Česká zemědělská univerzita v Praze
Provozně ekonomická fakulta
Katedra informačního inženýrství

Diplomová práce

Správcovské rozhraní mezinárodního webového odborného časopisu

Ondřej Badín

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Ondřej Badín

Systémové inženýrství a informatika
Informatika

Název práce

Správcovské rozhraní mezinárodního webového odborného časopisu

Název anglicky

Management Interface of the International Expert Magazine Website

---

Cíle práce

Cílem práce je vytvořit správcovské rozhraní pro mezinárodní webový odborný časopis Slovjani. Rozhraní bude umožňovat vkládání odborných článků a editaci bibliografických údajů o nich, přidávání a editaci citací a autorů. Přidávání autorů bude možné i na základě jejich ORCID, odborné články na základě jejich DOI.

Metodika

Nejprve bude provedena analýza funkčních požadavků. Na základě této analýzy bude provedena implementace v jazyce PHP s využitím standardů softwarového inženýrství, především UML a WebML, z bezpečnostních důvodů bez použití běžně dostupných CMS modulů. Součástí bude i uživatelská příručka.

Doporučený rozsah práce

60-100 stran

Klíčová slova

CMS, PHP, UML, WebML, Správcovské rozhraní

Doporučené zdroje informací

BARKER, Deane. Web content management: systems, features, and best practices. Boston: O'Reilly, 2016. ISBN 978-1491908129.

MACINTYRE, Peter. PHP: the good parts. Sebastopol, CA: O'Reilly, 2010. ISBN 9780596804374.

SILBERSCHATZ, Abraham, Henry F. KORTH a S. SUDARSHAN. Database system concepts. 6th ed. New York: McGraw-Hill, c2011. ISBN 978-0-07-352332-3.

WELLING, Luke a Laura THOMSON. PHP and MySQL Web development. 4th ed. Upper Saddle River, NJ: Addison-Wesley, 2009. Developer's library. ISBN 978-0-672-32916-6.

Předběžný termín obhajoby

2019/20 LS – PEF

Vedoucí práce

doc. Ing. Vojtěch Merunka, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 19. 2. 2020

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 19. 2. 2020

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 19. 02. 2020

Čestné prohlášení

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

V Praze dne 5. dubna 2020 ----------

Ondřej Badín

Poděkování

Rád bych touto cestou poděkoval doc. Ing. Vojtěchu Merunkovi, Ph.D. za příkladné vedení práce, za jeho podporu a trpělivost. Dále bych rád poděkoval panu děkanovi Ing. Martinu Pelikánovi, Ph.D., za umožnění studia na Provozně ekonomické fakultě ČZU.

Správcovské rozhraní mezinárodního webového odborného časopisu

Abstrakt

Práce se zabývá tvorbou správcovského rozhraní pro odborný webový časopis zabývající se slovanskou kulturou a jazyky. Toto rozhraní bylo navrženo za využití modelovacích technik definovaných specifikacemi UML a WebML. Rozhraní je vytvořeno v jazyce PHP (uživatelská část v kombinaci PHP, HTML a JavaScriptu), propojeno s MySQL databází a umožňuje uživatelům do databáze vkládat, upravovat a v případě potřeby i mazat data o autorech, kteří do časopisu přispívají, a článcích v časopise publikovaných. Tato data se poté v přehledné podobě zobrazují čtenářům a lze je využít i například pro generování metadat. Dále umožňuje spravovat autorizace uživatelů přistupujících k obsahu - neregistrovaní uživatelé mohou pouze procházet stránky, registrovaní uživatelé stahovat jednotlivá vydání časopisu a správci mohou využívat rozhraní pro správu obsahu. Aplikace by měla svými funkcemi zjednodušit stávající redakční procesy. Vytvořené modely by mělo být možné využít pro další rozšiřování aplikace, případně by se daly využít pro vytváření nových obdobných systémů.

Klíčová slova

CMS, WCM, PHP, JavaScript, JS, UML, WebML, Správcovské rozhraní, Hypertext, HTML

Management Interface of the International Expert Magazine Website

Abstract

This thesis deals with creation of a management interface for an expert web magazine about Slavic culture and languages. This interface was designed using modeling techniques defined in UML and WebML specifications. The interface was programmed in PHP (user interface in a combination of PHP, HTML and JavaScript), it is connected to a MySQL database and allows its users to insert, edit and, when needed, remove from the database information about authors contributing to the magazine and articles which are published in it. This data is then presented to the reader in clear form and can be further used, for example for generating metadata. The interface furhter allows the administrators to manage access authorization of the users - unregistered users may only navigate the pages, registerd users may download electronic issues of the magazine and administrators may use the management interface itself. The application thru its functionalities should make the current editorial processes easier. Models created as part of the application should be usable for futher expansion of the application, alternatively they could be used for creating new simmilar systems.

Keywords

CMS, WCM, PHP, JavaScript, JS, UML, WebML, Management interface, Hypertext, HTML

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Posledních několik desítek let žijeme v době internetové. Více než čtyři miliardy lidí na celém světě mají přístup ke globální počítačové síti internet a pravidelně jej využívají za různými účely. Na internetových serverech bychom v současnosti nalezli téměř dvě miliardy webových stránek. Každý den jsou zobrazeny a nahrány uživateli internetu miliardy videí, fotografií, textových článků a krátkých příspěvků nebo komentářů[1]. Uživatelé internetu vytvářejí a konzumují obsah.

Tento obsah je ovšem potřeba nějak spravovat, zpřístupňovat jej, upravovat, pokud to vyžaduje, mazat, pokud nesplňuje očekávání nebo je přímo nevhodný. Odborné časopisy možná netvoří velký podíl obsahu na internetu, ale i tyto musí řešit stejný problém - jak spravovat jimi vytvářený obsah. Pro tento účel existují nástroje pro správu webového obsahu, tzv. CMS systémy (zkratka z ang. content management system). Tyto nástroje za lidi sice obsah vytvářet samy neumí, umožňují jim však tento obsah snáze zpřístupnit širší veřejnosti.

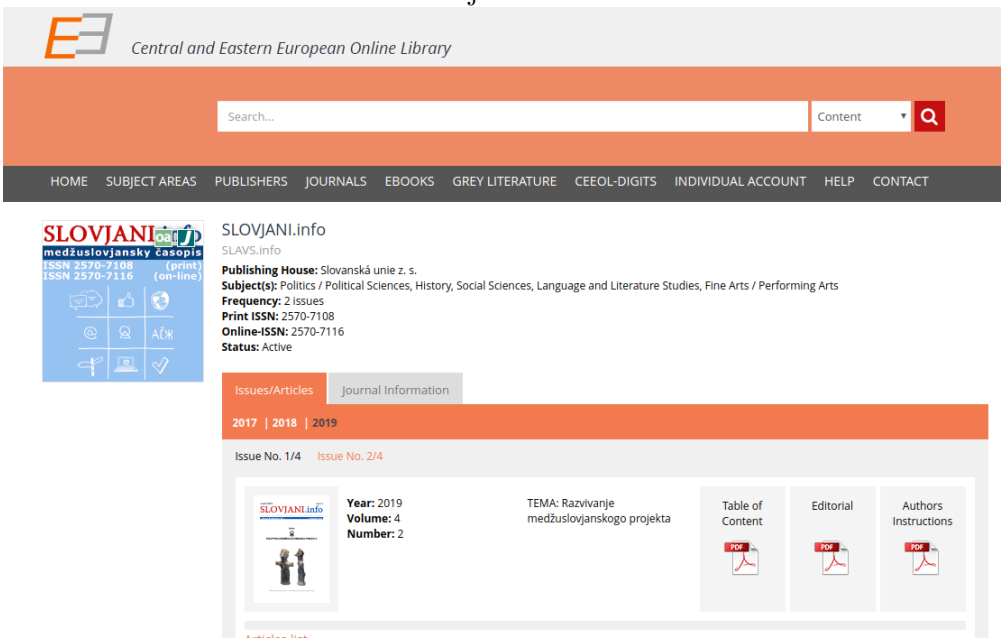
CMS systém pro odborný webový časopis by měl mimo jiné umožnit redakci podle potřeby vkládat, upravovat a mazat veškeré potřebné bibliografické informace k publikovaným článkům, tj. název, autory, abstrakt, klíčová slova, čerpané zdroje aj., spravovat informace o autorech těchto článků, tedy minimálně jejich jména, ale pokud je to možné, tak i kontakty na ně, instituce, ve kterých působí a další. Mimo to by měl umožnit i správu přístupu k tomuto obsahu. Náhodný uživatel přicházející odjinud z internetu například může zobrazovat články, stahovat elektronická vydání časopisu, ale už by neměl být schopen zasahovat do redakčního procesu, to by mělo být výsadou autorizovaných uživatelů s příslušným oprávněním.

Pro takový web je třeba tedy navrhnout takový systém, který umožňuje provádět všechny výše jmenované činnosti. Tento systém by měl být schopen do databáze ukládat všechny potřebné informace, dle potřeby je aktualizovat a mazat, vyhledávat v databázi a zobrazovat získaný obsah ve formě příslušné uživatelem zamýšlené činnosti (upravování, prohlížení atd.).

V rámci této práce by ke konci měl vzniknout systém pro správu obsahu pro

mezinárodní odborný webový časopis Slovjani.info. Tento časopis, zabývající se slovanskými národy, jejich kulturou a jazyky, je součástí mezinárodní online knihovny CEEOL[2].

Obrázek 1: Slovjani.info a CEEOL[2]



CEEOL (Central and Eastern European Online Library), sídlící ve Frankfurtu nad Mohanem, je předním poskytovatelem elektronických akademických časopisů a knihv oblasti humanitních a společenských věd pocházejících z a zabývajících se střední a východní Evropou. V současnosti je evidována více než tisícovka různých vydavatelů, kteří poskytují skrz CEEOL své knihy a časopisy. CEEOL poskytuje studentům, výzkumným a akademickým pracovníkům přístup k velké škále akademického obsahu. V této době je v CEEOL dostupných přes dva tisíce časopisů, téměř půl milionu článků, více než dva tisíce dvě sta e-knih a dva a půl tisíce dokumentů šedé literatury. CEEOL pokrývá velké množství výzkumných oblastí, od antropologie přes kulturu, ekonomii, náboženství, umění, lingvistiku až po filosofii a právo. Vydavatelé využívající služeb CEEOL svým členstvím získávají mnohé výhody, například indexaci publikací systémy Google Scholar, Primo a Summon, statistiky o čtenosti na úrovni jednotlivých článků nebo příležitost oslovit nové čtenáře skrz odebírající instituce. CEEOL také pořádá a

účastní se řady mezinárodních konferencí a sympózií o evropské kultuře, slavistice a knihovnictví[3].

Jakožto studnice středoevropské a východoevropské kultury lze CEEOL, založenou roku 1999, považovat za duchovního nástupce neziskové organizace Palais Jalta, která mezi lety 1989 a 2003 pořádala mnohá sympózia, kulturní a politické debaty a výstavy se zaměřením na střední a východní Evropu[4].

2 Cíl práce a metodika

2.1 Cíl práce

Cílem práce je vytvořit systém pro správu obsahu pro odborný webový časopis *Slovjani.info*. Systém bude umožňovat vkládání nových odborných článků, editaci bibliografických údajů o těchto článcích jejich autory, vkládání, editaci a možné znovupoužití citovaných zdrojů, přidávání nových autorů, pokud již nejsou v systému zaneseni. Systém by měl nahradit stávající metodu vkládání údajů, tj. přímý zápis do databáze. Součástí vypracování bude uživatelská příručka provádějící uživatele prací se systémem.

2.2 Metodika

Nejprve budou nastudovány potřebné teoretické podklady, tj. systémy pro správu dat, skriptovací jazyk PHP, systémy pro správu báze dat a systém MySQL, modelovací standardy UML a WebML. Na základě těchto podkladů a konzultací se zadavatelem bude následně vytvořen systém pro správu obsahu na webu odborného časopisu Slovjani, a to tak, že budou vypracovány podklady pro tvorbu systému v UML a WebML a následně bude vytvořen samotný systém za použití skriptovacího jazyka PHP. Z bezpečnostních důvodů nebudou pro tvorbu systému využity žádné předpřipravené CMS moduly (jako např. WordPress) ani existující PHP frameworky (např. Nette). Nakonec bude vytvořena příručka, která bude uživatele provádět prací se systémem, např. vkládáním nových článků, editací článků stávajících atd.

3 Teoretická východiska

V této části budou přiblíženy koncepty správy obsahu na webu, skriptovacího jazyka na straně serveru PHP a databázových systémů. Dále pak budou nastíněny principy modelovacích jazyků UML a WebML.

3.1 Databázové systémy

Systém řízení báze dat (SŘBD, angl. DBMS, Database Management System) je jednak kolekce vzájemně propojených dat a dále pak také sada aplikačního vybavení používaná pro přístup k těmto datům a manipulaci s nimi. Účelem těchto systémů je přinést efektivní a pohodlný způsob ukládání, získávání a manipulace s daty.

Správa data zahrnuje jednak definici struktur, do kterých se data mají ukládat, tak implementaci metod, pomocí kterých se datům bude přistupovat. Systém by měl také zajistit zabezpečení konzistence dat, např. v případě havárie, neoprávněného přístupu či současného přístupu k datům vícero uživateli[6].

3.1.1 Relační databázový model

Velmi častým způsobem reprezentace dat v SŘBD je relační datový model. V tomto datovém modelu jsou jak data, tak i vztahy mezi nimi, interně reprezentována jako sada tabulek.

Relační databáze sestává ze sady tabulek - relací, každá z nich má přidělen jedinečný název. Dále pak má každá relace hlavičku, která definuje názvy sloupců - atributy, a řádky se záznamy (n-ticemi, angl. n-tuple, entity). Atributy relace mají určenou doménu, což je sada všech přípustných hodnot, kterých tento atribut může nabývat. Nemá totiž smysl, aby např. atribut plat nesl záporná čísla nebo nečíselné hodnoty. Atributy relace mají zároveň být atomické, tj. dále nedělitelné - jeden atribut nesmí nést více než jednu hodnotu[6].

Je nutné mít určeno, jak jednoznačně identifikovat záznamy v databázi. Žádné

dva záznamy v databázi by neměly nést naprosto identické hodnoty. K tomuto účelu v relačních databázích slouží klíče. Množina jednoho a více atributů, které jednoznačně identifikují záznam v relaci se nazývá superklíč (angl. superkey). Pro identifikaci záznamů většinou hledáme minimální superklíče, tedy takové, které v sobě neobsahují další podmnožinu superklíčů. Takovýmto minimálním superklíčům se říká kandidátní klíče (angl. candidate key). Kandidátní klíč, který byl návrhářem databáze zvolen k identifikaci záznamu se nazývá primární klíč (angl. primary key). Je možné zahrnout v relaci r_1 primární klíč jiné relace (r_2) a tím tedy odkázat na data v této druhé relaci. Takovému klíči se potom říká cizí klíč (angl. foreign key), relaci r_1 odkazující relace a relaci r_2 odkazovaná relace. Platí potom, že pro každou hodnotu cizího klíče v n-tici v r_1 musí existovat v r_2 taková n-tice, hodnota jejíhož primárního klíče odpovídá hodnotě tohoto klíče cizího. Vztah mezi dvěma relacemi je orientovaný[6].

Pro interakci s databází existují tzv. dotazovací jazyky. Takové jazyky jsou obvykle vysokoúrovňové a mohou být jak procedurální - uživatel systému přímo zadává postup, jak získat kýžená data, tak neprocedurální - uživatel popíše, jaká data po systému požaduje, nespecifikuje však již, jakým způsobem mají tato data být získána. V praxi používané dotazovací jazyky tyto dva přístupy kombinují.

Všechny dotazovací jazyky umožňují nad jednou či párem relací provádět množinu tzv. relačních operací, nebo také operací relační algebry. Mezi tyto operace patří selekce, projekce, kartézský součin, sjednocení a mnohé další. Výsledkem těchto operací je vždy jedna relace, díky čemuž je možné tyto operace provádět v řadě za sebou, používat výstupní data z jedné operace jako vstupní data pro další[6].

- Základní operace relační algebry

  Selekce (σ)

  Bere jednu relaci na vstupu a vybírá z ní n-tice na základě podmínky, podmínka může být jak jednoduchá, tak složená pomocí booleovských operací and ($\wedge$), or ($\vee$) a not ($\neg$).

  $\sigma_{plat>15000}(zamestnanec)$ - nalezne všechny záznamy v tabulce zamestnanec, kde je plat vyšší než 15 000

$\sigma_{plat<100000 \wedge pozice=\text{“}senator\text{”}}(zamestnanec)$ - nalezne všechny záznamy v tabulce zamestnanec, kde plat je nižší 100 000 a pozice zaměstnance je senátor

Projekce (Π)

Z jedné relace na vstupu vytvoří na výstupu relaci novou, která bude obsahovat pouze uživatelem specifikované atributy.

$\Pi_{ID,jmeno,pozice}(zamestnanec)$ - vybere z tabulky zamestanec pouze sloupce ID, jmeno a pozice

Sjednocení ($\cup$)

Operace, která spojuje dvě relace. Funguje tak, jak je definována v teorii množin - záznamy z obou relací se spojí do jedné, shodné záznamy se ignorují. Relace na vstupu musí být vzájemně kompatibilní, tj. musí mít shodný počet atributů n a atribut $a_{i=<1,n>}$ v obou vstupních relacích musí mít shodnou doménu. Jelikož výsledky relačních operací jsou opět relace, je možné sjednocovat i výsledky těchto operací.

