Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Návrh a realizace e-shopu v ASP.NET MVC

Bakalářská práce

Autor: David Šupík
Studijní obor:ai3p

Vedoucí práce: prof. RNDr. PhDr. Antonín Slabý, CSc.

Hradec Králové duben 2022

Čestné prohlášení

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně pod vedením vedoucího bakalářské práce, s použitím výhradně odborné literatury a dalších informačních zdrojů, které jsou v práci uvedeny v seznamu literatury a použitých zdrojů.

V Hradci Králové dne 29.4.2022 David Šupík

Poděkování

Děkuji vedoucímu bakalářské práce prof. RNDr. PhDr. Antonínu Slabému, CSc. za metodické vedení práce, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

Anotace

Práce pojednává o moderních postupech tvorby webových stránek a jejich použití při navrhování a implementaci internetového obchodu s použitím aplikačního rámce ASP.NET MVC. Navrhování zahrnuje analýzu firemních procesů a jejich popis pomocí diagramů v jazyce UML a BPMN. Práce také pojednává o datovém modelování a grafickém designu, což jsou disciplíny, které jsou nedílnou součástí návrhu webové aplikace. Poslední částí práce je implementace, která popisuje vytvoření databáze v MSSQL na základě datových modelů a vygenerování a napsání kódu aplikace v ASP.NET.

Annotation

Title: Design and implementation of e-shop in ASP.NET MVC

Thesis deals with modern methods of creating websites on ASP.NET platform and their use in the design and implementation of e-commerce. Design includes analysis of business processes and their description using diagrams in UML and BPMN. It also describes the creation of the company's visual style and graphic design of the website. The thesis also deals about data modeling graphic design which are integral parts of website design. Last part of the work is implementation which describes creating a database in MSSQL based on data models and generating and writing application code in ASP.NET.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Cílem práce je seznámení se s moderními postupy tvorby webových stránek a jejich použití při navrhování a implementaci internetového obchodu. Řešení e-shopu je určeno pro podnikatele, kteří chtějí své produkty nabízet prostřednictvím internetového obchodu. Internetový obchod umožňuje zákazníkům vyhledávat a objednávat produkty z pohodlí domova. V práci je realizován příklad internetového obchodu, který může být rozšířením pro klasickou kamennou prodejnu bot. Pokud chce tedy zákazník nakupovat přímo tam, může přes e-shop zjistit, jaké zboží prodejce nabízí. Smyslem projektu je vytvořit snadnější a ohodlnější cestu nákupu bot.

Digitální obchod přináší majiteli e-shopu vyšší zisky a nové potencionální zákazníky, a to díky prodejům, které by se jinak neuskutečnily kvůli „lenosti zákazníka“, nebo nedosažitelnosti prodejny. Majitel navrhovaného internetového obchodu může spravovat zboží v internetovém obchodě a zároveň sledovat aktuální stav skladu. Rovněž bude schopen spravovat objednávky zákazníků a mít přehled o veškerých financích. Tento obchod by měl splňovat všechny hlavní funkce, které lze od internetového obchodu očekávat. K tomu patří např. možnost vytvářet a spravovat objednávky, produkty, reklamace, uživatelské účty, obchodní podmínky, platební metody a faktury.

2 Modelování procesů v BPMN

Tato kapitola pojednává o modelování procesů v BPMN. Business Process Model and Notation (BPMN) poskytuje podnikům možnost porozumět jejich interním obchodním procesům v grafickém zápisu a umožňuje organizacím komunikovat tyto postupy standardizovaným způsobem. Grafický zápis kromě toho také usnadňuje pochopení spolupráce mezi organizacemi. To zajistí, že podniky porozumí sobě a účastníkům svého podnikání a umožní organizacím rychle se přizpůsobit novým okolnostem.

BPMN má mnoho výhod. Jednou z těchto výhod je, že neodborní uživatelé získají snadný a intuitivní způsob, jak porozumět procesnímu diagramu. Sémantika složitých procesů je reprezentována snadno a v čitelném formátu., což odstraňuje komunikační propast mezi návrhářským týmem vytvářejícím obchodní proces a týmem, který jej implementuje. Vzhledem k intuitivnosti BPMN je velmi snadné usnadnit integraci dalších odborníků. Návrhu se mohou účastnit například obchodní analytici, zaměstnanci přímo pracující s procesy, také i manažeři mající přístup k datům používaným k řízení a monitorování procesu. S BPMN mohou obchodníci jednoduše definovat, co chtějí, a to s vysokou mírou přesnosti. IT profesionálové mohou komunikovat mezi sebou a s obchodníky o modelu v jasném společném rámci. BPMN funguje pro jakýkoli druh procesu řízení, provozu a podpory.

BPMN 2.0 je nejnovější verzí standardu BPMN, který vyvinula společnost Object Management Group (OMG) s cílem vytvořit jednotný modelovací jazyk, který je srozumitelný všem zúčastněným. Odstraňuje propast mezi návrhem a implementací obchodních procesů. To zjednodušuje uživatelskou úlohu stanovením přehledného softwarového diagramu. Verze 2.0 staví na předchozích verzích tím, že poskytuje bohatší standardní sadu symbolů a notací, což umožňuje více podrobností pro ty, kteří to potřebují. Modely BPMN 2.0 jsou konstruovány z grafických prvků představujících různé části procesu. Základní prvky se skládají z aktivit, událostí a bran, které jsou propojeny šipkami znázorňujícími tok sekvence. Výsledek obchodního procesu je svou technikou podobný vývojovým diagramům a UML.

2.1 Základní elementy

BPMN se skládá z těchto čtyř typů prvků:

- Objekty toku (Flow objects) – událost, aktivita, brána
- Spojovací objekty (Connecting objects) – sekvenční tok, tok zpráv, asociace
- Plavecké dráhy (Swim lanes) – bazén, dráha
- Artefakty (Artifacts) – dokument, skupina, anotace

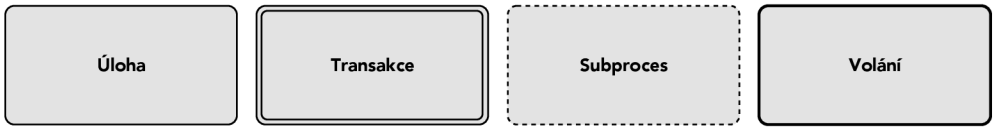
Níže jsou popsány jednotlivé prvky, jak se používají k definování obchodního procesu.

Událost – znázorněná kruhem, popisuje něco, co se děje v průběhu procesu.
Každý proces musí začínat iniciační událostí, nazývanou startovací událost.
V rámci modelování obchodních procesů existují tři hlavní události:

Obrázek 1 – Typy událostí
Zdroj: Vlastní zpracování

Události mají různé typy, a to zprávu, časovač, chybu, kompenzaci, signál, zrušení, eskalaci a odkaz. Jsou znázorněny kroužkem obsahujícím další symbol uvnitř podle typu události. Jsou klasifikovány jako vyvolávané (throwing) nebo odchytávané (catching) v závislosti na jejich funkci.

Aktivita – je konkrétní činnost nebo úkol prováděný osobou nebo systémem.
Je znázorněna obdélníkem se zaoblenými rohy. Může být popsána podrobněji pomocí dílčích procesů, smyček, kompenzací a více instancí.



Obrázek 2 – Typy aktivit

Zdroj: Vlastní zpracování

Úloha – je definována jako jednotka práce, která má být provedena. Při označení symbolem ⊞ se označuje jako sub proces, je dále dělitelná na další úlohy. Úloha má typy, které určují povahu akce, které má být provedena. Těmito typy jsou:

- odesílací úkol
- přijímací úkol
- uživatelský úkol
- manuální úkol
- úkol služby
- skriptovací úkol

Obrázek 3 - Typy aktivit

Zdroj: Vlastní zpracování

Transakce – je soubor činností, které po sobě logicky následují; může se řídit specifikovaným transakčním protokolem.

Sub proces události – je umístěn do procesu nebo sub procesu. Aktivuje se při spuštění události spuštění, může přerušit kontext procesu vyšší úrovně, nebo běžet paralelně (nepřerušovaně) v závislosti na události spuštění.

Aktivita volání – je obal pro globálně definovaný úkol nebo proces znovu použitý v aktuálním procesu. Volání procesu je označeno symbolem ⊞.

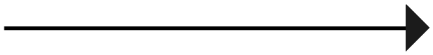
Brána – je rozhodovací bod, který může upravit cestu na základě podmínek nebo událostí. Je vyobrazena jako diamant. Může být exkluzivní, zahrnující, paralelní, komplexní, nebo založená na datech či událostech. Brány mají několik typů:

- Exkluzivní – při rozdělení směruje tok sekvence přesně do jedné z odchozích větví. Při slučování čeká na dokončení jedné příchozí větve, než spustí odchozí tok.
- Založená na události – je vždy následována vyvoláváním událostí nebo přijímáním úkolů. Tok sekvence je směrován k následující události/úloze, která nastane jako první.
- Paralelní – při použití k rozdělení toku sekvence jsou všechny odchozí větve aktivovány současně. Při slučování paralelních větví čeká na dokončení všech příchozích větví, než spustí odchozí tok.
- Inkluzivní – může aktivovat více výstupů současně. Podporuje podmínky na tocích odchozí sekvence.
- Exkluzivní, (založená na události) – každý výskyt následující události spustí novou instanci procesu.
- Komplexní – složité slučování a větvení chování, které není zachyceno jinými bránami.
- Paralelní (založená na události) – každý výskyt následující události spustí novou instanci procesu.

Obrázek 4 – Typy bran

Zdroj: Vlastní zpracování

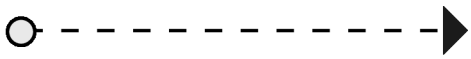
Sekvenční tok – zobrazuje pořadí činností, které mají být provedeny. Je zobrazen jako přímka se šipkou. Může zobrazovat podmíněný tok nebo výchozí tok.



Obrázek 5 – Sekvenční tok

Zdroj: Vlastní zpracování

Tok zpráv – zobrazuje zprávy, které proudí přes *pooly* nebo hranice organizace, jakými jsou oddělení. Neměl by spojovat události nebo aktivity v rámci *poolu*. Je znázorněn přerušovanou čarou s kruhem na začátku a šipkou na konci.



Obrázek 6 – Tok zprávy

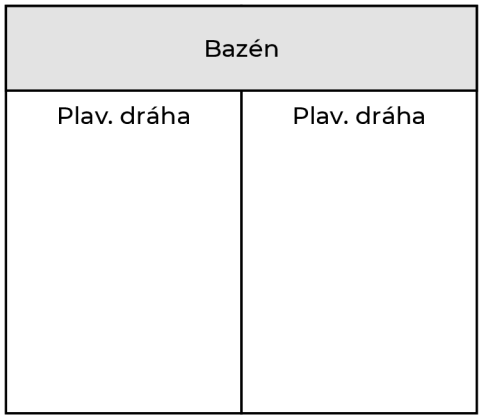
Zdroj: Vlastní zpracování

Asociace – zobrazená tečkovanou čarou, přiřazuje artefakt nebo text k události, aktivitě nebo bráně.

Obrázek 7 – Asociace

Zdroj: Vlastní zpracování

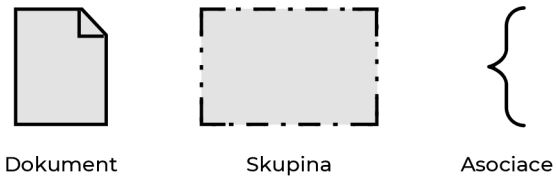
Bazén a plavecké dráhy – představují hlavní účastníky procesu. Některý z bazénů může být v jiné společnosti nebo oddělení, ale stále je zapojen do procesu. Plavecké dráhy zobrazují aktivity a tok pro určitou roli nebo účastníka a zároveň definují, kdo je odpovědný, za jaké části procesu.



Obrázek 8 – Bazén a plavecké dráhy

Zdroj: Vlastní zpracování

Artefakt – představuje další informace, které vývojáři přidávají, aby do diagramu přinesli nezbytnou úroveň podrobností. Existují tři typy artefaktů: datový objekt, skupina nebo anotace. Datový objekt ukazuje, jaká data jsou pro aktivitu nezbytná. Skupina zobrazuje logické seskupení aktivit, ale nemění tok diagramu.
Anotace poskytuje další vysvětlení k části diagramu.



Obrázek 9 – Typy artefaktů

Zdroj: Vlastní zpracování

2.2 Typy diagramů

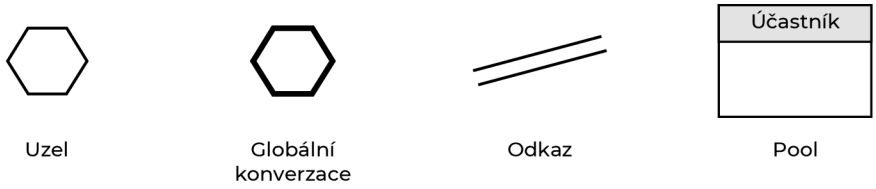
Diagramy se používají ke komunikaci s různorodým publikem, netechnickým i technickým. Dílčí modely umožňují různým divákům najít to, co je pro ně nejvhodnější. Modely mají různé typy, srov. níže:

- Soukromé obchodní procesy. Jsou interní pro konkrétní organizaci a nepřekračují *pooly* nebo organizační hranice.
- Abstraktní obchodní procesy. K nim dochází mezi soukromým/interním procesem a jiným účastníkem nebo procesem. Abstraktní proces ukazuje vnějšímu světu sled zpráv potřebných k interakci se soukromým procesem. Nezobrazuje samotný soukromý/interní proces.
- Spolupráce obchodních procesů. Ty ukazují interakce mezi dvěma nebo více podnikatelskými subjekty.

V BPMN 2 jsou tyto další typy diagramů: choreografický diagram, diagram spolupráce, diagram konverzace.

- **Choreografický diagram** – se zaměřuje na interakce mezi procesy a toky zpráv. Dalším způsobem, jak se dívat na choreografii, je pohlížet na ni jako na typ obchodní „smlouvy" mezi dvěma nebo více organizacemi. Choreografie je typ procesu, ale účelem a chováním je od standardního procesu BPMN odlišná. Standardní proces definuje tok aktivit konkrétního účastníka nebo organizace. Naproti tomu choreografie formalizuje způsob, jakým účastníci koordinují své interakce. Důraz tedy není kladen na orchestraci díla prováděného v rámci těchto účastníků, ale spíše na výměnu informací (zpráv) mezi těmito účastníky. Choreografický diagram lze použít k analýze toho, jak si účastníci vyměňují informace za účelem koordinace svých interakcí.
- **Diagram spolupráce** – zobrazuje interakce mezi dvěma nebo více procesy pomocí více než jednoho *poolu*, kde každý jednotlivý proces představuje osobu, roli nebo systém. V diagramu spolupráce lze použít všechny kombinace skupin, procesů a choreografií.
- **Diagram konverzace** – zobrazuje skupinu souvisejících výměn zpráv v obchodním procesu. Může být rozšířen o dílčí konverzace. Konverzace definuje sadu logicky souvisejících výměn zpráv. Na podrobnější úrovni lze konverzační diagramy modelovat v diagramu choreografie nebo diagramu spolupráce. Vyobrazení diagramu konverzace diagramu

spolupráce obsahuje dva další grafické prvky, které v jiných zobrazeních BPMN neexistují: prvky uzlu konverzace (šestiúhelník) a odkaz konverzace (dvojitá čára).



Obrázek 10 – Elementy diagramu konverzace

Zdroj: Vlastní zpracování

3 Modelovací jazyk UML a CASE nástroje

Tato kapitola pojednává o modelovacím jazyce UML, což je společný jazyk pro analytiky, softwarové architekty a vývojáře. Modelovací jazyk se používá k popisu, specifikaci, návrhu a dokumentaci stávajících nebo nových obchodních procesů a softwarových systémů. UML lze použít v různých odvětvích jako např. v bankovnictví, letectví, zdravotnictví aj. Na základě nějakého konkrétního UML diagramu není 100% jisté, že pochopíme vyobrazenou část nebo chování. Některé informace jsou z diagramu záměrně vynechány, část informací znázorněná na diagramu má různé interpretace a ostatní koncepty v UML nemají žádnou grafickou notaci, takže neexistuje žádný způsob, jak je znázornit. Existují dvě hlavní kategorie diagramů UML, a to diagramy struktury a diagramy chování. Strukturální diagramy zobrazují objekty v modelovaném systému z technického hlediska. Behaviorální diagramy ukazují chování systému. Popisují, jak objekty vzájemně reagují a vytvářejí fungující systém.

3.1 Modelovací koncepty specifikované v UML

Vývoj systému se zaměřuje na tři celkově odlišné modely systému:

- **Funkční:** Jedná se o diagramy případů užití, které popisují funkčnost systému z pohledu uživatele.
- **Objektový:** Jedná se o diagramy tříd, které popisují strukturu systému z hlediska objektů, atributů, asociací a operací.
- **Dynamický:** Interakční diagramy, diagramy stavových strojů a diagramy aktivit se používají k popisu interního chování systému.

Objekty v UML jsou entity reálného světa. Při vývoji softwaru lze objekty použít k popisu nebo modelování vytvářeného systému. Níže jsou popsány základní koncepty objektově orientovaného světa a jejich vlastnosti:

- **Třídy** – definují strukturu a funkce objektu této třídy.
- **Objekty** – proměnné tříd, pomáhají rozkládat velké systémy, a též nám pomáhají je modulovat. Modularita pomáhá rozdělit náš systém na srozumitelné komponenty, abychom mohli náš systém budovat „kousek po kousku“. Objekt je základní blok systému, který se používá k zobrazení entity.
- **Dědičnost** – je mechanismus, kterým podřízené třídy dědí vlastnosti svých nadřazených tříd.
- **Abstrakce** – je mechanismus, kterým jsou podrobnosti implementace skryty před uživatelem.
- **Zapouzdření** – je spojení dat dohromady, jejich ochrana před vnějším světem se označuje jako zapouzdření.
- **Polymorfismus** – je mechanismus, pomocí kterého mohou funkce nebo entity existovat v různých formách.

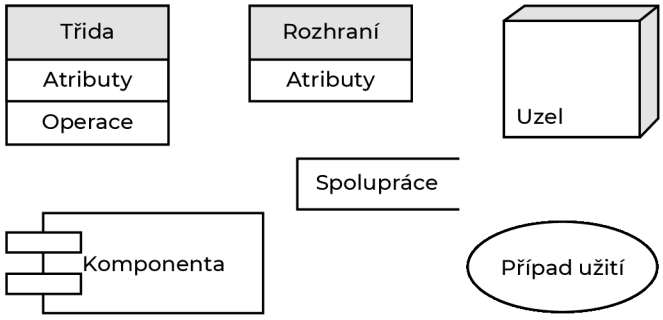
Elementy používané v UML lze rozdělit na prvky, vztahy a diagramy. Tyto elementy si pro lepší pochopení modelovacího jazyka UML popíšeme níže.

