

Pistolník má hráčem zadané jméno a statistiky z kterých se odvíjí jeho síla a schopnost plnit složitější úkoly. Dále nosí předměty které nadále navyšují jeho statistiky.

Statistiky pistolníka jsou následující:

- Jméno, to je zadáno uživatelem.
- Obrázek, obrázek je textový atribut uložený do databáze. Tvůrce frontendu by potom v tomto textovém řetězci měl uloženou adresu s uložištěm obrázku a načítal ho ze složky.
- Level (úroveň), se zvyšuje s načerpanými zkušenostmi. Čím vyšší je úroveň postavy, tím více je třeba zkušeností pro další novou úroveň.
- HitPoints (body života), jedná se celkové množství poškození, které je postava při souboji schopna utrpět, než souboj prohraje. Na začátku má každý pistolník 100 životů. Jejich počet se odvíjí od úrovně hráče a statistik poskytovaných nošenými předměty.
- Defense (obrana), je statistika která snižuje utržené poškození při soubojích. Defense je konstanta, která se odečítá od poškození. Pakliže ho převyšuje, poškození je vždy 1.
- Damage (poškození), je číselná hodnota určující množství čistého a o boranu protivníka neponíženého poškození uštědřeného soupeři při souboji za jeden úspěšný zásah. Výše poškození se odvíjí od úrovně hráče a statistik z jeho předmětů. Každá postava začíná s poškozením 10.
- Critical Hit Chance (šance na kritický zásah), je procentuální šance, která určuje šanci na zdvojnásobení poškození při úspěšném zásahu. Tuto statistiku zvyšují předměty.
- Hit Chance (šance na zásah), určuje přesnost postavy, čili šanci na úspěšný zásah nepřítele při souboji. Tuto statistiku zvyšují předměty.

- Experience Points (body zkušeností), určují počet zkušeností, které postava hráče právě má. Zkušenosti sbírá chozením na úkoly.
- Dollars (dolary), herní měna pro nákup vybavení.

Ve hře je záložka s přehledem hráčova pistolníka včetně jeho předmětů.

4.1.3 Shop

Shop je obchodním místem hry. Zde se každý den generují 4 nové předměty. Od každého typu předmětu jeden. Generace probíhá vždy při spuštění obchodu po zjištění, že je již nový den. Dále obchod negeneruje stejné předměty jako den před tím. Předměty se generují v závislosti na úrovni hráče.

Každý předmět v obchodě má svoji nákupní cenu. S nákupem mohou vzniknout tři scénáře.

- Hráč má dostatek peněz. Pak se mu odečtou peníze z konta a automaticky se předmět nasadí a starý předmět prodá za jeho prodejní cenu jež bude přičtena ke kontu dolarů.
- Hráč nemá peníze na předmět, ale po přičtení prodejní hodnoty jeho předmětu v daném slotu by na něj měl. Hráč tak může předmět nakoupit a prodat ten co má oblečený a tím nový předmět splatit.
- Hráč nemá dostatek financí. V tom případě mu hra řekne, že peníze nemá a hráč si daný předmět nekoupí.

4.1.4 Quest

Úkoly jsou zdrojem příjmů hráče. Úkoly jsou rozděleny do 4 kategorií. Podle kategorie se odvíjí jejich obtížnost, délka trvání i odměny. Úkoly trvají 5, 10, 15 a 20 minut. V každé kategorii je možné potkat určitý typ nepřítele.

Po přijmutí úkolu je hráč informován, kdy úkol skončí. Po jeho skončení proběhne souboj s náhodně vygenerovaným soupeřem. O jeho výsledku je vzápětí hráč informován.

4.1.5 Combat

V souboji je hráčova postava postavena proti nepřátelské entitě, které se většinou ve hrách říká mob. Souboj probíhá na kola. Každé kolo začíná pistolník výstřelem. Při výstřelu se spočítá šance na zásah. Pakliže zásah dopadne, spočítá se šance an kritický zásah. Od

vypočteného poškození se odečte soupeřova obrana a výsledný počet se odečte od bodů zdraví zasaženého. Jestliže životy nepřítele jsou vyšší jak 0, počítá se stejně i jeho pokus o zásah. Na konci kola proběhne kontrola jestli oba účastníci žijí. Pakliže ano, probíhá další kolo dokud jeden ze zúčastněných neklesne úrovní životů pod 1.

Bez ohledu na výsledek souboje, hráčova postava má v každém novém souboji opět všechno své zdraví zpět. Z logiky hry je neúspěšný souboj brán tak, že pistolník zraněný radši unikl.

4.1.6 Pub

V hospodě si hráč vybírá a plní úkoly. Při vstupu do hospody se mu vždy vygeneruje 7 úkolů pro každý den. Mechanismus opětovné generace funguje stejně jako u Obchodu.

Po zvolení daného úkolu a uběhnutí doby trvání úkolu probíhá souboj právě s jedním soupeřem. Při úspěšném zabití nepřítele jsou hráči přiznány odměny. Při prohře hráč nezískává nic.

Po dokončení ať již úspěšném či neúspěšném je daný úkol z hospody odstraněn. Po dokončení všech 7 úkolů již hráč v daný den na žádné další úkoly jít nemůže.

4.2 Přípravná fáze vývoje

Před tím, než započalo samotné programování aplikace bylo třeba připravit analýzu a jako její součást některé diagramy.

Analýza v případě této práce je přibližně 30 stránkový strukturovaný dokument obsahující popis postupu jednotlivých úloh a prací. Nejedná se o dokument psaný formálně, ale spíše v podobě poznámek a myšlenek autora, tudíž pro jeho silně neformální úpravu není vhodné ho přikládat jakožto přílohu této práce. Některé jeho části a skutečnosti je však vhodné zmínit.

Například skutečnost, že vždy nebyly některé části analýzy naplněny, nebo uskutečněny tak, jak bylo původně zamýšleno. Zkušenosti autora s užívanou technologií byly ze začátku prací o poznání slabší, než v průběhu, či ke konci. Tudíž se i metodika užitá při práci mohla změnit.

Dále při jednotlivých pracích autor vytvářel dílčí analýzy, které podrobněji rozepisovaly postup dané úlohy.

Funkční požadavky na aplikaci byly následující:

- Administrátorské rozhraní pro manipulaci s předměty
- Přehled pistolníka
- Přihlášení a registrace
- Nákup předmětů
- Chození a plnění úkolů
- Soubojový systém po dokončení úkolů
- Odměny za úspěšně splněné úkoly

4.2.1 Persony

Dále bylo třeba pro potřeby práce a testování vytvořit persony. Tedy ideální uživatele. Práce má dva typy uživatelů a to hráče hry a vývojáře. Nad projektem lze smýšlet jako nad hrou a tudíž cílový uživatel by byl hráč. Ovšem cílem projektu není vytvořit celistvou hru, ale backend část a ta je pak dále určena pro dalšího vývojáře, který s ní bude pracovat a na ní stavět.

4.2.1.1 Persona hráč

- Jméno: Karel Hravý
- Věk: 20 let
- Pohlaví: Muž
- Koníčky: Skládání modelů letadel, videohry, filmová kritika a historie

Informace: Karel je od dětství zapáleným hráčem videoher. Na střední škole trávil u her celé noci. Ale po úspěšném přijmutí na vysokou školu, kde studuje dějiny lidstva se setkává s problémem menšího množství volného času. Karel nemá zatím řidičské oprávnění a proto všude jezdí hromadnou dopravou.

Po vystudování školy by Karel chtěl dělat korekturu historické literatury a historických snímků.

Běžný den: Karel vstane ráno a jelikož má každý všední den školu, nasnídá se a vydá se do školy. Škola je poměrně náročná a proto se jí věnuje studiem před i po každý den. Ve volných chvílích kouká na seriály, nebo hraje s kamarády videohry. Poslední dobou objevil hraní her na telefonu v hromadné dopravě.

Důvod užívání aplikace: Díky nedostatku volného času a absenci vlastního dopravního prostředku se Karel ubral k hraní her na svém telefonu například v časech cestování hromadnou dopravou. Hra jako tato nevyžaduje vysokou časovou investici a proto pro něj představuje ideální řešení.

4.2.1.2 Persona vývojář

- Jméno: Jitka Kódová
- Věk: 32 let
- Pohlaví: Žena
- Koníčky: Programování, rybaření, box a četba

Informace: Jitka studovala na střední průmyslové škole programování a vývoj aplikací. Obor ji zaujal natolik, že se po úspěšně složené maturitě vydala na vysokou školu, kde nadále studovala ten samý obor. Školu po 5 letech úspěšně dokončila a získala tak inženýrský titul. Při svých studiích již pracovala jako vývojář junior. Po dokončení školy nastoupila na plný úvazek jako frontend vývojář. Po pár letech se však naučila i backend a stala se tak full-stack vývojářskou.

Ve volném čase zápasí v boxu a tak tráví spoustu času na tréninccích. O víkendech pak jezdí ráda rybařit se svým manželem. Během několika let by si přála stát se maminkou.

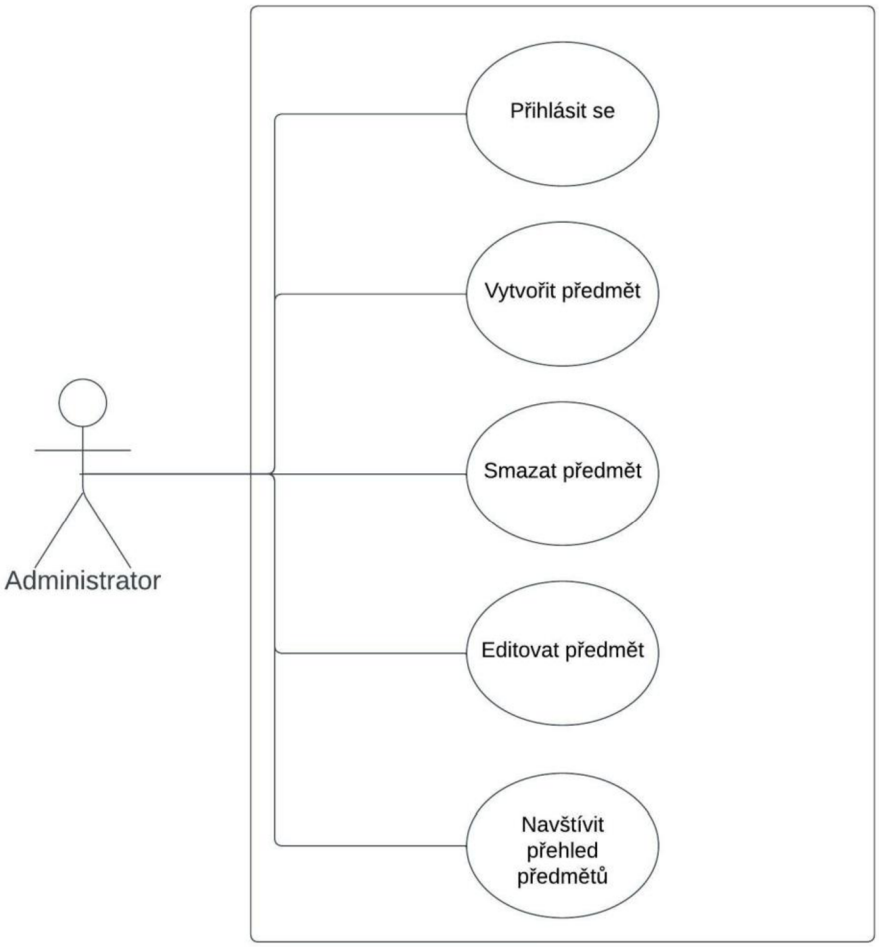
Běžný den: Jitka ráno vstane a vydá se do práce. Po pracovní době každý všední den trénuje. Večer tráví čas se svým manželem.

Důvod užívání aplikace: Jitka je sice full-stack vývojář, ale primárně se věnuje frontendu. Aplikace by jí tak sloužila jako jakási šablona, na které může dále stavět. Díky silným znalostem backendu by byla sama plně schopna aplikaci upravit a rozšířit dle svých představ.

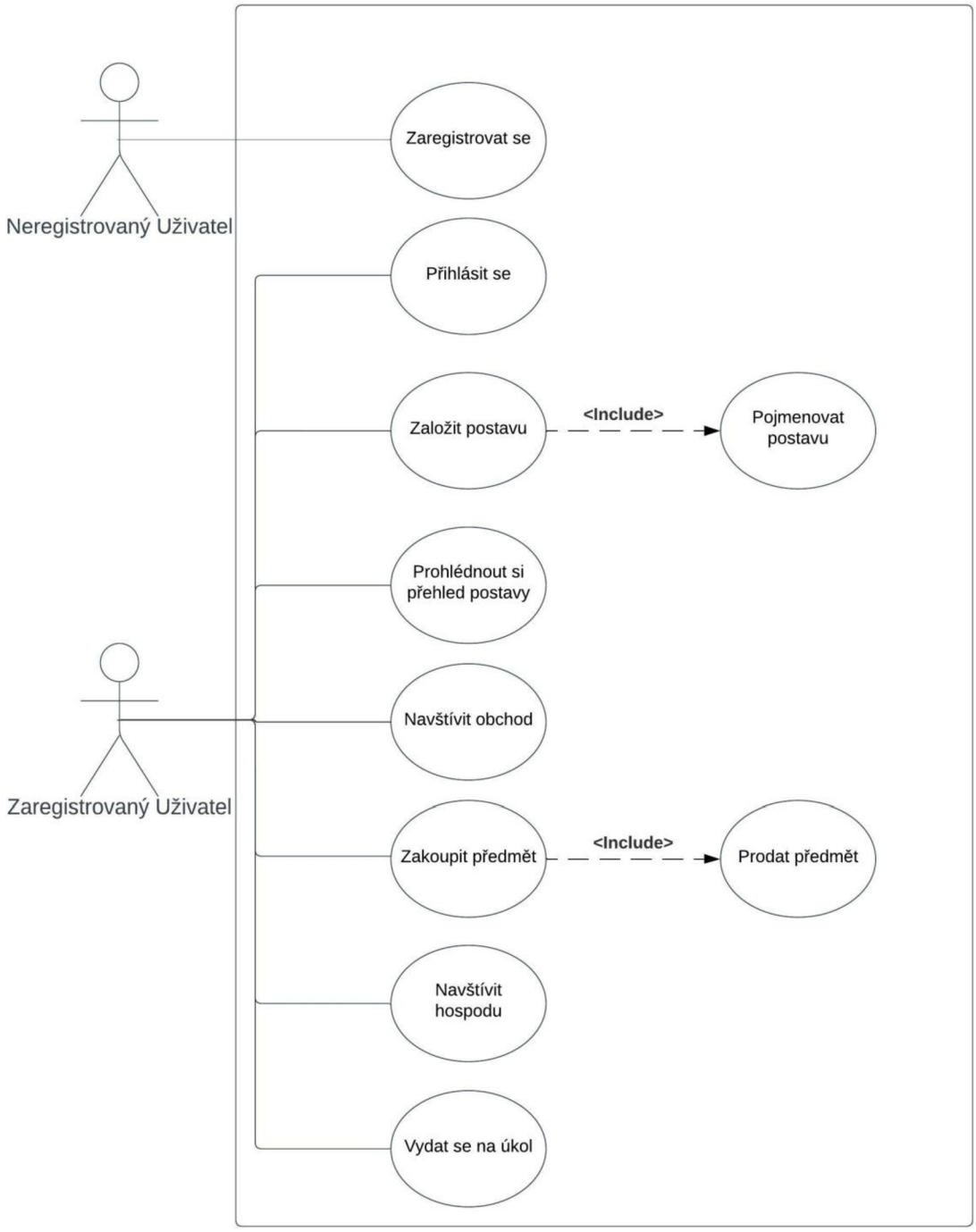
4.2.2 Diagramy užité pro práci

4.2.2.1 Use case diagram

Následují dva diagramy užití. První je z pohledu běžného uživatele a druhý je pak z pohledu administrátora. Oba diagramy by šlo spojit v jeden, ale pro lepší přehlednost a kompaktnost došlo k jejich rozdělení.