Rozdíl ($-$)

Z dvou relací na vstupu vytváří novou relaci, která obsahuje pouze prvky z relace první, které nejsou v relaci druhé, tedy při $r - s$ vrací pouze prvky v r, které nejsou v s

Kartézský součin ($\times$)

Tato operace umožňuje kombinovat jakékoliv dvě relace na vstupu. Výsledkem operace je nová relace, která obsahuje všechny atributy vstupních relací. Jelikož se může stát, že tyto atributy budou mít v obou relacích stejné názvy, je nutné definovat názvy nové. To se běžně provádí tak, že se k názvu atributu připíše název relace, ze které tento atribut původně pocházel (např. atribut plat z tabulky zamestnanec je nově identifikován jako plat.zamestnanec). Kartézský součin vrací všechny možné kombinace n-tic z obou relací.

$zamestnanec \times adresa$ - vrací všechny možné kombinace záznamů v relacích zamestanec a adresa

$\sigma_{zamestnanec.adresa=adresa.id}(zamestnanec \times adresa)$ - pomocí následovné selekce vrací pouze ty kombinace záznamů, kde záznam z relace adresa odpovídá bydlišti zaměstnance

Přejmenování (ρ)

Dočasně přejmenovává relaci (ať už předem existující, nebo výsledek operace relační algebry) pro použití v dalších operacích. Operace je využitelná např. při provádění kartézského součinu na dvou shodných relacích[6].

- Dodatečné operace relační algebry

  Základní operace relační operace jsou dostačující pro vyjádření jakéhokoliv dotazu, pokud však budeme používat pouze tyto základní operace, může nastat situace, že jednoduchý dotaz budeme složitě vyjadřovat dlouhou řadou těchto operací. Proto jsou definovány dodatečné operace, které slouží ke zjednodušení zápisu dotazů.

  Průnik ($\cap$)

  Párová operace, která vrací relaci obsahující pouze ty n-tice z první vstupní relace, které se zároveň vyskytují ve druhé vstupní relaci.

  $r \cap s = r - (r - s)$

  Přirozené spojení (natural join, $\bowtie$)

  Zjednodušuje dotazy vyžadující kartézský součin, konkrétně ty operace, které vyžadují následnou selekci a tato selekce zajišťuje, že shodné atributy v obou relacích na vstupu kartézského součinu jsou si rovny. Pokud nemají relace na vstupu žádné shodné atributy, je výsledek operace roven kartézskému součinu.

  $r \bowtie s = \Pi_{R \cup S}(\sigma_{r.a_1=s.a_1 \wedge r.a_2=s.a_2 \wedge \ldots \wedge r.a_n=s.a_n}(r \times s))$

  Přiřazení (assigment, $\leftarrow$)

  Přiřadí výsledku relační operace dočasný název (uloží jej do proměnné), pomocí kterého je možné s tímto výsledkem dále pracovat. Slouží pro usnadnění zápisu.

  Vnější spojení (outer join)

Vnější spojení rozšiřuje přirozené spojení o způsob, jak zacházet z řádky, kde na jedné straně spojení chybí informace.

Levé vnější spojení (left outer join, $⟕$) - n-tice na levé straně, kterým neodpovídají n-tice na pravé straně, jsou doplněny o prázdné hodnoty a zahrnuty do výsledné relace.

Pravé vnější spojení (right outer join, $⟖$) - symetrické k levému vnějšímu spojení - n-tice na pravé straně, kterým neodpovídají n-tice na levé straně, jsou doplněny o prázdné hodnoty a zahrnuty do výsledné relace.

Úplné vnější spojení (full outer join, $⟗$) - kombinace pravého a levého vnějšího spojení[6]

- Rozšířující operace relační algebry.

Rozšířující operace relační algebry jsou takové operace, které umožňují psát dotazy, které pomocí základních a dodatečných operací zapsat nelze.

Obecná (generalizovaná) projekce

Obecná projekce rozšiřuje standardní projekci o možnost zápisu dalších operací, např. aritmetických funkcí.

Agregace ($\mathcal{G}$)

Agregace umožňuje na množině n-tic provádět agregační funkce, jako zjištění počtu výskytů (count), maximální a minimální hodnoty (max, min), průměru (avg), nebo sumy hodnot (sum). Pokud je nutné před aplikací agregační funkce eliminovat shodné hodnoty, dělá se to připsáním klíčového slova distinct odděleného pomlčkou za název funkce.

$\mathcal{G}_{count(ID)}(zamestnanec)$ - vrátí počet záznamů v relaci zamestnanec

Výsledky agregačních funkcí je možno seskupovat podle hodnoty jednoho z atributů vstupní relace.

${}_{pozice}\mathcal{G}_{avg(plat)}(zamestnanec)$ - vrátí průměrný plat zaměstnance seskupený podle pracovní pozice[6]

Při manipulaci s daty v databázi může nastat situace, kdy uživatel udělal chybu a potřebuje se vrátit do předchozího stavu. K tomuto a dalším účelům slouží transakce. Transakce jsou logické celky práce, příkazů.

Transakce jsou atomické - dále nedělitelné, musí se provést celé nebo vůbec, jsou izolované od ostatních právě prováděných dotazů nad databází, čímž se zachovává konzistence databáze, a jakmile jsou vykonány, musí se jimi provedené změny projevit v databázi, nesmí být zahozeny nebo ztraceny (jsou trvanlivé). Těmto vlastnostem transakcí se někdy také říká ACID podle prvních písmen anglických názvů těchto vlastnosti - atomicity, consistency, isolation, durability.

3.1.2 SQL

V současné době nejčastěji používaným dotazovacím jazykem je SQL (Structured Query Language). SQL, tehdy pojmenovaný Sequel, byl vyvinut na začátku 70. let 20. století společností IBM. Různé implementace tohoto jazyka se od sebe mohou lehce lišit, existuje však referenční verze jazyka - v roce 1986 byl jazyk SQL standardizován Americkým národním standardizačním institutem (ANSI) jako SQL-86. Standard byl od té doby několikrát aktualizován[6], aktuální verze v době publikace této práce je SQL:2016[7].

SQL neslouží pouze k dotazování. Jazyk je rozdělena na několik částí[6]:

- Data definition language (DDL)

  příkazy pro definici struktury dat - definice schémat relačních tabulek, jejich modifikace a mazání

  Integritní omezení

  podmínky, které musí splňovat data ukládaná do databáze

  Pohledy do databáze

  uložené tabulky vzniklé jako výsledky dotazů, není možné je upravovat

  Autorizace

v SQL DDL je možné definovat přístupová práva jednotlivých uživatelů k relacím a pohledům

- Data manipulation language (DML)

  příkazy pro manipulaci s daty v databázi - dotazování na data v databázi, vkládání, upravování a mazání záznamů

- Transakce

  SQL definuje příkazy pro specifikaci počátku a konce transakce, tedy uceleného bloku příkazů, který je možné v případě potřeby odvolat a vrátit tak databázi do předchozího stavu

Jazyk pro definici dat SQL umožňuje specifikaci schématu jednotlivých relací, datových typů jednotlivých atributů a jejich integritních omezení, ale i přístupová práva uživatelů k relaci atd[6].

Tabulka 1: Standardní datové typy v SQL[6]

char(n)	řetězec znaků o pevné délce n, také *character*
varchar(n)	řetězec znaků o proměnlivé délce s maximální délkou n, také *character varying*
int	celé číslo, rozsah závisí na architektuře serveru, také *integer*
smallint	malé celé číslo
bigint	velké celé číslo
numeric(p, d)	číslo s pevnou desetinnou čárkou, p je počet číslic, d je počet číslic z p, které se nacházejí za desetinnou čárkou
real, double-precision	číslo s plovoucí desetinnou čárkou s jednoduchou nebo dvojitou přesností
float(n)	číslo s plovoucí desetinnou čárkou s přesností na alespoň n desetinných míst
date	datum
time	časové razítko
binary	
binary large object	binární data
	a další...
	všechny datové typy mohou, pokud je to v návrhu relace povoleno, obsahovat také prázdnou hodnotu - NULL

Pro vytváření datových struktur v jazyce SQL slouží příkaz CREATE. Pomocí tohoto příkazu se dají vytvářet databáze, relace a pohledy[8].

Pro úpravu schématu relačních tabulek v rámci SQL DDL slouží příkaz ALTER. Tento příkaz se dá použít jednak pro přidávání a odstraňování sloupců z relace, potom také pro změnu datových typů jednotlivých sloupců v rámci relační tabulky[8].

Obrázek 2: Příkaz CREATE v SQL (vlastní)

```
-- vytvoření databáze
CREATE DATABASE nazev_databaze;
-- vytvoření relační tabulky
CREATE TABLE nazev_relace
 (atribut_1 datovy_typ,
  atribut_2 datovy_typ,
  ...,
  atribut_n datovy_typ,
  integitni_omezeni_1,
  ...,
  integritni_omezeni_k);
-- vytvoření pohledu
CREATE VIEW nazev_pohledu AS
  /* selekce */;
```

Pro mazání relačních tabulek, pohledů a databází se v SQL DDL používá příkaz DROP[8].

Jazyk pro manipulaci s daty v SQL definuje příkazy pro selekci a projekci dat, spojování tabulek, vkládání záznamů do tabulky, jejich úpravu a mazání. V rámci selekce dále poskytuje agregační funkce a příkazy pro seskupování a řazení dat[6].

Příkaz SELECT v SQL DML slouží k selekci a projekci dat z vybraných relací. Při zápisu dotazu po příkazu SELECT následuje projekce - výběr sloupců, poté klíčové slovo FROM následované relací, na které provádíme selekci, poté případně následuje klíčové slovo WHERE, po kterém následuje omezující podmínka pro tuto selekci. Složené podmínky spojujeme klíčovými slovy AND, OR a NOT.

Je-li vhodné získat pouze naprosto unikátní data, tedy vyřadit z výsledků dotazu ty záznamy, které se shodují v hodnotě alespoň jednoho sloupce, použije se namísto klíčového slova SELECT slovo SELECT DISTINCT.

Obrázek 3: Příkaz ALTER v SQL (vlastní)

```
-- přidání sloupce
ALTER TABLE nazev_relace ADD nazev_sloupce datovy_typ;
-- odstranění sloupce
ALTER TABLE nazev_relace DROP COLUMN nazev_sloupce;
-- změna datového typu sloupce
ALTER TABLE nazev_relace ALTER COLUMN nazev_sloupce datovy_typ;
```

Obrázek 4: Příkaz DROP v SQL (vlastní)

```
-- smazání tabulky
DROP TABLE nazev_tabulky;
-- smazání pohledu
DROP VIEW nazev_pohledu;
-- smazání databáze
DROP DATABASE nazev_databaze;
```

Výsledky dotazu je možné řadit podle hodnot jednoho či více sloupců. Za tímto účelem existuje klíčové slovo ORDER BY, za které se zapisuje seznam sloupců, podle kterých se mají výsledky řadit, následované klíčovým slovem ASC nebo DESC, podle toho, zda se má řadit vzestupně nebo sestupně. Pokud řadíme podle více sloupců, postupuje se v seznamu sloupců zleva doprava, pokud v jednom sloupci narazíme na více stejných hodnot, řadí se záznamy podle následujícího sloupce v seznamu do té doby, než se v předchozím sloupci narazí na novou hodnotu[8].

Obrázek 5: Selekce v SQL (vlastní)

```
SELECT [DISTINCT] sloupec_1, sloupec_2, ..., sloupec_n
FROM tabulka
[WHERE podmínka_1 [AND|OR|NOT ... AND|OR|NOT podmínka_k]]
[ORDER BY sloupec_1, ..., sloupec_i ASC|DESC];
```

Pro spojování relací v SELECT dotazech SQL DML existuje několik způsobů zápisu, které se však mírně liší podle implementace, standardizovaným a tedy i univerzálním zápisem je tzv. tečková notace. Spočívá v tom, že za klíčové slovo FROM zapíšeme seznam relací, na kterých se provádí selekce a v klauzuli WHERE se poté specifikují podmínky spojení tak, že se názvům sloupců předsadí tečkou oddělený název relace, ve které se vyskytují, čímž se mimo jiné zabrání případným kolizím v názvech sloupců[6].

Obrázek 6: Spojení relací v rámci selekce v SQL (vlastní)

```
SELECT t_1.s_1, t_1.s_2, ..., t_n.s_k
FROM t_1, ..., t_n
WHERE t_1.s_1 = t_2.s_1, ...;

-- příklad, spojení relací osoba a adresa
-- osoba(id, jmeno, adresa_id, datnar)
-- adresa(id, ulice, cp, mesto)
SELECT osoba.jmeno, osoba.datnar, adresa.ulice, adresa.cp, adresa.mesto
FROM osoba, adresa
WHERE osoba.adresa_id = adresa.id;
```

SQL DML podporuje v rámci příkazu SELECT agregační funkce, konkrétně COUNT, AVG, SUM, MIN a MAX. Tyto funkce se zapisují do sekce projekce a jako argument přijímají název sloupce, na jehož hodnotách se vykonávají. Při provádění funkce se

ignorují prázdné (NULL) hodnoty.

Ve výsledné relaci lze také seskupovat řádky, které mají stejnou hodnotu vybraného sloupce. Seskupování se často používá v kombinaci s agregačními funkcemi. Pro seskupování slouží konstrukt GROUP BY následovaný seznamem sloupců, podle kterých budou řádky seskupeny. Pokud je potřeba seskupení omezit podmínkou, slouží pro to klíčové slovo HAVING následované seznamem podmínek. Klíčové slovo HAVING funguje stejně jako slovo WHERE, ale vztahuje se pouze na seskupení, proto, je-li třeba předem omezit podmínkou neseskupené řádky, je nutno zapsat nejprve klauzuli WHERE a až potom GROUP BY HAVING[8].

Obrázek 7: Seskupování v rámci selekce v SQL (vlastní)

```
SELECT sloupec_1, ..., sloupec_n
FROM tabulka
[WHERE podmínka]
GROUP BY sloupec_k
[HAVING podmínka];

-- příklad, počet osob podle města
-- osoba(id, jmeno, adresa_id, datnar)
-- adresa(id, ulice, cp, mesto)
SELECT count(osoba.id), adresa.mesto
FROM osoba, adresa
WHERE osoba.adresa_id = adresa.id
GROUP BY adresa.mesto;
```

Pro sjednocení výsledků dvou selekcí v SQL DML slouží klíčové slovo UNION. Jedná se o párový operátor, který na obou stranách očekává dotazy typu SELECT. Relace, na kterých se provádí selekce, musí mít stejný počet sloupců, tyto sloupce musí mít shodné nebo alespoň obdobné datové typy (např. float a real) ve stejném pořadí v obou relacích. UNION standardně ignoruje duplicitní hodnoty, pokud je třeba povolit

duplicitní hodnoty, používá se klíčové slovo UNION ALL[8].

Obrázek 8: Sjednocení v SQL (vlastní)

```
SELECT sloupce FROM tabulka_1 ...
UNION [ALL]
SELECT sloupce FROM tabulka_2 ...;
```

Pro podmínky v rámci klauzulí WHERE a HAVING jsou v SQL DML definovány jednoduché porovnávací operátory, operátor ověření rozsahu, operátor pro zjištění, zda hodnota existuje v seznamu hodnot a operátor pro ověření, zda textový řetězec obsahuje hledaný vzor[8].

Tabulka 2: Porovnávací operátory v SQL[8]

=	rovnost
<>	nerovnost
>	hodnota větší než
<	hodnota menší než
>=	hodnota větší nebo rovna
<=	hodnota menší nebo rovna
BETWEEN	hodnota se nachází v rozsahu od–do
IN	hodnota se nachází v seznamu hodnot
LIKE	řetězec odpovídá vzoru

Operátor LIKE umožňuje ověřovat, zda hodnota textového řetězce odpovídá zadanému vzoru. Tento vzor, zadaný také jako textový řetězec, může být buď přesná hodnota, nebo lze využít tzv. wildcards (nebo také zástupných) znaků, které zastupují jeden nebo více znaků v řetězci[8].

Tabulka 3: Zástupné znaky v SQL[8]

%	procento, nula a více znaků	$h\% = \{ha, ho, hi, had, hod, ...\}$
_	podtržítko, právě jeden znak	$h\_ = \{ha, hb, hc, ...\}$
[]	právě jeden ze znaků v hranatých závorkách	$h[ao]d = \{had, hod\}$
^	stříška, právě jeden znak, který se nenachází v hranatých závorkách	$h[\hat{}ao]d = \{hbd, hcd, hdd, ...\}$
-	pomlčka, právě jeden znak z rozsahu	$h[a-c]d = \{had, hbd, hcd\}$

Operátor IN slouží k zjištění, zda se hodnota argumentu nachází v seznamu hodnot. Tento seznam může být zadaný nejen explicitně, jako seznam hodnot oddělených čárkami v kulatých závorkách, ale i jako příkaz SELECT. Operátor IN se používá v zastoupení pro větší množství podmínek, kde se porovnává hodnota stále stejného argumentu a jsou vzájemně propojeny logickým operátorem OR[8].