3.2 Prvky

Prvky jsou nejdůležitějšími stavebními kameny UML. Prvky mohou být, strukturální, behaviorální, sdružovací či anotační.

Strukturální prvky – definují statickou část modelu. Představují fyzické a koncepční elementy následované stručným popisem konstrukčních elementů.

- Třída – představuje sadu objektů s podobnou odpovědností.
- Rozhraní – definuje sadu operací, které určují odpovědnost třídy.
- Spolupráce – definuje interakci mezi prvky.
- Případ užití – představuje soubor akcí prováděných systémem pro konkrétní cíl.
- Komponenta – popisuje fyzickou část systému.
- Uzel – lze definovat jako fyzický prvek, který existuje za běhu.



Obrázek 11 – Strukturální prvky

Zdroj: Vlastní zdroj

Behaviorální prvky – se skládají z dynamických částí modelů UML. Zobrazují aspekty týkající se chování:

- Interakce – je definována jako chování, které se skládá ze skupiny zpráv vyměňovaných mezi prvky za účelem provedení určitého úkolu.
- Stavový stroj – je užitečný, když je důležitý stav objektu v jeho životním cyklu. Definuje posloupnost stavů, kterými objekt prochází v reakci na události. Události jsou vnější faktory odpovědné za změnu stavu.

Obrázek 12 – Behaviorální prvky

Zdroj: Vlastní zpracování

Prvky seskupování – lze definovat jako mechanismus pro seskupování prvků modelu UML dohromady. K dispozici je pouze jeden prvek seskupení:

- Balíček – je jediný prvek seskupování, který je k dispozici pro shromažďování strukturálních a behaviorálních prvků.

Anotační prvky – s poznámkami lze definovat jako mechanismus pro zachycení poznámek, popisů a komentářů prvků modelu UML.

- Poznámka – je jediný dostupný prvek s poznámkami. Poznámka se používá k vykreslení komentářů a omezení.

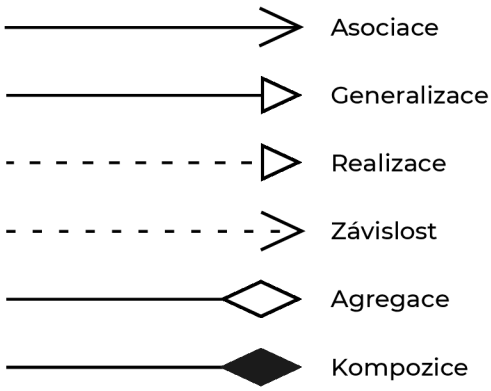
3.3 Vztahy

Vztah je dalším důležitým „stavebním kamenem“ UML. Vztahy přidávají informace do diagramu tím, že objasňují způsob, jakým prvky interagují, nebo na sobě závisí. Popisují chování, které je žádoucí, nebo které lze očekávat mezi prvky.

- **Závislost** – v UML naznačuje, že zdrojový prvek, nazývaný také klient, a cílový prvek, nazývaný také dodavatel, spolu souvisí tím způsobem, že zdrojový prvek využívá nebo závisí na cílovém prvku. Změny v chování nebo struktuře cíle mohou znamenat změny ve zdroji.
- **Asociace** – je v podstatě sada odkazů, které spojují prvky modelu UML. Také popisuje, kolik objektů se tohoto vztahu účastní.
- **Generalizace** – je vztah zobecnění v UML, může existovat mezi konkrétním prvkem a obecnějším prvkem stejného druhu. Konkrétní prvek dědí atributy, vztahy a další charakteristiky z obecného prvku.

Typy, nediferencované třídy, implementační třídy a rozhraní mohou využívat zobecněné vztahy. Za zobecnění lze považovat vztah rodič-dítě, kdy dítě dědí od rodiče, a může tedy přistupovat, využívat strukturu a chování rodičovského prvku.

- **Realizace** – je vztah mezi dvěma prvky v diagramu UML, kde jeden prvek specifikuje chování a druhý prvek toto chování implementuje nebo vykonává, jinými slovy realizuje. Existuje zdrojový prvek (nazývaný realizační prvek) a cílový prvek (nazývaný prvek specifikace). Vztah mezi nimi je označován jako vztah mezi dodavatelem a klientem. V mnoha případech je prvkem specifikace rozhraní nebo soubor operací, přičemž prvkem realizace je implementace těchto chování nebo operací.
- **Agregace** – je typ asociačního vztahu, který označuje, že prvek je tvořen souborem jiných prvků. Například, společnost má oddělení nebo knihovna má knihy. Souhrnný prvek se spoléhá na jiné prvky jako části, ale tyto další prvky mohou existovat i nezávisle na něm.
- **Kompozice** – je jiný typ agregačního vztahu, ve kterém prvky části nemohou existovat bez agregátu. Například, místnosti v domě nemohou dále existovat, pokud je dům zničen.
- **Multiplicita** – určuje mohutnost mezi prvky. Například. 1 ku 1.

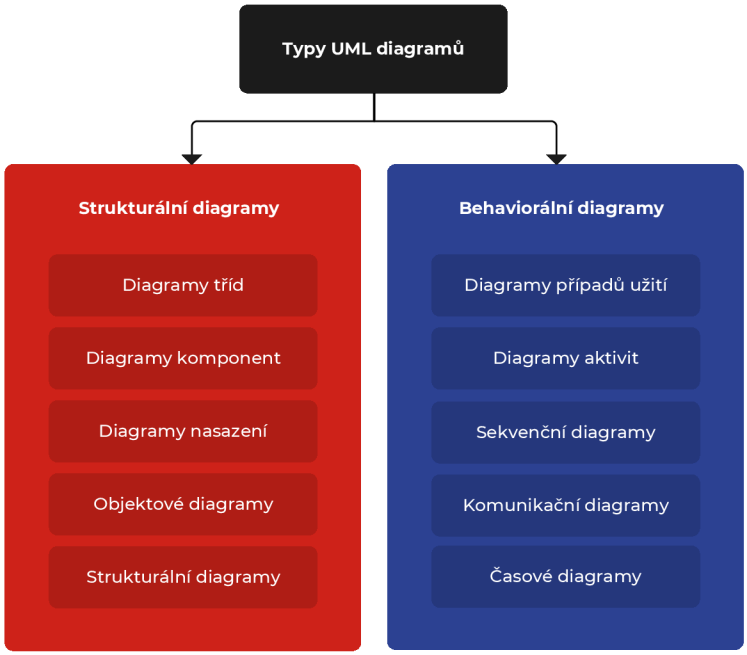


Obrázek 13 – Vztahy

Zdroj: Vlastní zpracování

3.4 Diagramy

UML diagramy jsou konečným výstupem celé diskuse o modelování. Všechny prvky, vztahy jsou použity k vytvoření kompletního UML diagramu, diagram představuje systém. UML diagram je nejdůležitější součástí celého procesu. Všechny ostatní prvky slouží k tomu, aby mohl být diagram kompletní. Na obrázku níže můžeme vidět rozdělení a typy UML diagramů. Digramy se dělí na dva typy. Těmito typy jsou strukturální a behaviorální diagramy.



Obrázek 14 – Typy UML diagramů

Zdroj: Vlastní zpracování

Strukturální diagramy

- **Diagramy tříd** – jsou hlavním stavebním kamenem jakéhokoli objektově orientovaného řešení. Zobrazují třídy v systému, atributy a operace každé třídy a vztah mezi každou třídou. Ve většině modelovacích nástrojů má třída tři části. Název nahoře, atributy uprostřed a operace nebo metody dole. V systému s mnoha souvisejícími třídami jsou třídy seskupeny a vytvářejí

diagramy tříd. Různé vztahy mezi třídami jsou znázorněny různými typy šipek.

- **Diagramy komponent** – zobrazují strukturální vztah komponent softwarového systému. Většinou se používají při práci se složitými systémy s mnoha komponenty. Komponenty spolu komunikují pomocí rozhraní. Rozhraní jsou propojena pomocí konektorů.
- **Diagramy nasazení** – ukazují hardware systému a software v tomto hardwaru. Schéma nasazení je užitečné, když je softwarové řešení „nasazeno“ na více počítačích, přičemž každý má jedinečnou konfiguraci.
- **Objektové diagramy** – někdy označované jako instanční diagramy, jsou velmi podobné diagramům tříd. Stejně jako diagramy tříd také ukazují vztah mezi objekty, ale používají příklady z reálného světa. Ukazují, jak bude systém v danou dobu vypadat., protože jsou v objektech k dispozici data, používají se k vysvětlení složitých vztahů mezi objekty.
- **Diagramy balíčků** – jsou diagramy, které ukazují závislosti mezi různými balíčky v systému.

Behaviorální digramy

- **Diagramy případu užití** – zobrazují grafický přehled aktérů zapojených do systému, různých funkcí potřebných těmito aktéry a způsobu interakce těchto různých funkcí. Je to dobrý výchozí bod pro jakoukoliv diskusi o projektu, protože lze z modelu snadno identifikovat hlavní zúčastněné subjekty a hlavní procesy systému.
- **Diagramy aktivit** – představují pracovní toky grafickým způsobem. Lze je použít k popisu obchodního pracovního postupu nebo provozního pracovního postupu jakékoliv komponentu v systému.
- **Sekvenční diagramy** – v UML ukazují, jak objekty interagují navzájem, a v jakém pořadí k těmto interakcím dochází. Je důležité si uvědomit, že ukazují interakce pro konkrétní scénář. Procesy jsou zobrazeny svisle, interakce jsou zobrazeny jako šipky.

- **Komunikační diagramy** – jsou podobné sekvenčním diagramům, ale důraz je kladen na zprávy předávané mezi objekty. Stejné informace lze znázornit pomocí sekvenčního diagramu a různých objektů. V UML 1 se jim říkalo diagramy spolupráce.
- **Časové diagramy** – jsou typ UML diagramů popisující chování nebo interakce, zaměřují se na procesy, které probíhají v určitém časovém období. Jsou speciální instancí sekvenčního diagramu, kromě toho, že se čas zvyšuje zleva doprava místo shora dolů.

4 Grafický design

Existuje mnoho výzkumů, které naznačují, že „první dojem“ z webové stránky je definován kvalitou designu webu. Zjednodušeně řečeno, vizuální stránka webu určuje, zda uživatel na webu zůstane, či nikoliv. Díky tomu je fáze návrhu jednou z nejdůležitějších fází webového projektu. Návrh webu vyžaduje důkladné plánování, a s tím je spojeno také mnoho technických záležitostí. V této kapitole se seznámíme s teorií grafického designu, principy a postupy tvorby webového designu a uživatelského rozhraní a s tvorbou vizuální identity.

4.1 Fáze tvorby webového designu

Vytvořit webovou stránku, která je vizuálně přitažlivá, intuitivní a která se odlišuje od konkurence, je náročný úkol. Rozdělením návrhu do snadno zvládnutelných fází se toho však dá docílit. Těmito fázemi jsou: shromáždění počátečních informací, návrh, vývoj a testování použitelnosti.

Návrh architektury webu a *wireframy* zajistí, že byly zváženy všechny klíčové stránky na webu, ukazující jejich vzájemný vztah a definující, jak by měla být strukturována celková navigace. *Wireframy* poskytují detailní pohled na obsah, který se objeví na každé stránce. Přestože *wireframy* nezobrazují žádné skutečné prvky návrhu, poskytují vodítko pro definování hierarchie obsahu na stránce. Tvorba *wireframu* je levný a časově nenáročný způsob, jak zobrazit ne zcela věrný způsob zobrazení designu. Jedná se o grafické znázornění aplikace nebo webu obsahující nejdůležitější prvky a obsah. *Wireframe* je podobný plánu stavby

budovy. Při stavbě masivní budovy se nezačíná hned správně. Místo toho se načrtává, kreslí, vytváří plány, počítá atd. Totéž se děje s designem webových stránek a aplikací. Nelze začít hned, protože tím je podstoupeno riziko, že něco přehlédneme, nebo že bude chybět nezbytně důležitý prvek. *Wireframe* se vyznačuje následujícími vlastnostmi:

- Zobrazuje hlavní části obsahu.
- Určuje obrys a strukturu rozvržení.
- Zobrazuje nejzákladnější uživatelské rozhraní.

Jednou z obrovských výhod vytváření *wireframu* je skutečnost, že není tak nákladný jako tvorba *mockupu a*, jeho vytvoření je rychlé. Je proto možné ho ukázat potenciálním uživatelům nebo klientům a požádat o zpětnou vazbu, což je užitečné, protože budou věnovat více pozornosti funkčnosti a uživatelské zkušenosti než estetice. V každém případě je později třeba doladit vizuální stránku.

Vizuální design, tedy tvorba vizuálního stylu, je dalším krokem po vytvoření struktury webu a *wireframů*, které definují plán webu. Celkový vizuální styl musí být v souladu s vizuální značkou organizace. Cílem je propojit web se všemi ostatními formami komunikace organizace. Značka organizace hraje v této části procesu důležitou roli, protože je třeba v rámci designu vizuálně zprostředkovat klíčové myšlenky jejího vnímání. Tvorba *mockupu* je vizuální způsob reprezentace produktu. Zatímco *wireframe* většinou představuje strukturu produktu, *mockup* ukazuje, jak produkt bude vypadat. Na rozdíl od wireframu, je *mockup* buď středním, nebo vysoce věrným zobrazením designu. *Mockup* pomáhá učinit konečná rozhodnutí ohledně barevných schémat produktu, vizuálního stylu, typografie. S *mockupem* je možné si dovolit experimentovat s vizuální stránkou produktu a zjistit, co vypadá nejlépe. Na rozdíl od *wireframu*, *mockup* nelze načrtnout. Pro vytvoření *mockupu* je třeba speciální software, jako je např. Sketch, Adobe XD, nebo Figma.

4.2 Tvorba vizuální identity

Vizuální identita byla vždy důležitou součástí podnikání, ale nyní je důležitější než kdykoli předtím. Díky sociálním sítím jsou spotřebitelé každý den vystaveni

novým značkám. To může být výhodné pro spotřebitele, kteří díky tomu mají spoustu možností, a jsou schopni provést průzkum, aby našli tu nejlepší, ale firmám to počínání znesnadňuje. V dnešní době existuje velká konkurence, což znamená, že podniky musí udělat „něco navíc“ a zajistit, aby vynikly v davu. Aby toho bylo dosaženo, je třeba investovat do vytvoření silné značky, která si získá a udrží pozornost publika. Se správně nastavenou vizuální identitou je šance získat určitou kontrolu nad tím, jak lidé firmu vnímají.

Silná značka vyniká na hustě přeplněném trhu. Lidé se do značek zamilují, důvěřují jim a věří v jejich nadřazenost. To, jak je značka vnímána, ovlivňuje její úspěch, bez ohledu na to, zda se jedná o start-up, neziskovou organizaci nebo produkt. Tvorba vizuální identity je disciplinovaný proces používaný k budování povědomí a rozšíření loajality zákazníků. Tvorba vizuální identity je o využití každé příležitosti k vyjádření toho, proč by si lidé měli vybrat jednu značku před jinou. Touha vést, předběhnout konkurenci, dát zaměstnancům ty nejlepší nástroje k oslovení zákazníků, jsou důvody, proč společnosti využívají vizuální identitu. Efektivní strategie značky poskytuje centrální, sjednocující myšlenku, kolem níž je sladěno veškeré chování, akce a komunikace. Funguje napříč produkty a službami a je efektivní v průběhu času.

4.3 Design webových stránek

Webový design má několik základních principů, které je nutné dodržet pro docílení pozitivní uživatelské zkušenosti. Důležitým krokem tvorby webového designu je vytvoření vhodné barevné palety. Doplňkové barvy vytvářejí rovnováhu a harmonii. Použití kontrastních barev pro text a pozadí usnadní čtení. Saturované barvy vyvolávají emoce a měly by být používány střídmě. Je vhodné je použít například pro tlačítka a výzvy k akci.

Výběr vhodných obrázků pro web může pomoci s navázáním kontaktu s cílovým publikem. Je dobré také zvážit použití infografiky a videa, protože tento typ komunikace je mnohem efektivnější než samotný text.

Kromě udržení konzistentnosti navigace na webu by měl být celkový vzhled webu shodný na všech stránkách. Pozadí, barevná schémata, písmo, a dokonce i tón psaní jsou oblasti, ve kterých může být důsledností pozitivně ovlivněna

použitelnost aplikace. Není však podmínkou, aby každá stránka na webu měla přesně stejné rozvržení. Místo toho je vhodné vytvořit různá rozvržení pro konkrétní typy stránek. Důsledným používáním těchto rozvržení usnadníme návštěvníkům pochopit jaký typ informací na dané stránce pravděpodobně najdou.

Při tvorbě designu není vhodné celou obrazovku naplnit obsahem. Prázdný prostor je stejně důležitý, protože poskytuje uživatelům možnost si „vizuálně“ oddechnout. Na webových stránkách s informacemi je kognitivní zatížení opravdu únavné a jediný způsob, jak jej vyrovnat, je negativní prostor. Slouží k rozdělení obsahu do několika oblastí, a k vytvoření cesty, která uživatelům pomáhá snáze získávat informace.

Každý web má jedinečné informace, které je třeba zobrazit. Pro snadné dosažení konkrétních informací by navigace a organizace prvků měla být intuitivní. Aby uživatel nebyl frustrovaný, je třeba, aby elementy na webu byly organizovány, a uspořádány. Návštěvníci zůstávají více času na webových stránkách, které mají snadnou navigaci. Pro efektivní navigaci je třeba zvážit vytvoření logické hierarchie.

Vzhledem k variabilitě velikostí používaných elektronických zařízení musí být webový design přizpůsobený pro různé obrazovky. Responzivní web je navržen tak, aby reagoval, nebo se přizpůsobil na základě technologie a typu výpočetního zařízení, které návštěvník používá k zobrazení webu. Je to jedno designové řešení webových stránek, které bude vypadat dobře v jakékoliv velikosti – od velkého stolního LCD monitoru po malé obrazovky, které používáme na tzv. chytrých telefonech. Díky responzivnímu designu mají návštěvníci webu podobnou zkušenost, která je nezávislá na velikosti zařízení používaného k prohlížení webu. Pokud webové stránky nepodporují všechny velikosti obrazovky, je pravděpodobné, že stránky ztratí potencionální návštěvníky. Text je nejdůležitějším prvkem na webu, protože poskytuje uživatelům požadované informace. Typografie by tedy měla být pro návštěvníky vizuálně přitažlivá a čitelná. Je vhodné zvážit použití písem, která jsou snáze čitelná.