Obrázek 15 - Use case diagram z pohledu administrátora

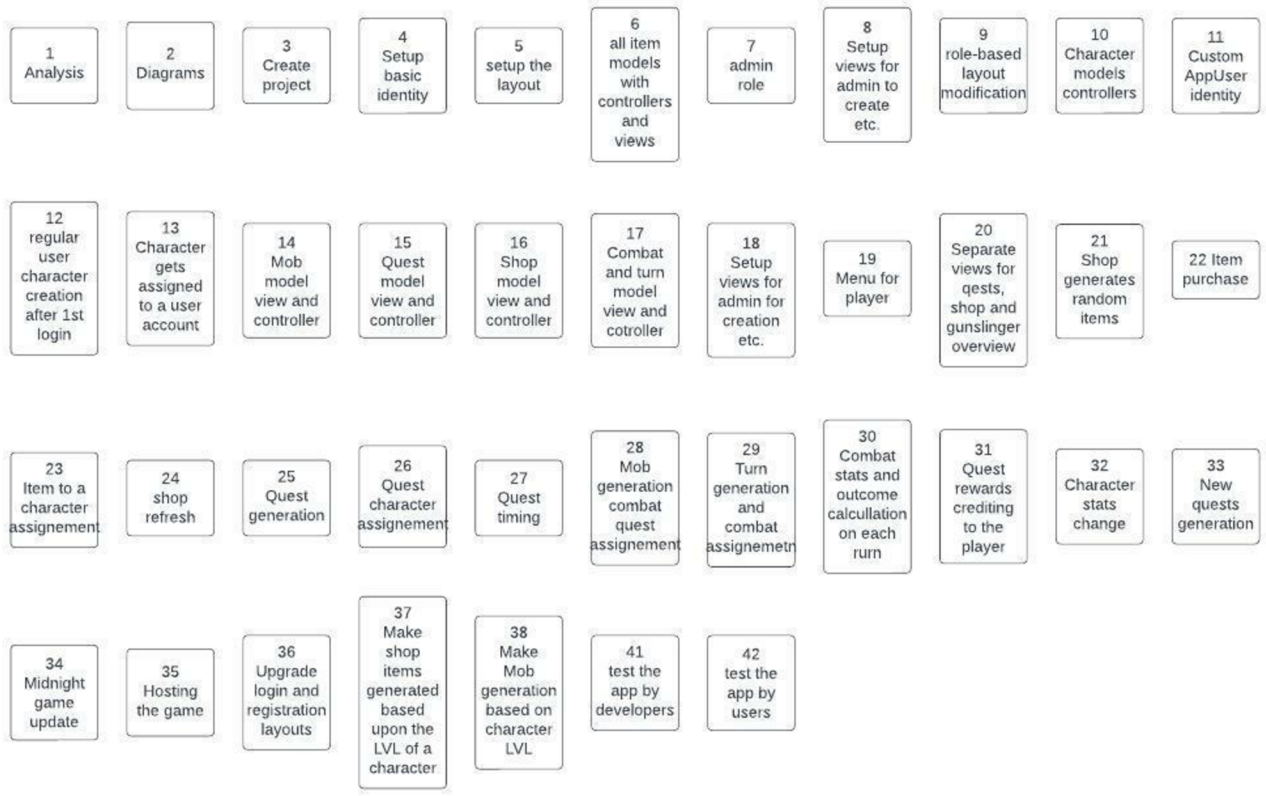


Obrázek 16 - Use case diagram z pohledu uživatele

4.2.2.2 Workflow diagram

První diagram, který bylo třeba vytvořit byl Workflow diagram. Ten zjednodušeně zachycuje postup prací a úloh, které je třeba uskutečnit.

Tento diagram by šel nahradit i jednoduchým seznamem, či tabulkou v Excelu. Jelikož se nejedná o rozsáhlý a složitý týmový projekt, kde je třeba uvádět například rozsahy prací, nebo jim přídržné zdroje, tak pro jeho potřeby postačí pouze zjednodušená verze.



Obrázek 17 - Workflow diagram

Nutno podotknout, že některé úlohy v tomto diagramu byly pozměněny v průběhu prací, ale nikdy zásadně či přímo vynechány.

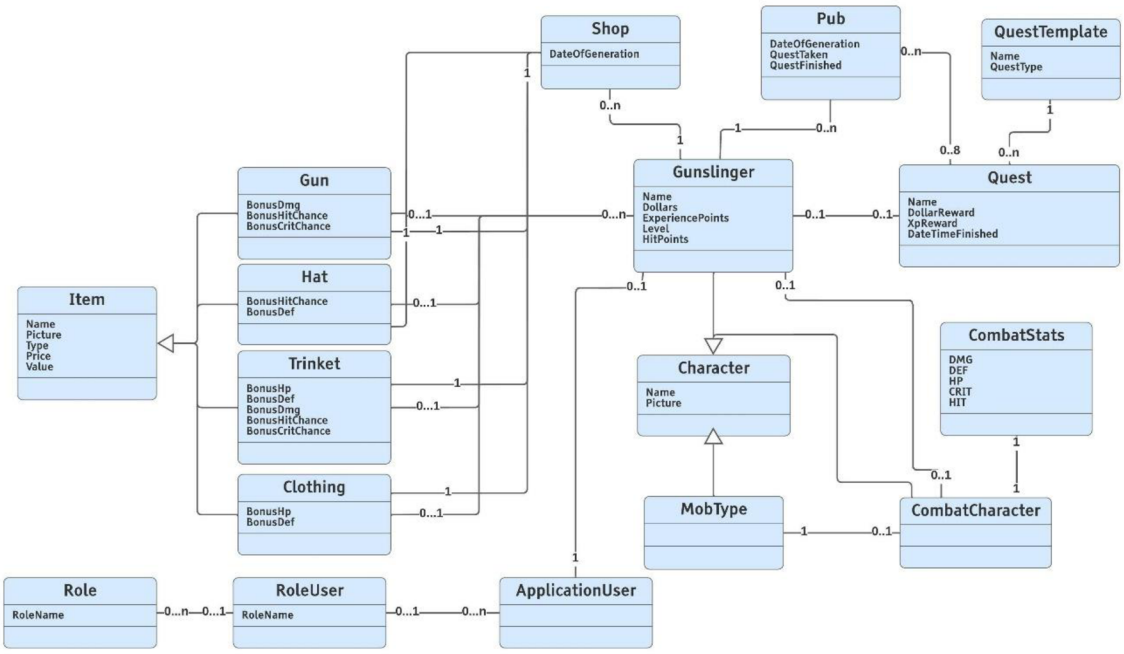
4.2.2.3 Konceptuální diagram

Dalším diagramem je diagram tříd. Tento diagram zachycuje rozložení tříd v aplikaci a jejich atributy. Důležitým prvkem je vyobrazení kardinalit, tedy to kolik instancí jedné entity může náležet entitě druhé.

V diagramu jsou zachyceny pouze aktivně využívané třídy, neb defaultně implementované rozhraní .NET Core Identity přináší velké množství tříd, které však nejsou součástí aktivní prohlížeče projektu. Jsou uloženy v knihovnách, které nejsou programátorovi hned přístupné. Při migraci se však do databáze propíší.

Tato práce využívá pouze vlastní poupravený model uživatele a roli pak nezměněnou.

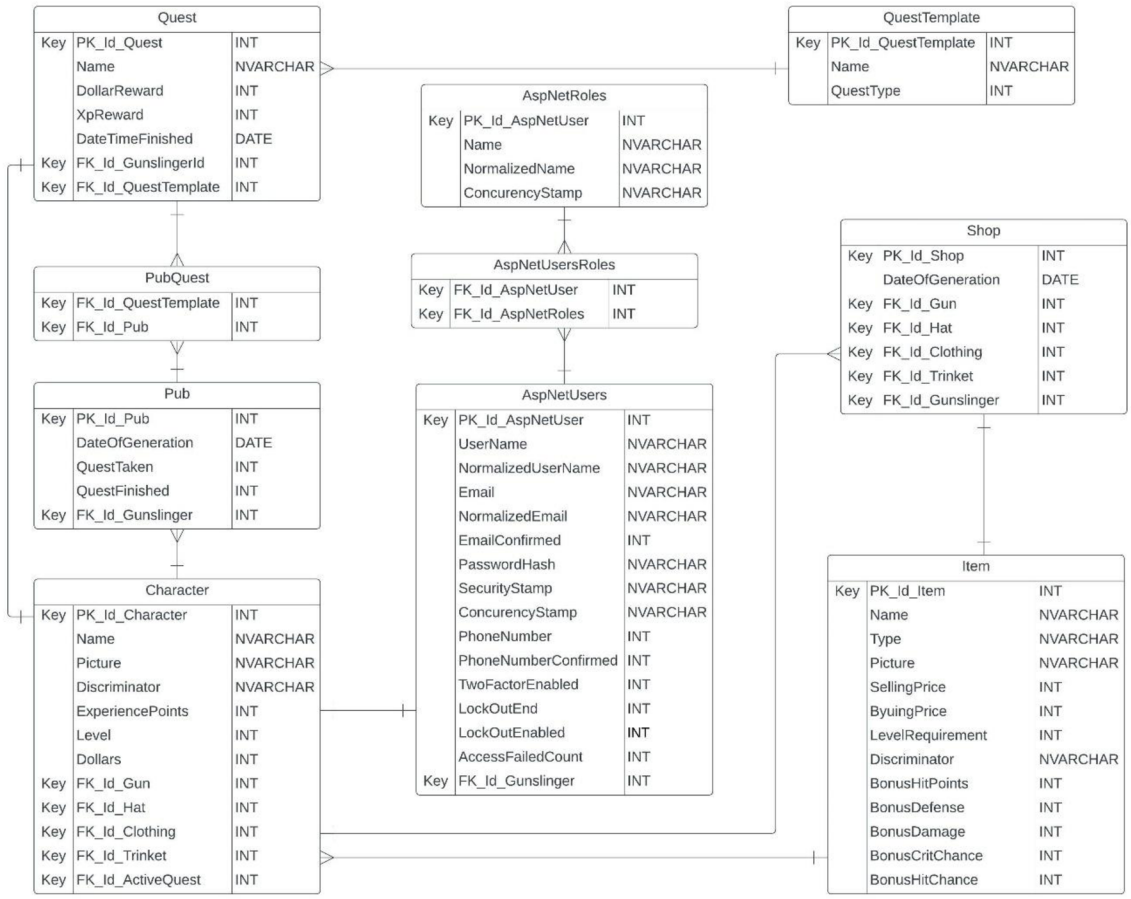
Ne všechny třídy v diagramu jsou součástí DbContext třídy a tudíž se v rámci migrací nemapují do databáze.



Obrázek 18 – Konceptuální diagram

Jelikož ne každá třída je vhodná pro mapování do databáze, nebo ne všechny údaje je třeba v databázi uchovávat, některé třídy jsou pouze pro aplikaci a jejich instance vznikají a zanikají v průběhu běhu aplikace.

4.2.2.4 Databázový (relační) diagram



Obrázek 19 - Databázový diagram

Databázový diagram nebyl navrhnut autorem práce, avšak je výsledkem práce mapování Entity Frameworku. Tudíž některé třídy z diagramu tříd zde zanikají a stávají se součástí sloučených tabulek. K tomu dochází v důsledku dědění.

4.3 Vývoj aplikace

Pro vývoj aplikace bylo zvoleno vývojářské prostředí Microsoft Visual Studio od firmy Microsoft. Konkrétně pak ve verzi community, která je zadarmo, ale její produkty by neměly být užity pro komerční účely.

Jako SŘBD byl užit Microsoft SQL Server Management Studio. Toto studio nebylo užíváno primárně pro tvorbu databáze, ale spíš pro kontrolu dat a přidávání položek do různých relací.

V neposlední řadě bylo třeba aplikaci uložit na uložiště služby Github, která umožňuje verzování a v případě potíží s vývojem snadný způsob jak aplikaci uvést zpět do funkčního stavu.

Při vývoji aplikace bylo přihlíženo k SOLID principům. Jejich využití pak záviselo na konkrétních implementacích. V některých případech by se držení sofistikovaných a mnohdy složitých postupů nemuselo vzhledem k rozsahu daných částí vyplatit. Tudíž hlavním principem, pro tuto práci byl princip O. Nadále byly využity návrhové vzory MVC, Factory a Singleton.

Samotný vývoj probíhal formou prototypu. Autor vždy naprogramoval dílčí v analýze připravenou část práce, kterou poté otestoval a připojil k celku. Vývoj probíhal vždy v oddělených větvích v prostředí GitHub. Každá větev po úspěšném naprogramování byla připojena k celku. Tento fakt však nevylučuje testování po dokončení vývoje. O tom také pojednává jedna z následujících kapitol.

4.3.1 Příprava modelů

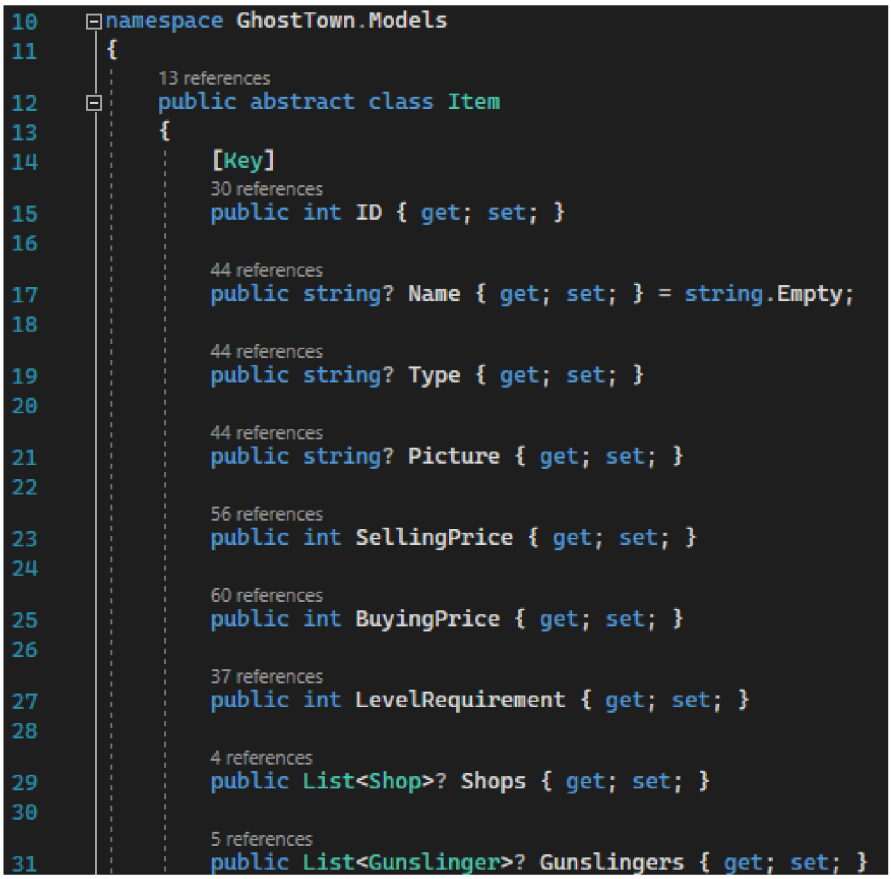
Jako první krok v programování práce bylo třeba připravit modely, které budou základem pro práci. Pro představu budou předvedeny nejdůležitější modely.