Obrázek 9: Operátor IN v SQL (vlastní)

```
-- explicitně zadaný seznam hodnot
SELECT sloupce
FROM tabulka
WHERE sloupec IN (hodnota1, hodnota2, hodnota3, ...);

-- seznam hodnot zadaný z dotazu
SELECT sloupce
FROM tabulka
WHERE sloupec IN (SELECT sloupec_ FROM tabulka_);
```

Za účelem vkládání dat do relací v rámci SQL DML je definován příkaz INSERT INTO. Tento příkaz umožnuje kromě vložení dat tak, jak jsou specifikována v návrhu relace, i vložení dat pouze do uživatelem specifikovaných sloupců, pokud to návrh relace umožňuje, tedy pokud vynechané sloupce mají povoleno zůstat nevyplněny (obsahovat

NULL hodnotu) nebo mají povolenu automatickou inkrementaci hodnoty předchozího záznamu[8].

Obrázek 10: Příkaz INSERT INTO v SQL (vlastní)

```
INSERT INTO tabulka [(sloupec1, sloupec2, sloupec3, ...)]
VALUES (hodnota1, hodnota2, hodnota3, ...);
```

Kromě explicitního zadávání dat lze pro specifikaci vkládaných dat využít i příkaz SELECT. Zdrojová tabulka ovšem potom musí splňovat určité podmínky - sloupce tabulky musí mít shodné datové typy a datové typy sloupců se musí vyskytovat ve stejném pořadí jako v cílové tabulce[8].

Obrázek 11: INSERT INTO SELECT v SQL (vlastní)

```
INSERT INTO tabulka2
SELECT * FROM tabulka1
[WHERE podmínka];

-- se specifikací sloupců
INSERT INTO tabulka2 (sloupec1, sloupec2, sloupec3, ...)
SELECT sloupec1, sloupec2, sloupec3, ...
FROM tabulka1;
```

Jelikož databáze jsou zřídkakdy statické, je třeba mít možnost modifikovat existující záznamy. K tomu v SQL DML slouží příkaz UPDATE. Ten na základě omezující podmínky přiřadí uživatelem specifikovaným sloupcům v relaci nové hodnoty. Je-li tato omezující podmínka vynechána, budou změněny všechny záznamy v relaci[8].

Obrázek 12: Příkaz UPDATE v SQL (vlastní)

```
UPDATE tabulka
SET sloupec1 = hodnota1, sloupec2 = hodnota2, ...
WHERE podmínka;
```

Pro mazání záznamů z delace v SQL DML slouží příkaz DELETE. Tento příkaz vymaže všechny záznamy specifikované zadanou omezující podmínkou[8].

Obrázek 13: Příkaz DELETE v SQL (vlastní)

```
DELETE FROM tabulka
WHERE podmínka;
```

Uživatel při práci s databází může vytvořit příkazem SET TRANSACTION novou transakci, v rámci které se budou provádět všechny jeho dotazy od vytvoření transakce až po její potvrzení příkazem COMMIT. Výsledky dotazů se až do potvrzení transakce neukládají fyzicky do databáze, zůstávají pouze v paměti. Díky tomu má uživatel možnost v případě chyby odvolat příkazem ROLLBACK všechny změny provedené od vytvoření transakce. Někdy není žádoucí odvolávat všechny změny. Pokud tomu tak je, lze vytvořit v rámci transakce příkazem SAVEPOINT záchytné body, ke kterým se dá poté příkazem ROLLBACK TO vrátit[9].

Obrázek 14: Transakce v SQL (vlastní)

```
SET TRANSACTION; -- vytvoří novou transakci
ROLLBACK; -- vrátí změny zpět do stavu při vytvoření transakce
SAVEPOINT jmeno; -- vytvoří záchytný bod
ROLLBACK TO jmeno; -- vrátí zpět změny od vytvoření záchytného bodu
COMMIT; -- potvrdí transakci, zapíše změny do databáze
```

3.2 PHP

PHP (zkratka původně znamenala Personal Home Page, dnes PHP: Hypertext Preprocessor) je interpretovaný skriptovací jazyk pro vývoj dynamických webových stránek používaný na straně serveru. Koncový uživatel nemá na rozdíl např. od jazyka JavaScript přístup ke zdrojovým kódům. Umožňuje vývojářům mimo jiné dynamické reakce na uživatelský vstup nebo napojení webu na databázový systém. Výsledné stránky generované interpretací PHP skriptů jsou ve skutečnosti statické, ale v praxi se většinou díky napojení na průběžně aktualizovanou databázi v kombinaci se zpracováním uživatelského vstupu koncovému uživateli jeví jako dynamické[10].

3.2.1 Syntaxe a základní operátory

Jazyk PHP svou základní syntaxí vychází z programovacího jazyka C. Bloky kódu jsou ohraničeny složenými závorkami, jednotlivé příkazy jsou zakončeny středníkem. Jména proměnných jsou uvozena znakem dolaru. Funkce jsou volány jménem funkce následovaným závorkami ve kterých se nacházejí případné argumenty. Je-li funkce či proměnná součástí instance třídy, je zpráva o volání tohoto objektu předávána pomocí šipky vpravo - ->, v případě proměnných se tyto potom nezapisují s dolarem.

Komentáře v kódu se zapisují pomocí kombinací lomítka a hvězdičky, případně mřížkou:
// pro jednořádkový inline komentář (může být na stejném řádku jako funkční kód, ale všechny znaky za ním se berou jako součást komentáře), /* */ pro víceřádkový inline

komentář (může být obklopen funkčním kódem) a # pro řádek komentáře (na stejném řádku nemůže být funkční kód).

PHP skript může stát jako samostatný soubor nebo být začleněn do struktury HTML souboru, v obou případech ale výsledný soubor bude mít příponu *.php*. PHP skript je vždy umístěn v PHP tagu - *<?php ?>*, ukončuje-li skript soubor, není nutné tento tag ukončovat. Je možný i krátký zápis tohoto tagu - *<? ?>*, obvykle však není implicitně povolen[11].

Obrázek 15: Příklad PHP syntaxe (vlastní)

```
<?php
// toto je příklad jednořádkového inline komentáře,
// může být vložen za funkční kód v jednom řádku

/*
* toto je příklad
* víceřádkového komentáře
*/

# toto je řádkový komentář

$promenna = 1; // příklad proměnné, s přiřazením hodnoty
funkce($argument); // volání funkce
$objekt->funkce($argument); // volání funkce z objektu třídy
$objekt->promenna = 0; // přiřazení hodnoty proměnné,
                       //která je součástí objektu třídy
?>
```

Jazyk PHP nabízí pro manipulaci s proměnnými následující operátory[12]:

- Aritmetické operátory

+	součet	$x + $y = suma $x a $y
-	rozdíl	$x - $y = rozdíl $x a $y
*	násobek	$x * $y = násobek $x a $y
/	podíl	$x / $y = podíl $x a $y
%	modulo	$x % $y = zbytek po dělení čísla $x číslem $y
**	exponent	$x ** $y = x^{\$y}$
++	inkrementace	$x++ = $x + 1, potom vrátí $x
		++$x = vrátí $x, potom $x + 1
--	dekrementace	$x-- = $x - 1, potom vrátí $x
		--$x = vrátí $x, potom $x - 1

- Přiřazovací operátory

=	přiřazení	proměnné na levé straně přiřadí hodnotu výrazu na pravé straně
+=	součet	k proměnné na levé straně přičte hodnotu výrazu na pravé straně
-=	rozdíl	od proměnné na levé straně odečte hodnotu výrazu na pravé straně
=	násobek	proměnnou na levé straně vynásobí hodnotou výrazu na pravé straně
/=	podíl	proměnnou na levé straně vydělí hodnotou výrazu na pravé straně
%=	modulo	proměnné na levé straně přiřadí zbytek po dělení proměnné hodnotou výrazu na pravé straně

- Porovnávací operátory
  - `==` rovnost, porovnává pouze hodnotu, při porovnání provádí konverzi mezi datovými typy
  - `===` identita, kromě hodnoty proměnné porovnává i shodu datového typu
  - `!=` nerovnost
  - `<>` nerovnost, jiný styl zápisu
  - `!==` neidentičnost
  - `>` větší než
  - `<` menší než
  - `>=` větší nebo rovno
  - `<=` menší nebo rovno
  - `<=>` tzv. spaceship operátor, vrací hodnotu menší, větší nebo rovnu jedné, podle toho, zda je hodnota na levé straně větší, menší nebo rovna hodnotě na pravé straně
- Logické operátory
  - `and` logické and
  - `or` logické or
  - `xor` logické xor - exclusive or
  - `&&` logické and
  - `||` logické or
  - `!` logické not
- Operátory pro práci se znakovými řetězci
  - `.` konkatenace, spojení dvou řetězců
  - `.=` konkatenace přiřazením, k řetězci v proměnné na levé straně se připojí řetězec z výrazu na pravé straně
- Operátory pro práci s poli

+	spojení polí
==	rovnost, zda pole obsahují shodné páry klíč/hodnota
===	identita, jako rovnost, ale porovnává se i pořadí hodnot a jejich datový typ
!=	nerovnost
<>	nerovnost, jiný způsob zápisu
!==	neidentičnost

3.2.2 Datové typy a struktury

Jazyk PHP podporuje deset primitivních datových typů rozdělených do třech skupin - skalární, složené a speciální.

Ze skalárních datových typů se jedná o typy boolean - pravdivostní hodnota true/false, integer - celé číslo, float - číslo s plovoucí desetinnou čárkou a string - řetězec znaků.

Složené datové typy jsou array - pole prvků, object - instance třídy, callable - volatelné struktury - funkce a iterable - pole nebo objekt implementující interface Traversable, dá se nad nimi iterovat smyčkou foreach.

Do skupiny speciálních datových typů patří typy resource - odkaz na externí zdroj, NULL - prázdná hodnota[11].

Tabulka 4: Rozsahy hodnot skalárních datových typů v PHP

boolean	$< 0, 1 >, < false, true >$
integer (32-bit)	$< -2^{32}, 2^{32} - 1 >$
integer (64-bit)	$< -2^{64}, 2^{64} - 1 >$
float	$< -1.79769E + 308, 1.79769E + 308 >$
	max. přesnost - 14 des. míst
string (32-bit)	$2^{31} - 1$ znaků
string (64-bit)	délka není omezena, resp. je
	omezena nastaveným paměťovým limitem

Základní datovou strukturou v PHP je pole. Pole jsou v PHP implementovány

jako řazené mapy. Mapa obecně je struktura, která přiřazuje hodnoty klíčům, pomocí kterých jsou tyto hodnoty následně dohledatelné. Datová struktura pole se v PHP dá použít i jako seznam, hashovací tabulka - hodnoty jsou dohledatelné pomocí jejich hashů, slovník - pár klíč => hodnota definován v obou směrech, kolekce, fronta atd. Pole mohou obsahovat jako hodnoty další pole, tímto se dají vytvořit vícerozměrná pole a stromy.

Pole se v PHP vytváří konstruktem array(). Ten jako argumenty přijímá n hodnot oddělených čárkou, případně n párů klíč => hodnota.

Při přiřazování hodnot klíčům se provádí konverze datových typů těchto klíčů následujícím způsobem:

- řetězce obsahující platné celé číslo jsou převedeny na celé číslo s touto hodnotou,
- čísla s plovoucí desetinnou čárkou jsou zaokrouhlena na nejbližší celé číslo dolu,
- pravdivostní hodnoty jsou převedeny na celé číslo - false = 0, true = 1,
- NULL je převeden na prázdný řetězec,
- pole a objekty nemůžou být klíčem, jejich použití vyústí ve varování.

Protože PHP nerozlišuje mezi indexovanými a asociativními poli, je možné kombinovat číselné a znakové klíče.

Obrázek 16: Vytváření polí v PHP (vlastní)

```
// bez specifikace klíčů, klíče budou celá čísla počínaje 0
$pole = array("hodnota_1", "hodnota_2");
$pole = array(1, 5, 10, 100, 0);
// se specifikací klíče
$pole = array( "foo" => "bar", "bar" => "foo");
// kombinace číselných a znakových klíčů
$pole = array( "foo" => "bar", 1 => "foo", "foobar" => 150);
```

K hodnotám v poli se přistupuje pomocí hranatých závorek, mezi nimiž je specifikován klíč, ke kterému je přiřazena požadovaná hodnota. Tento zápis se používá i pro modifikaci hodnot v poli. Pokud je takto přiřazena hodnota v poli zatím neexistujícímu klíči, je vytvořen nový pár klíč => hodnota a přidán na konec pole. Páry se dají z pole odstranit pomocí funkce unset[11].

Obrázek 17: Práce s poli v PHP (vlastní)

```
//přistupování k hodnotám v poli
$pole["klíč"];

//modifikace hodnot
$pole["klíč"] = "hodnota";

//odstraňování párů klíč => hodnota z pole
unset($pole["klíč"]);
```

3.2.3 Funkce

Funkce jsou způsobem vytváření znovupoužitelného kódu, který můžeme opakovaně volat, může transformovat data na vstupu a případně vracet hodnotu.

V PHP jsou funkce definovány klíčovým slovem function následovaným názvem funkce a seznamem jejích parametrů oddělených čárkami vepsaným do kulatých závorek. Název funkce může být jakýkoliv kromě rezervovaných slov jazyka PHP. Za definicí následuje blok kódu ve složených závorkách - tělo funkce, který zajišťuje funkcionalitu. Pokud má funkce vracet hodnotu, je toto zajištěno pomocí příkazu return v těle funkce následovaném návratovou hodnotou.

Parametrům se při definici funkce dají přiřadit implicitní hodnoty. Pokud je potom taková funkce zavolaná bez předání argumentů - hodnot parametrů, je provedena za použití těchto implicitních hodnot.

Obrázek 18: Definice funkce v PHP (vlastní)

```
function nazev_funkce($argument_1, $argument_2, ..., $argument_n) {
// tělo funkce s příkazy
}

//funkce s implicitními hodnotami argumentů
function implic_argumenty($argument = "hodnota") {
// tělo funkce
}
```

Funkce se volají zapsáním jména funkce se seznamem argumentů v kulatých závorkách. Argumenty mohou být zadány přímým zápisem, předáním hodnoty proměnné nebo z vypočteného výrazu.

Obrázek 19: Volání funkce v PHP (vlastní)

```
nazev_funkce(1, $promenna, 3+1, "hello" . "world");
```

Argumenty se funkcím normálně předávají hodnotou, tj. pokud předáme funkci jako argument proměnnou, funkce bude pracovat pouze s hodnotou této proměnné a původní hodnota předávané proměnné zůstane nezměněna, pokud jí nebude přiřazena případná návratová hodnota funkce. Je ale možné argumenty předávat i odkazem. To se dělá tak, že se před předávanou proměnnou připíše ampersand (&). Funkci bude potom předán odkaz na místo v paměti, kde je uložena předávaná proměnná a jakékoliv změny hodnoty parametru v těle funkce budou odraženy v hodnotě původní proměnné[10].

3.2.4 Třídy a objekty

Jazyk PHP od verze PHP 5 umožňuje objektově orientované programování. Umožňuje vytváření abstraktních a finálních tříd a metod, dědičnost, interfacy, zapouzdření, na-

Obrázek 20: Předávání odkazem v PHP (vlastní)

```
function nazev_funkce($argument) {
$argument = "world";
}
$promenna = "hello";
nazev_funkce(&$promenna);
//po zavolání je v $promenna uloženo "world"
```

stavení viditelnosti proměnných a funkcí třídy (modifikátory přístupu) atd.

Předpisem pro objekt v PHP je třída. Třída je definována klíčovým slovem class a názvem třídy následovaným tělem třídy ve složených závorkách. Název třídy může být jakýkoliv kromě rezervovaných slov jazyka PHP. Tělo třídy obsahuje konstanty, proměnné a metody, které definují její účel a chování[11].

Obrázek 21: Definice třídy v PHP (vlastní)

```
class trida {
  public $promenna = "implicitni hodnota";

  public function zobraz_promennou() {
    echo $this->promenna;
  }
}
```

Pro přístup k proměnným a metodám třídy v rámci metod této třídy se používá pseudoproměnná $this, která obsahuje odkaz na volající objekt, tedy objekt této konkrétní třídy.

Třída může obsahovat konstruktor, speciální metodu, pomocí které se při vytváření objektu třídy dá např. přiřazovat předané parametry proměnným třídy. Stejně tak může být definován destruktor, který se volá, pokud již neexistují žádné odkazy na objekt

této třídy [11].

Obrázek 22: Konstruktor a destruktor třídy v PHP (vlastní)

```
class trida {
  function __construct() {
   echo "Konstruktor";
  }

  function __destruct() {
    echo "Destruktor";
  }
}
```

Objekt třídy - instance - se vytváří pomocí klíčového slova new následovaného názvem třídy a případným seznamem parametrů v kulatých závorkách. Pokud konstruktor třídy nevyhazuje výjimku, je vždy vrácen objekt požadované třídy. Místo názvu požadované třídy je možné použít i řetězec s tímto názvem, včetně řetězce uloženého v proměnné [11].

Obrázek 23: Vytváření objektu třídy v PHP (vlastní)

```
$instance = new trida();

//následující volání je funkčně shodné s předchozím
$promenna = "trida"
$instance = new $promenna();
```

Konstanty, proměnné a metody třídy mohou mít definovány tzv. modifikátory přístupu, které určují, odkud jsou tyto viditelné, odkud je k nim možno přistoupit. Definují se předsazením definice konstruktu klíčovým slovem public, private nebo protected [11].