4.4 Barvy

Oči něco vidí (např. oblohu) a data odeslaná z očí do mozku je interpretují jako určitou barvu (např. modrou). Předměty odrážejí světlo v různých kombinacích vlnových délek. Mozek zachytí tyto kombinace vlnových délek a převede je do jevu, který nazýváme barva. Barvu lze chápat jako speciální jazyk umožňující (na rozdíl od jiných jazyků) bezprostřední komunikaci. Tento intuitivní smyslový jazyk nevyžaduje podobně jako hudba žádná slova. Lidé se rozhodnou, zda se jim produkt líbí za méně než 90 sekund. Velká část tohoto rozhodnutí je založena pouze na barvě. Velmi důležitá část značky by se tedy měla zaměřit na barvu.

Odstín – je základní vlastnost barvy, podle níž též barvy povětšinou pojmenováváme (zelená, žlutohnědá, vínová). Dalšími vlastnostmi barvy jsou jas a sytost, které ovšem, nedosahují-li extrémních hodnot, jsou pro rozpoznávání odstínů spíše doplňkové. Změna odstínu představuje pohyb po spektru barev.

Obrázek 15 – Odstíny barvy

Zdroj: Vlastní zpracování

Sytost – popisuje intenzitu barvy, spolu s odstínem a hodnotou představuje jednu ze tří vlastností barvy. Čím více je barva sytá, tím je vnímána jako živější. Naopak méně syté barvy jsou vnímány jako nudné či zašedlé.

Obrázek 16 – Sytost barvy

Zdroj: Vlastní zpracování

Teplota – představuje vnímané teplo nebo chlad barvy. Žlutá, oranžová a červená a variace těchto tří barev jsou teplé barvy. Jedná se o barvy ohně, západu a východu slunce, obecně jsou „vášnivé“ a pozitivní. Červená a žlutá jsou obě primární barvy. Oranžová je sekundární barva a nachází se uprostřed nich, což znamená, že teplé barvy jsou skutečně teplé a nevznikají kombinací teplé barvy s chladnou barvou. Zelená, modrá a fialová jsou chladné barvy. Jsou to barvy noci, vody, přírody a obvykle jsou uklidňující a relaxační. Modrá je primární barva v chladném spektru. Ostatní chladné barvy jsou vytvářeny kombinací modré a teplé barvy. Kombinace modré se žlutou tvoří zelenou, modrá s červenou fialovou.

Obrázek 17 – Teplota barvy

Zdroj: Vlastní zpracování

Hodnota – je relativní stupeň světlosti nebo tmavosti barvy. Přidání bílé nebo černé změní hodnotu barvy. Hodnota může být použita pro zdůraznění. Světlá postava na tmavém pozadí ihned upoutá pozornost, podobně jako tmavá postava na převážně bílém pozadí. Postupná změna hodnoty se také používá k vytvoření iluze hloubky. Místa světla a tmy mohou působit trojrozměrně, například při zastínění oblastí nějakého objektu.

Obrázek 18 – Hodnota barvy

Zdroj: Vlastní zpracování

Kontrast – Každý odstín barvy má opačný odstín, jehož kontrast k původnímu odstínu je mnohem větší než k jakékoliv jiné barvě. Pomocí barevného kola

na obr. 19 můžeme najít opak každé konkrétní barvy. Jednoduše vyhledáme barvu na opačném konci kruhu.

Barvy obecně v lidech vyvolávají mnoho emocí. Ať už jsou jejich preference jakékoli, většina lidí pozitivně reaguje na harmonické používání barev. Harmonii lze definovat jako příjemné uspořádání částí, stejně jako v hudbě. Harmonická kombinace barev vytváří vnitřní smysl pro pořádek a vizuální rovnováhu, která zaujme. Harmonie barev jednoduše znamená, že barevná kompozice není chaotická či neuspořádaná.

Pro reprezentaci barev je možné využít barevné kolo, které poskytuje vizuální reprezentaci toho, které barvy jsou harmonické (tj. hodí se k sobě). Většina modelů se skládá z 12 barev. Teoreticky by se však barevné kolo mohlo rozšířit o nekonečný počet odstínů. Bylo vyvinuto mnoho modelů pro vizuální srovnání barev a jejich interakce s jinými barvami. Tyto diagramy se běžně používají k zobrazení vztahů mezi jednotlivými odstíny barev. Barvy jsou roztříděny podle toho, jak reagují na jiné odstíny. Pochopení nejznámějších typů párování barev umožní vybrat si ty barvy, které mají předvídatelné pozitivní vizuální výsledky.

Obrázek 19 – Barevné kolo

Zdroj: Johannes Itten

Tato reprezentace je obzvláště zajímavá. Uprostřed se nachází barvy prvního řádu: modrá, červená, žlutá. Okolo jsou barvy druhého řádu: zelená, oranžová, fialová, získány smícháním barev prvního řádu modro-žluté, žluto-červené, červeno-modré. Pokud do kola vložíme pravidelný tvar, jako je čtverec, obdélník nebo rovnostranný trojúhelník, barvy, které dostaneme na vrcholech, jsou harmonické.

Ve svém textu využijeme v designu používaný model RYB (zkratka označuje červená, žlutá, modrá), což je model subtraktivního míchání barev, který je běžně používanou sadou základních barev. Je primárně používán při výuce umění. RYB (červená, žlutá, modrá) tvoří primární trojici barev ve standardním uměleckém barevném kruhu. Sekundární barvy fialová, oranžová, zelená jsou další trojicí. Trojice jsou tvořeny třemi rovnoměrně vzdálenými barvami na zvoleném barevném kruhu. Další běžně používané modely počítačové grafiky jsou např. RGB a CMYK (pro tisk).

Primární barvy – se skládají z červené, žluté a modré. Jsou to čisté odstíny, které na sobě nezávisí. Smícháním správného množství primárních barev je možné vytvořit jakoukoli barvu ve spektru.

Obrázek 20 – Primární barvy

Zdroj: Vlastní zpracování

Sekundární barvy – jsou fialová, oranžová a zelená. Tyto barvy jsou vytvořeny kombinací stejného množství dvou základních barev.

Obrázek 21 – Sekundární barvy

Zdroj: Vlastní zpracování

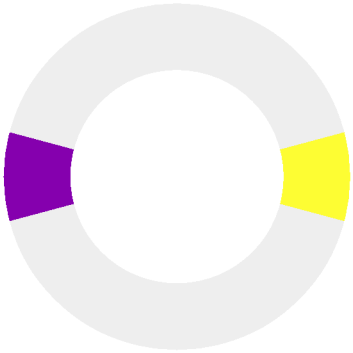
Terciární barvy – jsou umístěny mezi primární a sekundární barvou na barevném kole a skládají se z více než jedné primární barvy. Jak terciární barva vypadá, závisí na tom, která primární barva je ve směsi dominantní.

Pravděpodobně nejdůležitějším aspektem teorie barev je barevná harmonie, která se týká použití barevných kombinací, které jsou pro lidské oko vizuálně příjemné. Nedostatek harmonie v barevné paletě může mít za následek buď nedostatečně stimulující (nudné), nebo příliš stimulující (chaotické) rozhraní. Existuje několik zavedených schémat, která můžeme k vytvoření harmonických barevných palet použít.

Obrázek 22 – Barevné kolo

Zdroj: Vlastní zpracování

Doplňkové schéma – jsou jakékoli dvě barvy, které jsou proti sobě na barevném kole. Například, modrá a oranžová, nebo červená a zelená. Takové barvy vytvářejí vysoký kontrast, takže je vhodné je použít, když chceme, aby něco vyniklo. V ideálním případě použijeme jednu barvu jako pozadí a druhou pro akcenty. Můžeme také použít různé odstíny; světlejší odstín modré v kontrastu, například s tmavší oranžovou.

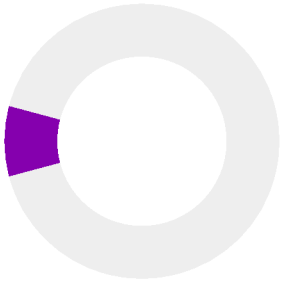


Obrázek 23 – Doplňková paleta

Zdroj: Vlastní zpracování

Rozdělené doplňkové schéma – obsahuje jednu dominantní barvu a dvě barvy přímo sousedící s doplňkem dominantní barvy. Tímto postupem je vytvořena bohatší paleta barevných odstínů než u doplňkového barevného schématu při zachování výhod kontrastních barev. Pozitivní i negativní aspekt rozdělené doplňkové barevné palety spočívá v tom, že ve schématu lze použít libovolné dvě barvy a dosáhnout dobrého kontrastu. Zároveň je ale obtížné najít správnou rovnováhu mezi barvami.

Monochromatické schéma – se skládá z různých tónů v určitém odstínu. Tato schémata se snadno vytvářejí, ale mohou působit nudně, pokud jsou vytvořena špatně. Přidání silné neutrální barvy, jako je bílá nebo černá, pomůže vytvořit dobrou barevnou paletu.

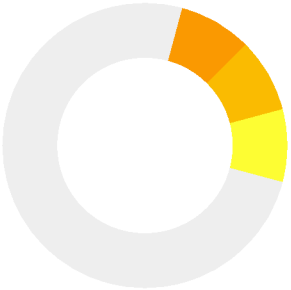


Obrázek 24 – Monochromatická paleta

Zdroj: Vlastní zpracování

Analogické schéma – používá barvy, které jsou na barevném kole vedle sebe, jako je červená a mandarinková, nebo modrá a zelená. Analogická schémata jsou

vytvořena pomocí tří barev. Použitím tónů a odstínů můžeme této paletě přidat na zajímavosti a přizpůsobit ji našim potřebám při navrhování webových stránek.



Obrázek 25 – Analogická paleta

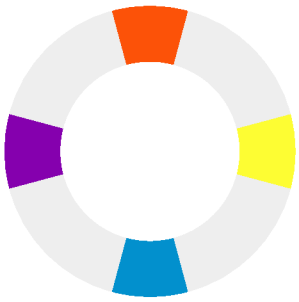
Zdroj: Vlastní zpracování

Triadické schéma – používá tři barvy, které jsou rovnoměrně rozmístěny, a vytvářejí tak rovnoměrný trojúhelník na barevném kole. Primární barvy žlutá, červená a žlutá by mohly být použity společně v barevném schématu k vytvoření živého výsledku.

Obrázek 26 – Triadická paleta

Zdroj: Vlastní zpracování

Tetradické schéma – používá čtyři barvy uspořádané do dvou komplementárních párů. Toto bohaté barevné schéma nabízí spoustu možností variace. Tetradická barevná schémata fungují nejlépe, pokud necháme jednu barvu dominovat. Při návrhu bychom také měli dbát na vyváženost teplých a studených barev.



Obrázek 27 – Tetradická paleta

Zdroj: Vlastní zpracování

Zmíníme ještě barevné modely, které v teorii barev využívané v počítačové grafice, které matematicky popisují, jak mohou být barvy reprezentovány. Většina moderních barevných modelů má tři rozměry (například RGB), a proto je lze zobrazit jako 3D tvary, zatímco jiné modely mají více rozměrů (například CMYK). V následujícím textu zmíníme barevné modely RGB, HSV a HSL, které v současných nástrojích pro digitální design a programovacích jazycích převládají.

- **RGB** – je barevný model se třemi rozměry – hodnotami zastoupení červené, zelené a modré barvy světelného zdroje, které po smíchání vytvářejí určitou barvu. Při definování barev v těchto rozměrech je třeba dodržet posloupnost barev v barevném spektru. Barevný model RGB je často zobrazen jako krychle mapováním červené, zelené a modré dimenze na osy x, y a z v 3D prostoru. Barevný model RGB není zvlášť intuitivní pro vytváření barev v kódu. I když jsme schopni uhodnout kombinaci hodnot pro některé barvy, jako je žlutá (stejné množství červené a zelené), méně čisté barvy jsou u tohoto barevného modelu mnohem těžší uhodnout.
- **HSV** – je barevný model, který používá místo tří primárních barev RGB, tři rozměry, které jsou pro člověka srozumitelnější. Těmito rozměry jsou odstín, sytost a hodnota. Odstín určuje úhel barvy na barevném kruhu RGB. Odstín 0° má červenou barvu, 120° zelenou barvu a 240° modrou barvu. Sytost určuje množství použité barvy. Barva se 100 % nasycením bude nejčistší možnou barvou. Hodnota určuje jas barvy. Barva s 0 % jasem je čistá černá, zatímco barva se 100 % jasem neobsahuje žádnou černou.

Je důležité si uvědomit, že tři rozměry barevného modelu HSV jsou vzájemně závislé. Pokud je hodnotový rozměr barvy nastaven na 0 %, nezáleží na hodnotě odstínu a sytosti, protože barva bude vždy černá.

- **HSL** – je další barevný model, který sdílí dvě dimenze s HSV a nahrazuje hodnotový rozměr rozměrem světlosti. Odstín určuje úhel barvy na barevném kruhu RGB, přesně jako HSV. Sytost řídí čistotu barvy, stejně jako u HSV. Světlost se řídí svítivostí barvy. Tato dimenze se liší od dimenze hodnoty HSV v tom, že nejčistší barva je umístěna uprostřed mezi černými a bílými konci stupnice. Barva s 0 % světlostí je černá, 50 % je nejčistší možná barva a 100 % je bílá. I když je dimenze sytosti teoreticky podobná mezi dvěma barevnými modely (řízení množství použité čisté barvy), výsledné stupnice sytosti se u jednotlivých modelů liší v důsledku přeměny jasu na světlost. Stejně jako HSV je barevný model HSL nejlépe zobrazen jako válec.

4.5 Typografie

Správně použitá typografie usnadňuje čtení obsahu webových stránek, zatímco špatná typografie čtenáře od webu odradí. Při tvorbě webového designu je třeba použít jednoduchý, rozeznatelný a snadno čitelný typ písma. Obsah se ale může stát monotónním, pokud je typografie příliš prostá. Čitelnost je klíčová. Není vhodné použít kaligrafická písma do té míry, že čtenáři stěží rozeznají slova. Použití standardních písem ve skutečnosti může zaujmout čtenářovu mysl lépe, protože je s nimi již podvědomě obeznámen. Jednoduchost zvyšuje čitelnost webu a zvyšuje jeho vizuální přitažlivost. Pro udržení struktury webu je nutné použít minimální počet druhů písma.

Počet znaků v řadě textů do značné míry určuje snadnost, s jakou může uživatel číst a pochopit zprávu. Kratší věty nabízejí lepší čitelnost než delší, a proto by řádky neměly být příliš dlouhé nebo příliš krátké. Obecným pravidlem pro typografii na webu je omezit rozsah znaků na řádek na přibližně 50–60. Správné použití mezer mezi řádky textu nejen zvyšuje viditelnost a vizuální přitažlivost, ale také zlepšuje čitelnost. Pokud mezi řádky textu nebudeme mít optimální mezery, může to způsobit, že web bude vypadat nepřehledně a nebude

přitahovat zájem čtenáře. Pro lepší typografii bychom měli zvážit použití správné výšky řádků mezi řádky textu na svém webu.

Barva hraje v typografii na webu obrovskou roli. Barevný kontrast mezi textem a pozadím zvyšuje čitelnost textů. Aby webové stránky dobře, fungovaly pro cílené publikum, je vhodné porozumět koncovým uživatelům, a brát v úvahu jejich potřeby a očekávání. Je důležité zvážit, pro koho navrhujeme (věková skupina uživatelů, pohlaví apod.). Odpověď na otázky týkající se cílového publika nám pomohou vybrat správné typy písma pro předání zprávy, kterou jim chceme sdělit.

4.6 Logo

Logo může od tvaru, přes barvu, až po typ písma v mysli diváka pomoci vytvořit vnímání značky. Proto je důležité věnovat pozornost drobným detailům. Je třeba vytvořit prvky loga, které odpovídají osobnosti značky. Logo musí odrážet společnost čestným a jedinečným způsobem. Logo musí být lehce rozpoznatelné a dobře zapamatovatelné. Pokud se do prvků loga zakomponují klíčové prvky, které firma představuje, je větší šance, že si ho lidé zapamatují.

Čím jednodušší logo je, tím snazší je ho rozpoznat. Klíčem je vytvoření loga, které je koncepčně jednoduché, aniž by bylo zjednodušené. Je potřeba se ujistit, že logo dobře funguje – barevně i černobíle po převedení do odstínů šedi. Pokud logo nefunguje v černobílém provedení, „bude s ním problém“, například při tisku v novinách. Je potřeba zajistit, aby se barevné provedení loga dobře převedlo na černobílou barvu. Aby logo dobře fungovalo, musí být čitelné ve všech velikostech. Logo musí vypadat stejně dobře jak na hlavičkovém papíře, tak na vizitce. Žádná část loga by neměla převažovat nad druhou. Nejlepší způsob, jak toho dosáhnout, je vzít si příklad z největších obrazů na světě. Spojují určité barvy takovým způsobem, že žádná nevyčnívá nad ostatními, aby odvrátila pozornost od celkového obrazu. Logo by mělo být navrženo tak, aby vydrželo i napříč časem. Je třeba se ujistit, že návrh loga nevyužívá aktuální trend.

5 Databáze a databázové systémy

V této kapitole si přiblížíme důležitost a princip fungování databází. Většina webových stránek využívá nějaký druh databáze. Kdykoli někdo na webu použije například vyhledávání, tak jej databázový server musí zpracovat. Různá průmyslová odvětví vytvořila své vlastní normy pro návrh databáze, od letecké dopravy, až po výrobu vozidel. Databáze je sbírkou dat, která je uchovávána pro různé programy, které tvoří počítačové informační systémy. Hlavním rysem databází je to, že data jsou oddělena od programů, které je používají. Díky tomu mohou různé programy přistupovat a upravovat stejnou databázi a sdílet společná data. Databáze je běžně řízena systémem pro správu databáze (DBMS – *database management system*). Data a systém DBMS společně s přidruženými aplikacemi tvoří databázový systém, často zkráceně označovaný databáze.

5.1 Datové modelování

Datové modelování je proces analýzy datových objektů a jejich vztahu k ostatním objektům. Používá se k analýze požadavků na data, která jsou vyžadována pro obchodní procesy. Datové modely jsou vytvořeny pro data, která mají být uložena v databázi. Datový model se zaměřuje především na to, jaká data jsou potřebná, a jak musíme data organizovat, spíše než na to, jaké operace musíme provádět. Existuje mnoho druhů datových modelů. Mezi nejčastější patří hierarchický databázový model, relační model, síťový model, objektově orientovaný databázový model a mnoho dalších. Datový model je v podstatě architektův stavební plán. Je to proces dokumentace návrhu komplexního softwarového systému v diagramu, který lze snadno pochopit. Diagram bude vytvořen pomocí textu a symbolů, které budou představovat, jak budou data proudit. Je také známý jako plán pro konstrukci nového softwaru nebo přepracování jakékoli aplikace.