4.3.1.1 Abstraktní model Item

Prvním modelem je abstraktní třída Item, která slouží jakožto základ pro další 4 třídy, neb všechny variace předmětů od modelu Item dědí.

V kódu třídy jsou deklarovány všechny její atributy, které budou automaticky součástí všech jejích dětí, tedy oddděděných tříd. Dále vidíme, že u atributu ID je určeno, že se bude jednat o klíčový atribut.

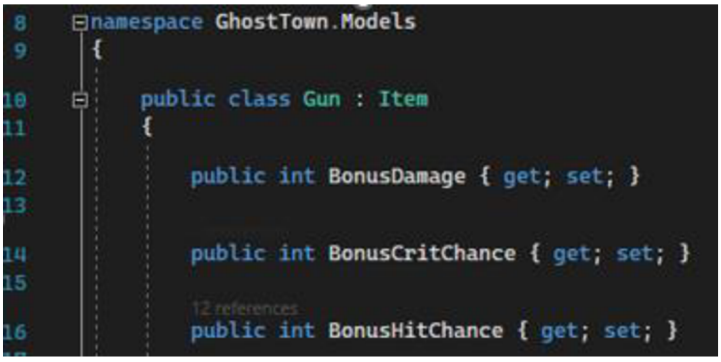
Poslední poznámka připadá na poslední dva atributy. Jelikož předměty nosí pistolník a mohou se objevit v obchodě, mají odkaz na předmět. Ten pak má list všech shopů a pistolníků, kteří jej nosí, aby vazba byla oboustranná.



Obrázek 20 - Kód modelu Item

4.3.1.2 Model Gun

Od třídy Item je oděděna třída Gun, tedy zbraň. Jedná se o jeden z předmětů, které hráčova postava může nosit. Předmět Gun tedy dědí vlastnosti předmětu Item a tudíž je již není nutné deklarovat. Stačí doplnit pro něj vlastní a již Itemu se netýkající vlastnosti. Ostatní běžné modely v kódu jsou již velmi podobné tomuto a není třeba jejich kód předvádět.



Obrázek 21 – Kód modelu Gun

4.3.1.3 Nedatabázový model Combat Character

Následuje model, který není přidán do DbContextu a tudíž nebude mapován do databáze. Jeho existence je však stále pro funkčnost aplikace důležitá. A jeho užití bude vysvětleno v následujících kapitolách.

Třída CombatCharacter užívá generiky, jelikož do ní může vstoupit více tříd oddděných od třídy Character. V případě této hry to je Gunslinger a MobType. Dále tato třída má vlastnost CombatStats, což je další nedatabázová třída.

4.3.1.4 Application User

Pro možnost vlastní modifikace uživatele a lepší práce programátora s ním byla vytvořena třída Application User. Jedná se o model založený na defaultním modelu .NET frameworku Identity User a tudíž je od něj oddděn.

Model má jako vlastnost pouze identifikaci Gunslingera. Aby identifikace byla kompletní, je třeba přidat ještě číselný atribut s klíčem Gunslingera.

4.3.1.5 Třída ApplicationDbContext

Tato třída slouží jakožto DbContext aplikace. Umisťují se do něj modely, které mají být namapovány do databáze. Dále se zde ošetřuje databáze například z hlediska relační integrity.

```
20          public DbSet<Item> Item { get; set; }
21
            12 references
22          public DbSet<Gun> Gun { get; set; }
23
            11 references
24          public DbSet<Hat> Hat { get; set; }
25
            12 references
26          public DbSet<Clothing> Clothing { get; set; }
```

Obrázek 22 – Ukázka modelů mapovaných do databáze

V další části kódu DbContextu je seznam všech tříd, které mají být mapovány do databáze.

```
base.OnModelCreating(builder);

builder
    .Entity<Gunslinger>()
    .HasOne(g => g.ActiveQuest)
    .WithOne(g => g.Gunslinger)
    .HasForeignKey<Quest>(q => q.GunslingerId);
```

Obrázek 23 – Ukázka kódu nastavení databáze

V poslední části této třídy jsou pak provedena nastavení databáze. Například zde je ošetřena relační integrita. Kdyby tak nebylo učiněno, při smazání předmětu by se smazal jej nosící pistolník a pak i účet k němuž náleží. Těchto omezení je v této části kódu více, ale všechna jsou velmi podobná, pouze týkající se jiných modelů.

Po naprogramování všech modelů dochází k migraci pomocí Entity Frameworku. Součástí migrací je pak i migrace, která nastavuje zvolený účet jako administrátorský.

Součástí praktiky O z principů SOLID je pak nemutovaní daných modelů. Jakékoliv následující migrace pak nesměly obsahovat změny již existujících atributů tříd, pouze přidávat nové. To nebylo ve všech částech vývoje dodrženo a proto musela být databáze několikrát kompletně smazána.

4.3.2 Služby

Služby, neboli Services jsou programátorem vytvořené třídy, které pomáhají přenést logiku některých funkcionalit do oddělené části kódu. Díky tomu je pak dodržen princip S z principů SOLID.

4.3.2.1 Combat Character Factory

Jednou z mnoha továren, které spadají pod návrhový vzor Factory je Combat Character Factory. Ta vytváří Combat Character, který nese statistiky určené pro boj. Jedná se o třídu, které se pak říká Service, tedy nějaká služba.

Metoda pro tvorbu výsledné kopie charakteru dostane na vstupu buď pistolníka, nebo MobType. Dle úrovně přihlášené postavy vygeneruje statistiky a vrátí kopii postavy. Ta je následně užita v další části kódu.

4.3.2.2 ShopFactory

```
public class ShopFactory : IShopFactory
{
    private readonly DateTime todayDate = DateTime.Today;
    private readonly IItemFetcher _itemFetcher;

    0 references
    public ShopFactory(IItemFetcher itemFetcher)
    {
        this._itemFetcher = itemFetcher;
    }

    2 references
    public Shop Create(Gunslinger gunslinger)
    {
        int gunslingerLevel = gunslinger.Level;

        var yesterdayShop = gunslinger.Shops?
            .FirstOrDefault(s => s.DateOfGeneration == todayDate.AddDays(-1));

        var gun = this._itemFetcher.FetchItem(gunslingerLevel, yesterdayShop?.Gun);
        var hat = this._itemFetcher.FetchItem(gunslingerLevel, yesterdayShop?.Hat);
        var clothing = this._itemFetcher.FetchItem(gunslingerLevel, yesterdayShop?.Clothing);
        var trinket = this._itemFetcher.FetchItem(gunslingerLevel, yesterdayShop?.Trinket);

        var shop = new Shop()
        {
            DateOfGeneration = todayDate,
            Gun = gun,
            GunId = gun?.ID,
            Hat = hat,
            HatId = hat?.ID,
            Clothing = clothing,
            ClothingId = clothing?.ID,
            Trinket = trinket,
            TrinketId = trinket?.ID
        };

        return shop;
    }
```

Obrázek 24 – Příklad kódu služby ShopFactory

Služba ShopFactory má pouze metodu Create, která získá pistolníka přihlášeného uživatele. Uloží si jeho předchozí obchod a začne chystat předměty. Dané předměty nesmí být ty samé, jako byly zobrazeny ve včerejším obchodě. Pak již vytvoří nový obchod, naplní ho předměty a vrátí.

4.3.2.3 ItemFetcher

Služba ItemFetcher dostane informaci o úrovni přihlášeného pistolníka a konkrétní včerejší předmět. Poté mu v metodě FetchItem nalezne náhodný předmět ze zadané kategorie příchozího předmětu, který odpovídá úrovni pistolníka a nebyl den před tím v obchodě. Předmět pak vrátí.

4.3.2.4 QuestGenerator

Služba QuestGenerator užívá připojení do databáze. Dále ve své metodě Create náhodně vybere jeden vzor úkolu questTemplate. Poté mu přiřadí pistolníka. Dle jeho úrovně a také typu úkolu, které jsou 4 mu nastaví odměny. Ty jsou větší s vyšším typem úkolu Tělo celého switche není třeba zobrazit, neb se jeho případy opakují. Na konci metoda vrátí vygenerovaný úkol.

4.3.2.5 PubFactory

Služba PubFactory generuje jako ShopFactory nové hospody a volá generaci jejich úkolů. Užívá aktuální čas a službu questGenerator, který jí nosí úkoly. Služba pouze vygeneruje novou hospodu a zaplní ji úkoly. Potřebuje k tomu pistolníka jemuž pro daný den hospodu přiřadí.

4.3.2.6 CombatFactory

Služba CombatFactory je volána při dokončení úkolu. Má metodu Fight, která počítá průběh souboje. Logika souboje se odehrává uvnitř cyklu. Ten běží dokud oba zúčastnění mají své zdraví výš než 0. Souboj začíná pistolník. Dle jeho šance na zásah je náhodně zvoleno jestli se trefí. Jestliže ano, je počítána stejně šance na kritický zásah, který zdvojnásobí způsobené poškození. Výsledné poškození je poníženo o obranu nepřítele. Jestliže má nepřítel vyšší obranu jak pistolník útok, utrží 1, nebo 2 body poškození. Jestliže protivník má po této části kola více jak 0 bodů života, stejnou logikou je počítán i jeho útok.

Jestliže cyklus skončí, přijde kontrola životů pistolníka. Jsou-li nad nulou, metoda vrátí pravdivostní hodnotu pravdu. Ta se v jiné části kódu interpretuje jako úspěšný souboj. Jestliže tomu tak není, vrácena je nepravda.

```
    if (mobHitPoints > 0)
    {
        if (random.Next(100) <= mobHitChance)
        {
            if (random.Next(100) <= mobCritChance)
            {
                if (gunslingerDefense >= mobDamage)
                {
                    gunslingerHitPoints -= 2;
                }
                else
                {
                    gunslingerHitPoints -= (2 * (mobDamage - gunslingerDefense));
                }
            }
            else
            {
                if (gunslingerDefense >= mobDamage)
                {
                    gunslingerHitPoints--;
                }
                else
                {
                    gunslingerHitPoints -= (mobDamage - gunslingerDefense);
                }
            }
        }
    }
}
if (gunslingerHitPoints > 0)
{
    return true;
}
else
{
    return false;
}
```

Obrázek 25 – Kód výpočtu výsledku souboje

4.3.3 Vývoj administrátorského rozhraní

Jako první část aplikace bylo vytvořeno administrátorské rozhraní. K tomu bylo třeba naprogramovat ke každému ze 4 předmětu kontroler. Jelikož jsou v tomto všechny předměty shodné, bude zde předveden pouze jeden z nich. V kontrolérů je vždy několik metod a ke každé z nich příslušné View, které se pak zobrazí uživateli.

4.3.3.1 Ukázka kódu pro administrátorskou obsluhu zbraně

Každý kontroler používá třídu DbContext pro komunikaci s databází.

```
[Authorize(Roles = "Admin")]
0 references
public IActionResult Index()
{
    IEnumerable<Gun> gunList = _db.Gun;
    return View(gunList);
}
```

Obrázek 26 - Metoda Index předmětu

Metoda index slouží k vyobrazení předmětů. Je zavolán list předmětů vytažených z databáze z tabulky Gun a předám do View k vyobrazení.

Následuje metoda pro tvorbu předmětu. Get metoda pouze zobrazí dané View a post metoda dostane poskládaný předmět a ten uloží do databáze.

```
//GET - CREATE
[Authorize(Roles = "Admin")]
0 references
public IActionResult Create()
{
    return View();
}
//POST - CREATE
[HttpPost]
[ValidateAntiForgeryToken]
[Authorize(Roles = "Admin")]
0 references
public IActionResult Create(Gun obj)
{
    _db.Gun.Add(obj);
    _db.SaveChanges();
    return RedirectToAction("Index");
}
```

Obrázek 27 - Create metoda předmětu

První metoda dostane klíč konkrétního předmětu a zjistí, jestli existuje zadaný klíč. Dále vrátí View s daným předmětem. Post metoda pak opět zjišťuje existenci. Pakliže předmět existuje, smaže ho a uloží stav databáze.

Následují metody editace pro daný předmět. Tyto metody jsou velmi podobné metodám pro mazání předmětu.

V Get metodě je opět nutné zjistit, zdali klíč vůbec existuje. Jestliže ano, je dle něj nalezen předmět a ten je zobrazen na View.

Metoda Post potom příjme poskládaný předmět, který může, nebo nemusí být upravený a uloží ho do databáze.

```
//GET - EDIT
[Authorize(Roles = "Admin")]
0 references
public IActionResult Edit(int? id)
{
    if (id == 0 || id == null)
    {
        return NotFound();
    }
    var obj = _db.Gun.Find(id);
    if (obj == null) return NotFound();
    return View(obj);
}
//POST - EDIT
[HttpPost]
[AutoValidateAntiforgeryToken]
[Authorize(Roles = "Admin")]
0 references
public IActionResult Edit(Gun obj)
{
    _db.Gun.Update(obj);
    _db.SaveChanges();
    return RedirectToAction("Index");
}
```

Obrázek 28 - Edit metoda pro předmět

4.3.4 Vývoj Gunslingera

První částí uživatelské části aplikace bylo nutné naprogramovat samotnou herní postavu.

4.3.4.1 Gunslinger Controller

Logika hlavních ovládacích prvků je opět napsána v kontroléru postavy.

V konstruktoru jsou opět uvedeny některé služby, které kontrolér užívá. První je DbContext pro komunikaci s databází. Následuje UserManager, který je nativním manažérem pro obsluhu přihlášeného uživatele. Díky němu je například možné zjišťovat stav přihlášeného uživatele, nebo mít přístup k jeho postavě. Poslední služba je autorem naprogramovaná CombatCharacterFactory. Její logika je vysvětlena v předchozí kapitole. Pro připomenutí,

její funkce je tvorba jakési kopie existující postavy v databázi doplněné o statistiky užité v boji. Zde jsou později používány pro výpis informací o pistolníkovi.