Tabulka 5: Modifikátory přístupu v PHP

public	konstanta, proměnná nebo metoda je viditelná všem, přístup k ní není omezen
protected	přístup pouze z instancí dané třídy nebo tříd od ní odvozených
private	přístupné pouze z instancí dané třídy

Dědičnost se v PHP zajišťuje klíčovým slovem extends v hlavičce třídy následovaným názvem třídy, od které nová třída dědí. Potomek od rodičovské třídy přebírá veškeré konstanty, proměnné a metody v rodičovské třídě definované. Pokud potomek stávající metody nepřetíží, zachovávají si tyto metody svou původní funkcionalitu. Přetížením se rozumí definování nového těla metody, případně s novou definicí jejích parametrů. Přetížení metody se dá zabránit použitím klíčového slova final v hlavičce metody u rodičovské třídy[11].

Obrázek 24: Dědičnost v PHP (vlastní)

```
class trida {
  // tělo třídy
  public function fce() {
    echo("foo");
  }
}

class potomek extends trida {
  // tělo nové třídy
}

$instance = new potomek();
potomek->fce(); //vypíše foo
```

Zvláštním případem třídy je abstraktní třída, definovaná klíčovým slovem abstract

v hlavičce třídy. Je to taková třída, která obsahuje alespoň jednu abstraktní metodu, nedají se z ní vytvářet instance. Abstraktní metody, uvozené opět klíčovým slovem abstract, mohou být pouze deklarovány, nemůžou být ale definovány. Definice těchto metod je ponechána na třídách, které z těchto abstraktních tříd dědí.

Interface je struktura podobná třídě, která pouze předepisuje metody, které musí třída implementující tento interface definovat. Kromě deklarací metod může obsahovat i konstanty. V PHP se interface definuje klíčovým slovem interface v hlavičce tohoto interfacu. Třída implementující daný interface potom musí mít v hlavičce klíčové slovo implements následované názvem interfacu[11].

Obrázek 25: Interfacy v PHP (vlastní)

```
interface iPredpis {
  public function foo();
}

class trida implements iPredpis {
  public function foo() {
   echo("bar");
  }
}
```

Objekty se mezi sebou dají porovnávat. Používají se pro to operátory == - rovnost a === - identita. Při porovnání operátorem rovnosti se pouze porovnává shoda hodnot parametrů a zda se jedná o objekty stejné třídy, při porovnání identitním operátorem se zjišťuje, zda se jedná o tentýž objekt dané třídy[11].

Obrázek 26: Porovnávání objektů v PHP (vlastní)

```
$foo = new trida("foobar");
$bar = new trida("foobar");

$c = ($foo == $bar); // true
$c = ($foo === $bar); // false
$foo = $bar;
$c = ($foo === $bar); // true
```

3.2.5 Propojení s MySQL DBMS

Pro propojení PHP aplikace s MySQL databází slouží v základu dvě rozhraní, PDO - PHP Data Objects, umožňuje propojení i s jinými typy databází, než je MySQL (Oracle SQL, MS SQL aj.) a mysqli[11]. Tato sekce je zaměřena na mysqli.

Rozhraní mysqli slouží pro propojení s MySQL databází, neumí se však připojit k jiným typům SQL DBMS. Umožňuje jak procedurální, tak objektový přístup, je možné je i kombinovat, avšak z důvodu přehlednosti to není doporučováno. Procedurální funkce se až na výjimky jmenují stejně jako objektové metody, pouze s tím rozdílem, že jsou jejich jména předsazena slovem *mysqli_*. Jména funkcí budou dále uváděna bez této předpony, pokud se nebudou v procedurálním přístupu lišit.

K MySQL databázi se lze připojit buď vytvořením instance třídy mysqli nebo voláním funkce mysqli_connect. Jak funkce, tak konstruktor třídy jako parametry přijímají umístění databáze v síti, uživatelské jméno a heslo a název databáze, nad kterou se bude dále dotazovat. Konstruktor třídy mysqli vrací instanci této třídy, funkce mysqli_connect vrací identifikátor zdroje, oba jednoznačně identifikují nově vytvořené připojení k databázi[13].

Vybranou databázi je možné kdykoliv změnit funkcí select_db na databázovém objektu.

Obrázek 27: Připojení k databázi s mysqli (vlastní)

```
// objektově
$db = new mysqli($umisteni, $uzivatel, $heslo, $databaze);
// procedurálně
$db = mysqli_connect($umisteni, $uzivatel, $heslo, $databaze);
// změna databáze
// objektově
$db->select_db($databaze);
// proceurálně
mysqli_select_db($db, $databaze);
```

Nad databázovým objektem/zdrojem se dají provádět dotazy. K tomto účelu slouží funkce query. Její návratovou hodnotou je objekt/zdroj s výsledkem dotazu (result), případně booleovská hodnota *false* při selhání[13].

Obrázek 28: Dotaz nad databází v mysqli (vlastní)

```
// objektově
$result = $db->query($dotaz);
// procedurálně
$result = mysqli_query($db, $dotaz);
```

Data z výsledku dotazu lze získat v podobě objektu, asociativního pole nebo enumerovaného pole. Lze také získat počet výsledků dotazu. Počet výsledků dotazu se získá funkcí num_results. To platí pro dotazy typu SELECT, pro dotazy typu INSERT a UPDATE lze zjistit počet ovlivněných řádek pomocí funkce affected_rows. Pro získání jednoho záznamu z výsledku v objektové podobě slouží funkce fetch_object, v podobě asociativního pole funkce fetch_assoc a v podobě enumerovaného pole funkce fetch_row.

Objektová podoba záznamu obsahuje sloupce relace jako atributy objektu, v asociativní podobě jsou názvy sloupců relací použity jako klíče a v enumerovaném poli mají

jednotlivé hodnoty sloupců přiřazeny číselné klíče podle pořadí v jakém byly vybrány v dotazu.

Výsledek dotazu obsahuje též ukazatel na aktuální záznam. Pokaždé, když se z výsledku přečte záznam, se tento ukazatel posune na další, dokud jsou k dispozici další záznamy. Toho lze využít např. pro iteraci nad výsledkem dotazu[13].

Obrázek 29: Iterace nad výsledkem dotazu (vlastní)

```
while($row = $result->fetch_object()) {
// proveď něco s výslednými daty, např. ulož do proměnné
$promenna1 = $row->sloupec1;
$promenna2 = $row->sloupec2;
// aj.
}
```

Po skončení práce s databází je zvykem toto spojení zavřít. To se provádí pomocí funkce close. Uzavřít spojení není vždy nutné, jelikož se uzavře při dokončení provádění skriptu[13].

Obrázek 30: Uzavření spojení s databází (vlastní)

```
// objektově
$db->close();
// procedurálně
mysqli_close($db);
```

Rozhraní mysqli umožňuje také využití předpřipravených dotazů (prepared statements). Ty lze využít pro zrychlení při provádění většího množství stejného dotazu s různými daty a jako ochrana proti SQL injection útokům.

Předpřipravený dotaz je ve své podstatě předloha dotazu, který chceme vykonat. Ta se odešle databázovému systému, který jí předkompiluje a připraví se na příjem dat, která se odesílají odděleně od samotného dotazu.

Pro vytvoření předpřipraveného dotazu slouží metoda prepare nebo funkce mysqli_stmt_prepare. Parametry se poté svazují s dotazem metodou bind_param nebo funkcí mysqli_stmt_bind_param. Ty kromě samotných parametrů potřebují ještě řetězec, který říká, jakého datového typu jsou posílaná data[13].

Tabulka 6: Datové typy parametrů předpřipraveného dotazu[13]

i	celé číslo
d	číslo s plovoucí desetinnou čárkou, double
s	textový řetězec
b	binary large object, blob

Předpřipravené dotazy se spouští metodou execute nebo funkcí mysqli_stmt_execute. Výsledky dotazu lze svázat s proměnnými podobně, jako se to dělá s parametry. K tomu slouží metoda bind_result nebo funkce mysqli_stmt_bind_result. Data se poté do těchto proměnných zapíší metodou fetch nebo funkcí mysqli_stmt_fetch. Každé volání této metody/funkce vrací data z dalšího řádku výsledné relace, dokud existuje další řádek[13].

Obrázek 31: Příklad předpřipravených dotazů v mysqli (vlastní)

```
// předpokládáme existující spojení s databází $db
$rok = 1999;
$mesto = "Frydek";
$jmeno = "";
$prijmeni = "";
$dotaz = "SELECT jmeno, prijmeni FROM osoba WHERE roknar = ? AND mesto LIKE ?";
$stmt = $db->prepare($dotaz);
$stmt->bind_param("is",$rok, $mesto);
$stmt->bind_result($jmeno, $prijmeni);
$result = $stmt->execute();
while($stmt->fetch()) {
  echo $jmeno . " " . $prijmeni;
}
$stmt->close();
```

3.2.6 Metody zabezpečení

Nejzranitelnější částí jakékoliv webové stránky nebo aplikace je ta, kde se manipuluje s daty nebo je očekáván vstup dat od uživatele. Při vytváření webové aplikace je tedy nezbytně nutné se zamyslet, kde mají potenciální útočníci možný přístup k datům a kde mohou napáchat škodu a co nejlépe se pokusit jim v přístupu zamezit.

V případě vstupu dat je vhodné jakákoliv vstupní data považovat za škodlivá a tak se k nim chovat. Pokud tomu takto není, mohou útočníci použít metod, jako je XSS a SQL injection (popsány dále), k získání dat. Jakákoliv vstupní data získaná PHP skriptem ze superglobálních polí *$_GET* a *$_POST* je vhodné validovat, zjistit zda odpovídají našim vstupním parametrům, a uložit do nově inicializované proměnné. Není považováno za bezpečné pracovat přímo s těmito superglobáními poli.

S daty je poté dále možno manipulovat, aby odpovídala našim požadavkům, například

funkcí *ucfirst* pro vynucení velkého písmene na začátku jména, vynucením, aby vstupní pole obsahovalo hodnotu atd., pro dosažení datové integrity.

Další nebezpečí vyvstává při odesílání dat skriptem, buď webovému prohlížeči jako výstup nebo jako záznam do databáze. Může se stát, že útočník do vstupního formuláře zadá HTML značku. Ta bude následně skriptem vyhodnocena jako validní ale při zobrazování dat na stránce se vykreslí jako prvek touto značkou definovaný. Může to být něco relativně neškodného, jako zobrazení tučného textu, ale daleko pravděpodobněji se v případě útoku bude jednat o značku script a v ní definovaný kód v JavaScriptu, který se po vykreslení spustí.

Je proto vhodné veškerý text před jeho odesláním tzv. escapeovat. Ve značkovacím jazyce HTML jsou definovány tzv. escape sekvence, řetězce ve formátu *&KÓD;*, které nahrazují potenciálně nebezpečné znaky, jako např. špičaté závorky, které se dají použít pro vymezení HTML značky, dvojité a jednoduché uvozovky nebo ampersand. Voláním funkce *htmlspecialchars* v PHP jsou tyto znaky převedeny na jim odpovídající escape sekvence. Obdobně existuje funkce *mysql_real_escape_string*, která escapeuje textový řetězec tak, aby se dal použít v SQL dotazu bez toho, aby tento vyústil v chybu, například když ukládaný řetězec obsahuje apostrof, kterým se v určitých implementacích SQL mohou ohraničovat textové řetězce.

Hesla uživatelů je nutné ukládat v takové podobě, která není přímo čitelná případným útočníkům, neměla by se tedy ukládat v neupravené textové podobě. Za tímto účelem je vhodné hesla transformovat hashovacím algoritmem. Tyto algoritmy fungují jako jednocestné šifry, generují na základě vstupu hexadecimální řetězec o fixní délce - hash, který se však nedá zpět rozšifrovat bez použití hrubé síly, tj. zkoušení všech možných kombinací vstupních řetězců a ověřování, zda výstup odpovídá existujícímu hashi[10]. Nemusí se vždy však postupovat jen hrubou silou, jelikož dnes existují tzv. rainbow tables, které obsahují mnoho kombinací vstupních řetězců a jejich hashů (včetně obligátních tvarů jako „password“, „heslo“, „heslo123“ aj.)[14]. Z toho důvodu se doporučuje hesla kromě hashování také tzv. solit, tedy v rámci kódu přidávat k těmto heslům pevně dané řetězce. Tyto řetězce mohou být také samy hashované. Tímto se dále zvyšuje bezpečnost hesla[10].

Obrázek 32: Hashování a solení hesla (vlastní)

```
Kód:
$heslo = "heslo123";
$sul = "Tohle je naprosto originální textový řetězec";

echo 'Hash původního hesla: ' . hash("sha256", $heslo) . "<br>";
$sul = hash("sha256", $sul);

$solene_heslo = $heslo . $sul;
echo 'Solené heslo před hashováním: ' . $solene_heslo . "<br>";
echo 'Po hashování: ' . hash("sha256", $solene_heslo);

Výstup (zkrácen):
Hash původního hesla: b4240791ce0be7b92ddd5091ae67234e5dff778e786dc0...
Solené heslo před hashováním: heslo12348f1268a928fdfeb70a48e6714120e...
Po hashování: 700c01d5d2f17cb6bb2a679cec19e5560e75f8cbd6e027b7996c93...
```

Cross-site scripting (XSS) a **SQL injection** jsou metody útoku, při kterých dochází k vykonání kódu vloženého útočníkem do vstupního pole formuláře. Při XSS se do stránky vkládá kód v jazyce JavaScript. Tento kód může dále manipulovat s prvky na stránce, přesměrovávat na stránku útočníka atp. Proti XSS se dá bránit escapeováním vstupů. SQL injection využívá chybového mechanismu jazyka SQL tak, že předčasně ukončí prováděný dotaz a vykoná dotaz vlastní. Aby toto fungovalo je však nutné znát názvy tabulek a sloupců v databázi. Proti SQL injection se dá bránit filtrováním vstupu - zákaz středníků, omezením práv účtu, kterým v kódu přistupujeme k databází nebo použitím předpřipravených dotazů[10].

3.3 Web content management

Web Content Management (dále WCM) je způsob správy obsahu na webu - vytváření, ukládání, vyvolání, mazání atd. Koncept správy obsahu je starší než web, jedním z nejstarších způsobů správy obsahu jsou knihovny - ukládají obsah v podobě tištěného slova a spravují přístup k tomuto obsahu[5].

3.3.1 Co je to obsah

Obsah jsou data vytvářená člověkem v rámci redakčního procesu a určená pro konzumaci jinými lidmi. Redakční proces zahrnuje všechny činnosti prováděné člověkem za účelem zpracování informace pro její publikaci a komunikaci široké veřejnosti či úzkým skupinám. Patří do něj sběr informací, samotný tvůrčí proces, editace, recenze, schvalování, verzování, srovnávání atd.

Je to proces silně subjektivní, pokud například zadáte dvěma osobám zpracovat článek na stejné téma a poskytnete jim přístup ke stejným informacím, oba články budou s velkou pravděpodobností i tak odlišné díky vlivu lidského faktoru - obecné informovanosti člověka o tématu, schopnosti pochopit dostupné informace, ale i osobním přesvědčením.

Hlavními vodítky při samotné tvorbě obsahu jsou názory tvůrců na to, co by mělo být předmětem obsahu, komu má být obsah určen, z jakého úhlu téma uchopit, jak dlouhý by měl být obsah a zda potřebuje oporu v multimediální formě (obrázky, grafy, videa aj.)[5].

3.3.2 Systémy pro správu obsahu

Systémy pro správu obsahu (content management systems, CMS) jsou softwarové aplikace umožňující do určité míry automatizovat úkoly pro spravování obsahu. Jedná se zpravidla o víceuživatelské systémy na straně serveru, které komunikují s nějakým uložištěm obsahu, například databázovým systémem. Umožňují svým uživatelům vytvářet nový obsah, upravovat obsah existující atd. za konečným účelem poskytnutí tohoto ob-

sahu veřejnosti.

CMS se dají rozdělit podle svého účelu na čtyři skupiny, ačkoliv hranice mezi těmito skupinami mezi sebou často prolínají[5]:

- Web Content Management

  Tyto systémy slouží převážně pro komunikaci obsahu veřejnosti skrz webové rozhraní. Vynikají v oddělování obsahu od způsobu jeho prezentace, některé umožňují publikovat obsah skrz více webových kanálů.

- Enterprise Content Management

  Používají se pro správu obsahu, který nemusí být vždy určen široké veřejnosti (např. životopisy zaměstnanců, interní zprávy aj.), mají zpravidla velmi dobrou správu přístupu, souborů, velmi dobře umožňují spolupráci.

- Digital Asset Management

  Systémy pro správu multimediálního obsahu - obrázků, videa, audia. Zpravidla dobře umí spravovat metadata.

- Records management

  Slouží pro správu informací o transakcích a dalších záznamech vyprodukovaných v rámci podnikání.