5.2 Relační datový model

Nejběžněji používaným modelem je model relační. Třídí data do tabulek, známých také jako relace, z nichž každá se skládá ze sloupců a řádků. V každém sloupci je uveden atribut dané entity, například jméno, adresa nebo věk.

Společně se atributy ve vztahu nazývají doménou. V rámci databáze lze tabulky přivést do souladu s normalizačními pravidly, díky nimž je databáze flexibilní, přizpůsobitelná a škálovatelná. Při normalizaci je každá část dat atomická.

Správně navržená databáze má mnoho výhod. Proces přidávání, úpravy, mazání a načítání dat tabulky je značně usnadněn správně navrženou databází. A co je nejdůležitější, databáze je snadno rozšiřitelná a škálovatelná.

Každá tabulka v systému ukládá data o jedné entitě. Entita obvykle představuje skutečný objekt nebo událost. Příkladem objektů jsou zákazníci, zaměstnanci nebo produkty. Příkladem událostí jsou například objednávky, schůzky a návštěvy u lékaře.

Atribut – je označení pro sloupec tabulky. Jsou to popisné vlastnosti každého řádku tabulky. Bereme v potaz popis zákazníka, který nakupuje na internetu. Chceme uchovat jeho jméno, adresu a kontaktní údaje. Jsou to důležité informace, které pomohou odeslat zásilku. U daného zákazníka tedy můžeme mít uvedeny následující atributy (jméno, příjmení, ulice, číslo popisné, PSČ, telefon, email).

Propojení tabulek – je řešeno pomocí primárního klíče nebo zvláštního sloupce v databázové tabulce, který jedinečně popisuje všechny záznamy.

Normalizace návrhu databáze – je označení pro proces, ve kterém upravujeme strukturu databáze tak, abychom mohli efektivně nakládat s daty, zajistit konzistentnost databáze a zabránit redundanci dat. Pro určitý stupeň normalizace musí databáze splňovat některá pravidla (normální formy). Čím je stupeň normalizace vyšší, tím je databáze navržená lépe, a je jednodušší manipulovat s daty. Každý vyšší stupeň normalizace musí dodržovat pravidla nižších stupňů.

- **První normální forma –** informace jsou uloženy v relační tabulce, kde každý sloupec obsahuje dále nedělitelné neboli atomové hodnoty. Neexistují žádné opakující se skupiny sloupců.
- **Druhá normální forma –** tabulka splňuje podmínky první normální normy, všechny sloupce závisí na primárním klíči tabulky.
- **Třetí normální forma –** k dosažení třetí normální formy musí tabulka splňovat všechny požadavky první a druhé normální formy. Všechny sloupce, které nejsou klíčové, musí být vzájemně nezávislé.

Entitně-relační diagram – je typ vývojového diagramu, který ilustruje, jak se entity, jako jsou například lidé, objekty nebo koncepty, vzájemně propojují v systému. Tyto diagramy se nejčastěji používají k návrhu nebo ladění relačních databází v oblasti softwarového inženýrství, podnikových informačních systémů, vzdělávání a výzkumu. Používají definovanou sadu symbolů, jako jsou obdélníky, kosočtverce, ovály a spojovací čáry, aby znázornily propojenost entit, vztahů a jejich atributů.

5.3 Jazyk SQL

SQL je jazyk sloužící ke komunikaci s databází. Pomocí dotazů dokáže manipulovat s daty a vytvářet nové databáze, databázové tabulky či nastavovat přístupová práva. Všechny příkazy SQL začínají některým z klíčových slov, jako je například SELECT, INSERT, UPDATE, DELETE, ALTER, DROP, CREATE, USE, SHOW. Všechny příkazy jsou zakončeny středníkem (;). SQL nerozlišuje velká a malá písmena, což znamená, že příkazy SELECT a select mají v příkazech SQL stejný význam.

```
SELECT "sloupec1", "sloupec2" FROM "nazev_tabulky" WHERE "podmínka";
```

Příkaz SELECT se používá k dotazování na databáze a načtení vybraných dat, která odpovídají zadaným kritériím. Zde je formát jednoduchého příkazu pro výběr:

```
SELECT "sloupec1", "sloupec2" FROM "nazev_tabulky" WHERE "podmínka";
```

Názvy sloupců, které následují po vybraném klíčovém slovu, určují, které sloupce budou vráceny ve výsledcích. Je možné vybrat libovolný počet názvů sloupců, nebo je možné vybrat všechny sloupce pomocí *.
Název tabulky, která následuje za klíčovým slovem FROM, určuje tabulku, která bude dotazována pro získání požadovaných výsledků. Klauzule WHERE, která je volitelná, určuje, které datové hodnoty nebo řádky se vrátí, nebo zobrazí, na základě kritérií popsaných za klíčovým slovem WHERE.

Operátor přiřazování vzorů LIKE lze také použít při podmíněném výběru klauzule WHERE. LIKE je velmi výkonný operátor, který umožňuje vybrat pouze

řádky, které jsou „jako“ to, co zadáte. Znak procenta % lze použít jako *divokou kartu*, shoduje se s jakýmkoli možným znakem, který se může objevit před nebo za zadanými znaky, uvedeme na příkladu níže:

```
SELECT * FROM uzivatele WHERE prvni LIKE 'Er%';
```

Tento příkaz SQL vrátí všechna křestní jména začínající na písmeno R. Řetězce musí být v jednoduchých uvozovkách.

Příkaz INSERT se používá k vložení řádku dat do tabulky. Pro vložení záznamů do tabulky se používá klíčové slovo INSERT. Dále je třeba definovat název tabulky, do které se záznam má vložit, názvy sloupců, do kterých se mají vkládat data a požadované hodnoty.

```
INSERT INTO "nazev_tabulky" (sloupec1, sloupec2) VALUES (hodnota1,
hodnota);
```

Příkaz DELETE se používá k odstranění záznamů nebo řádků z tabulky. Pokud je vynechána klauzule WHERE, budou vymazány všechny záznamy.

```
DELETE FROM "nazev_tabulky" WHERE "nazev_sloupce" OPERATOR "hodnota"
[and|or "sloupec" OPERATOR "hodnota"];
```

Příkaz CREATE TABLE se používá k vytvoření nové tabulky. Níže je uveden formát jednoduchého příkazu pro vytvoření tabulky:

```
CREATE TABLE "nazev_tabulky" ("sloupec1" "datový typ", "sloupe2"
"datový typ", "sloupec3" "datový typ");
```

Pro odstranění celé tabulky (včetně všech řádků) je možné použít příkaz DROPTABLE následovaný názvem tabulky.

```
DROP TABLE "nazev_tabulky"
```

Pohledy – jsou v SQL druhem virtuálních tabulek. Pohled obsahuje řádky a sloupce, stejně jako skutečná tabulka v databázi. Pohled je možné vytvořit výběrem polí z jedné nebo více tabulek v databázi. Pohled může mít na základě určitých podmínek buď všechny řádky tabulky, nebo pouze konkrétní vybrané řádky. Pohled je možné vytvořit pomocí příkazu CREATE VIEW. Lze jej vytvořit z jedné tabulky, více tabulek, nebo jiného pohledu. Pro vytvoření pohledu musí mít uživatel příslušné systémové oprávnění podle konkrétní implementace.

Uložené procedury – jsou v. SQL kolekce příkazů SQL, které jsou kompilovány, a uloženy do databáze. Uložené procedury v SQL umožňují vytvářet dotazy SQL, které se ukládají a provádějí na serveru. Uložené procedury lze také ukládat do mezipaměti a znovu je použít. Hlavním účelem uložených procedur je skrýt přímé dotazy SQL, jako je výběr, aktualizace a mazání dat před kódem a zlepšit výkon operací databáze. Uložené procedury je možné vytvořit a spustit pomocí Průzkumníku objektů v SQL serveru nebo pomocí SQL Server Management Studio (SSMS). Na serveru SQL server jsou k dispozici dva typy uložených procedur:

- uživatelem definované uložené procedury
- systémové uložené procedury

Uživatelem definovaná uložená procedura může převzít vstupní parametry a návratové výstupní parametry. Uživatelem definované uložené procedury jsou dále rozděleny do dvou typů:

- Uložené procedury T-SQL: (Transact SQL) přijímají a vrací parametry. Tyto uložené procedury zpracovávají dotazy Vložit, Aktualizovat a Smazat s parametry nebo bez parametrů a vracejí data řádků jako výstup. Je to jeden z nejčastějších způsobů, jak psát uložené procedury na serveru SQL.
- Uložené procedury CLR: (Common Language Runtime) jsou psány v programovacím jazyce založeném na CLR, jako je C # nebo VB.NET, jsou prováděny rozhraním .NET Framework.

Systémové uložené procedury jsou vytvářeny a prováděny SQL serverem pro administrativní činnosti serveru. Vývojáři obvykle nezasahují do systémových uložených procedur.

6 Vývojové platformy

V této kapitole si přiblížíme vývojové platformy, které obecně označují operační systém a počítačový hardware. Platforma je soubor standardů, které umožňují vývojářům vyvíjet softwarové aplikace založené na správném technologickém zásobníku.

6.1 .NET

.NET je *open source* vývojová platforma vytvořená společností Microsoft pro vytváření mnoha různých typů aplikací. Aplikace . NET je možné psát v C#, F#, Visual C++ nebo Visual Basic. NET podporuje Common Language Infrastructure (CLI), což znamená, že zdrojový kód je kompilován do Common Intermediate Language (CIL), nezávisle na programovacím jazyce, který je používán. To zaručuje skvělou interoperabilitu mezi jazyky na platformě. Architektura .NET je založena na dvou hlavních komponentech:

- CoreCLR: toto je běhové prostředí .NET. Je zodpovědné za spouštění programů CLI a obsahuje kompilátor *just-in-time*.
- CoreFX: API platformy implementující standardní knihovny CLI, tedy sadu knihoven, které poskytují nejběžnější funkce, jako je správa systému souborů, zpracování výjimek, síťová komunikace, vytváření vláken, reflexe aj. Komponent CoreFX se někdy nazývá Unified Base Class Library.

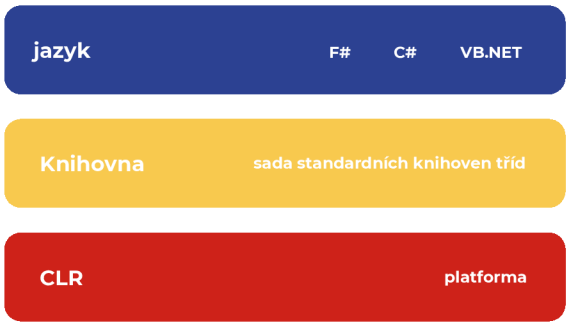
Nad základními komponenty jsou různé aplikační modelové rámce, tedy knihovny, které nabízejí podporu pro vývoj různých typů aplikací. V naší práci si uvedeme tyto:

- ASP.NET: aplikační rámec, který umožňuje vytvářet webové aplikace a webová rozhraní API.
- Windows PresentationFoundation (WPF): grafické uživatelské rozhraní pro desktopové aplikace Windows.

- Xamarin: rámec pro vytváření multiplatformních mobilních, televizních a desktopových aplikací.
- Blazor: rámec pro vytváření klientských webových aplikací pomocí C#. Umožňuje také generovat klientské webové aplikace v kódu Web Assembly.
- ML.NET: Aplikační rámec strojového učení, který zjednodušuje integraci modelů strojového učení v aplikaci .NET.

6.2 ASP.NET

ASP.NET je aplikační rámec pro webové aplikace vyvinutý a společností Microsoft, který umožňuje programátorům vytvářet dynamické webové stránky. Umožňuje používat programovací jazyk, jako je C # nebo VB.NET. Základní architektura aplikačního rámceASP.NET je znázorněna níže.



Obrázek 28 – Architektura ASP.NET

Zdroj: Vlastní zpracování

Architektura aplikačního rámce .NET je založena na následujících klíčových komponentech:

1. **Jazyk** – pro aplikační rámec.net existuje celá řada jazyků jako například VB.net a C#. Lze je použít k vývoji webových aplikací.
2. **Knihovna** –aplikační rámec .NET obsahuje sadu standardních knihoven tříd. Nejběžnější knihovnou používanou pro webové aplikace v .net je webová knihovna. Webová knihovna má všechny potřebné komponenty používané k vývoji webových aplikací založených na síti.

3. **Common Language Runtime** – Common Language Infrastructure nebo CLI je platforma. Na této platformě jsou spouštěny programy .NET. CLR, používá se k provádění klíčových činností.

ASP.NET podporuje tři hlavní vývojové modely, těmi jsou webové stránky, webové formuláře a MVC (model-view-controller). ASP.NET MVC je nenáročný a dobře testovatelný aplikační rámec, který je integrován s existujícími funkcemi ASP.NET jako je například ověřování. V rámci .NET je tento aplikační rámec definován v sestavení System.Web.Mvc. Nejnovější verze MVC aplikačního rámce je 5.0.

MVC – (model, view, controller) je architektonický vzor, který odděluje aplikaci od hlavních logických komponent, kterými jsou model, pohled a kontrolér. Každá z těchto komponent je vytvořena pro zpracování specifických aspektů vývoje aplikace. MVC je jedním z nejčastěji používaných průmyslových standardů pro vývoj webových aplikací k vytváření škálovatelných a rozšiřitelných projektů.

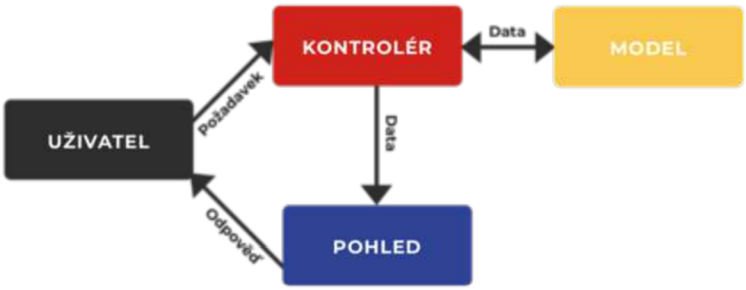
Komponenty mohou fungovat samostatně. Model, pohled a kontrolér na sobě nezávisí, což je důležité, protože obecně platí, že na softwaru pracují týmy, a ty mohou být velké. Rozdělení aplikace do komponent znamená, že členové týmu mohou pracovat na aplikaci současně. Rozdělení aplikace je také dobré pro údržbu, vývojáři mohou opravit chybu v jednom kódu, aniž by museli kontrolovat ostatní části kódu. MVC zjednodušuje i náročnost vývoje projektu. Definuje tři rozdílné části na třech různých místech, izolované tak, aby se navzájem neovlivňovaly, a nezpůsobily větší komplikovanost vývoje.

Model – představuje základní logiku a data. Modely poskytují vlastnosti a chování doménové entity, odhalují vlastnosti, které tuto entitu popisují. Například, třída Produkt představuje koncept *produkt* v aplikaci, vyznačuje se vlastnostmi, jako je název nebo cena.

Pohled – je zodpovědný za transformaci modelů do vizuální reprezentace. Ve webových aplikacích to nejčastěji znamená generování HTML, které se má vykreslit v uživatelském prohlížeči. Pohledy se ale mohou projevovat v mnoha podobách. Například, stejný model může být vizualizován v HTML, PDF, XML, nebo v tabulce.

Kontrolér – řídí aplikační logiku a působí jako koordinátor mezi pohledem

a modelem. Kontroléry přijímají vstupy od uživatelů prostřednictvím pohledu, poté pracují s modelem, provádějí konkrétní akce a výsledky předávají zpět do pohledů.



Obrázek 29 – Blokové schéma návrhového vzoru MVC

Zdroj: Vlastní zpracování.

Na obrázku výše je zobrazeno fungování MVC. Uživatel odešle požadavek na server, kontrolér ho přijme a provede změny v modelu. Poté od modelu „získá" data nazpět, které odešle je na pohled. Ten následně vykreslí odpověď uživateli.

6.3 C#

C# je hybridem C a C ++. Jedná se o programovací jazyk Microsoftu. C# je objektově orientovaný programovací jazyk používaný s webovými službami, založenými na XML na platformě .NET a navržený pro zvýšení produktivity při vývoji webových aplikací. C# odstraňuje některé složitosti a úskalí jazyků, jako je Java a C ++. Funkce, jako je ošetření výjimek, rozšiřitelné typy dat a zabezpečení kódu, jsou funkce, které se očekávají v moderním jazyce a C # má všechny vyjmenované funkce.

6.4 JavaScript

JavaScript je programovací jazyk, který umožňuje implementovat komplexní funkce na webových stránkách. Moderní JavaScript je bezpečný. Neposkytuje nízko-úrovňový přístup k paměti nebo CPU, protože byl původně vytvořen pro prohlížeče, které jej nevyžadují. Schopnosti JavaScriptu v prohlížeči jsou

z důvodu bezpečnosti uživatele omezené. Cílem je „zabránit webové stránce“ v přístupu k soukromým informacím, nebo poškození dat uživatele.

6.5 Objektově relační mapování

Objektově relační mapovač (ORM) je knihovna, která automatizuje přenos dat uložených v tabulkách relačních databází na objekty, které se běžně používají v aplikačním kódu. ORM umožňuje psát kód místo SQL pro vytváření, čtení, aktualizaci a mazání dat a schémat v databázi. V důsledku toho je možné používat programovací jazyk pro práci s databází místo psaní příkazů SQL, nebo uložených procedur. Pro ASP.NET existuje několik ORM knihoven. Tyto knihovny si níže popíšeme.

NHibernate – je *open source* objektově relační mapovač pro aplikační rámec .NET. Je aktivně vyvíjen, plně funkční a používán v tisících úspěšných projektech. Je postaven na ADO.NET. Je založen na Hibernate, což je populární objektově-relační mapovač používaný v Javě.

Entity Framework – je primární prostředek společnosti Microsoft pro interakci mezi .NET aplikacemi a relačními databázemi. Entity Framework může generovat potřebné databázové příkazy pro čtení nebo zápis dat do databáze, které může provádět za nás. Entity Framework má podrobnější mapovací vrstvu, kterou je možné přizpůsobit mapování, například mapováním jedné entity na více databázových tabulkách, nebo dokonce více entit na jednu tabulku. V Entity Frameworku je ohnisko vývoje koncepční model, což je model objektů v aplikaci, nikoli model databáze, kterou používáme k zachování dat aplikace.

Linq-to-SQL – je součást rozhraní .NET aplikačního rámce verze 3.5, která „poskytuje infrastrukturu“ pro správu relačních dat. V LINQ na SQL je datový model relační databáze mapován na objektový model v programovacím jazyce vývojáře. Při spuštění aplikace převede LINQ na SQL jazykově integrované dotazy v objektovém modelu do SQL, a odešle je do databáze k provedení. Když databáze vrátí výsledky, LINQ to SQL je převede zpět na objekty, se kterými můžeme pracovat v našem vlastním programovacím jazyce.