Důležitá je především metoda pro tvorbu nového pistolníka. Tato metoda dostane již vyzbrojeného pistolníka. Toho pak pošle do CombatCharacterFactory a ta vrátí Combat Character, který již nese všechny potřebné informace.

```
//GET - CREATE
[Authorize]
0 references
public IActionResult Create()
{
    return View();
}
//POST - CREATE
[Authorize]
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public IActionResult Create(Gunslinger obj)
{
    var userId = this.userManager.GetUserId(this.User);
    var user = this._db.Find<ApplicationUser>(userId);

    obj.Picture = "x";
    obj.Level = 1;
    obj.ExperiencePoints = 0;
    obj.Shekel = new Random().Next(0, 15);
    obj.Shops = new List<Shop>();
    obj.Pubs = new List<Pub>();


    _db.Gunslinger.Add(obj);
    user!.Gunslinger = obj;
    _db.SaveChanges();
    return RedirectToAction("Index");
}
```

Obrázek 29 - Metoda Create pistolníka

Metoda dostane daného pistolníka. Zjistí přihlášeného uživatele, dotvoří potřebné statistiky a přiřadí uživateli pistolníka. Vše je pak uloženo do databáze a hráč je již přesměrován na výpis své postavy.

4.3.5 Vývoj Shopu

Dále byl vytvořen obchod pro nakupování předmětů.

4.3.5.1 Shop Controller

V konstruktoru kontroléru obchodu jsou užity již známé služby. Dále je zde volána služba shopFactory, která opět bude vysvětlena později. Její funkce je tvorba obchodu a plnění ho předměty. Dále je zde volána funkce .NET frameworku pro určování aktuálního času.

Kontroler obsahuje metodu Index, která slouží k výpisu obchodu uživateli.

Metoda Create a některé další části tohoto kontroléru užívají ViewModel, který obsahuje kolektivní data o obchodu, pistolníkovi a předmětech. Jeho užití je nutné, neb rozhraní obchodu zobrazuje instance více tříd.

```
public IActionResult Create()
{
    var userId = this.userManager.GetUserId(this.User);

    var user = this._db.Users
        .Include(u => u.Gunslinger)
        // if this bugs just change it to [g.Shops]
        .ThenInclude(g => g == null ? null : g.Shops)
        .FirstOrDefault(u => u.Id == userId);

    var gunslinger = user?.Gunslinger;

    if (gunslinger is null)
    {
        return this.View(new ShopGunnerItemViewModel());
    }
    var shops = gunslinger.Shops?
        .Where(s => s.DateOfGeneration == this.dateToday)
        .ToList();

    if (shops is null or { Count: 0 })
    {
        return CreateAndAssignShop(gunslinger);
    }
    if(shops is { Count: > 1 })
    {
        return RemoveRedundantShops(gunslinger, shops);
    }
    var singleShop = shops.First();

    return this.View(
        new ShopGunnerItemViewModel
        {
            ClothingId = singleShop.ClothingId,
            Gun = this._db.Gun.Find(singleShop.GunId),
            Hat = this._db.Hat.Find(singleShop.HatId),
            Clothing = this._db.Clothing.Find(singleShop.ClothingId),
            Trinket = this._db.Trinket.Find(singleShop.TrinketId)
        });
```

Obrázek 30 - Metoda Create obchodu

V první části zjistí přihlášeného uživatele a získá jeho pistolníka s jeho obchody. Dále mu vrátí jeho aktuální obchod pro dnešní den. V následující části je zajištěna generace nového obchodu každý den. Tato metoda využívá dalších privátních metod, které jsou součástí kontrolérů. Jestliže navrácený obchod neexistuje, je zavolána metoda, která vytvoří nový obchod a přiřadí ho pistolníkovi. Jestliže došlo v databázi k poškození dat a chybě, která zapříčiní existenci více obchodů daného pistolníka v jeden den, je volána metoda pro jejich redukci. Pakliže pistolník již obchod má, je vrácen ViewModel, který obchod zobrazí.

```
public IActionResult BuyItem(int? id)
{
    var userId = this.userManager.GetUserId(this.User);

    var user = this._db.Users
        .Include(u => u.Gunslinger)
        .FirstOrDefault(u => u.Id == userId);

    var gunslinger = user?.Gunslinger;
    if (gunslinger is not null)
    {
        gunslinger.Gun = this._db.Gun.Find(gunslinger.GunId);
        gunslinger.Hat = this._db.Hat.Find(gunslinger.HatId);
        gunslinger.Clothing = this._db.Clothing.Find(gunslinger.ClothingId);
        gunslinger.Trinket = this._db.Trinket.Find(gunslinger.TrinketId);
    }
    if (id is null || id == 0)
    {
        return NotFound();
    }
    var item = this._db.Item.Find(id);
    if (item is Gun gun)
    {
        if (item?.BuyingPrice <= gunslinger?.Shekel)
        {
            gunslinger!.Gun = (Gun)item!;
            gunslinger.Shekel -= (item!.BuyingPrice - gunslinger!.Gun.SellingPrice);
            this._db.SaveChanges();
            return RedirectToAction("Create");
        }
        else if (item?.BuyingPrice <= (gunslinger?.Shekel + gunslinger?.Gun?.SellingPrice))
        {
            return RedirectToAction("Sell", new {ID = id});
        }
        else
        {
            return RedirectToAction("NotEnoughMoney");
        }
```

Obrázek 31 - Metoda Buy obchodu

Metoda nákupu získá přihlášeného pistolníka a tentokrát i jeho oblečené předměty. Ve své hlavičce získala klíč kupovaného předmětu. Dále pak vyhledá daný předmět a ptá se, jakého je druhu. Daný kód se pak opakuje pro každý typ předmětu. V ukázce je pouze kód pro zbraň. Kód se zeptá, jestli má hráč dostatečné finanční prostředky. Jestliže ano, starý předmět nahradí novým a změní stav konta peněz uživatele. Jestliže nikoliv, ale po prodání aktuální výbavy by již peníze byly, metoda odkáže hráče na View s touto informací a možností prodeje. Jestliže hráč nemá ani peníze ani s možným prodejem, je mu oznámeno, že nemá dostatek financí. Dále je přítomna metoda na prodej, která prodá aktuální předmět a provede změny ve financích postavy.

4.3.6 Vývoj Pubu

Jako poslední klíčová část aplikace k vývoji bylo prostředí hospody s úkoly.

4.3.6.1 Pub Controller

První důležitá metoda kontroléru je metoda Create. Ta spočívá ve stejné logice jako metoda vytváření obchodu. A tedy v tom, že na základě z databáze získaných dat o přihlášeném pistolníkovi vytvoří pro daný den novou hospodu s úkoly. Jestliže pistolník již hospodu má, pouze zobrazí tu existující. Jestliže dojde k chybě s existencí více hospod, je volána metoda pro jejich redukci. Metody jsou prakticky totožné s jejich protějšky u kontroléru obchodu.

Nejdůležitější je pak metoda UndertakeQuest. Tou se přihlášený pistolník vydává na úkol. Zároveň se jedná o metodu nejdelší. K metodě se váže View, které zobrazuje průběh souboje, nebo jeho výsledek. Metoda dostává klíč daného úkolu z View. Ten však dostane pouze při zadání úkolu. V tento moment danou instanci úkolu obohatí o všechny potřebné informace a uloží do databáze pro budoucí užití. V případě dalšího vstupu na tuto stránku je hráči na View vrácena tato instance se všemi již získanými informacemi. Pakliže čas úkolu vyprchal, spustí se logika volání souboje. Zde se opět získají informace o úkolu a pistolník je zbaven aktivního úkolu. Dále je hospoda informována o dokončení úkolu, tedy lze opět vybírat nové úkoly. To není při probíhajícím úkolu možné. Dále jsou připraveny CombatCharactery a je vygenerován dle délky úkolu příslušný typ nepřítele. Dále metoda pomocí CombatCharacterFactory dotvoří příslušné účastníky boje. Ty pak pošle metodou Fight do CombatFactory a ta vrátí výsledek. Dále je třeba vytvořit instanci ViewModelu, který je opět kombinací spousty atributů, které jsou třeba pro výsledné zobrazení. Její vlastnosti jsou nadále naplněny. V poslední části jsou v závislosti na výsledku souboje přidány odměny. Pro přidání zkušeností je volána metoda GiveXp. Dále je zobrazen výsledek boje ViewModelem.

Metoda GiveXp dostane zkušenosti získané v úkolu a příslušného pistolníka. Pak mu zkušenosti přičte. Jestliže zkušenosti přetečou přes s každou úrovní vyšší mez, získá pistolník novou úroveň a přebytečné zkušenosti jsou rekurzivně kontrolovány, kdyby měly přidat další úroveň. Skutečně přebytečné zkušenosti jsou potom pistolníkovi připsány a ty staré vynulovány.

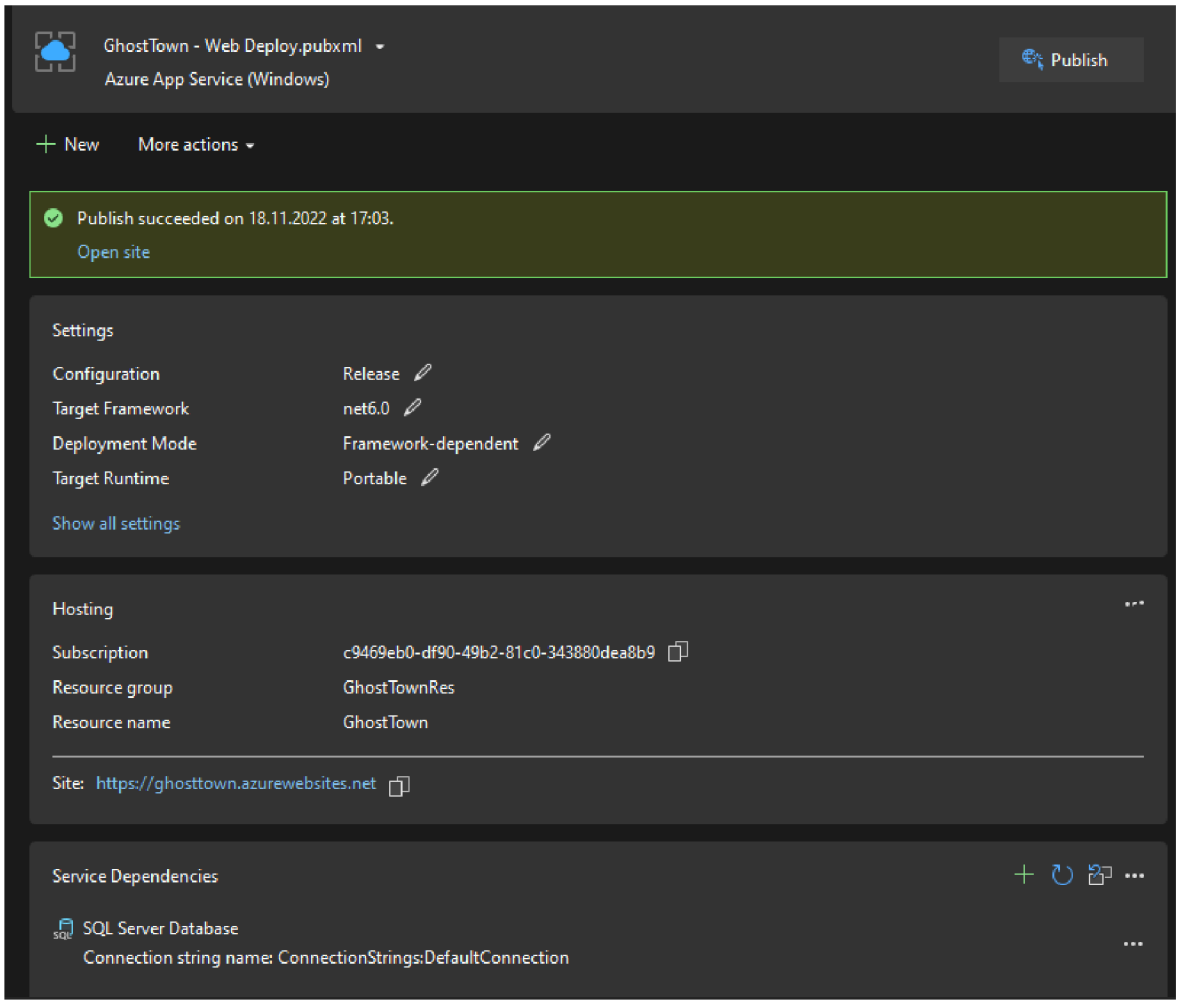
```
2 references
private void GiveXp(int xp, Gunslinger gunslinger)
{
    gunslinger.ExperiencePoints += xp;

    if (gunslinger.ExperiencePoints >= (gunslinger.Level * 65))
    {
        gunslinger.Level++;
        gunslinger.ExperiencePoints = (gunslinger.ExperiencePoints - (gunslinger.Level * 65));
        int overXp = (xp - gunslinger.Level * 65);
        if(overXp > 0)
        {
            gunslinger.ExperiencePoints = 0;
            GiveXp(overXp, gunslinger);
        }
    }

    this._db.SaveChanges();
}
```

Obrázek 32 - Metoda GiveXp

4.4 Nasazení aplikace

Aplikace byla nasazena na cloudový portál Azure. Toto vyžadovalo, aby si autor založil vlastní účet na této platformě. Poté existovali dvě možnosti, jak aplikaci nasadit. Auto se rozhodl pro nasazení ze samotného vývojového prostředí. Zde může vývojář zvolit možnost *publish*, nastavit potřebné parametry nasazení a pak z přehledového okna aplikaci zveřejnit a nasadit na server Microsoftu.

Tyto parametry jsou zvolení předplatného v rámci kterého aplikaci nasazuje, dále místo, kde jsou uloženy zdroje, server a potřebnou výpočetní paměť.



Obrázek 33 - Nasazení aplikace z vývojového prostředí

4.5 Testování aplikace

Testování aplikace v první řadě probíhalo při samotném vývoji. Autor vždy připravil jednu samostatně funkční část v jednotlivé větvi GitHubu a tu podrobil několika testům funkčnosti. Zejména pak co se týče správného fungování funkcionalit a mechanik hry, ale také vstupů, kde takové vstupy jsou.

Takové testování probíhalo bez scénářů a nebylo příliš dlouhé. Vždy šlo spíš o ověření funkčnosti napsaného kódů. Pakliže nově naprogramovaná část odpovídala předpokladům, byla připojena ke zbytku kódu. Následovala analýza nové části, vytvoření nové samostatné větve v Gitu a celý cyklus pokračoval znovu dokud kód nesplňoval na začátku vztyčené požadavky.

Po dokončení aplikace bylo testování rozděleno do dvou samostatných logických celků.

4.5.1 Testování vývojáři

V prvním se jednalo o testování jinými vývojáři, kteří kódu a vývoji rozumí. Tito vývojáři měli možnost do kódu nahlédnout v Gitu a pak aplikaci podrobit několika testům.

Průběh testů je popsán pomocí scénářů. Jejich správnost pak byla ověřována buď samotným výsledkem v aplikaci, nebo kontrolou údajů v databázi.