3.4 UML

UML (Unified Modeling Language) je nástroj pro modelování struktury a chování softwarových aplikací nezávisle na platformě a implementaci. Za tímto účelem se sestává ze třinácti typů diagramů ve třech skupinách[15]:

- Diagramy strktury

  reprezentují statickou strukturu aplikace

  class diagram, object diagram, component diagram, composite structure diagram, package diagram, deployment diagram

- Diagramy chování

  reprezentují chování aplikace v obecné podobě

  use case diagram, activity diagram, state machine diagram

- Interakční diagramy

  rozšiřují diagramy chování, reprezentují různé aspekty interakce

  sequence diagram, communication diagram, timing diagram, interaction overview diagram

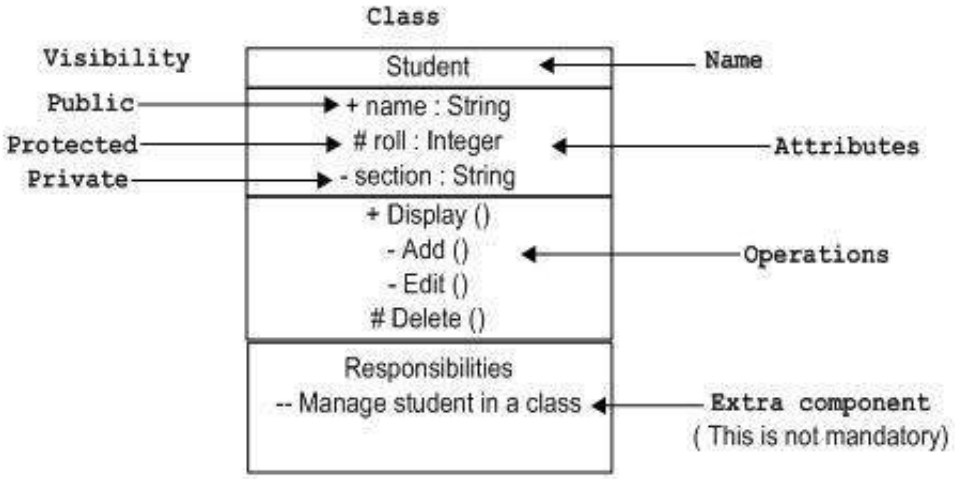
3.4.1 Diagramy struktury

V této sekci budou popsány některé strukturní diagramy, konkrétně diagram tříd a diagram objektů.

Diagram tříd (class diagram) slouží k popisu statické struktury aplikace. Popisuje atributy a operace tříd a omezení uvalená na systém. Často se používá pro popis systémů vyvíjených objektově orientovanou metodologií, jelikož je to jediný typ UML diagramu, který přímo odpovídá vlastnostem objektově orientovaných programovacích jazyků.

Diagram tříd slouží tedy ke statickému návrhu aplikace, popisuje vlastnosti systému a usnadňuje dokumentaci pro další rozšiřování nebo opravu chyb[16].

Obrázek 33: Zápis třídy v UML[21]



Třída je v UML reprezentována obdélníkem rozděleným na tři, eventuálně čtyři sekce. V horní sekci je zapsán název třídy. Druhá sekce obsahuje atributy třídy, jejich datové typy a modifikátory přístupu. V třetí sekci jsou zapsány operace prováděné touto třídou. Čtvrtá sekce je nepovinná, obsahuje jakékoliv doplňující komponenty a informace[16].

Při sestavování diagramu tříd by měly být dodrženy následující body:

- název diagramu by se měl vztahovat k části systému, kterou popisuje
- prvky a vazby mezi nimi by měly být identifikovány před sestavením diagramu
- měly by být jasně definovány atributy a operace všech tříd
- třídy by měly mít pouze vlastnosti nezbytně nutné pro svou funkčnost

Násobnost (kardinalita, multiplicita) vztahu vyjadřuje, kolik objektů konkrétní třídy se účastní toho konkrétního vztahu (1, n atd.), zapisuje se na spojovací šipku mezi třídami tak, že se násobnost dané třídy ve vztahu zapíše na stranu této třídy.

Zobecnění (dědičnost, generalizace) se zapisuje čarou s nevyplněnou šipkou na konci směřující od potomka k rodičovské třídě[16].

Obrázek 34: Class diagram[17]

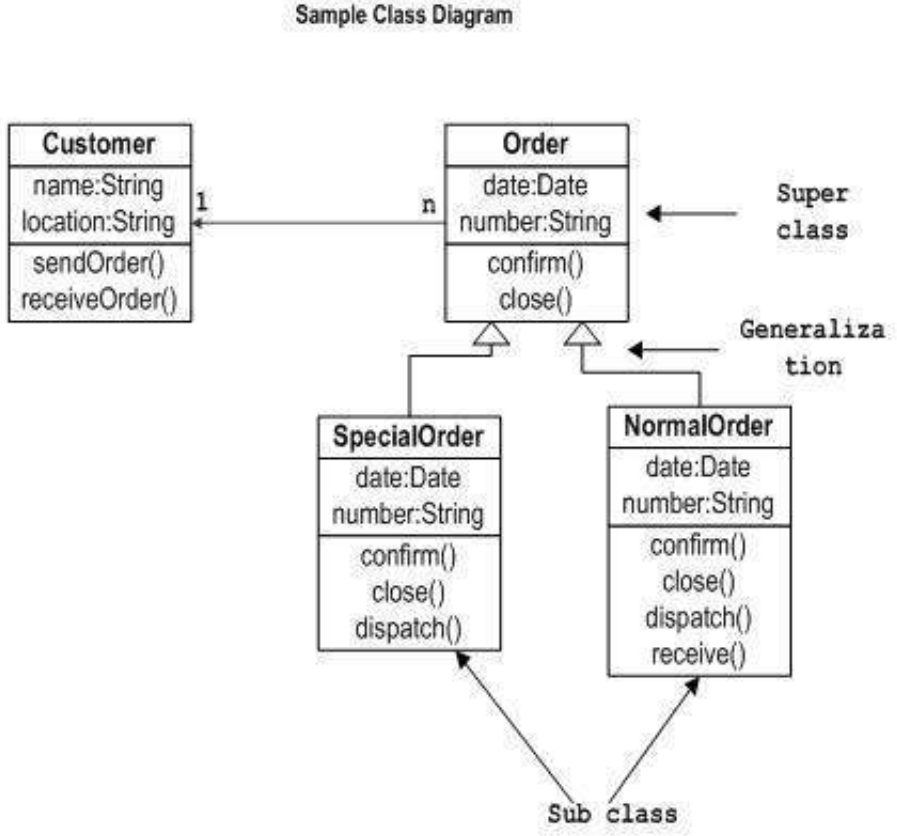
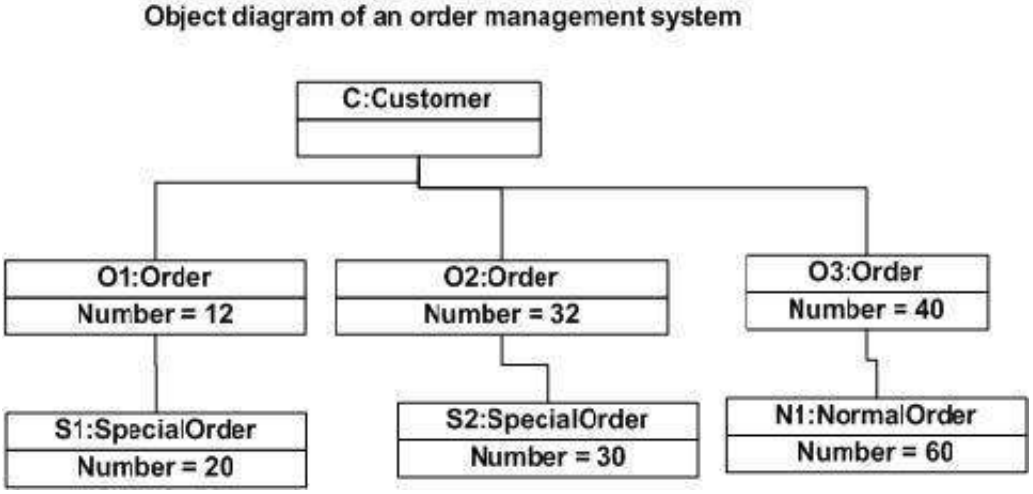


Diagram objektů (object diagram) vychází z diagramu tříd, reprezentují statický pohled na systém v určitém časovém okamžiku. Zobrazují množinu objektů (instancí tříd) a vztahy mezi nimi. Pomáhají pochopit vztahy a chování objektů z praktického pohledu, poskytují statický pohled na interakci.

Na rozdíl od diagramu tříd, který poskytuje pouze jeden pohled na systém, jich může diagram objektů poskytovat téměř nekonečně mnoho, jelikož je možné z každé třídy vytvořit téměř nekonečně mnoho od sebe odlišných instancí. Je proto třeba se před kreslením diagramu rozhodnout, které instance a vazby jsou důležité a zahrnout pouze ty instance, které v dostatečné míře pokryjí funkcionalitu systému. Dále by se měly dodržet následující body[16]:

- název diagramu by měl smysluplně odpovídat jeho účelu
- musí být identifikovány nejdůležitější atributy
- musí být znázorněny vazby mezi objekty
- atributy různých hodnot by měly být zaznamenány v diagramu

Obrázek 35: Object diagram[18]



3.4.2 Diagramy chování

V této sekci budou popsány některé diagramy chování, konkrétně diagram případů užití, diagram aktivit a diagram stavů.

Diagram případů užití (use case diagram) slouží k zachycení dynamického aspektu systému, konkrétně k zachycení požadavků na systém včetně vnějších a vnitřních vlivů. Využívá se při analýze funkcionalit systému, zachycuje případy užití a aktéry se systémem interagující.

Případy užití jsou jednoznačně identifikované izolované funkce systému. Aktoři jsou prvky, které se systémem přicházejí do styku. Může se jednat o lidi - uživatele systému, vnitřní aplikace systému, ale i aplikace přistupující k systému z vnějšku.

Při plánování diagramu případů užití je nutné identifikovat všechny klíčové funkce systému, které mají být reprezentovány jako případy užití, aktéry a vztahy mezi aktéry a případy užití, případně mezi dvěma případy užití.

Při kreslení by se měly dodržet následující body[16]:

- název diagramu by měl vystihovat jeho účel
- aktoři by měli být vhodně pojmenování
- vztahy a závislosti by měly být jasně vyznačeny
- není vhodné zahrnout všechny možné vztahy mezi prvky, smyslem diagramu je zachytit klíčové požadavky

Obrázek 36: Use case diagram[19]

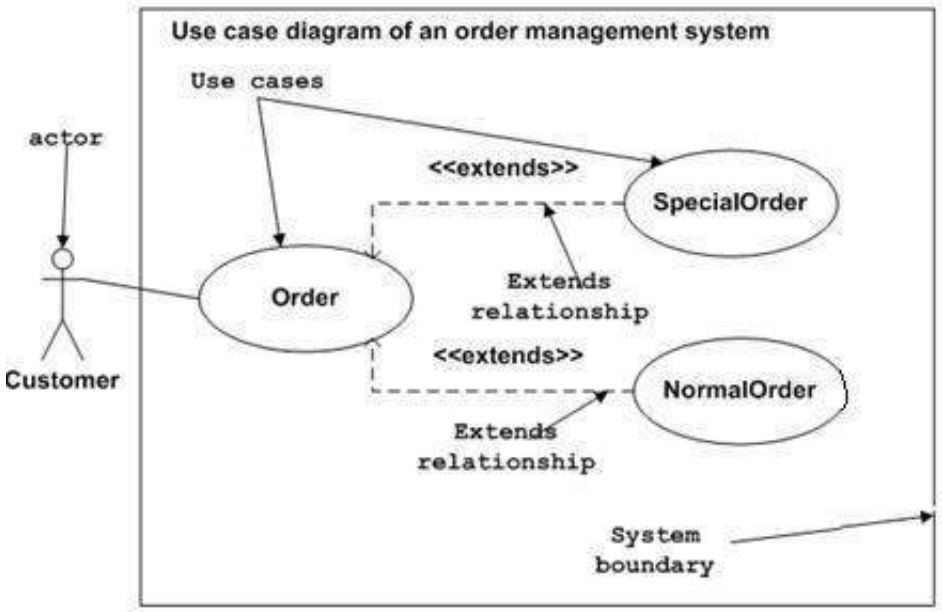


Figure: Sample Use Case diagram

Diagram aktivit (activity diagram) zobrazuje, jak za sebou postupují jednotlivé aktivity v systému. Aktivitou se rozumí jedna ucelená operace systému. Diagram aktivit zachycuje pouze posloupnost aktivit, v tomto ohledu připomíná klasické vývojové diagramy, neřadí se však mezi ně. Dá se pomocí něj zakreslit jak jednoduchý, tak i větvený nebo paralelní běh. Diagram aktivit není přímo rovný koncovému kódu.

Pro kreslení diagramu aktivit je nejprve nutné identifikovat aktivity nutné pro provedení funkcionality, logickou posloupnost aktivit, kontrolní podmínky a omezení[16].

Obrázek 37: Activity diagram[20]

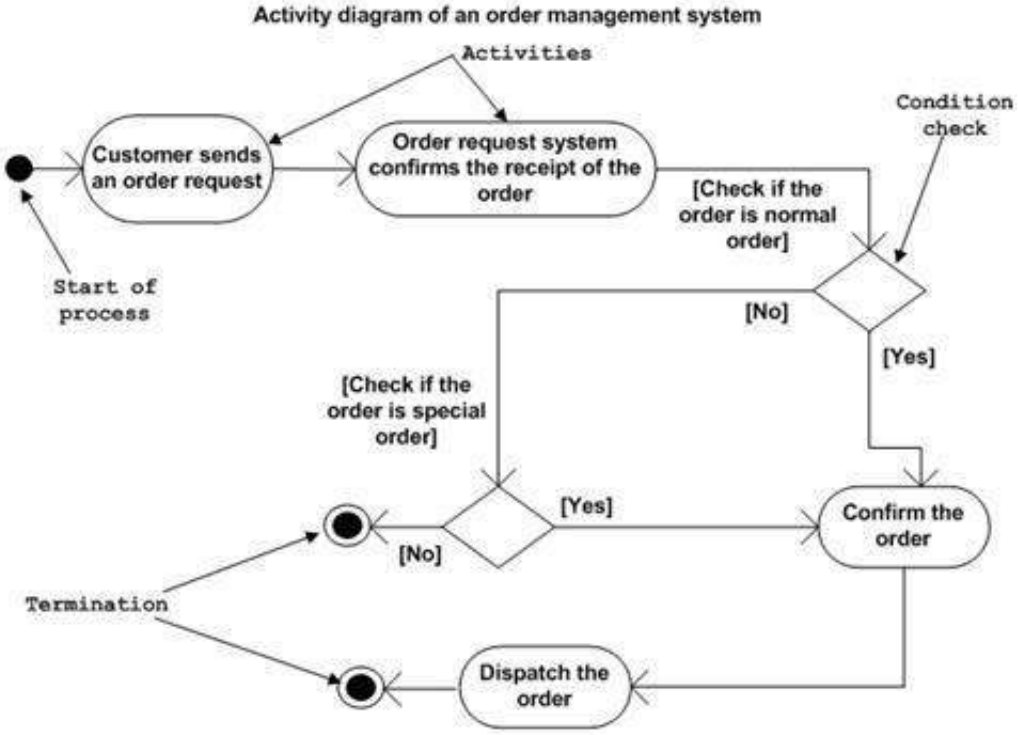
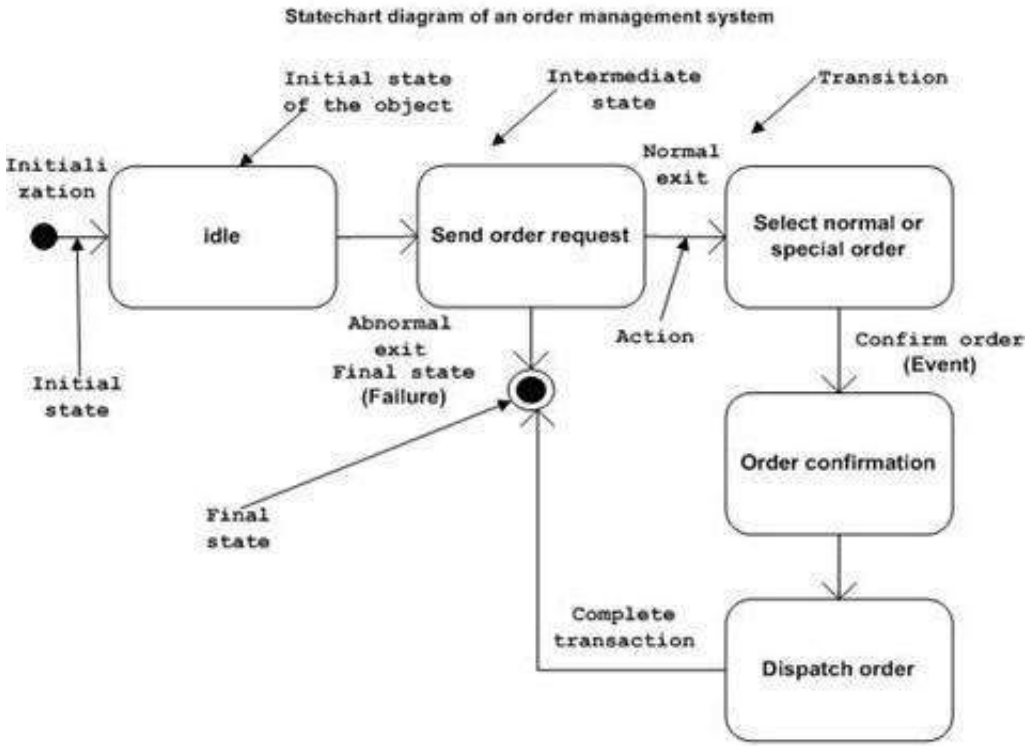


Diagram stavů (statechart diagram) se používá pro znázornění stavů, kterých může objekt nabýt během své existence. Znázorňuje hlavně přechody mezi těmito stavy na základě vnitřních či vnějších událostí. Stav objektu je potom reakcí na tuto událost.