Dapper – je *open source*, nenáročný ORM vyvinutý týmem StackOverflow. Dapper je (ve srovnání s jinými ORM) velmi rychlý, především díky své nízké náročnosti. Dapper byl vytvořen s ohledem na výkon a snadné použití. Poskytuje podporu statické i dynamické vazby objektů pomocí transakcí, uložených procedur, nebo hromadného vkládání dat.

6.6 Kaskádové styly

Kaskádové styly (neboli CSS) se používají ke stylizaci prvků psaných v značkovacím jazyce, jako je HTML. Oddělují obsah od vizuální reprezentace webu. HTML a CSS mají úzký vztah. CSS není technicky nutností, ale pravděpodobně bychom nechtěli hledat informace na webu, který obsahuje pouze HTML, protože by vypadal naprosto zmatečně a chaoticky.

Na internetu lze vidět webové stránky, které se úplně nenačtou, a mají bílé pozadí, přičemž většina textu je modrá a černá. To s velkou pravděpodobností znamená, že část webu obsahující CSS se nenačetla správně, nebo vůbec neexistuje. Takto vypadá web pouze s HTML. Před použitím CSS musela být všechna stylizace zahrnuta do značek HTML. To mělo za důsledek, že bylo potřeba samostatně popsat všechna pozadí, barvy písma, zarovnání atd. CSS představují poměrně jednoduché řešení tohoto problému. Je obtížné dosáhnout vytvoření funkce, opakovaného použití definice nebo dědičnosti. U větších projektů, nebo složitých systémů je údržba velký problém. Preprocesory se proto staly nedílnou součástí CSS. Preprocesory rozšiřují CSS o proměnné, operátory, interpolace, funkce, mixiny a mnoho dalších použitelných aktiv. SASS, LESS a Stylus jsou známé a používané preprocesory.

Stejně jako každý programovací jazyk, i preprocesory mají odlišnou syntaxi, ale svým chováním jsou obdobné. Každý preprocesor CSS má svou vlastní syntaxi, kterou zkompiluje do běžného CSS, takže jej prohlížeče mohou zobrazit na straně klienta. Všechny preprocesory CSS dělají podobné věci, ale jiným způsobem, a se svými vlastními syntaxemi. Každý z nich má některé pokročilé funkce, které jsou pro něj jedinečné a také svůj vlastní ekosystém (nástroje, aplikační rámce, knihovny).

7 Zabezpečení

ASP.NET Core MVC je webový vývojový aplikační rámec, který je možné použít pro vývoj webových aplikací. Ukázalo se však, že tyto webové aplikace jsou zranitelné vůči útokům z různých zdrojů. Je proto třeba webovou aplikaci řádně zabezpečit. Nyní podrobněji rozebereme některé z potencionálních hrozeb:

SQL injekce – je nebezpečný útok, kdy neautorizovaní uživatelé vkládají škodlivý kód SQL, který se poté spustí v databázi, a umožní útočníkům přístup k důvěrným informacím v něm uloženým. Útoku SQL injekce je možné zabránit následujícími způsoby:

- ukládáním šifrovaných dat
- ověření vstupů
- použitím uložených procedur
- použitím ORM
- použitím parametrizovaných dotazů
- použitím nejméně privilegovaného přístupu k databázi

Ověření vstupů – obranu proti SQL injekci je vhodné ověřit uživatelské vstupy na straně klienta i serveru. Je vhodné zakázat speciální znaky, které se používají ve skriptech SQL. K ověření vstupů je vhodné použít regulární výrazy.

ORM – je zkratka pro objektově relační mapovač, který mapuje objekty SQL na objekt vaší třídy v aplikaci. Pokud je ORM použito správně, není web „náchylný“ k útokům SQL injekce, protože ORM interně používá parametrizované dotazy.

Ukládání šifrovaných dat – je nevhodné ukládat důvěrné informace (, jako jsou e-mailové adresy a hesla) jako prostý text do databáze. Obsah by měl být uložen v šifrovaném formátu.

Stránky se zobrazením chyb – na webu může být nastavené nesprávné zobrazení chybových hlášek. To může vést k odhalení citlivých informací, jako jsou informace o konfiguraci databáze, názvy tabulek, uložené procedury a datové struktury.

Ověřování uživatele – je možnost, jak zabezpečit webovou aplikaci. Je nutné povolit uživateli přihlášení, a je potřeba uživatelům určit roli. Podle rolí uživatelů

webové stránky udělují přístupy do určitých částí aplikace, a povolují, či blokují vybrané funkce.

Potvrzení e-mailem – je nutné při nové registraci uživatele, abychom ověřili, že se nevydává za někoho jiného, je vhodný mechanismus, který lze použít k ověření platnosti e-mailové adresy. Předpokládejme např., že uživatel Jakub se omylem zaregistroval jako *jakuv@priklad.cz* a nevšiml si toho. Nemohl by použít funkci obnovení hesla, protože aplikace nemá správný e-mail. E-mailové potvrzení poskytuje pouze omezenou ochranu před roboty, ale neposkytuje ochranu před *spammery*. *Spammery* mají mnoho e-mailových aliasů, které mohou použít k registraci.

8 Návrh a realizace internetového obchodu

Tato kapitola pojednává o našem návrhu a realizaci internetového obchodu na aplikačním rámci ASP.NET. Navrhovaný internetový obchod by měl splňovat všechny hlavní funkce, jež lze od internetového obchodu očekávat. Je rozdělen na administrační a zákaznickou část. K administrační části by měli přístup pouze zaměstnanci, její hlavní funkčnost pokrývá možnost spravovat objednávky, produkty, uživatelské účty, ale i obchodní podmínky, kontaktní údaje majitele a bankovní spojení pro objednávky s platbou přes bankovní převod. V zákaznické části by měl e-shop obsahovat stránku se seznamem produktů s možností filtrování a řazení, stránku s detailem produktu, stránku s informacemi o společnosti a kontaktní stránku. Internetový obchod by měl být řádně zabezpečený, lehce škálovatelný a lehce použitelný pro zákazníky i zaměstnance.

Navrhovaný internetový obchod by měl umožňovat zákazníkům vyhledávat a objednávat produkty z pohodlí domova. V e-shopu by mělo být možné zboží filtrovat podle kategorie a značky. Navrhovaný internetový obchod bude rozšířením ke klasické kamenné prodejně bot. Pokud bude chtít tedy zákazník nakupovat přímo tam, může předtím zjistit, jaké zboží prodejce nabízí.

Majiteli e-shopu by měl e-shop přinést vyšší zisky, nové potencionální zákazníky, a to díky prodejům, které by se jinak neuskutečnily, kvůli „lenosti zákazníka“ nebo nedosažitelnosti prodejny. Majitel navrhovaného internetového obchodu by měl mít možnost spravovat zboží a zároveň sledovat aktuální stav

zásob. Rovněž by měl být schopen spravovat objednávky zákazníků a měnit např. jejich aktuální stav.

8.1 Použité technologie

Prvním důležitým krokem návrhu internetového obchodu bylo zvolení vhodných technologií. Pro tvorbu modelování jsme použili Enterprisearchitect verzi 13.0, což je komplexní nástroj pro analýzu a návrh UML, SysML, BPMN a mnoho dalších technologií. Pokrývá vývoj softwaru od shromažďování požadavků, až po fáze analýzy, návrhové modely, testování a údržbu. Pro návrh grafického designu jsme pro vektorovou grafiku použili programu Adobe Ilustrator, Sketch, Figma, pro rastrovou grafiku Adobe Photoshop. Webová aplikace je postavená na aplikačním rámci ASP.NET MVC. Jako datové úložiště je použit MS SQL server 2012, což je systém pro správu relačních databází vyvinutý a uváděný na trh společností Microsoft. Primární funkcí SQL serveru je ukládání a načítání dat používaných jinými aplikacemi. Pro objektově relační mapování je použit NHibernate, který umožňuje ukládání a čtení dat v relační databázi bez psaní SQL. Web dále využívá Javascript a knihovnu JQuery, která je použita např. pro Carousel s produkty. Další použitou technologií je Ajax, který je použit např. pro přidávání zboží do košíku. Web také využívá Bootstrap, což je nejpopulárnější HTML, CSS a JavaScript aplikační rámec pro vývoj responzivních mobilních webů.

8.2 Modelování procesů

Modelování je ústřední součástí všech činností, které vedou k nasazení dobrého softwaru. Vytvořili jsme proto UML a BPMN model, abychom lépe porozuměli systému, který budujeme. Často jsme přitom „odkrývali příležitosti“ pro zjednodušení aplikace a její opětovné použití. Popis procesů, které webová aplikace řeší či automatizuje**:**

- **nákup produktů** (vložení produktů do košíku, vybrání způsobů platby a dopravy, zadání dodací adresy, rekapitulace objednávky, odeslání e-mailu)

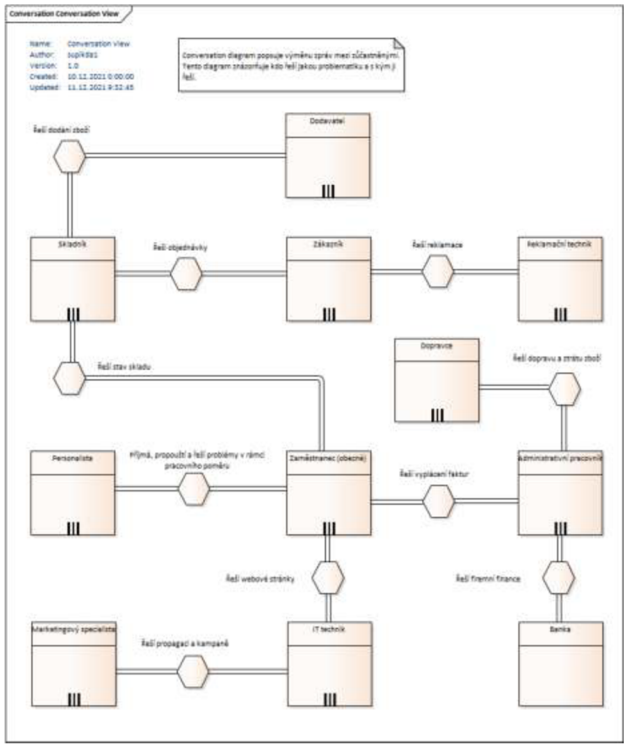
- **zapomenuté heslo** (zadání e-mailové adresy, odeslání e-mailu s tokenem pro autorizaci a následná změna hesla)
- **odesílání e-mailů** (odesílání e-mailů při registraci uživatelského účtu, dokončení objednávky, změně stavu objednávky, žádosti o změnu hesla)
- **vyhledávání** (produktů, kategorií, značek atd. v administraci)
- **registrace, autentifikace a autorizace uživatelů**
- **správa uživatelů a rolí** (změna role, upravení údajů uživatele)
- **správa produktů, kategorií, značek, objednávek, skladu, způsobů dodání, platebních metod, bankovního spoje** (přidávání, editace, mazání, vyhledávání)
- **správa faktur** (přidání, editace, mazání, označení data zaplacení)
- **správa kampaní**
- **správa reklamací** (přidání, editace, mazání, změna typu kompenzace)

Web je rozdělen do několika uživatelských rolí a uživatel má na jejich základě různá oprávnění. Těmito rolemi jsou:

- **Zákazník** – má možnost vytvořit a editovat svůj účet, prohlížet produkty, nakupovat zboží a prohlížet své objednávky.
- **Skladník** – má dovoleno spravovat produkty, velikosti produktů, kategorie produktů, značky produktů, slevy produktů. Může také editovat stav skladu. Smí spravovat objednávky a jejich adresu, e-mail, telefon, stav.
- **IT technik** – může si zobrazit stav skladu, logy, může editovat uživatele
- **Reklamační technik** – může spravovat reklamace. Může je přijímat, odmítat a upravovat typ případné kompenzace.
- **Marketingový specialista** – může vytvářet a spravovat marketingové kampaně.
- **Administrativní pracovník** – může si zobrazit seznam všech plateb provedených přes internetový obchod.
- **Administrátor** – má nejvyšší pravomoci. Může v administraci upravit vše, co uživatelé s ostatními rolemi, a navíc může spravovat uživatele a uživatelské role. Smí spravovat produkty, kategorie produktů, značky

produktů, velikosti produktů, slevy produktů, objednávky, stav skladu, platební metody, způsoby dodání, uživatelské účty a role, číslo bankovního účtu pro platby bankovním převodem, stavy objednávek.

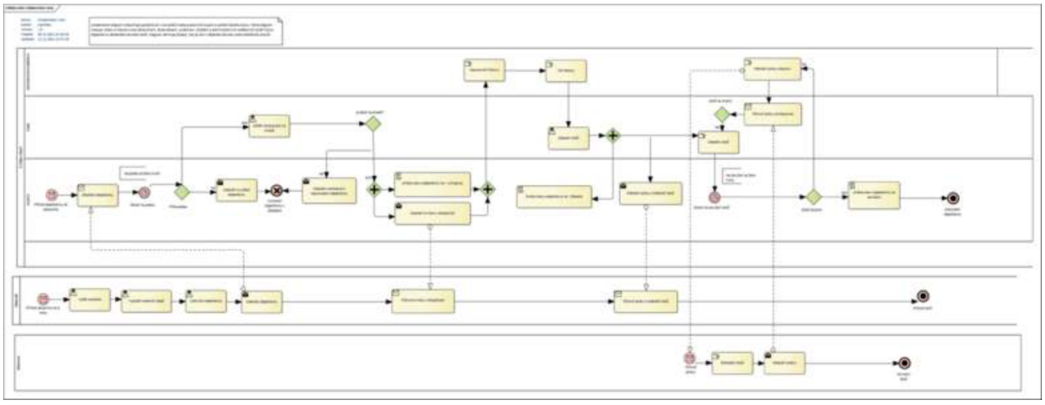
První z BPMN diagramů, které jsme navrhli, je konverzační diagram, popisující výměnu zpráv mezi zúčastněnými. Diagram nám následně pomohl s návrhem rolí v administrační části internetového obchodu. Je na něm zobrazeno, kdo a s kým jakou problematiku diskutuje. Všichni zúčastnění (kromě banky) mají nastavenou multiplicitu, mohou se tedy v systému vyskytovat vícekrát než jednou. Na modelu můžeme vidět, že zákazník vyřizuje reklamace s reklamačním technikem.



Obrázek 30 – Konverzační diagram

Zdroj: Vlastní zpracování

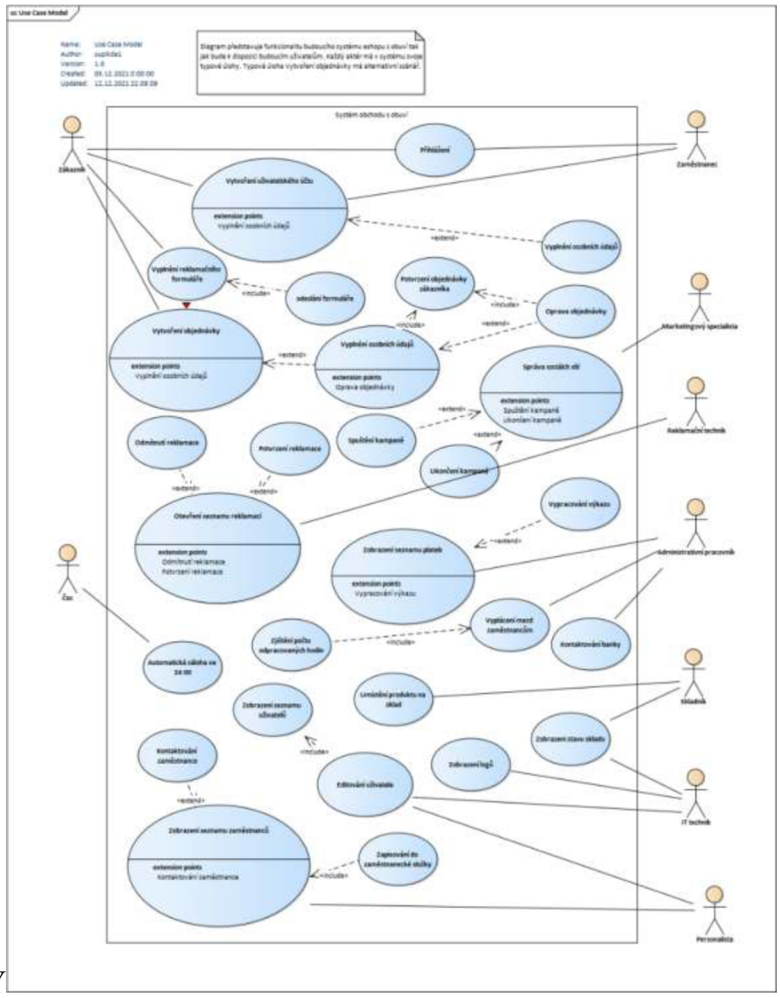
Další z BPMN diagramů, který jsme vypracovali, je diagram kolaborační znázorňující společné úsilí více aktérů nebo pracovních skupin na splnění daného úkolu. Tento diagram ukazuje vztahy a interakce mezi zákazníkem, dodavatelem, systémem, skladem a administrativním oddělením při plnění úkolu objednání následného odeslání zboží.



Obrázek 31 – Kolaborační diagram

Zdroj: Vlastní zpracování

Následným krokem bylo použití UML, což je nástroj, který může výrazně zlepšit kvalitu analýzy a návrhu systému. Před implementací internetového obchodu jsme vytvořili analytický a návrhový model tříd. V analytickém modelu tříd je zachována přehlednost a jednoduchost bez zanášení implementačních detailů. První iterace analýzy je na velmi vysoké úrovni, aby bylo možné identifikovat celkové cíle systému a ověřit požadavky prostřednictvím analýzy případu užití. Součástí této první iterace je identifikace aktérů, definování počátečního modelu případu užití, který zobrazuje role a popisuje jejich privilegia. Každý z případů užití má jasně strukturovaný scénář po sobě jdoucích událostí, popisujících interakce prováděné buď aktérem, nebo systémem.