4.5.1.1 Testovací scénáře

Pro potřeby testování byla vytvořena sada testovacích scénářů. Testovací scénáře jsou označeny jako TS (testovací scénář) + číslo.

4.5.1.1.1 TS01

Název: Registrace

Prerekvizita: Připojení k internetu a funkční webový prohlížeč.

Kroky testu:

- Spuštění aplikace.
- Kliknutí na tlačítko registrace
- Zadání mailové adresy a dvakrát hesla
- Potvrzení registrace

Očekávaný výsledek: Úspěšná registrace

4.5.1.1.2 TS02

Název: Přihlášení

Prerekvizita: Úspěšná registrace

Kroky testu:

- Kliknutí na tlačítko přihlášení
- Zadání přihlašovacích údajů
- Potvrzení přihlášení

Očekávaný výsledek: Úspěšné přihlášení a přesměrování na přehled postavy.

4.5.1.1.3 TS03

Název: Vytvoření postavy

Prerekvizita: Přihlášení

Kroky testu:

- Zadání jména postavy
- Potvrzení vytvoření

Očekávaný výsledek: Úspěšní vytvoření postavy a zobrazení jejího přehledu.

4.5.1.1.4 TS04

Název: Přijmutí úkolu

Prerekvizita: Vytvoření postava a přihlášení

Kroky testu:

- Kliknutí na hospodu
- Zvolení kýženého úkolu
- Potvrzení zvolení a podstoupení úkolu

Očekávaný výsledek: Přesměrování na přehled probíhajícího úkolu

4.5.1.1.5 TS05

Název: Dokončení úkolu

Prerekvizita: Přijmutí úkolu

Kroky testu:

- Kliknutí na hospodu
- Při probíhajícím úkolu zobrazení informací o jeho průběhu
- Při dokončení přesměrování na přehled informací o dokončeném úkolu

- Návrat do hospody, nebo na přehled pistolníka

Očekávaný výsledek: Úspěšné a správné zobrazení výsledků

4.5.1.1.6 TS06

Název: Nákup předmětu

Prerekvizita: Přihlášení a existující postava

Kroky testu:

- Kliknutí na obchod
- Výběr chtěného předmětu
- Zvolení předmětu
- Potvrzení nákupu

Očekávaný výsledek: Správná změna financí na kontě hráče a změna nesených předmětů.

4.5.1.1.7 TS07

Název: Pokus o vstup na tvorbu postavy po vytvoření postavy

Prerekvizita: Přihlášení s existující postavou

Kroky testu:

- Kliknutí na profil postavy
- Kliknutí na řádek pro zadání URL adresy
- Připsání za aktuální adresu /Create
- potvrzení

Očekávaný výsledek: Přesměrování na profil a nezobrazení možnosti tvorby postavy

4.5.1.1.8 TS08

Název: Správné zvýšení úrovně a výpočet aktuálních zkušeností

Prerekvizita: Jít s postavou na úkol a úspěšně jej dokončit

Kroky testu:

- Zvýšení odměn za splnění úkolu
- Kliknutí na hospodu
- Zvolení úkolu
- Zobrazení výsledků úkolu
- Kontrola správnosti aktuálních zkušeností a úrovně

Očekávaný výsledek: Správně přičtené úrovně a upravené zkušenosti

4.5.1.1.9 TS09

Název: Pokus o nákup vybavení s nedostatkem financí

Prerekvizita: Přihlášení s existující postavou

Kroky testu:

- Změna financí na 0 dolarů
- Kliknutí na obchod
- Zvolení nového předmětu
- Pokus o jeho nákup
- Kontrola potvrzení aplikací, že hráč nemá dostatek financí

Očekávaný výsledek: Odkázání na stránku s informací o nedostatku financí

4.5.1.1.10 TS10

Název: Odhlášení

Prerekvizita: Přihlášení s existující postavou

Kroky testu:

- Kliknutí na tlačítko odhlášení

Očekávaný výsledek: Úspěšné odhlášení a zobrazení přihlašovací obrazovky

4.5.1.2 Samotné testování vývojáři

První bylo třeba otestovat funkčnost všech funkcionalit hry. Oba vývojáři byli seznámeni s naplánovanými mechanikami hry. Vytvořili si vlastní postavu a s ní testovali funkčnost nákupů a chození na úkoly.

Pro účely testování byla zkrácena doba úkolů na jednotky vteřin. V první řadě testů se vše zdařilo úspěšně a díky průběžnému testování vše fungovalo. Nákup předmětů i plnění úkolů bylo funkční.

Pro druhou dávku testování bylo třeba zvýšit možné odměny za jednotlivé úkoly. Zejména pak odměny na poli zkušeností. Pakliže zkušenosti pistolníka přesáhnou určitou úroveň, je třeba mu přiznat novou úroveň a jeho aktuální zkušenosti ponížit o správné množství.

Pro tento případ byla zvýšena zkušenostní odměna za plněné úkoly stonásobně. To odhalilo nedostatek v kódu, které způsoboval nesprávné přičítání úrovní a stav, kdy pistolník měl záporné zkušenosti. Bylo třeba přepracovat metodu na přidávání odměn. Díky rekurzy se cyklilo odčítání zkušeností a způsobovalo to záporné zkušenosti. Do kódu byly přidány podmínky pro kontrolu stavu zkušení a metoda byla upravena. Následující testy již tuto chybu vyloučily a následující testy potvrdili správnou funkčnost metody a tedy i přičítání zkušeností.

Další sada testování se zabývala logickým pohybem po aplikaci. Při přesání URL adresy bylo možné se dostat do nesprávných částí aplikace. Například bylo možno navštívit znova tvorbu pistolníka, i když profil hráče již pistolníka měl. Pro tento případ byla do kódu přidána kontrola existence pistolníka na profilu hráče. Pakliže hráč již postavu má, bude při vstupu na tuto adresu okamžitě přesměrován zpět na svůj profil a postavu tak nebude moci vytvořit.

Po odhalení těchto nedostatků proběhlo poslední testování a vše již fungovalo tak jak bylo zamýšleno.

4.5.2 Testování uživateli

Z hlediska logického využití takovéto aplikace, která představuje backend, není hráč takové hry cílovým uživatelem. Tím, jak již bylo výše vysvětleno, je spíše vývojář, který potřebuje funkční backend na jehož základě postaví frontend a finální softwarový produkt.

Ovšem i tak bylo třeba aplikaci podrobit testování uživateli, neb takový uživatel se většinou v aplikaci nepohybuje s cílem zjistit její nefunkčnost a tak se chová zcela intuitivně a přirozeně. Tento intuitivní a přirozený pohyb po aplikaci testujícímu vývojáři může chybět, a proto je třeba produkt dát k otestování i běžnému uživateli, který s vývojem nemá zkušenosti.

Autor práce hraje dlouhá léta hru Shakes & Fidget a je součástí jednoho z předních cechů, který sdílí s dalšími 49 hráči. Tyto hráče prezentoval s žádostí o otestování hry.

Testování se zúčastnilo 16 nahlášených spoluhráčů a je možné že i několik, kteří se testování zúčastnili, ale nenahlásili se.

Pro potřeby testování hráči již nebylo třeba vytvářet nové testovací scénáře, neb stačí znovu využít některé z těch, co posloužili pro testování vývojáři.

4.5.2.1 Průběh uživatelského testování

Pro účely uživatelského testování již byly odměny a časy úkolů nastaveny do normy.

Každý hráč byl vyzván k vytvoření vlastní postavy a pak hraní hry, jak by jí hrál přirozeně. Hráči chodili na úkoly a s našetřenými financemi nakupovali předměty. Většina hráčů hru hrála v průběhu jednoho odpoledne.

Jelikož toto testování probíhalo až po vývojářském testování, hráči neodhalili žádné nedostatky, co se týče funkčnosti hry.

Měli však připomínky k možným vylepšením. Nejčastější byl požadavek na tvorbu frontendu. Byla jim však vysvětlena podstata projektu a s pochopením testovali dále. Další velmi častý požadavek byl na arénu, kde by se hráči mohli vzájemně utkávat v soubojích. Dále pak se hráči ptali na možnost inventáře, kam by si odkládali nepoužívané předměty. Poslední za zmínku vhodný návrh hráčů pak bylo přidání druhé herní měny, která by se získávala za speciálních podmínek, nebo by ji bylo možno zakoupit. Ta by hráče zvýhodňovala a například by mu umožňovala přeskakovat čekání na dokončení úkolu.

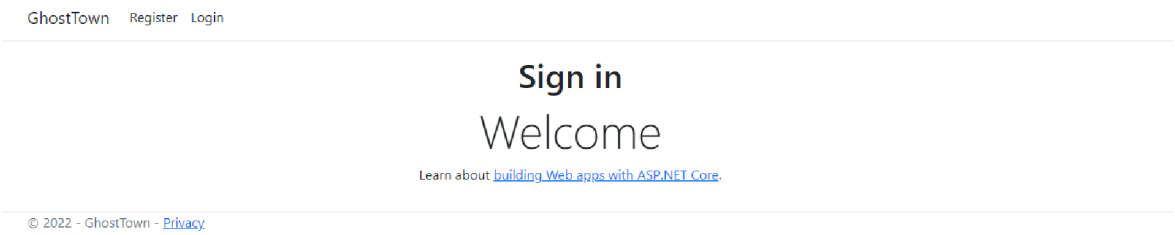
4.6 Výsledný projekt

Předmětem této části je ukázka, jak výsledný prototyp hry vypadá ze strany uživatele.

Je nutné mít na paměti, že se jedná o backendový projekt, nikoliv frontendový. Výsledný vzhled je tedy velmi prostý a jeho existence je čistě pro snazší ovládání projektu. Výsledný projekt by ve své nejhrubší podobě mohl být sadou knihoven. Ovšem pro ukázku jeho funkčnosti a lepší možnosti testování byla zvolena tvorba nejzákladnějšího uživatelského rozhraní.

4.6.1 Úvodní stránka

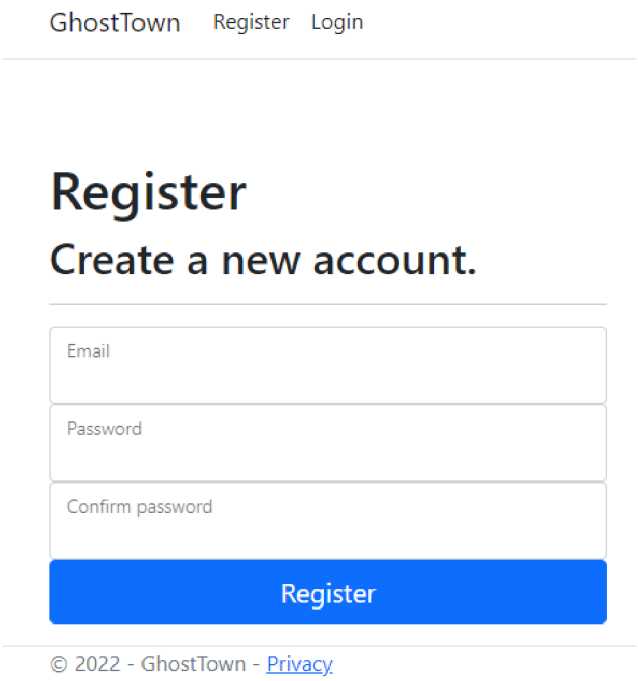
Úvodní stránka bez přihlášení vyzívá uživatele k přihlášení.



Obrázek 34 - Úvodní obrazovka

4.6.2 Registrace

Registrace je prováděna prostřednictví zadání mailové adresy a hesla. Projekt umožňuje přidání modulů pro přihlašování pomocí účtů na vybraných sociálních sítích či přes platformu Google. Někteří vývojáři však takové přihlašování nemusí chtít podporovat, a tak je taková možnost ryze na vývojáři, který by tento backend používal pro další vývoj hry.

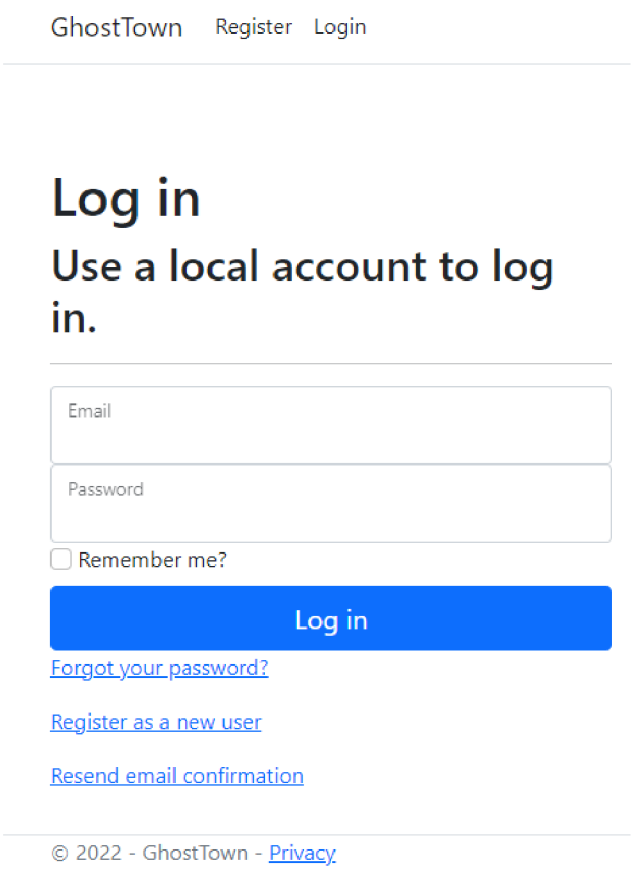


Obrázek 35 - Registrace uživatele

Při registraci zadá uživatel svoji mailovou adresu, heslo a jeho potvrzení. Po kliknutí na tlačítko registrace potvrdí svoji mailovou adresu a může se přihlásit do aplikace.

4.6.3 Přihlášení

Pro přihlášení musí uživatel zadat potvrzený mail a své heslo.

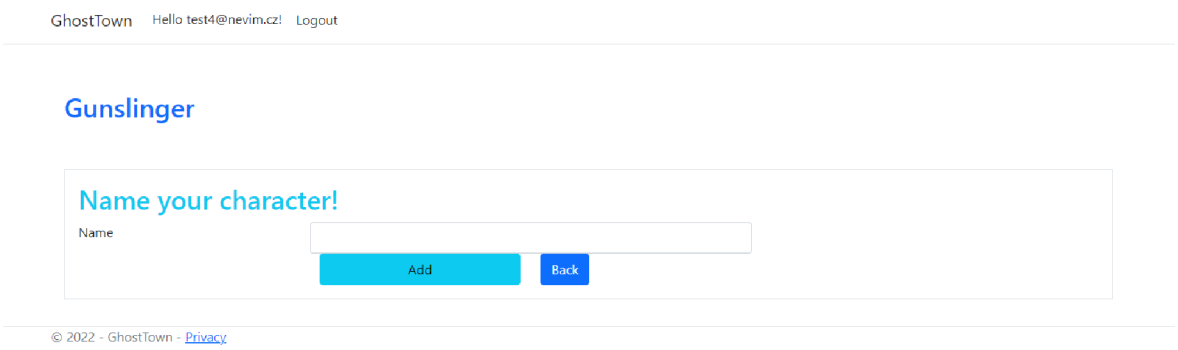


Obrázek 36 - Přihlášení uživatele

Po přihlášení se uživatelský zážitek různí. Postup je rozdílný pro první přihlášení a každé následující.