Před kreslením diagramu stavů je nutné identifikovat objekt, který analyzujeme, stavy, kterých může nabýt a události, které mohou nastat.

Obrázek 38: Statechart diagram[22]

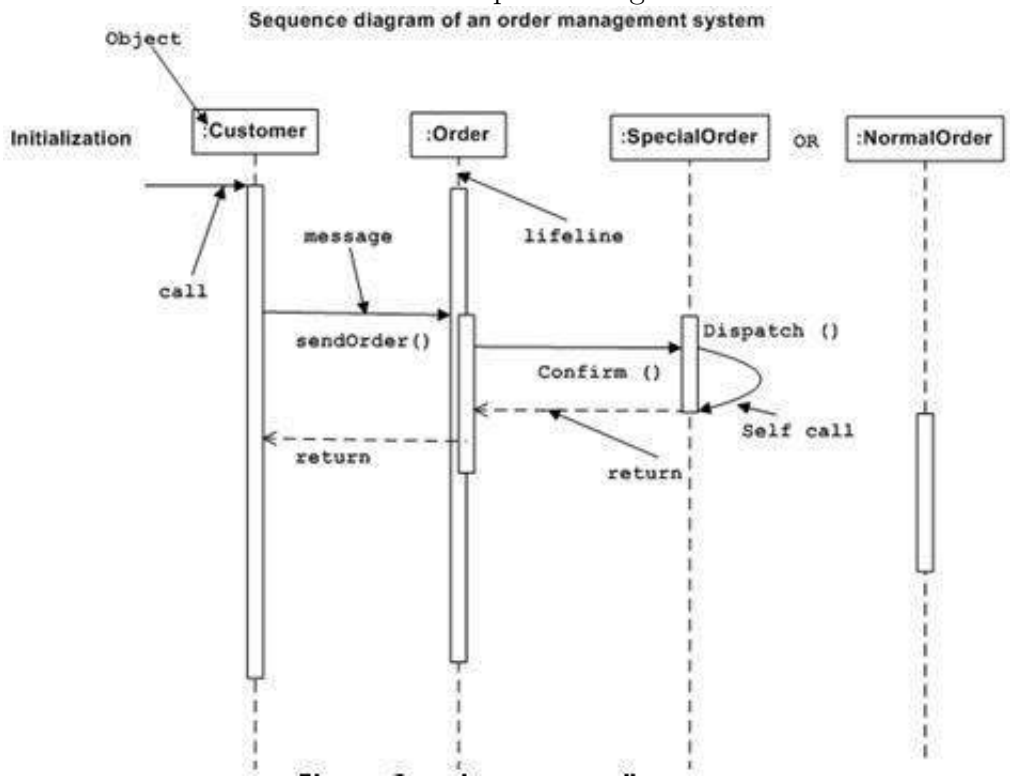


3.4.3 Interakční diagramy

V této sekci budou popsány některé interakční diagramy, konkrétně sekvenční diagram a diagram spolupráce.

Tyto dva typy diagramů jsou vzájemně téměř izomorfní, lze převádět z jednoho typu diagramu do druhého. Výjimka nastává v UML 2.0, kde do sekvenčních diagramů byly přidány strukturované mechanismy umožňující mimo jiné skládání sekvenčních diagramů z dalších sekvenčních diagramů. Sekvenční diagramy využívající tyto mechanismy nelze převést[25].

Obrázek 39: Sequence diagram[23]



Sekvenční diagram (sequence diagram) zachycuje objekty a zprávy zasílané mezi nimi a časovou posloupnost těchto zpráv. Nezachycuje však vztahy mezi objekty. Je tedy vhodný tehdy, kdy je důležité vidět časové souvislosti interakcí.

V jednoduchém sekvenčním diagramu jsou na vodorovné ose zaneseny účastníci,

časová osa je svislá. Účastníci (participanti, většinou objekty) jsou reprezentovány čarou života (lifeline), která také ukazuje kdy tento účastník *žije*, tedy kdy byly v rámci interakce vytvořeny a kdy zanikají (v obrázku 32 - objekty existovaly již před posláním první zprávy, nevíme kdy zaniknou nebo nás to nezajímá). Aktivita objektu (focus of control, execution of specification), znázorněný zdvojenou čarou či obdélníkem na čáře života, vyjadřuje dobu, po kterou je objekt aktivní a provádí nějakou operaci, včetně čekání na návratovou zprávu.

Zprávy se vyznačují plnou čarou s plnou šipkou. Návratové zprávy není povinné vyznačovat, je však vhodné je doplnit pro přehlednost, zakreslují se přerušovanou čarou s jednoduchou šipkou. Návratové zprávy mohou vracet hodnotu[25].

Obrázek 40: Collaboration diagram[24]

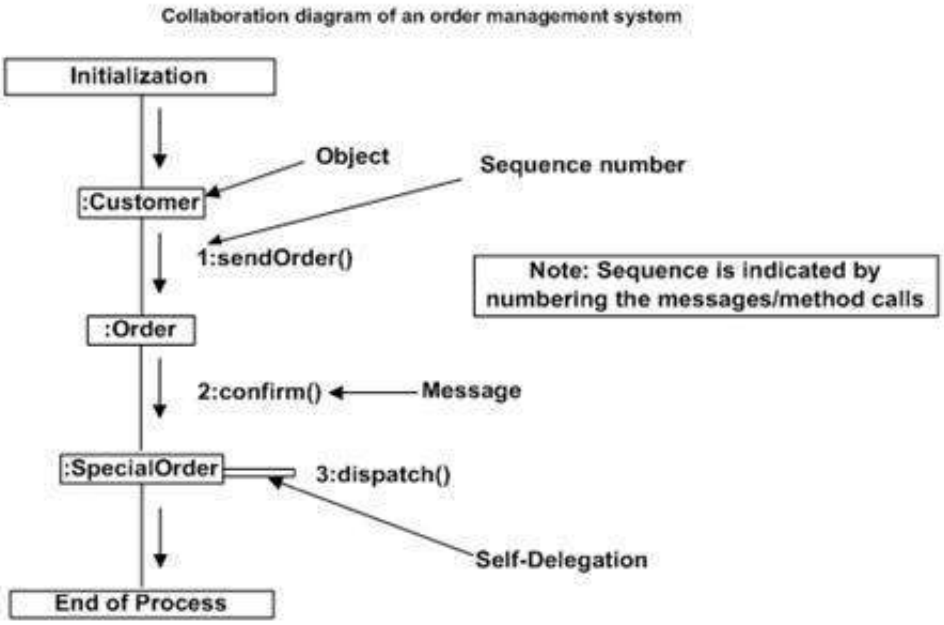


Diagram spolupráce (collaboration diagram), ve specifikaci UML 2.0 přejmenován na diagram komunikace (communication diagram), zobrazuje objekty a spojení a zprávy, které si mezi sebou posílají. Není zde zachycena dimenze času, posloupnost zpráv a souběžnost vláken musí být určena sekvenčním číslováním. Je vhodný tehdy, chceme-li zachytit strukturu spolupráce, kdo s kým komunikuje, spíše než časové souvislosti.

Uzly v diagramu reprezentují role ve spolupráci (*objekty*), odpovídají čarám života v sekvenčním diagramu. Čáry mezi objekty vyjadřují komunikační cesty. Můžou být pojmenovány a mohou na nich být vyjádřeny multiplicity. Zprávy jsou zakresleny formou malých šipek blízko spojovacích čar. Tyto šipky jsou pojmenovány podle zprávy, která se zasílá. Před názvem zprávy je zapsán sekvenční výraz. Ten určuje pořadí, ve kterém jsou zprávy zasílány. U zprávy může být zapsána návratová hodnota, její zapsání však není povinné.

3.5 WebML

WebML (Web Modeling Language) je modelovací jazyk pro modelování webových aplikací. Umožňuje vyjádřit formou diagramů strukturu webové aplikace, která je potom použitelná pro implementaci, ověření správnosti, údržbu a budoucí rozšiřování. Poskytuje různé pohledy na specifikaci webové aplikace, umožňuje při návrhu separovat zobrazované informace od formy a kompozice jejich zobrazení, specifikovat operace pro manipulaci s daty za účelem aktualizace obsahu stránek nebo pro interakci s externími službami internetu, navrhovat verze aplikace přizpůsobené různým skupinám uživatelů atd.

WebML definuje tři základní modely - datový, hypertextový a prezentační model. Tyto modely jsou dostačující pro návrh *read-only* aplikací - takových aplikací, kde je možné z webového rozhraní pouze číst informace, ale není možné je měnit. Přidáním operací k hypertextovému modelu je možné navrhnout WCM funkcionalitu a integraci s externími službami[27].

3.5.1 Datový model

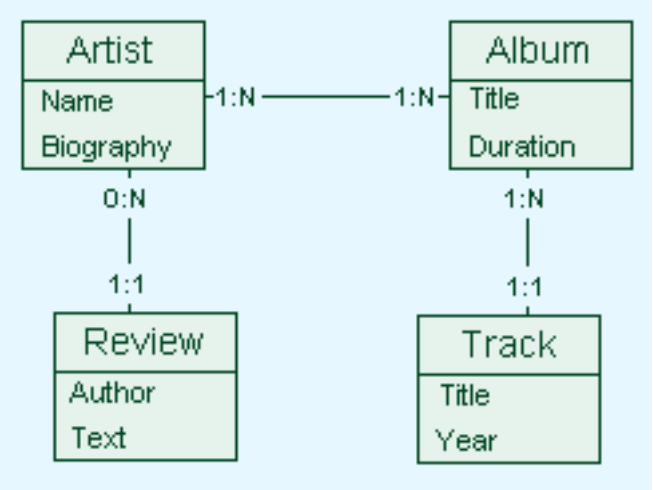
Datový model WebML adaptuje existující modely pro návrh datových struktur. Je kompatibilní s E-R diagramy používanými pro návrh databází a s UML Class diagramy.

Základní jednotkou v datovém modelu je **entita**. Entita je množina společných vlastností identifikujících množinu objektů. Množina objektů popsaná touto entitou se nazývá populace, jednotlivé objekty jsou instance této entity. Entita se zakresluje

jako obdélník rozdělený na dvě části - horní část obsahuje název entity, dolní část vlastnosti popisující tuto entitu. Každá instance entity je v datovém modelu WebML adresovatelná unikátním identifikátorem (**OID**, object identifier), tento identifikátor je čistě abstraktní, v modelu není vyjádřen a jeho implementace závisí na způsobu uložení dat, např. jako primární klíč v relační databázi.

Mezi entitami jsou definovány **vztahy**. Tyto vztahy reprezentují sémantické vazby mezi entitamy, sémantický význam, účel těchto vazeb je určen názvem vztahu (např. vazba mezi entitami učitel a předmět se může jmenovat [učitel]—učí—[předmět]). Základní formu vztahu je vztah binární, tedy vztah mezi dvěma entitami. Specifikace WebML dovoluje i vztahy mezi třemi a více entitami, nejsou však doporučovány a většině situací se dají nahradit binárními vztahy. Kromě účelu vztahů lze definovat i jejich kardinalitu, tedy násobnost entit ve vztahu.

Obrázek 41: Datový model WebML[28]



3.5.2 Hypertextový model

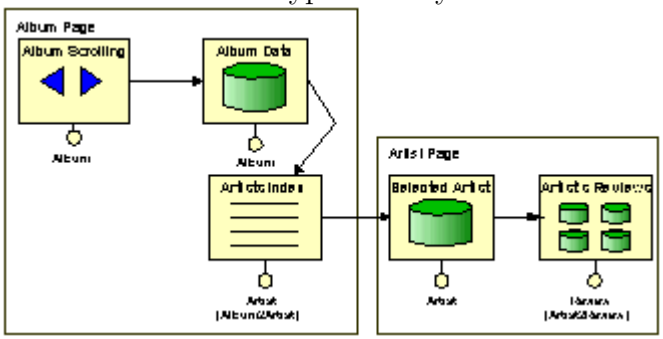
Hypertextový model WebML slouží pro specifikaci kompozice a navigace webové aplikace.

Kompozice specifikuje, z jakých **stránek** se skládá hypertext a z kterých **jednotek obsahu** se skládá stránka. Stránky jsou jednotky informace doručené uživateli/čtenáři.

Jednotky obsahu jsou dále nedělitelné prvky používané pro zobrazení informace popsané datovým modelem.

Navigace hypertextem je daná **odkazy**. Odkazy mohou být definovány mezi jednotkami na jedné stránce, mezi jednotkami na různých stránkách a mezi stránkami jako takovými. Informace nesená odkazem se označuje jako **kontext** (nebo taky navigační kontext). Odkazům, které nesou kontext, se říká kontextové odkazy, těm, které žádný kontext nenesou, se říká bezkontextové odkazy. Kontextová informace je často nezbytná pro vypočtení jednotek obsahu[27].

Obrázek 42: Hypertextový model[29]



Oblasti a pohledy jsou způsoby organizace komplexních webových aplikací do hierarchické struktury.

Pohled (siteview) je největší jednotka modularity, popisuje souvislý hypertext určený pro specifickou skupinu uživatelů. Pohledy mohou být veřejné nebo soukromé, soukromé pohledy vyžadují oprávnění pro přístup.

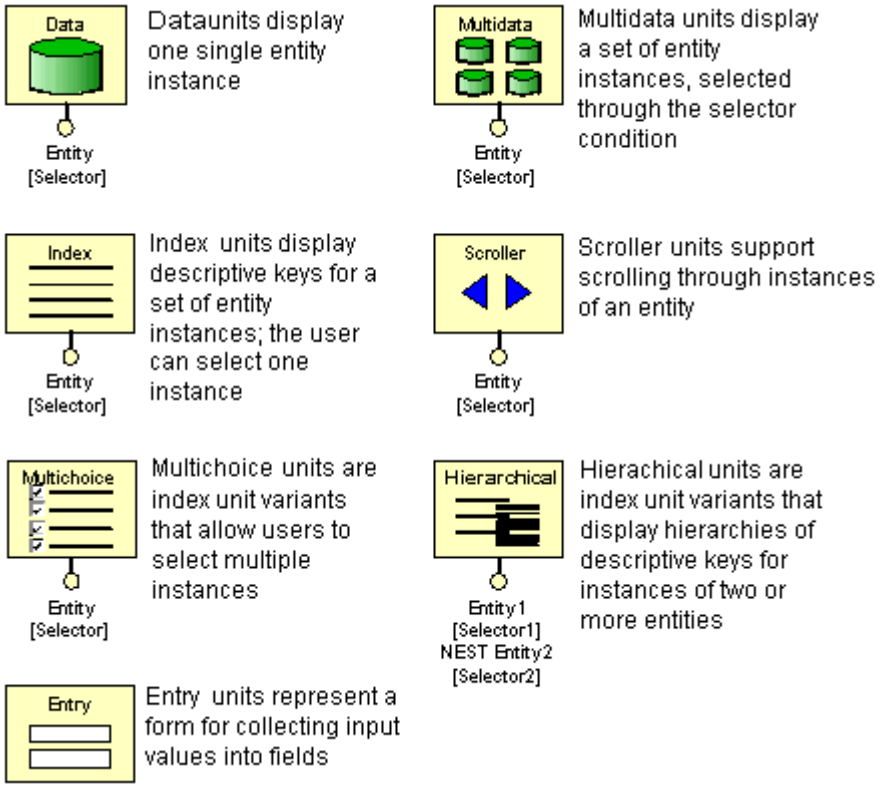
Oblast (area) je souvislá skupina stránek vytvořená za jedním účelem (např. oblast produktů, oblast technické podpory aj.). Oblasti mohou být označeny jako „landmark“ (orientační), tzn., že je možné se k nim dostat ze všech stránek v nadřazené oblasti nebo pohledu.

Stránky (pages) jsou prvky uživatelského rozhraní seskupující jednotky obsahu do smysluplné formy za účelem komunikace informace. Mezi stránkami mohou být vytvořeny odkazy pro vytvoření hypertextové struktury. Každá jednotlivá stránka může být designována jako „home“ (domovská), tedy stránka, která se zobrazí jako první při

načtení webové aplikace, jako „landmark“ (orientační), což je stránka, kterou je možné dosáhnout ze všech ostatních stránek, nebo jako „default“ (výchozí), která se načte po vstupu do oblasti, ve které se nachází.

V rámci WebML lze modelovat i vnořené (nested) stránky, a to jak v konjunktivní formě - stránky jsou zobrazeny spolu, tak v disjunktivní formě - při zobrazení jedné stránky se skryje obsah druhé[27].

Obrázek 43: Content/composition unit[30]



Jednotky obsahu (content units) jsou základní komponenty zobrazovaného obsahu. Každá jednotka buď zobrazuje informace získané z objektů datového modelu nebo vstupní formulář pro zadání dat, která mají být aplikací zpracována. Jednotky zobrazující informace jsou spojené s entitou nebo vztahem datového modelu, ze kterých se vypočítává zobrazovaný obsah.