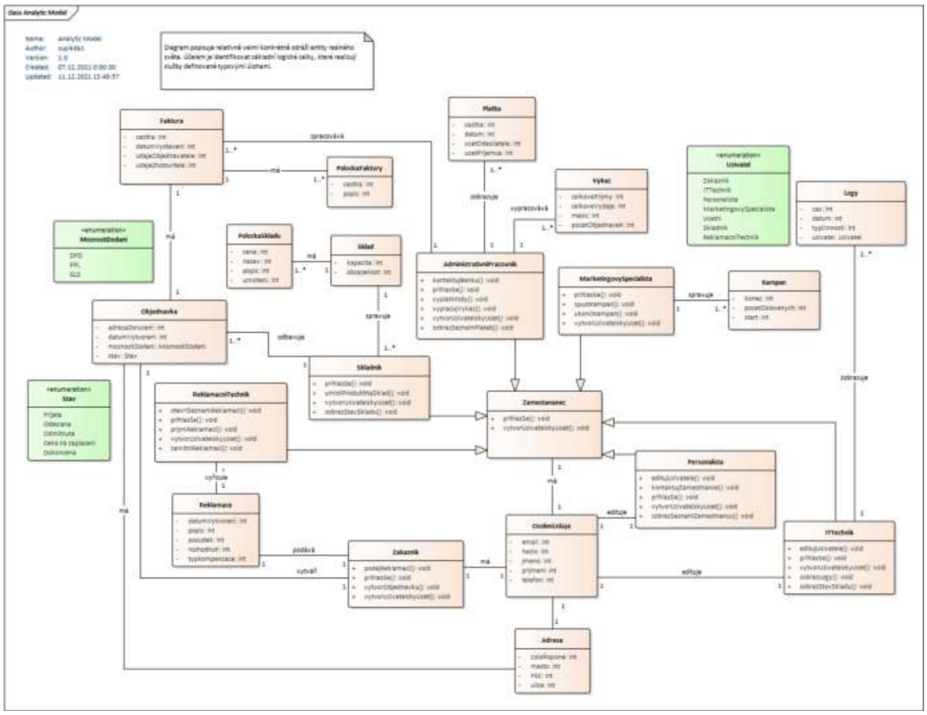


Y

Obrázek 32 – Diagram případů užití

Zdroj: Vlastní zpracování

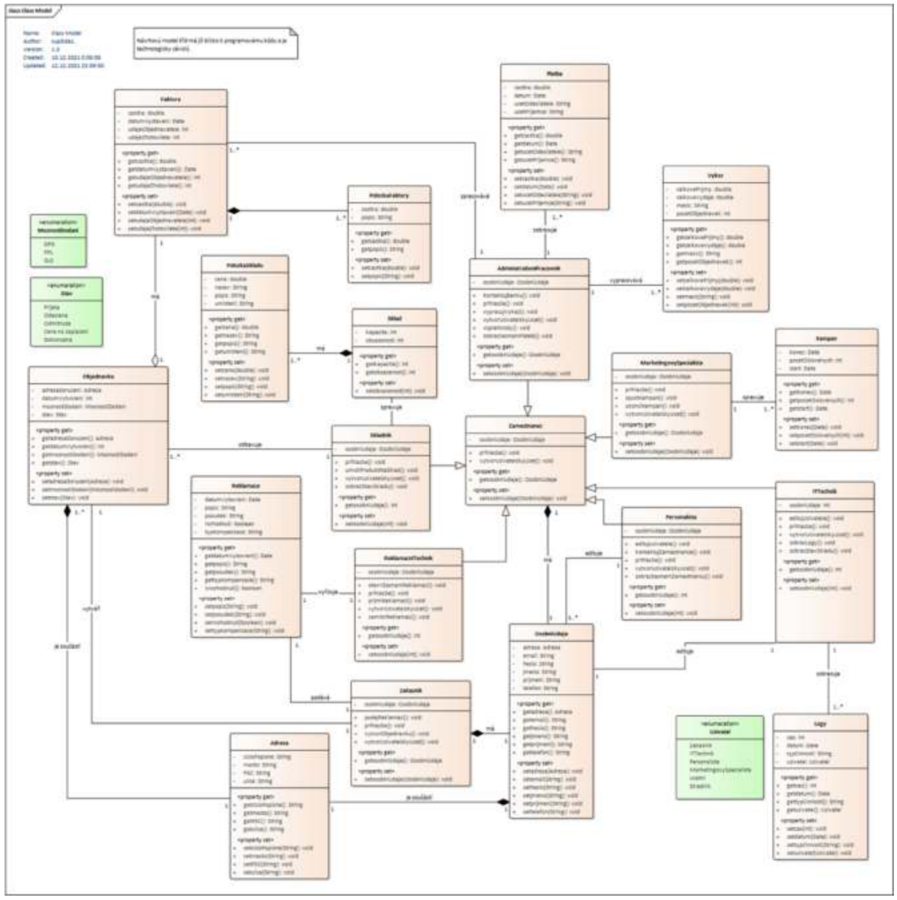
Iterace analýzy dále zpřesňují systémové požadavky prostřednictvím vývoje scénářů užití, diagramů tříd atd. Každá iterace se postupně podrobněji dívá na návrh systému, dokud nejsou věci a vztahy v systému jasně a přesně definovány v dokumentech UML.



Obrázek 33 – Analytický model tříd

Zdroj: Vlastní zpracování

Model zobrazuje třídy v systému, atributy, operace každé třídy a vztah mezi každou třídou. Třída má tři části. Název nahoře, atributy uprostřed a operace nebo metody dole. Různé vztahy mezi třídami jsou znázorněny různými typy šipek. Na návrhovém modelu tříd již můžeme vidět použití generalizace, např. u třídy „Zamestnanec“ a „Skladnik“. Dále je zde využita kompozice, například ve vztahu Faktura a „Polozka faktury“.



Obrázek 34 – Návrhový model tříd

Zdroj: Vlastní zpracování

Po dokončení analýzy a návrhu jsme získali přesnou a podrobnou sadu specifikací pro třídy, scénáře a aktivity v systému. Návrhový model jsme následně využili jako vzor při konstrukci tříd navrhovaného internetového obchodu.

8.3 Grafický design

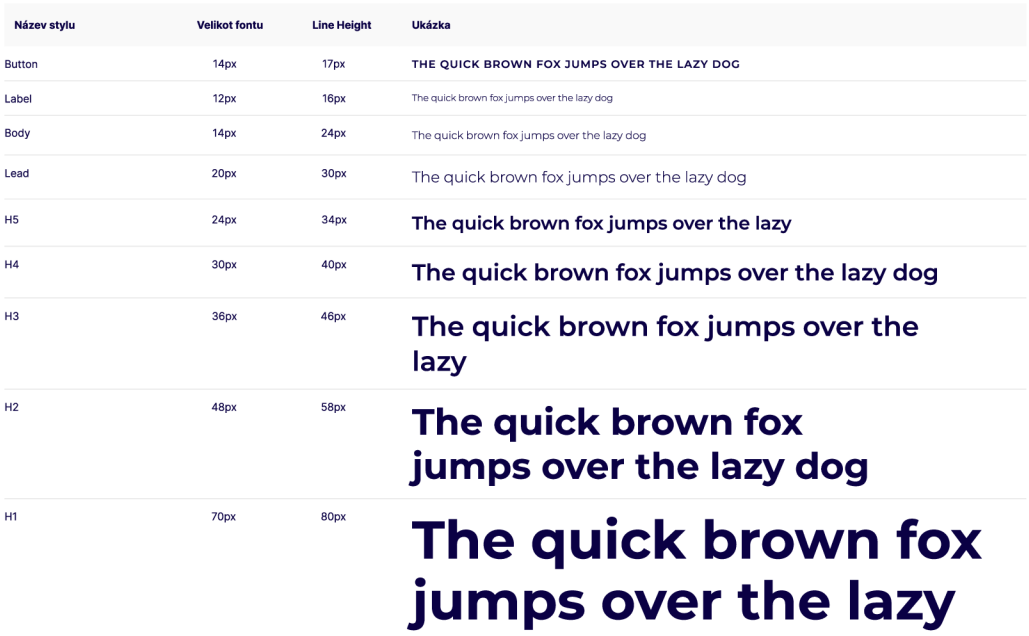
Dalším krokem návrhu tvorby internetového obchodu bylo vytvoření grafického designu. V prvním kroku jsme (pro docílení sjednoceného vizuálního stylu) vytvořili manuál, který specifikuje všechny vizuální detaily, vč. tónu hlasu a zpráv společnosti. Manuál definuje jasná pravidla, jak se pracuje s logem, barvami, typografií při digitálním použití i při tisku. Manuál je ve formě fyzické i digitální brožury, obsahuje rovněž návod k tomu, jak vhodně se značkou zacházet, nebo naopak nezacházet. Jedním z důležitých kroků při tvorbě manuálu byl výběr

barevné palety. Při výběru barev jsme využili metodu triadického schématu. Vybrali jsme tři primární barvy, a to modrou (#2D4293), žlutou (#F8C9E4) a červenou (#CF221A). Modrá je použita jako dominantní barva, zbylé dvě barvy jsou použité k vytváření kontrastu. Modrá barva je použita jako primární a tvoří 60 %. Žlutá barva je použita na 30 % obsahu. Červená barva slouží k vytvoření kontrastu pro zvýraznění důležitých elementů, je použita z posledních 10 %.

Obrázek 35 – Výběr barev

Zdroj: Vlastní zpracování

V práci jsme využili písmo z kolekce Google fonts Montserrat, které obsahuje některé hrany zakončené ostře. Pro udržení přehlednosti používáme napříč celým návrhem internetového obchodu pouze jedno jediné písmo. Písmo je moderní a dobře použitelné, protože obsahuje všechny řezy. Montserrat je geometrické bezpatkové písmo. Vytvořili jsme typografické měřítko, které přesně definuje styly písma na webu pro desktop, pro telefon i tablet. Při implementaci internetového obchodu se typografické měřítko použije jako předloha pro vytvoření znovupoužitelných textových stylů v css. Například, nadpis úrovně H1 má velikost 70px a Line Height 80px. Stejně mají jasně definovaný styl nadpisy ostatních úrovní, odstavce, seznamy, či tlačítka. Díky definovanému typografickému měřítku je zachována hierarchie. Uživatel se díky tomu lépe dokáže orientovat na webu.



Obrázek 36 – Typografické měřítko – Desktop

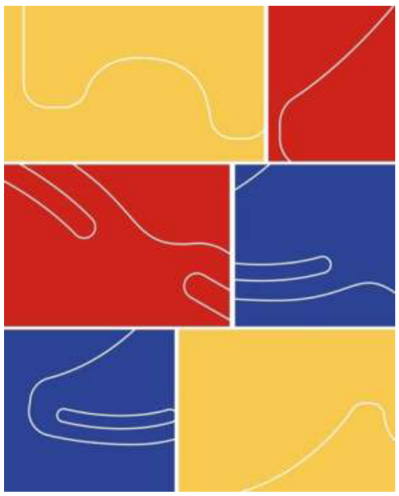
Zdroj: Vlastní zpracování

Dalším krokem bylo navrhnout logo, které představuje zjednodušený model tenisky. Je jednoduché, a proto je snadno zapamatovatelné a dobře použitelné. Zároveň jsme se ujistili, že logo funguje i v černobílém provedení. V ukázkách použití demonstrujeme různé návrhy vzorů, které lze použít například jako pozadí na webových stránkách nebo pro potisk obalů, ve kterých se tenisky budou odesílat. Naším cílem bylo, aby logotyp obsahoval i samotný název. Proto bylo velmi důležité zvolit písmo, které by se hodilo, k již vytvořenému logu, a zároveň by vyjadřovalo hlavní myšlenku celého projektu. Chtěli jsme vybrat písmo, které bude mít oblé a kulaté tvary, aby vyvolávalo přátelský dojem, ale aby přitom působilo seriózně. Snažili jsme se vybrat bezpatkové písmo, protože působí elegantně a moderně. Po celkovém uvážení jsme se rozhodli pro písmo Montserrat, které obsahuje některé hrany zakončené ostře.

Obrázek 37 – Logotyp

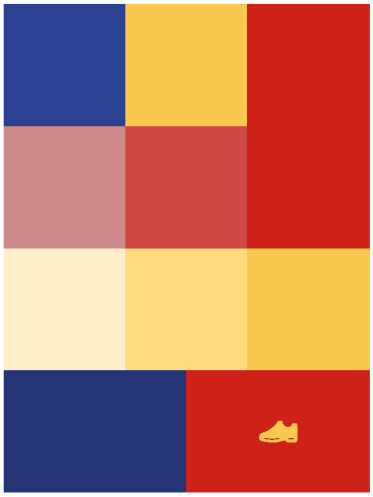
Zdroj: Vlastní zpracování

Po vytvoření loga jsme z jeho tvarů vytvořili dva vzory. Při jejich tvorbě jsme použili tvary a barvy firemního loga. Tyto vzory se dají využít jako různá pozadí na webu, či při fyzickém použití, např. na nákupní tašky, dlaždice v prodejně, na pozadí reklamních bannerů apod.



Obrázek 38 – Návrh vzoru č. 1

Zdroj: Vlastní zpracování



Obrázek 39 – Návrh vzoru č. 2

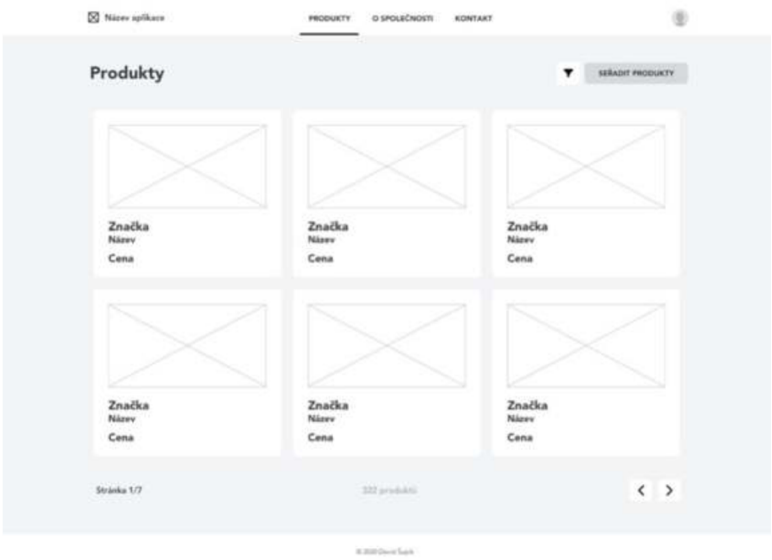
Zdroj: Vlastní zpracování

Pro lepší rozeznatelnost jsme design kamenné prodejny a internetového obchodu unifikovali. Jednotný vizuální styl využívá stejnou barevnou paletu, logo a písmo a vzory vytvořené z loga. Díky tomu je možné vytvořit silnější značku a zvýšit potencionální výnosnost.

Obrázek 40 – Vizuální identita

Zdroj: Vlastní zpracování

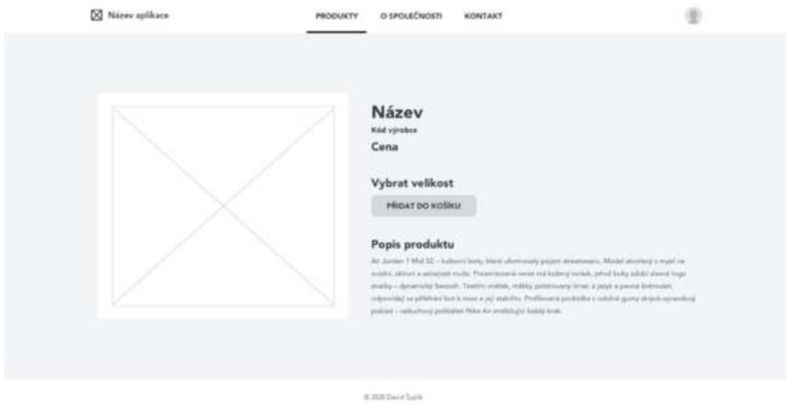
Prošli jsme několik významných českých i zahraničních internetových obchodů, porovnali jejich funkcionalitu, na základě tohoto zkoumání jsme vytvořili *wireframy* aplikace zohledňující diagramy případů užití, jsou zobrazené níže na obr. 49 v kapitole Uživatelské účty a role. Pro tvorbu *wireframů* jsme použili aplikaci Figma, která je aktuálně jednou z předních aplikací pro návrh uživatelských rozhraní. *Wireframy* zachycují pouze nejdůležitější prvky a obsah. Určují ne zcela věrný obrys a strukturu rozvržení. Při tvorbě *wireframů* jsme se soustředili na uživatelské potřeby, nezabývali jsme se estetikou. Při tvorbě *wireframů* jsme webovou aplikaci rozdělili do znovu použitelných komponent, které jsme využili ve vývoji. Vytvoření komponent snižuje čas vývoje a usnadňuje následné úpravy.



Obrázek 41 – Wireframe výpisu produktů

Zdroj: Vlastní zpracování

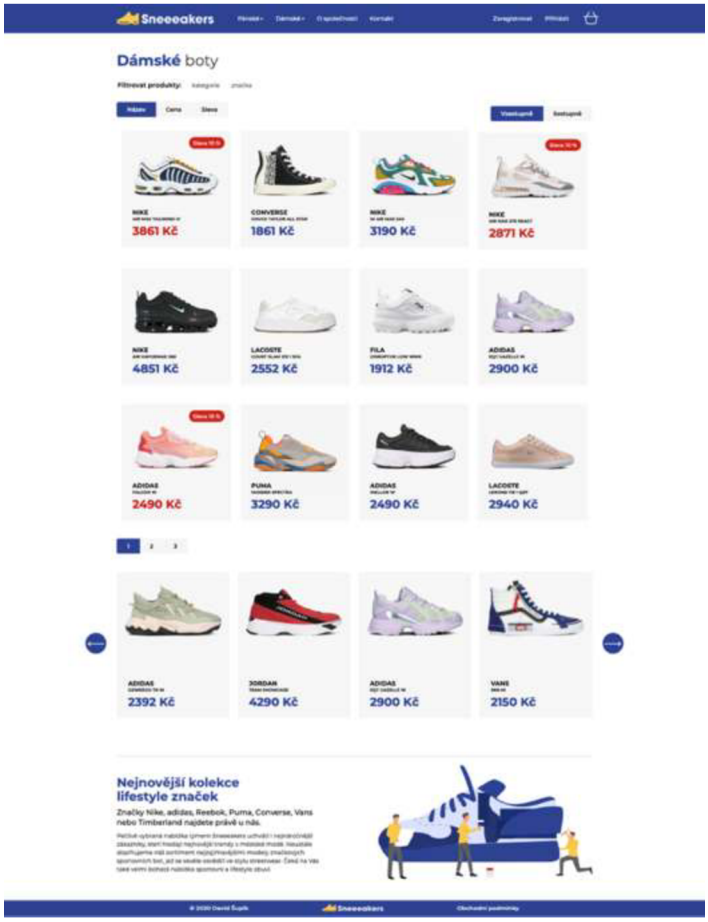
Na obrázku 42 je vytvořený *wireframe*, který zachycuje pohled zobrazení výpisu produktů navrhovaného internetového obchodu. Můžeme si povšimnout, že *wireframe* zobrazuje karty s produkty uspořádanými do mřížky. Také zobrazuje filtrování řazení a stránkování produktů, které uživatelům usnadní orientaci v katalogu produktů. Na obrázku 43 vidíme *wireframe* s detailem produktu, kde dominuje fotka produktu.