4.6.3.1 První přihlášení

Při prvním přihlášení si uživatel musí vytvořit svoji postavu. Na to stačí zadat její jméno.



Obrázek 37 - Vytváření postavy

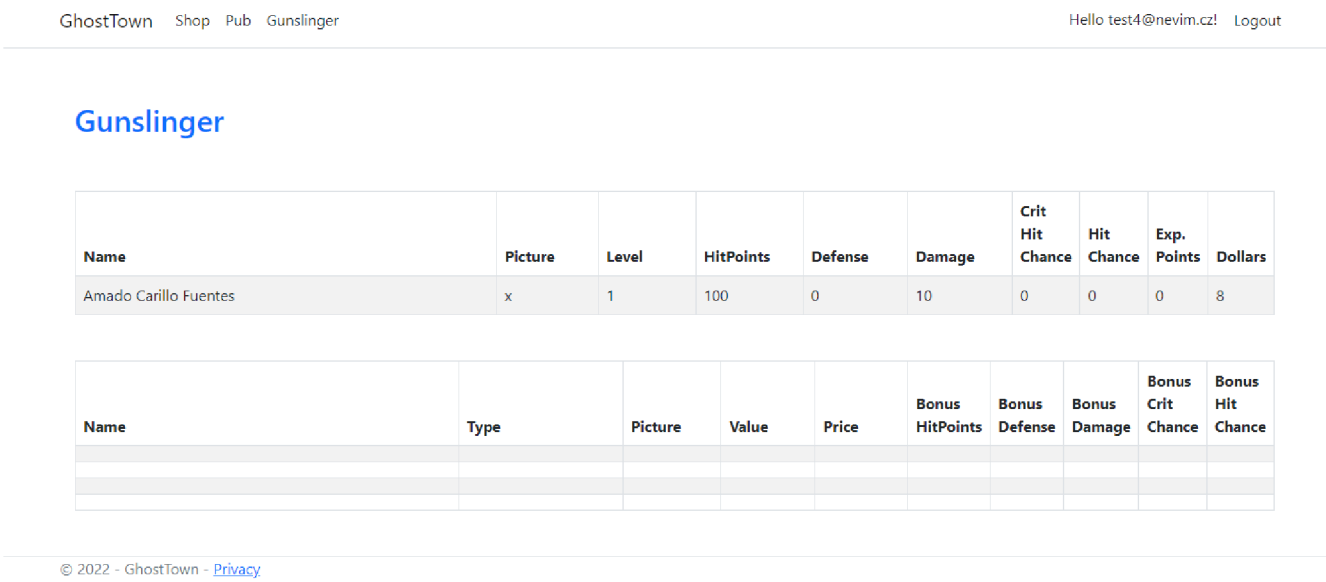
Po zvolení jména klikne hráč na tlačítko ADD a tím vznikne jeho postava, která bude provázána s jeho účtem.

4.6.3.2 Každé další přihlášení

Při každém jiném než prvním přihlášení se uživatel octne v rozhraní pro jeho Gunslingera.

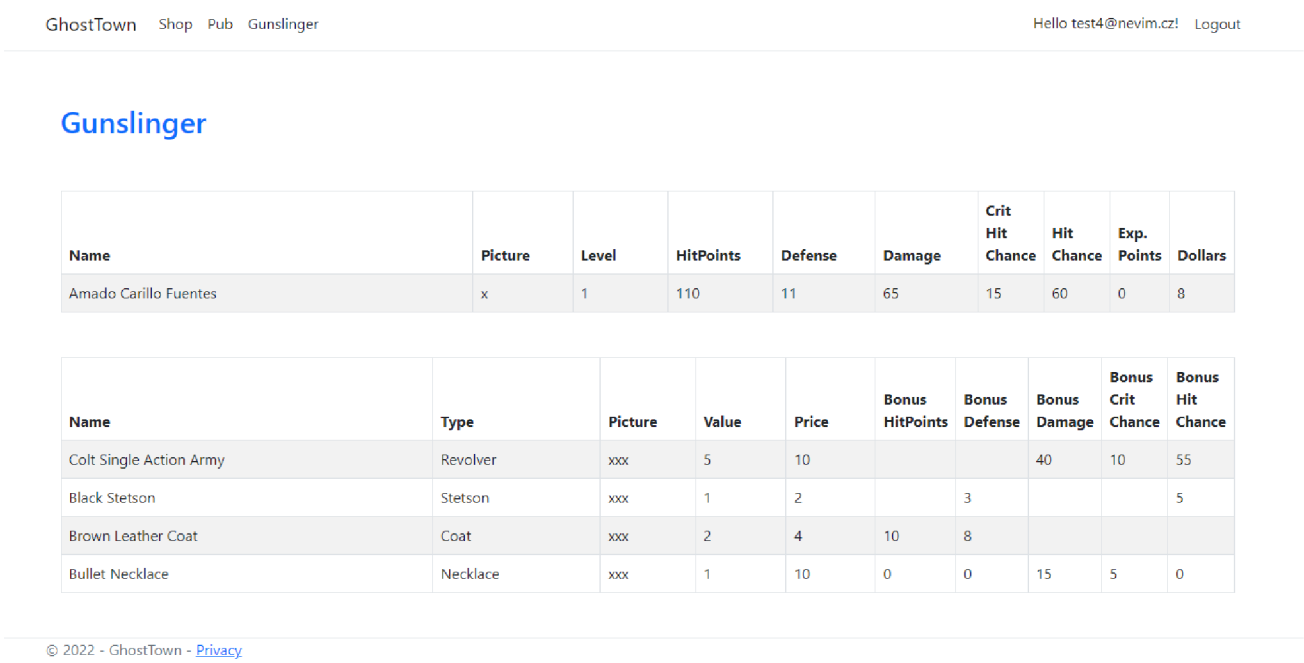
4.6.4 Gunslinger – rozhraní

V přehledu hráčova pistolníka lze vidět statistiky a jednotlivé předměty. Hráč začíná bez předmětů, tudíž jeho první rozhraní bude vypadat následovně.



Obrázek 38 - Rozhraní nového pistolníka

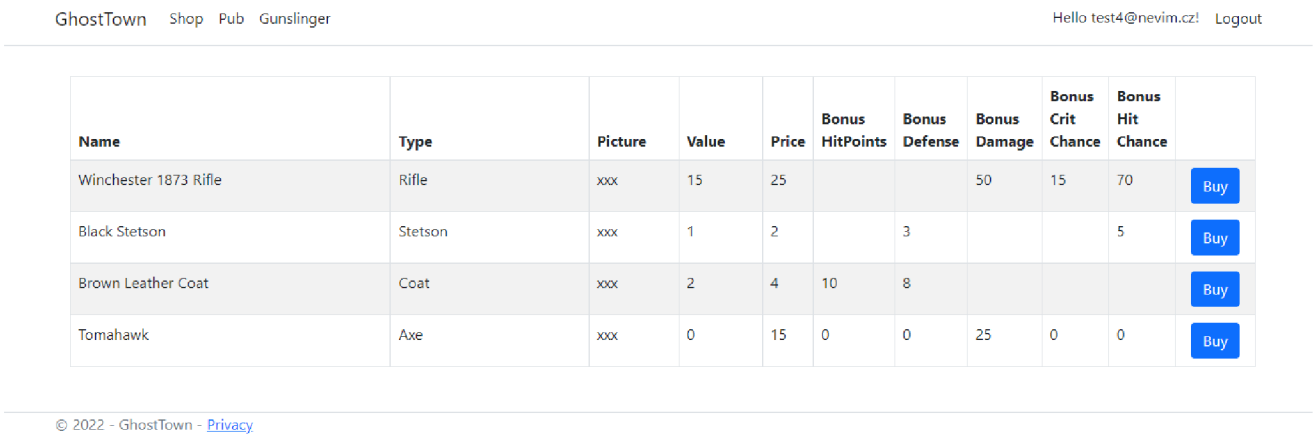
Pakliže již předměty má, rozhraní bude následující. U předmětů, které nemají určité statistiky jsou pole vynechána úmyslně právě pro tento důvod.



Obrázek 39 - Rozhraní pistolníka s předměty

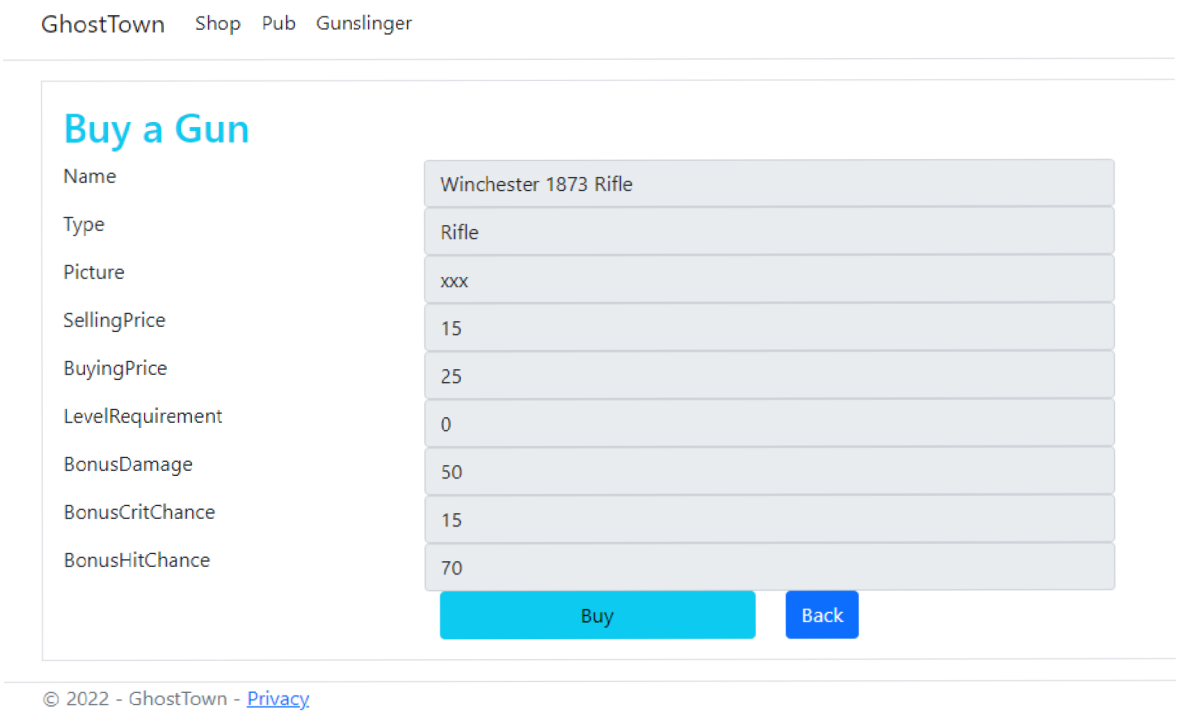
4.6.5 Shop – rozhraní

Jak již bylo výše vysvětleno a jak z názvu vyplívá, hlavním prvkem obchodu je nákup nových předmětů.



Obrázek 40 - Rozhraní obchodu

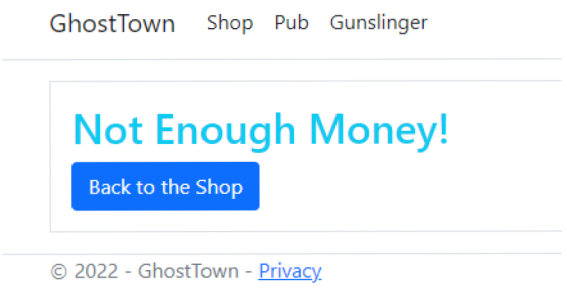
Obchod vyobrazí 1 od každého předmětu s tlačítkem možnosti nákupu. Po stisknutí tlačítka pro nákup se objeví přehled příslušného předmětu.



Obrázek 41 - Rozhraní nákupu předmětu

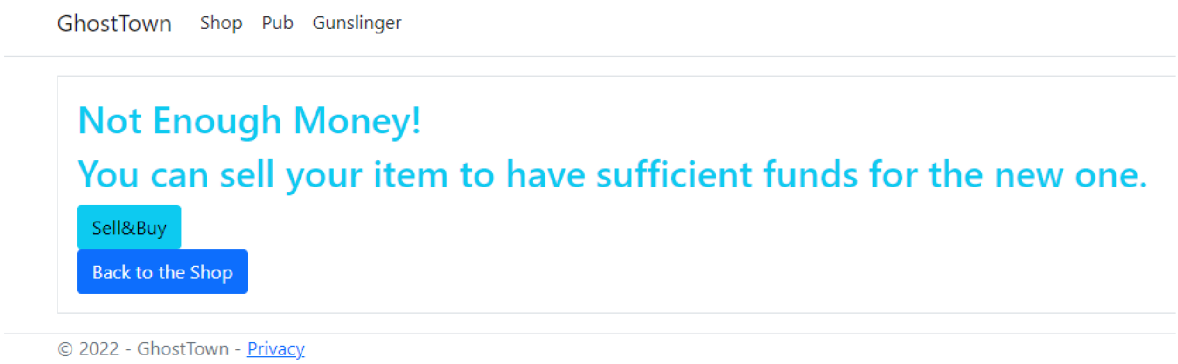
Po stisknutí tlačítka Buy nastane jeden ze tří výše zmíněných scénářů v závislosti na stavu dolarového konta hráče.

Při nedostatku financí:



Obrázek 42 – Nedostatek financí

Při nedostatku financí s předmětem ve slotu jehož prodejní hodnota by pokryla výdaje na nový předmět:



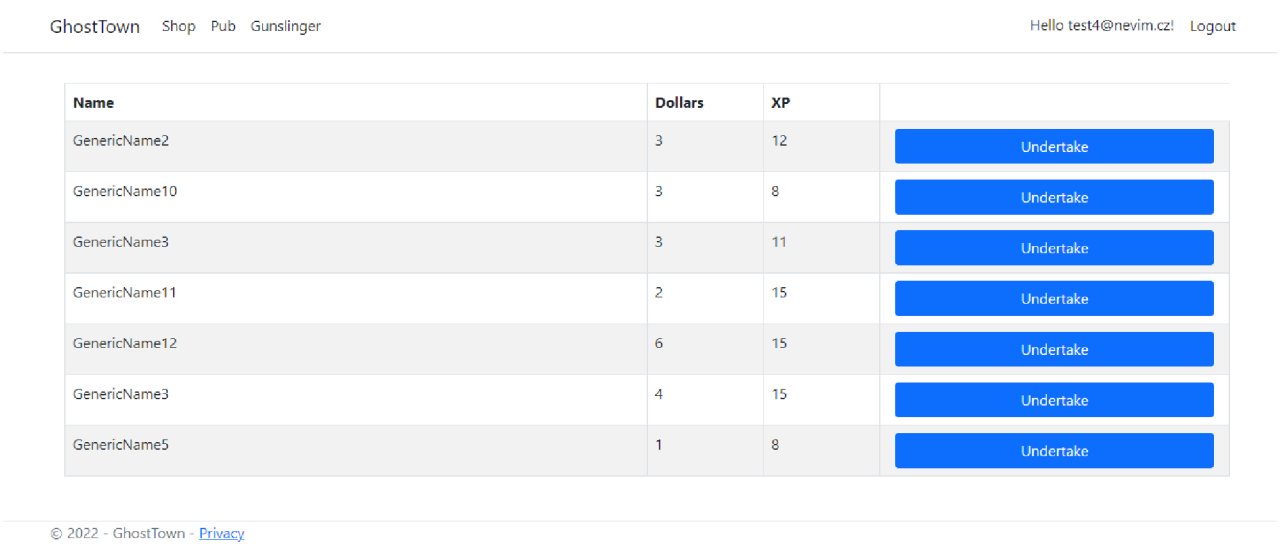
Obrázek 43 - Nákup předmětu za starý

Při nákupu nového předmětu bez potřeby prodeje starého se předmět nasadí, finance změní a zobrazí se opět obchod.