Pro zadávání obsahu slouží jednotka „entry“. Jednotka pro zobrazování obsahu „data“ slouží pro zobrazení informací o jednom datovém objektu, ostatní zobrazovací

jednotky reprezentují způsoby procházení množinou datových objektů.

Operace slouží pro modelování akcí, které mohou být spuštěny při procházení hypertextu. Operace mají vstupní a výstupní odkazy, mohou přijímat parametry na vstupu a emitovat data na výstupu. Operace nezobrazují data, zakreslují se tedy mimo stránky.

Za účelem odchycení možných chyb a selhání můžou z operace vycházet dva typy odkazů - OK odkaz vyvolaný úspěšným provedením operace a KO odkaz vyvolaný selháním operace.

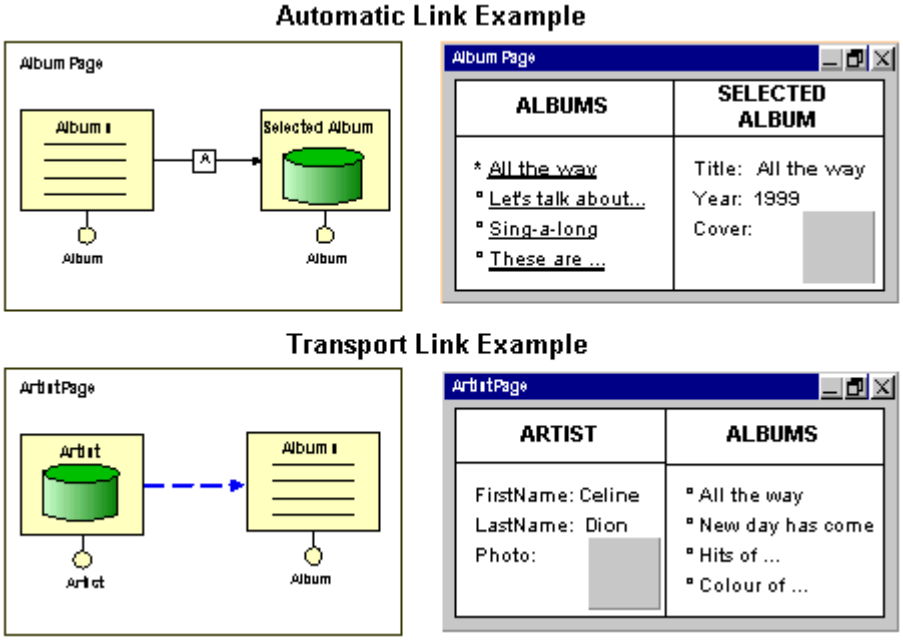
V rámci WebML jsou předdefinovány operace pro správu obsahu (create, delete, modify, connect, disconnect), správu přístupu (login, logout) a komunikaci (sendmail). Je ale také možné definovat operace nové, je však nutné specifikovat a komunikovat jejich vstupy, výstupy a činnost.

Odkazy (links) slouží v rámci hypertextu k přepínání mezi stránkami, předávání informace mezi jednotkami a případně k vyvolávání vedlejších akcí (např. spouštení operací)[27].

V některých aplikacích je nutné zobrazit obsah jednotky ihned po načtení stránky i když uživatel ještě nenavigoval po odkazu k tomuto obsahu vedoucímu. K tomu účelu slouží automatické odkazy. Automatický odkaz je kontextuální odkaz navigovaný bez uživatelského vstupu v momentě, kdy je navštívena uživatelem stránka, která obsahuje jednotku fungující jako zdroj tohoto odkazu.

Pro přenos kontextu mezi dvěma jednotkami slouží transportní odkazy. Tyto odkazy nelze použít pro navigaci, nevykresluje se při zobrazování stránky kotva ($< a >$, anchor) tomuto odkazu odpovídající.

Obrázek 44: Automatic/transport link[31]



3.5.3 Prezentační model

Prezentační model se používá pro návrh samotného vzhledu aplikace a jejích jednotlivých stránek. WebML nespecifikuje vlastní konceptuální modely pro návrh prezentační vrstvy aplikace, namísto toho využívá modely a přístupy již existující a v praxi široce využívané, např. wireframy[27].

4 Vlastní práce

Tato část práce se zabývá tvorbou správcovského rozhraní pro odborný webový časopis Slovjani.info. Toto rozhraní by mělo zjednodušit závěrečné fáze redakčního procesu tohoto časopisu. Umožňuje redakci časopisu vkládat, upravovat a mazat záznamy o jednotlivých publikovaných článcích, o citacích, ze kterých tyto články čerpají, a o autorech těchto článků. Zároveň umožňuje spravovat přístupová oprávnění k těmto informacícm. Součástí vypracovaného rozhraní je manuál provádějící uživatele prací s aplikací.

4.1 Návrh

Nejprve proběhl počáteční sběr požadavků. Z toho bylo vyvozeno, že každý uživatel systému se považuje potenciálně za autora a každý článek má mít o sobě kromě stávajících dat uloženy i použité zdroje. Po počátečním sběru požadavků na systém byly vypracovány modely podle specifikací UML a WebML.

Obrázek 45: Use Case diagram projektu (vlastní)

V rámci UML specifikace byl vypracován Use Case diagram. V něm byly identifikovány všechny hlavní případy užití - funkcionality - tohoto systému včetně jednotlivých aktérů. Diagram je navržený z pohledu uživatele, vstupuje do něj ale i systém.

Na základě navrženého Use Case diagramu byly sepsány hlavní a alternativní scénáře (plné znění scénářů viz příloha A). Tyto scénáře se odvíjí od případů užití a obsahují sekvence kroků, které musí podniknout aktéři, kteří jsou s tím daným případem užití asociovaní, aby došlo k jeho splnění. Případný alternativní scénář popisuje, co se bude dít, pokud nastane chyba.

Obrázek 46: Ukázka scénáře (vlastní)

Use Case - registrace

- **Aktéři**

  Neregistrovaný uživatel

  Systém

- **Hlavní scénář**

  1.1 Systém zobrazí registrační formulář

  1.2 Uživatel vyplní alespoň požadované údaje a formulář odešle

  1.3 Systém data převezme a validuje

  1.4 Systém uloží data o novém uživateli a zobrazí hlášení o úspěchu

- **Alternativní scénář**

  2.1 Uživatel zadal neplatný vstup

  2.2 Systém upozorní uživatele na chybu

  2.3 Uživatel opraví chybu a pokračuje dále od kroku 1.2

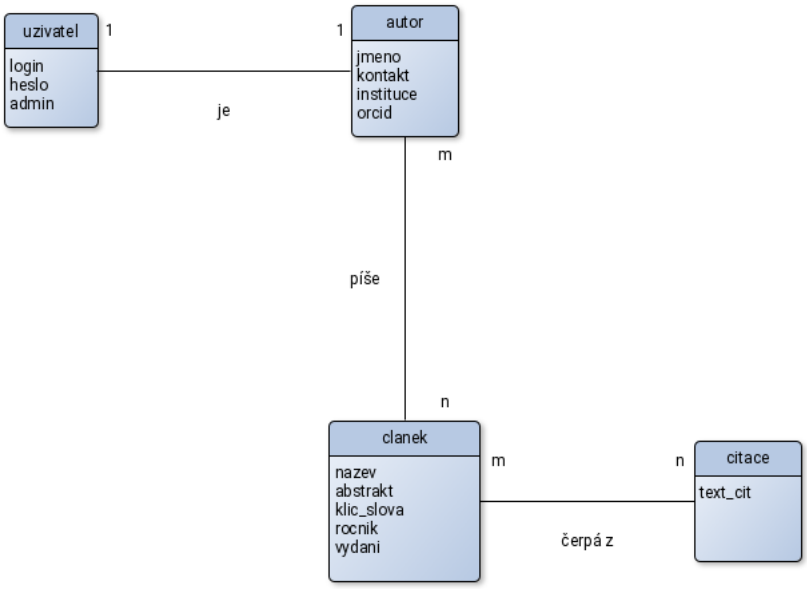
Po identifikaci funkcionalit systému byla identifikována data, se kterými bude systém pracovat. V případě tohoto systému se jedná o data o uživatelích, autorech, článcích a citacích. Tato data byla následně vyjádřena pomocí WebML data modelu. Tento model

zobrazuje ukládaná data jako entity a vazby mezi nimi. Těmto vazbám byl přiřazen jejich význam a byla určena kardinalita vztahů. Jedná se konkrétně o tyto vztahy (graficky viz diagram):

- Jeden uživatel je právě jedním autorem. Jeden autor je nejvýše jedním uživatelem.
- Jeden autor píše jeden a více článků. Jeden článek je napsán jedním a více autory.
- Jeden článek čerpá z jedné a více citací. Jedna citace je použita v jednom a více článcích.

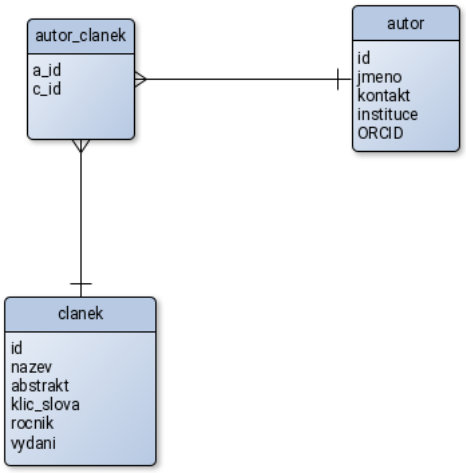
Je nutno dodat, že z důvodů bezpečnosti dat byly oproti skutečnému systému změněny názvy entit/tabulek a sloupců/atributů, stejně jako některých funkcí a proměnných, pokud jsou uvedeny.

Obrázek 47: Data model projektu (vlastní)



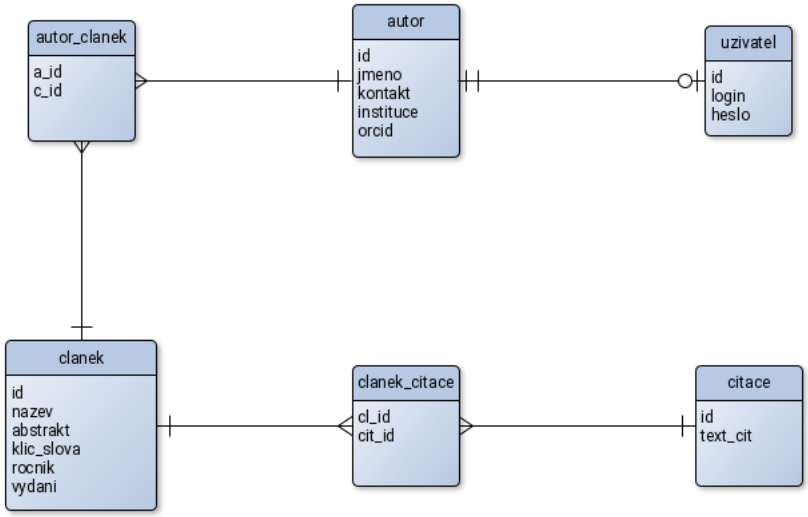
Při návrhu databáze se vycházelo z existující databázové struktury. Tato původní databáze nesla pouze data o článcích a autorech, která se dále využívala pro vyhledávání a zobrazování přehledů jednotlivých vydání časopisu. Pro potřeby aplikace byla ukládaná data nedostačující a databáze musela proto být rozšířena.

Obrázek 48: E-R diagram - původní schéma (vlastní)



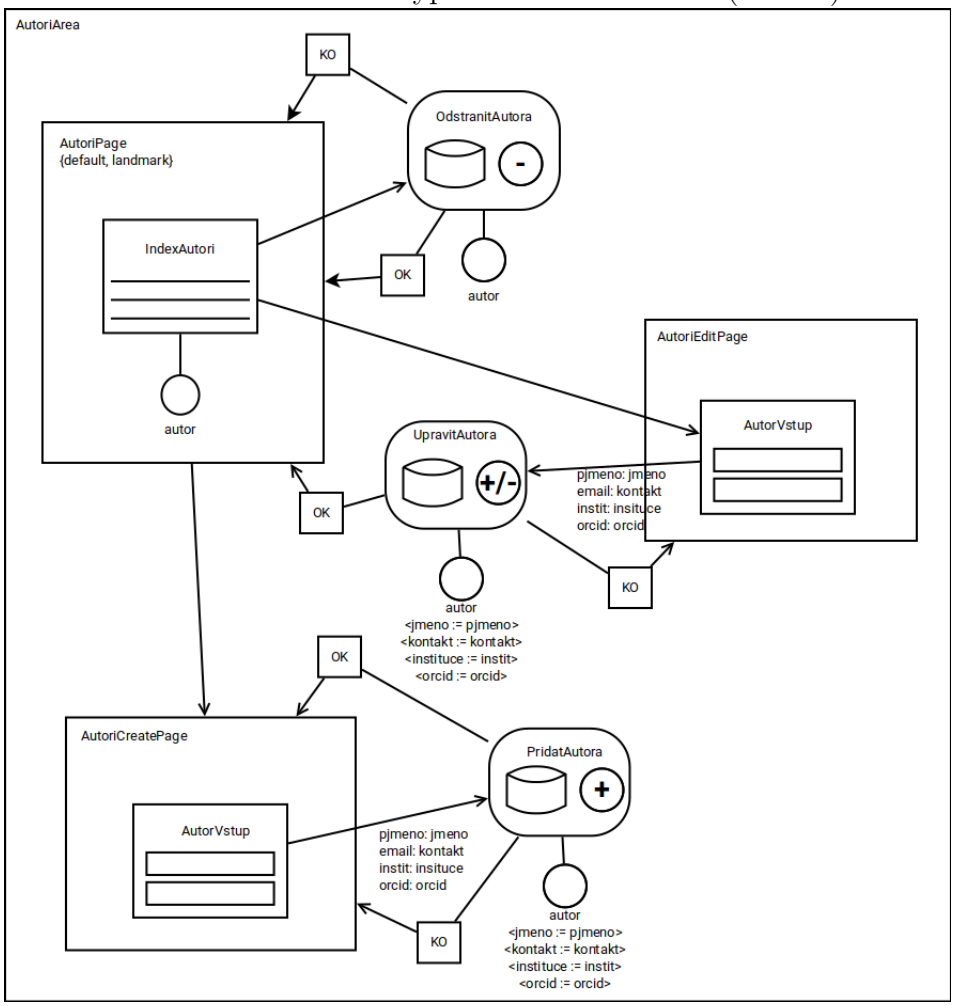
V souladu s navrženým WebML data modelem byla databáze rozšířena tak, aby pojala data o uživatelích a citacích, ze kterých čerpají stránky. Vzhledem k existenci vazeb typu m:n musely být použity asociační tabulky, konkrétně nová asociační tabulka *clanek_citace*. Tyto tabulky obsahují odkazy na primární klíče tabulek, mezi kterými vytváří vazbu mnoho - mnoho.

Obrázek 49: E-R diagram - rozšířené schéma (vlastní)



Pro zjednodušení a naplánování implementace projektu by navržen WebML hypertextový model. V tomto modelu je zobrazena aplikace jako množina stránek. Každá stránka obsahuje jednotky obsahu, které slouží k zobrazení dat nebo vstupů, mezi stránkami a jednotkami jsou odkazy sloužící k přesunu mezi stránkami nebo posílání informací mezi jednotkami (plná podoba modelu viz příloha B).

Obrázek 50: Ukázka hypertextového modelu (vlastní)



4.2 Implementace

Návrh systému následovala samotná implementace. Bylo nejprve nutné upravit stávající MySQL databázi tak, aby odpovídala požadavkům navrženým v datovém modelu. Byly tedy přidány tabulky *uzivatel*, *citace* a *clanek_citace* obsahující data potřebná pro funkčnost systému:

- *uzivatel*: id, login, heslo, spravce, autor_id
- *citace*: id, cit_text
- *clanek_citace*: clanek_id, citace_id

Obrázek 51: SQL skript pro úpravu databáze (vlastní)

```
; BOOLEAN bude převeden na SMALLINT(1), je to pouze rychlejší forma
; zápisu
CREATE TABLE uzivatel (
  id INT PRIMARY KEY,
  login VARCHAR(30) NOT NULL, heslo VARCHAR(30) NOT NULL,
  spravce BOOLEAN DEFAULT 0, autor_id VARCHAR(30) NOT NULL
);
CREATE TABLE citace (
  id INT PRIMARY KEY AUTO_INCREMENT, cit_text CHAR(255) UNIQUE NOT NULL
);
CREATE TABLE clanek_citace (
  clanek_id INT, citace_id INT,
  PRIMARY KEY(clanek_id, citace_id)
);
```

S upravenou databází bylo možné zahájit samotnou tvorbu systému. Serverová část systému, tedy ta, která zpracovává a ověřuje uživatelský vstup a komunikuje s databází,

byla vytvořena v PHP. Pro komunikaci s databází bylo využito rozhraní mysqli a aby se zabránilo případným SQL injection útokům, byly všude, kde se zpracovával uživatelský vstup, kromě standardních technik, jako je escapeování textových řetězců, použity předpřipravené dotazy (prepared statements), které umožňují předem připravit na databázovém serveru dotaz a poté do něj odděleně posílat data. Při dotazování se nad databází tímto způsobem téměř nehrozí, že by útočník mohl vyvolat chybu a vykonat vlastní dotaz.