Obrázek 42 – Wireframe detailu produktu

Zdroj: Vlastní zpracování

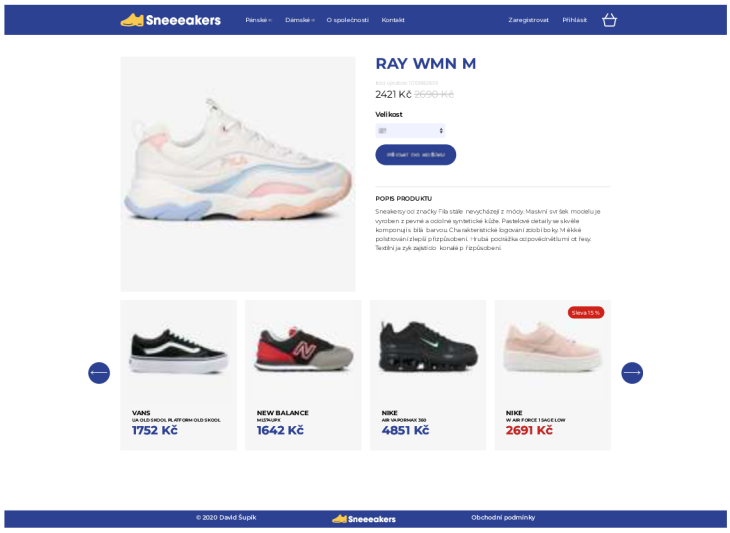
Po tvorbě *wireframů* jsme se věnovali tvorbě *mockupů*, které představují grafický návrh ve věrném provedení. Pro každou stránku navrhovaného e-shopu jsme navrhli *mockup* na základě *wireframů*. Při jejich navrhování jsme pracovali s vizuálním stylem vytvořeným na počátku vývoje internetového obchodu. Na rozdíl od *wireframů*, jsme dbali na estetickou stránku namísto funkčnosti. Kladli jsme důraz např. na velikost a řez písma, na barvy a na vizuální styl ikon. Vytvořili jsme několik ilustrací vycházejících z barevné palety a využili jsme je v designu. Výsledný design je minimalistický, zaměřený primárně na zobrazení produktů, které na webu vyniknou. Na obrázku 44 je zobrazen *mockup* výpisu produktů stránky.



Obrázek 43 – Mockup výpisu produktů stránky

Zdroj: Vlastní zpracování

Mockupy zachycují nejen design všech stránek, ale také stavy jednotlivých komponent, chybové hlášky formulářů, modalová okna, či interakce mezi jednotlivými stránkami. *Mockup* výpisu produktů zobrazuje produkty rozdělené na pánské a dámské v zobrazení pro desktop. Na *mockupu* jsme navrhli výpis produktů, filtr, stránkování a carousel s podobnými produkty. Červená barva ceny produktu a štítek uživateli ukazuje, že produkt je zlevněný. Design je rozdělený na komponenty se všemi svými stavy. To znamená, že změnit tenisku na zlevněnou v designu, je otázkou pouze přepnutím stavu v panelu nástrojů, místo složitého přesouvání všech vrstev souvisejících se zobrazením produktu. Všechny opakující se prvky jsou automatické rozložení, které simuluje vlastnost „display: flex“. Prvky jsou tedy jednoduše přeskupitelné v designu a dodržují *grid*, který jsem následně použil při tvorbě stylů v css. Na obrázku 45 můžeme vidět *mockup* detailu produktů.

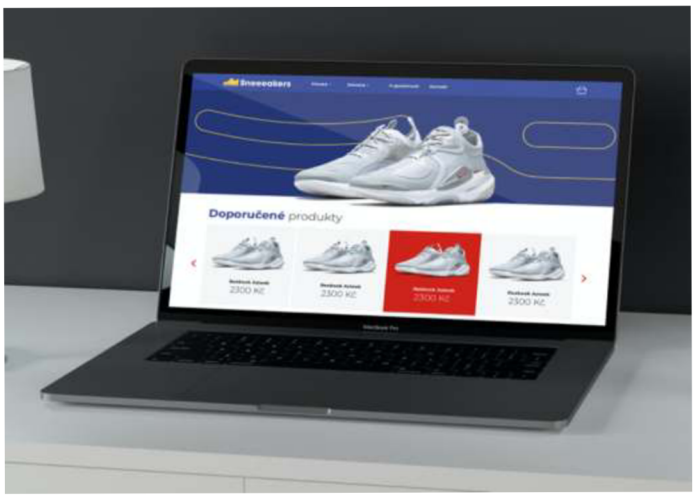


Obrázek 44 – Mockup detailu produktu

Zdroj: Vlastní zpracování

Na obrázku 46 pozorujeme *mockup* hlavní stránky, skládající se z navigace, bloku s carouselem s nejnovějšími produkty s fotografií ve velkém rozlišení, s cílem zaujmout uživatele na první pohled. Pod ním je sekce s doporučenými produkty, které vybírají zaměstnanci internetové obchodu a nejnovější produkty, které jsou aktuálně v nabídce. *Mockupy* jsem dále využil jako předlohu pro následný vývoj webové aplikace.

Při označení určitého elementu Figma v režimu průzkumníku umí našeptat určité CSS styly daného elementu, což značně snižuje čas potřebný k implementaci.

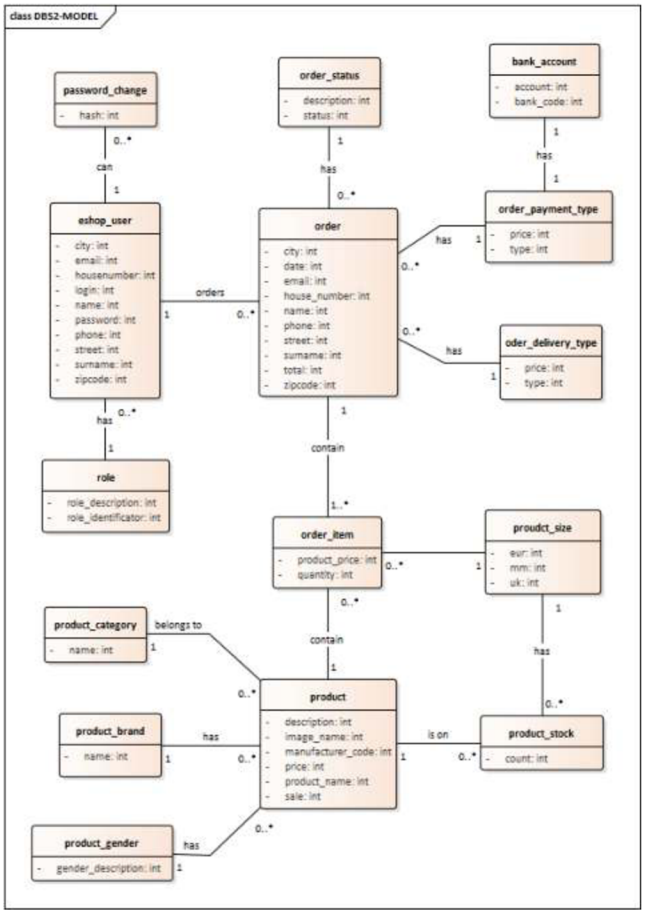


Obrázek 45 – Mockup hlavní stránky

Zdroj: Vlastní zpracování

8.4 Návrh datového modelu v MS SQL

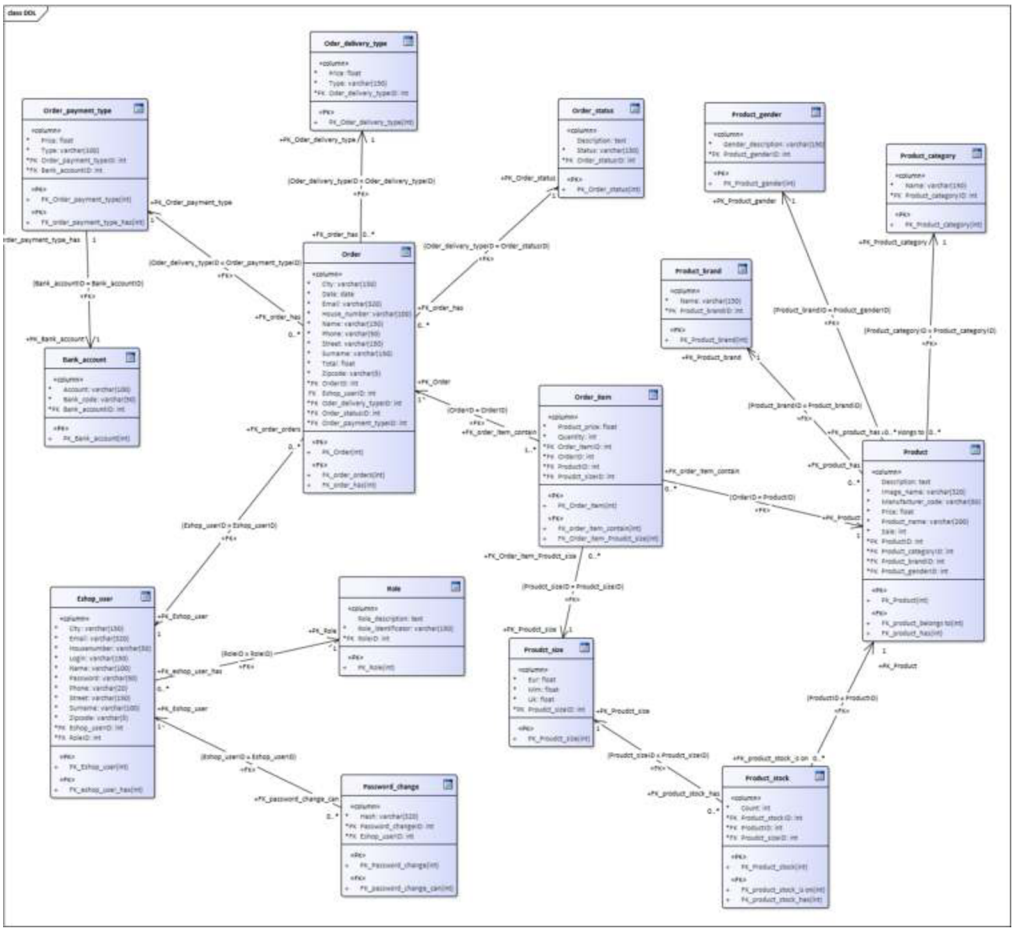
Ve chvíli, kdy byl grafický design e-shopu dokončen, jsme se věnovali datovému modelování. Vytvořili jsme entitně-relační diagram zobrazující datový model relační databáze. Návrh databáze jsme vytvořili s ohledem na normalizační formy. Na entitně-relačním diagramu na obrázku 47 můžeme vidět, že každý produkt má právě jednu kategorii, značku a je určen jednomu pohlaví.



Obrázek 46 – Entitně-relační diagram (ERD)

Zdroj: Vlastní zpracování

V datovém modelu na obrázku 48, který jsme částečně vygenerovali z entitně-relačního diagramu, a který jsme z poté doplnili, jsou již určeny i datové typy jednotlivých atributů v tabulkách, zároveň zobrazují i jejich klíče. Na digramu lze vidět, že každá entita má právě jeden primární klíč, některé z entit obsahují i cizí klíče. Například, entita „Order“ popisující objednávku, má primární klíč „OrderID“ a cizí klíče „Eshop_UserID“, „Order_delivery_type“,„Order_StatusID“ a „Order_payment_type“. Primární klíč „OrderID“ zajišťuje, že každá objednávka má unikátní klíč, podle kterého ji lze identifikovat. Cizí klíče zajišťují provázanost s tabulkami, které určují typ dopravy, status objednávky a typ platby.



Obrázek 47 – Datový model

Zdroj: Vlastní zpracování

S použitím aplikace Enterprise Architect jsme pomocí vygenerovaného kódu z datového modelu vytvořili tabulky v MS SQL serveru a naplnili je daty o produktech, značkách, velikostech, barvách atd. Dále jsme vytvořili několik pohledů pro získávání informací z několika tabulek zároveň ve námi požadovaném formátu. Těmito pohledy jsou:

- **Aktuálně objednané zboží**

```
AS SELECT EO.surname PRIJMENI, P.product_name PRODUKT, OI.product_size
VELIKOST, OI.quantity POCET
Z eshop_order EO
INNER JOIN order_item OI ON OI.order_id = EO.id
LEFT OUTER JOIN product P ON P.id = OI.product_id
KDE EO.order_status = 1 NEBO EO.order_status = 2 NEBO EO.order_status = 3
```

```
GROUP BY EO.surname, OI.product_id, P.product_name, OI.product_size,
OI.quantity
```

- **Celkové tržby**

```
CREATE VIEW CELKOVE_TRZBY
AS SELECT ISNULL(COUNT(EO.id),0) POCET_OBJEDNAVEK, ISNULL(SUM(EO.total),
0) TRZBA
FROM eshop_order EO
WHERE order_status = 8
```

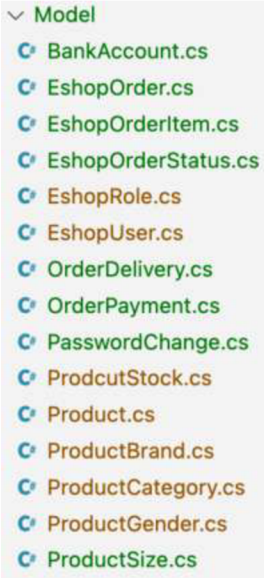
- **Seznam uživatelů**

```
CREATE VIEW SEZNAM_UZIVATELU
AS SELECT EU.name JMENO, EU.surname PRIJMENI, ER.role_identificator
FROM eshop_user EU
LEFT OUTER JOIN eshop_role ER ON ER.role_id = EU.role_id
GROUP BY EU.name, EU.surname, ER.role_identificator
```

8.5 Implementace

Při vytvoření nového projektu v ASP.NET MVC ve výchozím nastavení, získáme strukturu složek podobnou této: App_Data, App_Start, Content, Controllers, Fonts, Models, Scripts, Views. Do složky Content jsme vložili CSS styly, ikony, ilustrace a obrázky použité na webu. Dále jsme vytvořili složku „Uploads“, kam se nahrávají obrázky produktů, které jsou přidané v administraci. Adresář Scripts obsahuje všechny JavaScriptové soubory, které web využívá. Je zde např. skript pro carousel produktů nebo validátor formuláře. Další části aplikace jsou popsány detailněji níže.

Model (model) – obsahující základní logiku a strukturu dat. Složka Models obsahuje všechny modely, nebo data požadovaná pro webovou aplikaci. Například, ukládání informací souvisejících se zákazníky, je řešeno v modelu s názvem EshopUser.cs.



Obrázek 48 – Model aplikace

Zdroj: Vlastní zpracování

Pro ukázku si ukážeme model produktu prodávaného v internetovém obchodě, který lze vidět na obrázku 50. „System.ComponentModel.DataAnnotations" obsahuje třídy, které se používají jako datové atributy. Použitím těchto atributů na datovou třídu centralizujeme definici dat, a nemusíme znovu aplikovat stejná pravidla na více místech. Anotace Required určuje, že je konkrétní vlastnost vyžadována. Anotace Range se používá k určení rozsahu hodnot, které vlastnost může mít. Tento model popisuje, že produkt má svoje jedinečné ID datového typu integer. Dále musí mít název, kód výrobce a popis, což jsou parametry datového typu string. Produkt má svou cenu, datového typu integer, která musí být uvedena, a musí být větší než 0. Může, ale nemusí, se vázat na kategorii, značku, může být určen pro určité pohlaví. Produkt také může mít uloženou cestu k obrázku. Je možné produktu nastavit slevu v procentech (od 0 do 100 %).

```
using DataAccess.Interface;
using System;
using System.ComponentModel.DataAnnotations;

namespace DataAccess.Model
{
    public class Product : IEntity
    {
        public virtual int Id { get; set; }

         [Required(ErrorMessage ="Název produktu je požadován")]
        public virtual string Name { get; set; }


         [Required(ErrorMessage = "Kód výrobce je požadován")]
         public virtual string ManufacturerCode { get; set; }


         [Required(ErrorMessage = "Popis produktu je požadován")]
         public virtual string Description { get; set; }

         [RegularExpression(@"^(\d{1,30})$", ErrorMessage = "Cena musí obsahovat pouze čísla")]
         [Required(ErrorMessage = "Cena produktu je požadována")]
         [Range(0,Double.PositiveInfinity, ErrorMessage = "Cena musí být větší než 0")]
         public virtual int Price { get; set; }

         public virtual ProductCategory Category { get; set; }
         public virtual ProductBrand Brand { get; set; }
         public virtual ProductGender Gender { get; set; }

         public virtual string ImageName { get; set; }

         [RegularExpression(@"^(\d{1,30})$", ErrorMessage = "Sleva musí obsahovat pouze čísla")]
         [Range(0,100, ErrorMessage = "Sleva může být od 0 do 100 %")]
         public virtual int Sale { get; set; }
    }
}
```

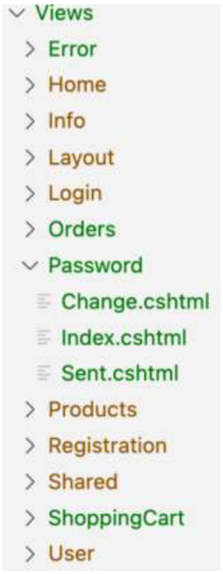
Obrázek 49 – Model produktu

Zdroj: Vlastní zpracování

Pohled (View) – je zodpovědný za transformaci modelu do vizuální reprezentace. Pohledy pro změnu hesla jsou například reprezentovány složkou Password, uvnitř složky Views. Složka Password obsahuje pohledy pro zadání e-mailu, informaci o zaslání e-mailu s odkazem na změnu hesla a pohled s formulářem pro změnu hesla. Pokud uživatel vstoupí na jednu z těchto tří webových stránek, kontrolér určí, který ze tří pohledů se použije k vytvoření a vrácení webové stránky uživateli. V ASP.NET MVC jsou pohledy.cshtml soubory, které používají programovací jazyk C# a značkovací jazyk Razor. Obvykle jsou soubory pohledů seskupeny do složek pojmenovaných podle každého z kontrolérů aplikace. Složky jsou uloženy ve složce Views v kořenovém adresáři aplikace.

Mnoho stránek v aplikaci také opakovaně používá struktury HTML. Všechny tyto sdílené prvky jsou definovány v layoutu, který je poté využit v ostatních pohledech v aplikaci. Layouty snižují duplicitní kód v pohledech. Podle konvence

jsme výchozí layout pojmenovali _Layout.cshtml. Layout definuje šablonu nejvyšší úrovně pro zobrazení v aplikaci. Řešení internetového obchodu využívá více než jeden layout. Např. zákaznická a administrační část aplikace má jiný layout. Níže můžeme sledovat strukturu pohledů aplikace.