4.6.6 Pub – rozhraní

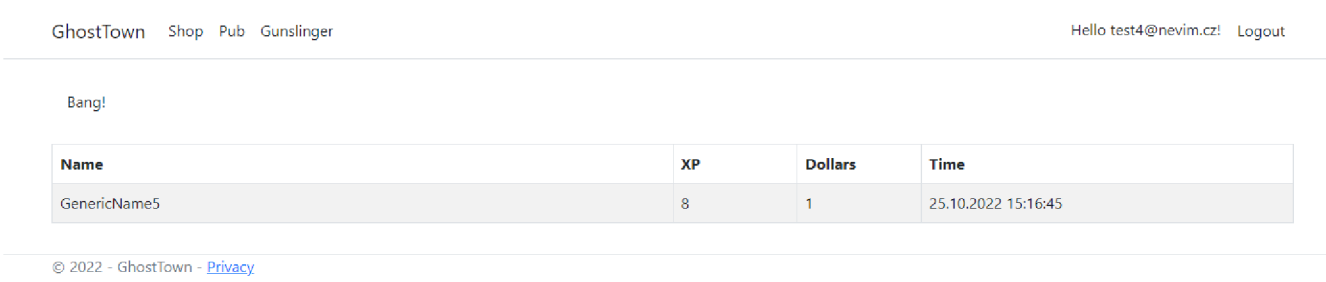
V hospodě si chodí pistolník pro své úkoly. Vyobrazí se mu 8 úkolů. Před seznámením čtenáře s rozhraním úkolů je nutno podotknout dvě skutečnosti. Úkoly nejsou konkrétně pojmenovány jelikož jsou dle typu losovány z databáze úkolů, kterých mohou pro pestrost hry být stovky. Vymýšlení tolik různých názvů úkolů by bylo pro potřeby této práce zbytečné.

Dále je třeba myslet na to že pro ulehčení pořizování snímků z programu byly časy úkolů sníženy na řádově vteřiny.



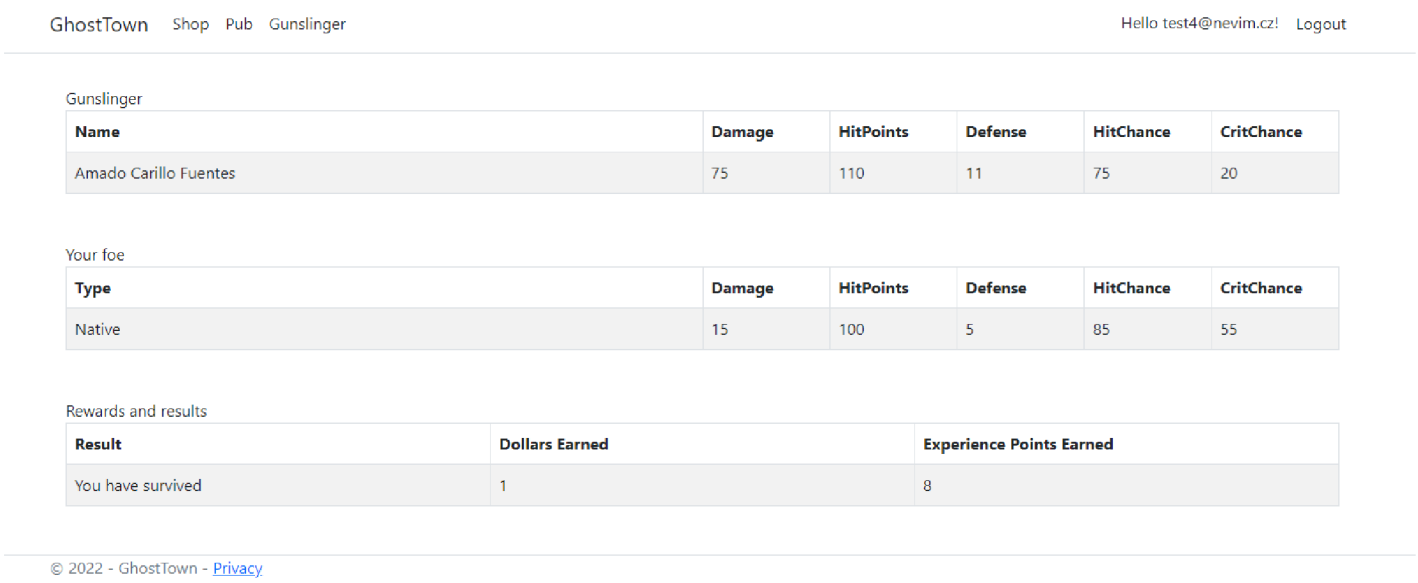
Obrázek 44 – Rozhraní hospody

Po zvolení úkolu a kliknutí na tlačítko Undertake se objeví informace o probíhajícím úkolu.



Obrázek 45 - Rozhraní probíhajícího úkolu

Po uplynutí času a znovunačtení stránky se objeví výsledná obrazovka s informacemi os pistolníkovi, nepříteli, úkolu a výsledku.

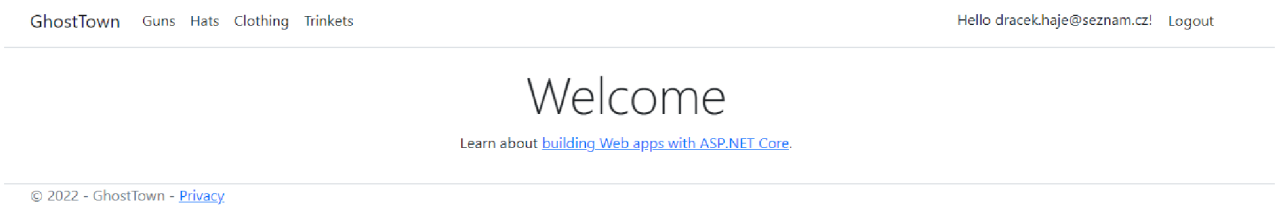


Obrázek 46 - Rozraní výsledku souboje a úkolu

Po dokončení úkolu se v hospodském rozhraní objeví zbývající úkoly a hráči jsou přičteny odměny.

4.6.7 Administrátorské rozhraní

Přihlásí-li se uživatel, který je administrátorem, jeho rozhraní je pak jiné než u obyčejného uživatele.

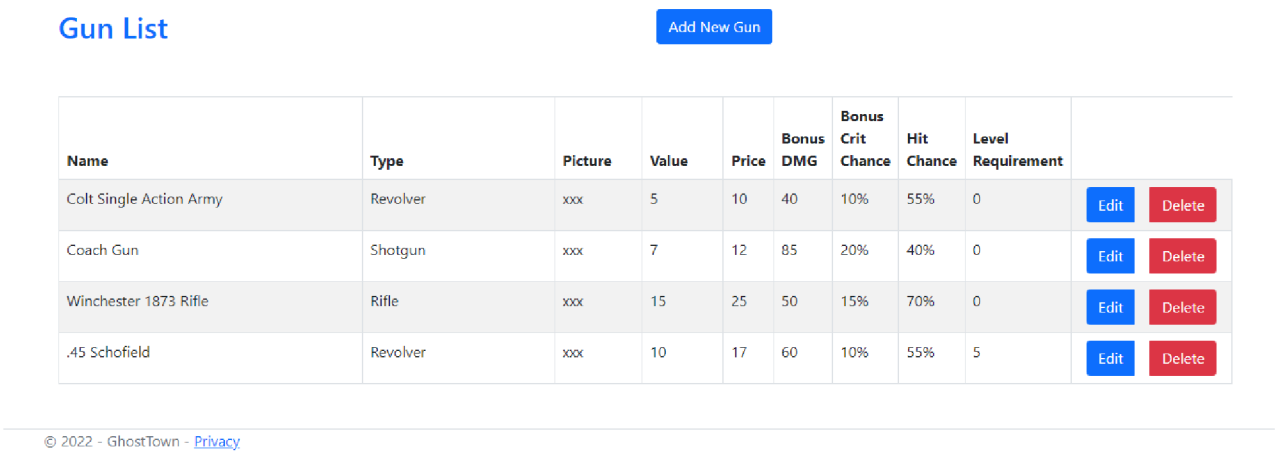


Obrázek 47 - Hlavní rozhraní administrátora

Administrátor umí obsluhovat předměty. Jeho rozhraní je přidáno hlavně pro ulehčení práce s předměty pro vývojáře. Dále jeho samotný vývoj sloužil jako testovací pole pro některé funkcionality což později ulehčilo testování a vývoj celé aplikace.

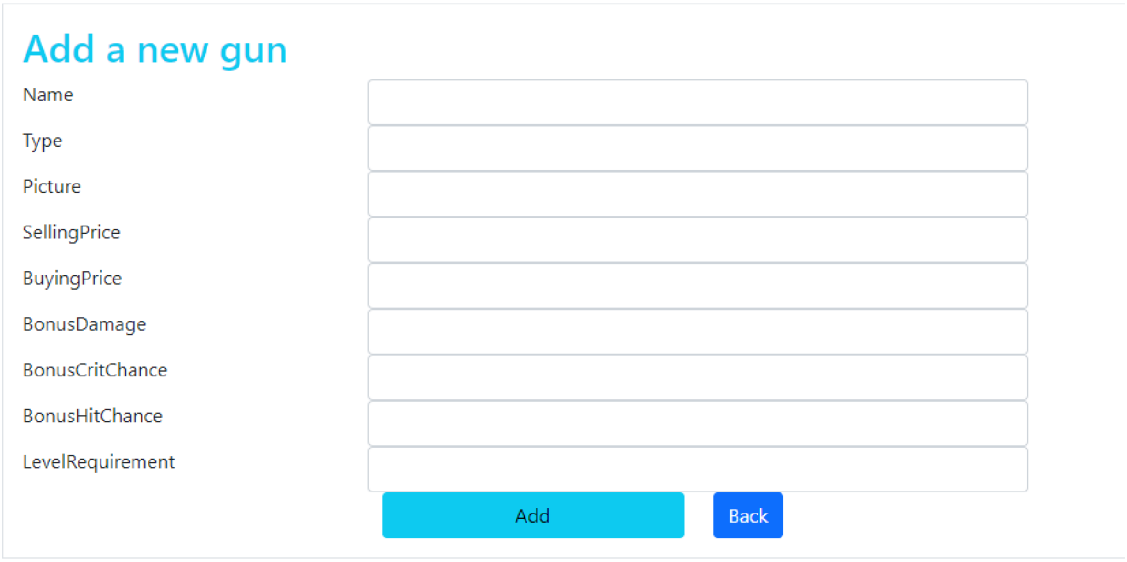
Všechna 4 rozhraní jednotlivých předmětů umí to samé. Tudíž stačí předvést jeden předmět.

4.6.7.1 Ukázka rozhraní předmětu



Obrázek 48 - Základní administrátorské rozhraní předmětu

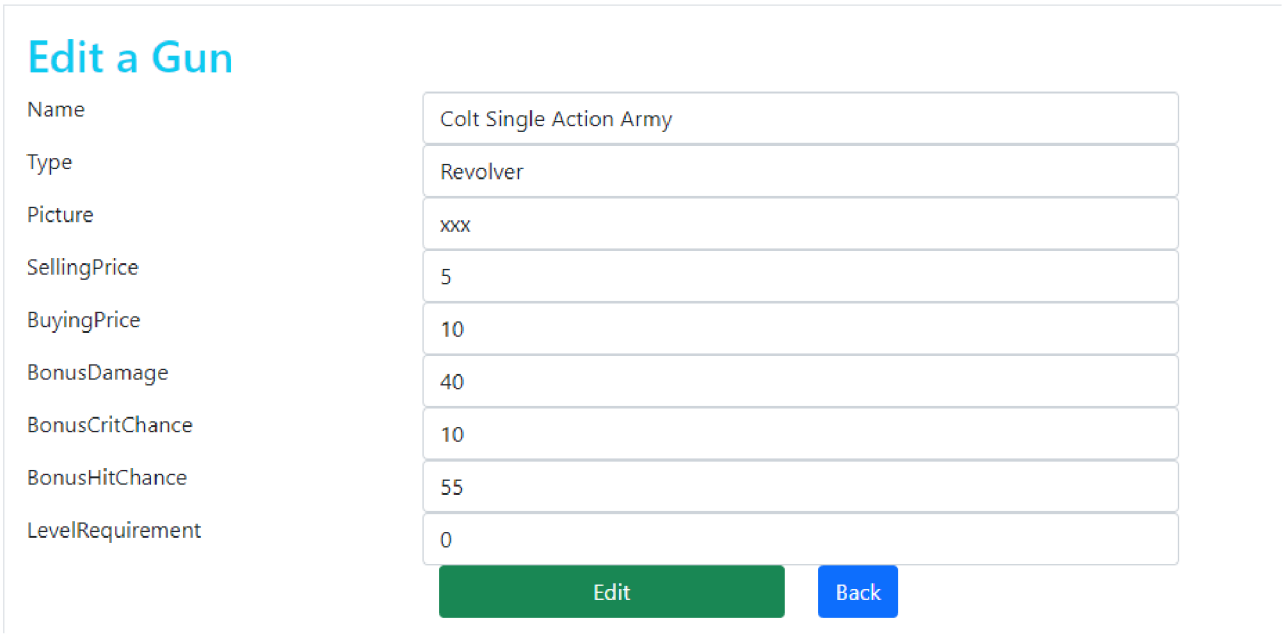
Daný předmět lze vždy přidat, editovat či smazat.



Obrázek 49 - Rozhraní přidání předmětu

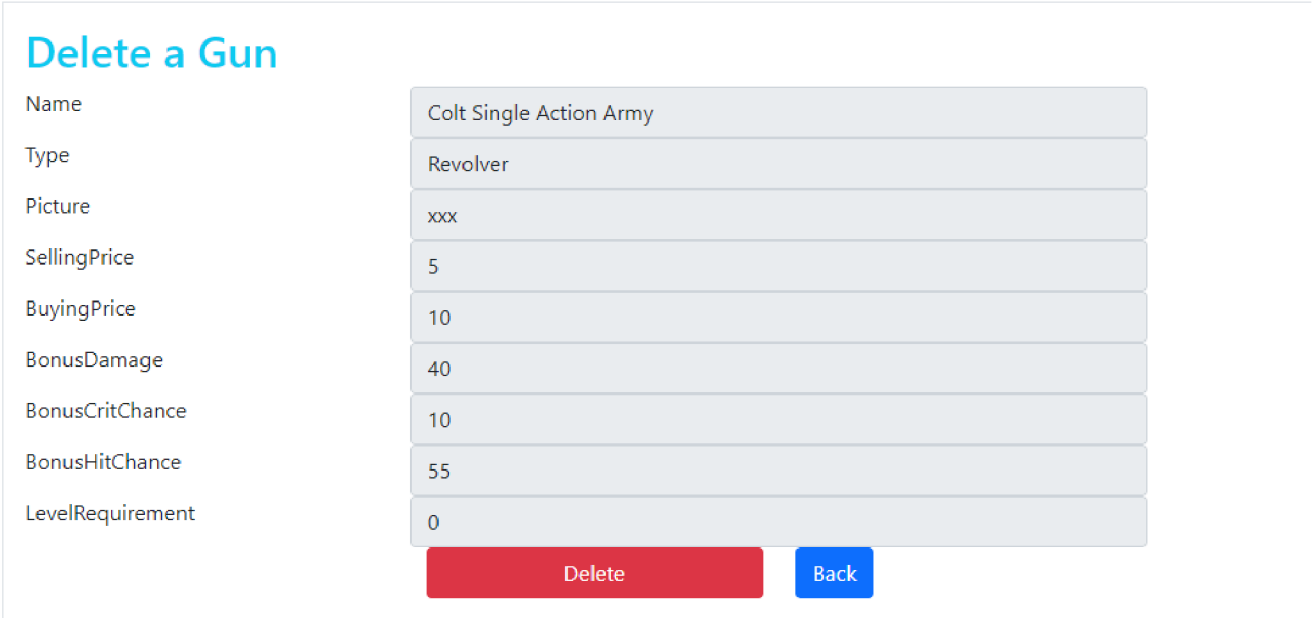
Uživatel vyplní jednotlivé údaje a tlačítkem Add předmět přidá.

Editace předmětu je pak velmi podobná. Rozdíl je v tom, že se konkrétní údaje předvyplní. Cokoliv se přepíše pak bude přepsáno i na předmětu v databázi po stisknutí tlačítka Edit.



Obrázek 50 - Rozhraní úpravy předmětu

Mazání předmětu potom vyvolá stejné rozhraní s rozdílem, že jednotlivé informace nelze přepisovat. Jejich vyobrazení je tudíž pouze informativního rázu. Po stisknutí tlačítka Delete se daný předmět vymaže z databáze.



Obrázek 51 - Rozhraní mazání předmětu

4.7 Možná zlepšení

Hlavním možným zlepšením by byla možnost souboje s jinými hráči v organizované aréně. Hráči by byli nabízeni ostatní hráči okolo jeho úrovně náhodně. Hráč by je pak mohl napadat vždy jednou za určité období a získávat tím část jejich peněz. Tento systém by nejspíš vyžadoval zavedení mechaniky slávy, která by se získávala úspěšnými útoky na jiné hráče. Podle té by se časem vybírali náhodní oponenti v této aréně.

Dalším možným zlepšením je vytvoření frontendu. I když ten není předmětem práce, autor bude v pracích na aplikaci pokračovat a její vzhled časem ke hře přidá. Frontend by vznikl za pomocí vlastní grafiky a javascriptu.

Posledním zlepšením, které je naplánováno je funkční inventář. Jeho velikost je ovšem předmětem otevřené debatě.

5 Závěr

Předmětem této práce byl návrh, analýza, tvorba a nasazení s testováním prototypu backendové aplikace pro online RPG hru ve frameworku ASP.NET. Samotnému programování předcházela analýza jejíž součástí byla tvorba příslušných diagramů v jazyce UML sloužících pro implementaci, tvorba person, pracovního plánu a vytyčení funkčních požadavků.

Práce byla naprogramována v prostředí Visual Studio ve frameworku ASP.NET Core 6 s užitím návrhového vzoru MVC. Hotový prototyp aplikace byl nasazen na cloudové uložiště prostřednictvím služby Azure. Při práci byla užita metodiky SOLID principů, softwarového inženýrství a jako metoda vývoje byla zvolena metoda prototyp.

V první části práce představuje poznatky, které sloužily jako výchozí teoretické základy samotného vývoje a celé práce, včetně principů softwarového inženýrství, programování a zejména pak objektového přístupu. Dále se lze dočíst o základní teorii tvorby backendu a databází. Nadále teoretická část slouží k představení užitých technologií při vývoji a to zejména ASP.NET a Entity Frameworku, a pak RPG her.

Druhá část se zabývá praktickou rovinou práce. Zde je předveden postupný vývoj aplikace od položení funkčních požadavků a definicí mechanik hry, přes analytickou část s tvorbou diagramů až po samotnou realizaci v programovacím jazyce C# a následné nasazení. A aplikace byla podrobnému kvalitativnímu testováním, jak vývojáři, tak běžnými uživateli na jehož základě byly opraveny dílčí problémy.

Za hlavní přínosy práce lze považovat vytvoření vhodné Backendové struktury pro tvorbu online RPG prohlížečových her se zaměřením na westernovou tématiku. Potenciální vývojář si může tento projekt dále upravit, zejména pak mu doplňovat specifické frontendové části, a dát tak vzniknout neomezenému množství unikátních a vlastních projektů.

6 Seznam použitých zdrojů

ALBAHARI, Joseph and Ben ALBAHARI, 2020. C# 8.0 in a Nutshell: The Definitive Reference. O'Reilly Media. ISBN 9781492051138.

ANDERSON Rick, *Introduction to Identity on ASP.NET Core.* [online]. [cit. 22.10.2022]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-6.0&tabs=visual-studio.

ANDERSON Rick, *Scaffold Identity in ASP.NET Core projects.* [online]. [cit. 22.10.2022]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/scaffold-identity?view=aspnetcore-6.0&tabs=visual-studio.

BILLESTRUP Jane, STAGE Jan, BRUUN Andres, NIELSEN Lene, NIELSEN Kira S., *Creating and Using Personas in Software Development: Experiences from Practice.* 5th International Conference on Human-Centred Software Engineering (HCSE), Paderborn, Germany 2014. [online]. [cit. 17.10.2022]. Dostupné z: https://hal.inria.fr/hal-01405082/document.

CAMBRIDGE Dictionary, *Algorithm.* [online]. [cit. 17.10.2022]. Dostupné z: https://dictionary.cambridge.org/dictionary/english/algorithm.

CodeAcademy Team, *What is programming?* [online]. [cit. 17.10.2022]. Dostupné z: https://www.codecademy.com/article/what-is-programming.

CONNOLLY, Thomas M., Carolyn E. BEGG a Jennifer WIDOM. Database systems: a practical approach to design, implementation, and management. 5th ed. London: Addison-Wesley, c2010. ISBN 978-0-32-152306-8.

ČÁPKA David, *Lekce 1 – Úvod do UML.* [online]. [cit. 17.10.2022]. Dostupné z: https://www.itnetwork.cz/navrh/uml/uml-uvod-historie-vyznam-a-diagramy.

ČÁPKA David, *Lekce 2 – UML – Use Case Diagram.* [online]. [cit. 17.10.2022]. Dostupné z: https://www.itnetwork.cz/navrh/uml/uml-use-case-diagram/?all-comments#comments.

ČERMÁK Petr, MARTINŮ Jiří, *Metodiky Vývoje Software*. Olomouc 2018. Moravská Vysoká Škola Olomouc, o.p.s., 2018 [online]. [cit. 12.10.2022]. Dostupné z:

https://dl1.cuni.cz/pluginfile.php/864918/mod_resource/content/1/Metodiky-v%C3%BDvoje-software-studijn%C3%AD-text.pdf.

DUPIRE François, *The Liskov Substituion Principle.* [online]. [cit. 18.10.2022]. Dostupné z: https://www.baeldung.com/cs/liskov-substitution-principle.

Entity Framework Tutorial, *What is Entity Framework?* [online]. [cit. 22.10.2022]. Dostupné z: https://www.entityframeworktutorial.net/what-is-entityframework.aspx.

ERINÇ Yiğit Kemal, *The Solid Principles of Object-oriented Programming explained in plain English.* [online]. [cit. 18.10.2022]. Dostupné z: https://www.freecodecamp.org/news/solid-principles-explained-in-plain-english/.

F2P, *BITEFIGHT SCREENSHOTS.* [online]. [cit. 23.10.2022]. Dostupné z: https://www.f2p.com/screenshot/bitefight/.

Hernimag, *Gladiatus.cz.* [online]. [cit. 23.10.2022]. Dostupné z: https://www.hernimag.cz/219/gladiatus-cz/.

InterviewBit, *MVC Architecture – Detailed Explanation.* [online]. [cit. 22.10.2022]. Dostupné z: https://www.interviewbit.com/blog/mvc-architecture/.

JavaPoint, *SQL Languages.* [online]. [cit. 22.10.2022]. Dostupné z: https://www.javatpoint.com/sql-languages.

LEGIERSKI Boguslaw, *Frontend vs. Backend.* [online]. [cit. 19.10.2022]. Dostupné z: https://www.evertop.pl/en/frontend-vs-backend/.

Geeks For Geeks, *Beta Testing Software Testing*. [online]. [cit. 17.10.2022]. Dostupné z: https://www.geeksforgeeks.org/beta-testing-software-testing/.

GILLIS Alexander S., *object-oriented programming (OOP).* [online]. [cit. 18.10.2022]. Dostupné z: https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP.

IBM, *Primary and foreign keys.* [online]. [cit. 20.10.2022]. Dostupné z: https://www.ibm.com/docs/en/ida/9.1?topic=entities-primary-foreign-keys.

IBM, *What are Workflow diagrams?* [online]. [cit. 12.10.2022]. Dostupné z: https://www.ibm.com/cloud/blog/workflow-diagrams.

JEVTIC GORAN, *What is SDLC? Phases of Software Develoment, Methods, Best & Practises.* [online]. [cit. 17.10.2022]. Dostupné z: https://phoenixnap.com/blog/software-development-life-cycle.

KETCHUNA, *WoW How to Transfer Key Bindings & Macros from Retail to Classic*. [online]. [cit. 23.10.2022]. Dostupné z: https://www.gosunoob.com/wow-classic/how-to-transfer-key-bindings-macros-from-retail-to-classic/.

KUMAR Subalh, *Strategy Pattern | Set 1 (Introduction).* [online]. [cit. 18.10.2022]. Dostupné z: https://www.geeksforgeeks.org/strategy-pattern-set-1/.

Management Mania, *Apliakční server (Application server).* [online]. [cit. 19.10.2022]. Dostupné z: https://managementmania.com/cs/aplikacni-server-aps.

Management Mania, *Třívsrtvá Architektura (Three-tier architecture).* [online]. [cit. 19.10.2022]. Dostupné z: https://managementmania.com/cs/trivrstva-architektura-three-tier-architecture.

MARTIN Matthew, *N Tier(Multi-Tier), 3-Tier, 2-Tier Architecture with EXAMPLE.* [online]. [cit. 19.10.2022]. Dostupné z: https://www.guru99.com/n-tier-architecture-system-concepts-tips.html.

MDN Contributors, *What is a web server?* [online]. [cit. 19.10.2022]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server.

Microsoft, *An Introduction to NuGet.* [online]. [cit. 22.10.2022]. Dostupné z: https://learn.microsoft.com/en-us/nuget/what-is-nuget.

Microsoft, *ASP.NET overview*. [online]. [cit. 22.10.2022]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/overview.

Microsoft, *Integrovaný dotaz jazyka (LINQ) (C#)*. [online]. [cit. 22.10.2022]. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/csharp/programming-guide/concepts/linq/.

MILLINGTON Sam, *A Solid Guide To SOLID Principles.* [online]. [cit. 18.10.2022]. Dostupné z: https://www.baeldung.com/solid-principles.

Oracle, *What Is a Database?* [online]. [cit. 19.10.2022]. Dostupné z: https://www.oracle.com/database/what-is-database/.

NAEEM Tehreem, *What is Data Integrity in a Database? Why Do You Need It?* [online]. [cit. 20.10.2022]. Dostupné z: https://www.astera.com/type/blog/data-integrity-in-a-database/.

PETKOV Alexander, *How to explain object-oriented programming concepts to a 6-year-old.* [online]. [cit. 18.10.2022]. Dostupné z: https://www.freecodecamp.org/news/object-oriented-programming-concepts-21bb035f7260/.

Refactoring.guru, *What's a design pattern?* [online]. [cit. 17.10.2022]. Dostupné z: https://refactoring.guru/design-patterns/what-is-pattern.

Refactoring.guru, *Factory Method.* [online]. [cit. 18.10.2022]. Dostupné z: https://refactoring.guru/design-patterns/factory-method.

ROTH Daniel, *ASP.NET Core overview.* [online]. [cit. 22.10.2022]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0.

ŘEZNÍČEK Josef, *Tvoříme persony pro obsahový marketing.* [online]. [cit. 17.10.2022]. Dostupné z: https://vceliste.cz/blog/tvorime-persony-pro-obsahovy-marketing/.

Source Making, *Prototype Design Pattern.* [online]. [cit. 18.10.2022]. Dostupné z: https://sourcemaking.com/design_patterns/prototype.

ThatJeffSmith, *How To: Generate an ERD for Selected Tables in SQL Developer.* [online]. [cit. 12.10.2022]. Dostupné z: https://www.thatjeffsmith.com/archive/2011/11/how-to-generate-an-erd-for-selected-tables-in-sql-developer/.

TechTarget, *Web application (Web app).* [online]. [cit. 19.10.2022]. Dostupné z: https://www.techtarget.com/searchsoftwarequality/definition/Web-application-Web-app.

UPADHYAY Mithlesh, *Second Normal Form (2NF).* [online]. [cit. 21.10.2022]. Dostupné z: https://www.geeksforgeeks.org/second-normal-form-2nf/.

VONDRÁK Ivo, *Úvod do softwarového inženýrství.* Ostrava 2002. VŠB – technická univerzita Ostrava, 2002. [online]. [cit. 12.10.2022]. Dostupné z: http://vondrak.cs.vsb.cz/download/Uvod_do_softwaroveho_inzenyrstvi.pdf.

W3Resource, *SQL Transaction.* [online]. [cit. 22.10.2022]. Dostupné z: https://www.w3resource.com/sql/controlling-transactions.php.

W3Schools, *ASP.NET Razor – Markup.* [online]. [cit. 22.10.2022]. Dostupné z: https://www.w3schools.com/asp/razor_intro.asp.

WILKINS Jessica, *What is Computer Programming*. [online]. [cit. 17.10.2022]. Dostupné z: https://www.codecademy.com/article/what-is-programming.

7 Seznam obrázků, tabulek, grafů a zkratek

7.1 Seznam obrázků