Obrázek 50 – Views

Zdroj: Vlastní zpracování

Pohled přihlášení vykresluje autorizaci uživatele. Kód začíná zobrazením nadpisu úrovně 2. Hned pod ním je podmínka, která rozhoduje, zda se má do odstavce vypsat chybová hláška způsobená neplatnými nebo prázdnými vstupy níže. Formulář obsahuje dva vstupy pro uživatelské jméno a heslo. Pod formulářem je odkaz na obnovení hesla. Dále jsou v pohledu dvě modalová okna aktivovaná přes id zadané do URL adresy pro zobrazení informačních hlášek.

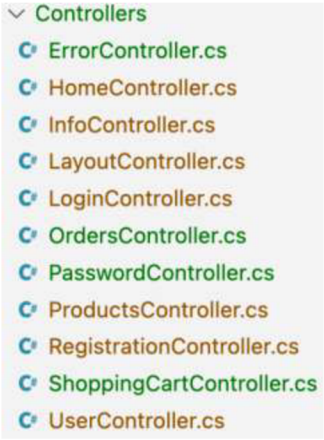
```
<div class="narrow-content">
    <div>
    <h2 class="form-signin-heading">Přihlášení</h2>
    @if (TempData["Error"] != null)
    {
        <p>@TempData["Error"]</p>
    }
    @using (Html.BeginForm("SignIn", "Login", FormMethod.Post, new { @class = "form-signin" }))
    {
        <div class="form-group">

            <label for="inputEmail">Login</label>
            <input type="text" class="form-control labeled" placeholder="Uživatelské jméno" name="login">
        </div>
        <div class="form-group">
            <label for="inputPassword">Heslo</label>
            <input type="password" class="form-control labeled" placeholder="Heslo" name="password">
        </div>
        <button class="btn btn-lg btn-primary btn-block" type="submit">Přihlásit</button>
    }
    <p class="registration-link">Zapoměl/a jste heslo? @Html.ActionLink("Vytvořit nové", "Index", "Password")</p>
        </div>
</div>


<div class="remodal" data-remodal-id="loggedOut">

    <h1><span style="font-weight:300; color: #1a1a1a">Byl jste úspěšně</span>  odhlášen</h1>
    <br>
    <button data-remodal-action="close" class="remodal-cancel">Zavřít</button>
</div>

<div class="remodal" data-remodal-id="badCredentials">

    <h1><span style="font-weight:300; color: #1a1a1a">Uživatel s těmito údaji </span>neexistuje</h1>
    <br>
    <button data-remodal-action="close" class="remodal-cancel">Zavřít</button>
</div>
```

Obrázek 51 – Pohled přihlášení

Zdroj: Vlastní zpracování

Kontrolér (Controller) – řídí aplikační logiku a působí jako koordinátor mezi pohledem a modelem. Kontroléry jsou v podstatě centrální jednotkou aplikace ASP.NET MVC. Jedná se o prvního příjemce, který komunikuje s příchozím požadavkem HTTP. Kontrolér tedy „rozhodne“, který model bude vybrán, a poté převezme data z modelu a předá je do příslušného pohledu. Kontroléry řídí celkový tok aplikace, přičemž přijímají vstup a vykreslují správný výstup.

Níže můžete vidět strukturu kontrolérů aplikace.



Obrázek 52 – Kontroléry

Zdroj: Vlastní zpracování

Orders Controller.cs zobrazený na obrázku 54, je kontrolér řídící aplikační logiku zobrazování seznamu a detailu objednávek. Logika seznamu objednávek si nejdříve ověřuje, jestli je přihlášen nějaký uživatel. Pokud ano, kontrolér načte z databáze seznam objednávek podle ID uživatele. Tento seznam objednávek je pomocí VieBag. Orders přenesen do pohledu. Pokud nikdo není přihlášen, je uživatel přesměrován na chybovou stránku, protože do této sekce má mít přístup pouze přihlášený uživatel. Logika detailu objednávky si z url nejdříve „vezme“ ID, poté ho porovná se záznamy v databázi. Pokud se v databázi nachází záznam se stejným ID, tak ho načte a předá přes ViewBag do příslušného pohledu. Pokud se v databázi záznam nenachází, je uživatel přesměrován na stránku s chybovou hláškou.

```
namespace Eshop.Controllers
{
    0 references
    public class OrdersController : Controller
    {
        // GET: Orders
        0 references
        public ActionResult Index()
        {
            EshopUser user = new EshopUserDao().GetByLogin(HttpContext.User.Identity.Name);
            if (user != null)
            {
                EshopOrderDao eshopOrderDao = new EshopOrderDao();
                IList<EshopOrder> eshopOrders = eshopOrderDao.GetOrdersByUser(user.Id);
                ViewBag.Orders = eshopOrders;
                return View();
            }
            else return RedirectToAction("BadUrl", "Error");

        }

        public ActionResult Detail(int ?id)
        {
            bool validOrder = false;
            0 references
            IList<EshopOrder> orders = new EshopOrderDao().GetAll();
            foreach (EshopOrder p in orders) { if (id == p.Id) validOrder = true; }

            if (validOrder)
            {
                int ids = id ?? 0;
                EshopOrderDao eshopOrderDao = new EshopOrderDao();
                EshopOrder order = eshopOrderDao.GetById(ids);
                ViewBag.Orders = order;
                IList<EshopOrderItem> orderItems = new EshopOrderItemDao().GetOrderItems(ids);
                ViewBag.OrderItems = orderItems;
                return View();
            }
            else return RedirectToAction("BadUrl", "Error");
        }
    }
}
```

Obrázek 53 – OrdersController.cs

Zdroj: Vlastní zpracování

Zabezpečení

Další důležitým krokem vývoje internetového obchodu bylo řešení zabezpečení. Pro zabránění útoků na stránku jsme zvolili a implementovali několik ochranných mechanismů. Formuláře jsou validovány na straně serveru pomocí Data Annotations, na straně klienta pomocí pluginuj Query. Validation, jejímž autorem je Jörn Zaefferer. Veškeré vstupy z URL jsou ošetřeny. Vstup je porovnáván se záznamy v databázi, pokud s žádným nesouhlasí, je uživatel přesměrován na chybovou stránku. Uživatelské e-maily a hesla jsou při registraci před uložením do databáze „zahashované“ pomocí algoritmu SHA-256. Při přihlašování nebo odesílání e-mailu se porovnává *hash* ze vstupu přihlašovacího formuláře a hash uložený v databázi. Zabezpečení na úrovni oprávnění uživatelských rolí je řešeno pomocí integrovaných funkcí MVC aplikačního rámce. Uživatel musí

pro vybrané funkce dokázat, že je skutečně ten, za koho se vydává, a to zadáním uživatelského jména a hesla. Výsledkem procesu je potvrzeni nebo vyvrácení identity uživatele. Na základě těchto údajů dochází k ověření, zda autentizovaný uživatel má povolen přístup k požadovanému zdroji. Webová aplikace používá režim autentizace Form. Pohled s formulářem pro přihlášení je zobrazen níže.

```
<div class="narrow-content">
    <div>
    <h2 class="form-signin-heading">Přihlášení</h2>
    @if (TempData["Error"] != null)
    {
        <p>@TempData["Error"]</p>
    }
    @using (Html.BeginForm("SignIn", "Login", FormMethod.Post, new { @class = "form-signin" }))
    {
        <div class="form-group">

            <label for="inputEmail">Login</label>
            <input type="text" class="form-control labeled" placeholder="Uživatelské jméno" name="login">
        </div>
        <div class="form-group">
            <label for="inputPassword">Heslo</label>
            <input type="password" class="form-control labeled" placeholder="Heslo" name="password">
        </div>
        <button class="btn btn-lg btn-primary btn-block" type="submit">Přihlásit</button>
    }
    <p class="registration-link">Zapoměl/a jste heslo? @Html.ActionLink("Vytvořit nové", "Index", "Password")</p>
        </div>
</div>


<div class="remodal" data-remodal-id="loggedOut">

    <h1><span style="font-weight:300; color: #1a1a1a">Byl jste úspěšně</span>  odhlášen</h1>
    <br>
    <button data-remodal-action="close" class="remodal-cancel">Zavřít</button>
</div>

<div class="remodal" data-remodal-id="badCredentials">

    <h1><span style="font-weight:300; color: #1a1a1a">Uživatel s těmito údaji </span>neexistuje</h1>
    <br>
    <button data-remodal-action="close" class="remodal-cancel">Zavřít</button>
</div>
```

Obrázek 54 – Formulář přihlášení

Zdroj: Vlastní zpracování

Pohled na obrázku 55 popisuje zobrazení formuláře, vypsání chybových hlášek ve formuláři a strukturu modalových oken zobrazující stavová hlášení. Modalová okna jsou skryta, zobrazí se v případě, že kontrolér odkáže na pohled s přidaným identifikátorem modalového okna. Obrázek 56 zobrazuje logiku kontrolérů přihlášení a odhlášení uživatele.

```
[HttpPost]
0 references
public ActionResult SignIn(string login, string password)
{
    if (Membership.ValidateUser(login, PasswordHelper.GetHashString(password)))
    {
        FormsAuthentication.SetAuthCookie(login, false);
        return RedirectToAction("Index", "Home");
    }

    return new RedirectResult(Url.Action("Index", "Login") + "#badCredentials");
}

0 references
public ActionResult Logout()
{
    FormsAuthentication.SignOut();
    Session.Clear();
    return new RedirectResult(Url.Action("Index", "Login") + "#loggedOut");
}
```

Obrázek 55 – Přihlášení a odhlášení

Zdroj: Vlastní zpracování

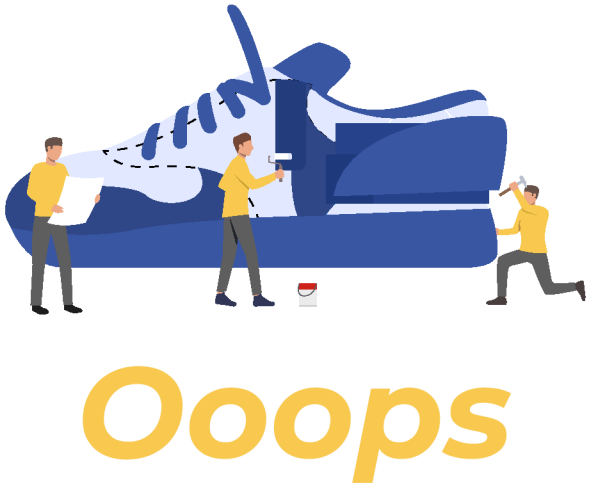
Obrázek 57 zobrazuje část kódu pohledu navigace, který zobrazuje položku seznamu pouze v případě, že uživatel je autorizovaný, a je v uživatelské roli „admin“.

```
@if (Roles.IsUserInRole("admin"))
{
    <li><a href="@Url.Action("Index", "User", new { area = "Admin" })">Správa uživatelů</a></li>
}
```

Obrázek 56 – Ověření role

Zdroj: Vlastní zpracování

Pro zabránění úniku citlivých informací jsme vytvořili vlastní chybové stránky pro zobrazení výjimečných stavů v internetovém obchodě, jako je například *chyba 404 (*tj. stránka nenalezena*)*, která je zobrazena na obrázku 58.



Obrázek 57 – Vlastní chybová hláška

Zdroj: Vlastní zpracování.

8.6 Nasazení

Webovou aplikaci jsme po implementaci na závěr „nasadili“ na webový hosting na doméně h*ttp://sneeeakers.cz*, kde je možné si aplikaci zobrazit a otestovat v několika rolích nastíněných v UML diagramu případů užití.

Ukázkové uživatelské účty

Pro demonstraci použití aplikace jsme vytvořili ukázkové uživatelské účty reflektující jednotlivé aktéry v diagramech případů užití.

Tabulka 1 – Ukázkové uživatelské účty

Název	Id	Heslo
Administrátor	administrator	UhkFim2021!
Skladník	coordinator	UhkFim2021!
Editor	editor	UhkFim2021!

Zdroj: Vlastní zpracování

9 Shrnutí výsledků

V teoretické části práce jsme rozebrali problematiku tvorby e-shopu. Definovali jsme, jak má vypadat vizuální identita značky, podle ní jsme vytvořili grafický návrh. Navrhli jsem modely v jazyce UML a BPMN, databázový model i programové řešení. V průběhu plánování a realizace projektu jsme o něm přemýšleli, a to v několika rovinách zároveň. Vytvořili jsme diagramy případů užití, na jejich základě jsme navrhli *wireframy*, které jsme následně převedli do *mockupů*. Na základě dat z předchozích činností jsme navrhli analytický a návrhový model, datovou vrstvu a naprogramovali jsme *front-end* i *back-end* aplikace. Při provádění každé z uvedených činností jsme přemýšleli, jak docílit toho, aby každá část systému byla ve všech směrech efektivní. Cílem i výsledkem je návrh řešení a implementace internetového obchodu.

10 Závěry a doporučení

Cílem práce bylo navrhnout a implementovat plnohodnotný internetový obchod. Zvolili jsme si projekt s názvem *Sneeeakers.cz*. Jde o e-shop s teniskami postavený na moderních technologiích. K dosažení vytyčeného cíle jsme si ujasnili, jak má samotný obchod fungovat, popsali jsme firemní procesy a vytvořili jsme diagramy za pomocí BPMN a UML. Navrhli jsme diagram případů užití a každému aktérovi jsme určili, jak se systémem bude pracovat. Vytvořili jsme konverzační diagram, který popisuje, jak spolu budou aktéři komunikovat. Dále jsme si upřesnili, jak se má firma prezentovat, na základě těchto informací jsme vytvořili jednotný vizuální styl skládající se z loga, ilustrací, vzorů, pravidel pro použití typografie a barev. Na základě vytvořeného vizuálního stylu jsme vytvořili design samotného webu a design kamenné prodejny.

Vybrali jsme použité technologie, a poté jsme navrhli entitně-relační diagram a datový model. Z datového modelu jsme vygenerovali a nakonfigurovali MSSQL databázi, kterou jsme následně namapovali na objekty pomocí NHibernate. Poté jsme napsali aplikační logiku, rovněž jsme naprogramovali zobrazení

jednotlivých pohledů. Na závěr jsme úspěšně vytvořený e-shop nasadili na doménu *sneeeakers.cz*, na které je možné ho přímo otestovat.

11 Seznam použité literatury

[1] WHITE, Stephen A. *BPMN modeling and reference guide: understanding and using BPMN:developrigorousyetunderstandablegraphicalrepresentationsof business processes*. Lighthouse Point: Future Strategies, c2008. ISBN 0977752720.

[2] *VisualParadigm: Whatis BPMN* [online]. 1802, Laford Center, 838 LaiChi Kok Road, KLN, Hong Kong: Visual Paradigm, c1999-2022 [cit. 2022-04-17]. Dostupné z: https://www.visual-paradigm.com/guide/bpmn/what-is-bpmn/

[3] SCHMULLER, Joseph. *Myslíme v jazyku UML*. Praha: Grada, 2001. Knihovna programátora (Grada). ISBN 80-247-0029-8.

[4] *Visual Paradigm: What is UML* [online]. 1802, Laford Center, 838 LaiChi Kok Road, KLN, Hong Kong: Visual Paradigm, c1999-2022 [cit. 2022-04-17]. Dostupné z: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/

[5] ARLOW, Jim a Ila NEUSTADT. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky*. Přeložil Bogdan KISZKA. Brno: Computer Press, 2007. ISBN 80-251-1503-8.

[6] SHERIN, Aaris. *Design elements, color fundamentals: a graphic style manualforunderstandinghowcolorimpacts design*. Beverly, Mass.: Rockport Publishers, 2011. ISBN 978-1-59253-719-8.

[7] WHEELER, A. *Designingbrand identity: a completeguide to creating, building, and maintainingstrongbrands*. 2nd ed. Hoboken: John Wiley, 2006. ISBN 0-471-74684-3.

[8] *Learn Design with Figma* [online]. San Francisco, CA: Figma, c2022 [cit. 2022-04-17]. Dostupné z: https://www.figma.com/resources/learn-design/

[9] NIEDERST ROBBINS, Jennifer. *LearningWeb design: a beginner'sguide to HTML, CSS, JavaScript, and web graphics*. Fourthedition. Beijing: O'Reilly, [2012]. ISBN 978-1-449-31927-4.

[10] CASTRO, Elizabeth a Bruce HYSLOP. *HTML5 a CSS3: názorný průvodce tvorbou WWW stránek*. Brno: Computer Press, 2012. ISBN 978-80-251-3733-8.

[11] FELKE-MORRIS, Terry. *Basicsof web design: HTML5 & CSS*. Fifthedition. NY, NY: Harper College, [2018]. ISBN 9780135225486.

[12] *W3Schools: Online Web Tutorials* [online]. c1999-2022 [cit. 2022-04-17]. Dostupné z: https://www.w3schools.com

[13] ATZENI, Paolo a Valeria DE ANTONELLIS. Relational database theory. Redwood City, Calif.: Benjamin/CummingsPub. Co., c1993. ISBN 0-8053-0249-2

[14] Jess, Chadwick. Programming ASP.NET MVC 4: Developing Real-World Web Applicationswith ASP.NET Mvc. vydání 1. United Statesof America: O'Reilly Media, Inc, 2012. 600 s. ISBN 978-1-449-32031-7.

[15] *Microsoft Docs: Dokumentace k ASP.NET* [online]. Redmond, Washington, USA: Microsoft, c2022 [cit. 2022-04-17]. Dostupné z: https://docs.microsoft.com/cs-cz/aspnet/core/?view=aspnetcore-6.0

[16] BAYER, Jürgen. *C# 2005: velká kniha řešení*. Brno: Computer Press, 2007. Programování. ISBN 978-80-251-1620-3.

[17] *Microsoft Docs: Dokumentace k C#* [online]. Redmond, Washington, USA: Microsoft, c2022 [cit. 2022-04-17]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/csharp/

[18] ŽÁRA, Ondřej. *JavaScript: programátorské techniky a webové technologie*. 2. vydání. Brno: Computer Press, 2021. ISBN 978-80-251-5026-9.

[19] *Microsoft: ASP.NET securityoverview* [online]. Redmond, Washington, USA: Microsoft, c2022 [cit. 2022-04-17]. Dostupné z: https://support.microsoft.com/en-us/topic/asp-net-security-overview-7c8562d3-7bea-306c-4c78-98dd6c6993b3

[20] *Kompletní e-shop v ASP.NET Core MVC: Online kurz* [online]. itnetwork.cz, 2022 [cit. 2022-04-27]. Dostupné z: https://www.itnetwork.cz/csharp/asp-net-core/e-shop

[21] *C# Corner: SecuringYour ASP.NET Web Applications* [online]. C# Corner, c2022 [cit. 2022-04-17]. Dostupné z: https://www.c-sharpcorner.com/article/securing-your-Asp-Net-web-applications/