Zobrazovací část byla vytvořena kombinací HTML/CSS, PHP pro doplňování informací získaných z databáze a JavaScriptu pro dynamizaci stránek na straně klienta a provádění některých operací, kterými není přímo nutné zatěžovat server.

Obrázek 52: Ukázka AJAX požadavku na DOI server (vlastní)

```
function getDataDOI(doi) {
  // vytvořit http požadavek
  request = new XMLHttpRequest();
  request.open("GET", "https://dx.doi.org/" + doi);
  // přijímaná data - json + styly citací
  request.setRequestHeader("Accept",
    "application/vnd.citationstyles.csl+json");
  request.onload = function() {
    if(request.status === 200) { // HTTP OK
      json.parse(request.responseText);
      // zpracovat data
    }
  };
  request.send();
}
```

Ze strany zadavatele byl požadavek, aby se daly autoři vkládat do databáze na základě jejich ORCID a články podle jejich DOI. Veřejné API organizací spravujících

tyto idenfikátory poskytuje téměř všechna data ukládaná do databáze, jsou však výjimky. ORCID například veřejně neposkytuje kontakty na přispěvatele a výzkumné pracovníky, metadata DOI zase nezahrnují informace jako abstrakt a klíčová slova. Z tohoto důvodu byl tento požadavek upraven, namísto vkládání dat do databáze se identifikátory využívají k předvyplnění dat do vstupních formulářů. Tato funkcionalita byla naprogramována v JavaScriptu a využívá AJAX požadavky pro komunikaci s API, data jsou vracena v JSON formátu, který je dále zpracován a data z něj přepsána do formuláře. Uživatel má potom možnost data upravit, doplnit chybějící informace a formulář odeslat.

Obrázek 53: Ukázka PHP skriptu odpovídajícího na AJAX požadavky (vlastní)

```
<?PHP
// vrácení jmen autorů podle vzoru
$c = new mysqli("10.0.0.1","login","heslo","databaze")
  or die("Error");
if(isset($_GET["q"])) {
  $poz = $c->escape_string(htmlspecialchars($_GET["q"]));
  $poz = "%" . $poz . "%"
  $data = $c->query("SELECT jmeno FROM autor WHERE jmeno LIKE '$poz'");
  $autori = array();
  while($z = $data->fetch_array()) {
   $tmp = array("value" => $z["jmeno"]);
   array_push($autori, $tmp);
  }
  $vysledek = array("autori" => $autori);
  echo json_encode($vysledek);
}
?>
```

Podobné funkce bylo posléze využito i v další části editačních formulářů, konkrétně při vkládání a editaci článků, pro realizaci našeptávače. Tento našeptávač umožňuje

uživatelům při vpisování citovaných zdrojů a jmen autorů vybrat z již existujících dat. Jelikož, pokud by se data pro zdroj našeptávače nahrávala všechna při načítání stránky, by při větším objemu dat v databázi mohlo dojít na straně serveru k time-outu a tím pádem k nedokončení vykreslování stránky, plní se tento zdroj za pomoci JavaScriptové funkce až potom, co uživatel začne psát. Funkce získává data v JSON podobě z PHP skriptu na serveru. Data se do zdroje zapisují až po tom, co server odpoví, ale protože HTTP požadavky se v JavaScriptu zpracovávají asynchronně, může uživatel pracovat bez přerušení. Je také možné ošetřit situaci, kdy získání dat ze serveru selže, buď proto, že data odpovídající požadavku nebyla nalezena, nebo proto, že došlo k time-outu. V tomto případě stačí, když se selhání ignoruje a jednoduše se při něm nezapíšou žádná data do datalistu.

Obrázek 54: Autentizace a vytvoření sezení (vlastní)

```
<?php
$sul = "Tohle je příklad soli";
$jmeno = htmlspecialchars($_POST["uziv_jmeno"]);
$heslo = htmlspecialchars($_POST["uziv_heslo"]);
$heslo = hash("sha256", $heslo . hash("sha256", $sul));
$c = new mysqli("10.0.0.1","login","heslo","databaze");
$d_ = "SELECT id, spravce FROM login WHERE jmeno = ? AND heslo = ?";
$d = $c->prepare($d_);
$d->bind_param("ss", $jmeno, $heslo);
$d->bind_result($id, $spravce);
$d->execute();
if($d->fetch()) {
  session_start();
  $_SESSION["id_uziv"] = $id;
  $_SESSION["uziv_spravce"] = $spravce;
}
?>
```

Ověřování oprávnění bylo realizováno pomocí sezení (sessions). Sezení se vytváří na základě autentizace pomocí uživatelského jména a hesla při přihlašování. Při vytvoření sezení se na straně klienta uloží cookie s identifikátorem tohoto sezení (tzv. session cookie) a na server se mohou do superglobálního pole *$_SESSION*, které je unikátní pro každé sezení a není přístupné z klienta, ukládat další data potřebná k identifikaci uživatele. V případě tohoto systému se jedná především o identifikátor uživatelského účtu a informaci o oprávnění, tedy zda je či není uživatel správcem. Ve skriptu každé stránky, která sezení využívá, je poté nutné jej spustit. Při spuštění sezení server ověří, zda v paměti klienta existuje session cookie. Pokud ano, načte z ní identifikátor sezení a bude jej v rámci skriptu využívat k přistupování k poli *$_SESSION*.

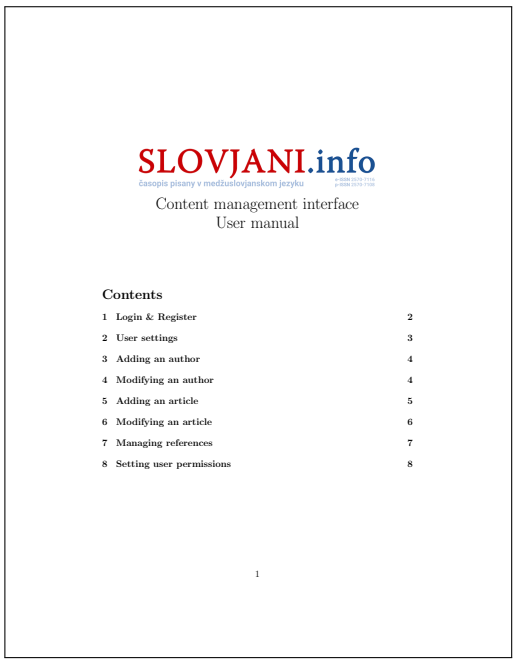
4.3 Příručka

Součástí práce je i uživatelská příručka (viz příloha C). Tato příručka byla vytvořena za pomoci sázecího značkovacího jazyka LaTeX a exportována pomocí nástrojů tohoto jazyka do formátů PDF a HTML.

Obrázek 55: Ukázka LaTeX kódu pro vytvoření příručky (vlastní)

```
\documentclass[a4,12pt]{article}
\usepackage{graphicx}
\graphicspath{{./img/}}
\usepackage{tabularx}
\usepackage[utf8]{inputenc}
\title{\centering{\includegraphics[scale=0.4]{logo-left}}\\
Content management interface\\User manual}
\date{}
\begin{document}
  \maketitle
  \tableofcontents % ...pokračuje dále
\end{document}
```

Obrázek 56: Výstup kódu na předchozím obrázku (vlastní)


SLOVJANI.info
časopis pisany v medžuslovjanskom jezyku
Content management interface
User manual
Contents
1 Login & Register 2
2 User settings 3
3 Adding an author 4
4 Modifying an author 4
5 Adding an article 5
6 Modifying an article 6
7 Managing references 7
8 Setting user permissions 8
1

HTML podoba příručky byla po exportu dále lehce přepracována z důvodu jejího začlenění do zbytku webu, jednalo se zejména o přidání hlavičky a patičky webu a nabídek. Je dostupná všem přihlášeným uživatelům rozhraní se správcovským oprávněním.

5 Výsledky a diskuse

Výstupem této práce je správcovské rozhraní pro webový odborný časopis Slovjani.info. Rozhraní umožňuje redakci časopisu manipulovat s daty o publikovaných článcích a autorech. Zároveň skrz něj lze řídit oprávnění pro přístup k tomuto rozhraní. Součástí rozhraní je uživatelská příručka. Tato příručka, psaná v angličtině a existující jak v podobě webové stránky dostupné z rozhraní samotného, tak v podobě elektronického dokumentu ve formátu PDF (viz příloha C), stručně provádí uživatele prací s rozhraním.

5.1 Vytvořené modely

Rozhraní bylo vytvořeno na základě WebML datového a hypertextového modelu (příloha B), které specifikují data ukládaná v rámci systému, stránky pro zobrazení, zadávání a úpravu těchto dat a operace nad těmito daty - vkládání, mazání, úpravy ap. Samotná funkcionalita modelovaná v těchto modelech byla identifikována v rámci Use Case diagramu UML a navazujících scénářích (příloha A), které definují hrubý sled operací potřebných pro provedení dané funkcionality. Ačkoliv byly tyto modely vytvořeny s úmyslem návrhu tohoto konkrétního systému a jeho rozhraní, není úplně vyloučeno, že by se daly využít jako základ pro vytvoření správcovského rozhraní pro jiný web podobného zaměření a po přepracování i pro obecněji zaměřené webové časopisy.

5.2 ORCID a DOI

Správcovské rozhraní umožňuje svým uživatelům vkládat údaje o autorech na základě ORCID identifikátorů. K tomuto se využívá veřejný ORCID rejstřík, který obsahuje o autorech informace jako jejich jméno, instituce, které je zaměstnávají nebo zaměstnávaly nebo jejich publikované práce. Ne vždy jsou bohužel dostupné kontaktní informace, které se v rámci systému časopisu také ukládají, je tedy nutné je doplňovat ručně. Nedostupnost kontaktních údajů je způsobena tím, že ne všichni držitelé ORCID mají pro tento účel ověřené adresy nebo se rozhodli pro jejich nezveřejnění[32].

Podobně jako ORCID se dá v rámci rozhraní využít DOI identifikátor k vkládání údajů o článcích. Veřejný DOI rejstřík obsahuje o v něm zahrnutých článcích informace jako název článku, jak v původním jazyce, tak v angličtině, jména autorů nebo čerpané zdroje. Ačkoliv je rejstřík připraven pro vkládání abstraktu a klíčových slov, tato data jsou u v současnosti publikovaných a v rejstříku zahrnutých článků často nedostupná, je proto podobně jako emailové adresy autorů nutné je doplnit ručně. Nedostupnost těchto dat může být způsobená tím, že při publikaci a generování DOI identifikátoru tato data vůbec nebyla zadána, například proto, že bylo zamýšleno je zpřístupnit až po zaplacení za přístup k článku samotnému.

5.3 Správa oprávnění

Dodatečným požadavkem byla změna přístupu k elektronickému vydání časopisu. Doposud měli k elektronickým vydáním časopisu přístup všichni návštěvníci webu, podle tohoto požadavku k nim mají přístup pouze registrovaní uživatelé. Proto musel být navíc vytvořen jednoduchý systém oprávnění, aby se nemohli registrovaní ale nepovolaní uživatelé dostat ke správcovskému rozhraní a měnit údaje uložené v systému.

6 Závěr

Cílem práce bylo vytvořit správcovské rozhraní pro mezinárodní odborný webový časopis Slovjani.info na praktickou objednávku mezinárodní knihovny CEEOL ve Frankfurtu, který by umožňoval vkládání a editaci údajů o autorech a publikovaných článcích, vkládání a možnost znovupoužití citací. To vše mělo být dosaženo pomocí jazyka PHP a neměly být použity předpřipravené moduly pro správu obsahu, jako jsou například Joomla nebo WordPress.

V rámci procesu návrhu byly vytvořeny UML a WebML modely, konkrétně Use Case, data model a hypertext model, ze kterých by teorieticky bylo možné vycházet nebo je přímo použít jako základ pro návrh obdobných systémů.

Rozhraní bylo nakonec vytvořeno kromě PHP v menší míře i v jazyce JavaScript. Při tvorbě nebyly použity existující CMS moduly, nedošlo ani na použití frameworků, které, ač by mohly proces tvorby mnohdy značně zjednodušit, by také mohly být spolu s CMS moduly díky své otevřenosti a častým závislostem na dalších modulech a frameworcích snadným cílem útoků.

Na základě zadání byla také vytvořena uživatelská příručka, která ve své PDF a HTML verzi provádí uživatele prací se systémem.

7 Seznam použitých zdrojů

[1] *Internet Live Stats* [online]. [cit. 2020-03-05]. Dostupné z: https://www.internetlivestats.com/

[2] Journal Detail. *CEEOL* [online]. [cit. 2020-03-31]. Dostupné z: https://ceeol.com/search/journal-detail?id=1890

[3] *CEEOL* [online]. [cit. 2020-03-31]. Dostupné z: https://ceeol.com/

[4] KLOTZ, Bea. The Central and Eastern European Online Library (http://www.ceeol.com). *The Serials Librarian* [online]. 2007, 53(1-2), 191-201 [cit. 2020-03-31]. DOI: 10.1300/J123v53n01_15. ISSN 0361-526X. Dostupné z: http://www.tandfonline.com/doi/abs/10.1300/J123v53n01_15

[5] BARKER, Deane. *Web content management: systems, features, and best practices.* Boston: O'Reilly, 2016. ISBN 978-1491908129.

[6] SILBERSCHATZ, Abraham, Henry F. KORTH a S. SUDARSHAN. *Database system concepts. 6th ed.* NEW YORK: McGraw-Hill, 2011. ISBN 978-0-07-352332-3.

[7] ISO/IEC JTC 1/SC 32: Data management and interchange. *ISO* [online]. [cit. 2020-03-08]. Dostupné z: https://www.iso.org/committee/45342/x/catalogue/p/1/u/0/w/0/d/0

[8] SQL Tutorial. *W3Schools Online Web Tutorials* [online]. [cit. 2020-03-09]. Dostupné z: https://www.w3schools.com/sql/

[9] SQL: TRANSACTIONS. *GeeksForGeeks* [online]. [cit. 2020-03-11]. Dostupné z: https://www.geeksforgeeks.org/sql-transactions/

[10] MACINTYRE, Peter. *PHP: the good parts.* Sebastopol, CA: O'Reilly, 2010. ISBN 9780596804374.

[11] PHP: PHP Manual. *PHP* [online]. 2020 [cit. 2020-02-06]. Dostupné z: https://www.php.net/manual/en/

[12] PHP Operators. *W3Schools Online Web Tutorials* [online]. [cit. 2020-03-03]. Dostupné z: https://www.w3schools.com/php/php_operators.asp

[13] WELLING, Luke a Laura THOMSON. *PHP and MySQL Web development. 4th ed.* Upper saddle River, NJ: Addison-Wesley, 2009. Developer's library. ISBN 978-0-672-32916-6.

[14] Understanding Rainbow Table Attack. *GeeksForGeeks* [online]. [cit. 2020-03-15]. Dostupné z: https://www.geeksforgeeks.org/understanding-rainbow-table-attack/

[15] What is UML. *Unified Modeling Language* [online]. [cit. 2020-03-14]. Dostupné z: https://www.uml.org/what-is-uml.htm

[16] UML Tutorial. *Tutorialspoint* [online]. [cit. 2020-03-14]. Dostupné z: https://www.tutorialspoint.com/uml/

[17] [UML Class Diagram]. In: *Tutorialspoint* [online]. [cit. 2020-03-14]. Dostupné z: https://www.tutorialspoint.com/uml/images/uml_class_diagram.jpg

[18] [UML Object Diagram]. In: *Tutorialspoint* [online]. [cit. 2020-03-14]. Dostupné z: https://www.tutorialspoint.com/uml/images/uml_object_diagram.jpg

[19] [UML Use Case Diagram]. In: *Tutorialspoint* [online]. [cit. 2020-03-14]. Dostupné z: https://www.tutorialspoint.com/uml/images/uml_use_case_diagram.jpg

[20] [UML Activity Diagram]. In: *Tutorialspoint* [online]. [cit. 2020-03-14]. Dostupné z: https://www.tutorialspoint.com/uml/images/uml_activity_diagram.jpg

[21] [UML Class Notation]. In: *Tutorialspoint* [online]. [cit. 2020-03-14]. Dostupné z: https://www.tutorialspoint.com/uml/images/notation_class.jpg

[22] [UML Statechart Diagram]. In: *Tutorialspoint* [online]. [cit. 2020-03-14]. Dostupné z: https://www.tutorialspoint.com/uml/images/um_statechart_diagram.jpg

[23] [UML Sequence Diagram]. In: *Tutorialspoint* [online]. [cit. 2020-03-14]. Dostupné z: https://www.tutorialspoint.com/uml/images/um_sequence_diagram.jpg

[24] [UML Collaboration Diagram]. In: *Tutorialspoint* [online]. [cit. 2020-03-14]. Dostupné z: https://www.tutorialspoint.com/uml/images/um_collaboration_diagram.jpg

[25] UML: sequence diagram - sekvenční diagram. *UML a OO, metodologie* [online]. [cit. 2020-03-14]. Dostupné z: http://mpavus.wz.cz/uml/uml-b-sequence-3-2-4.php

[26] UML: communication diagram - diagram komunikací. *UML a OO, metodologie* [online]. [cit. 2020-03-14]. Dostupné z: http://mpavus.wz.cz/uml/uml-b-communication-3-2-5.php

[27] *Webml.org* [online]. 2003 [cit. 2020-03-17]. Dostupné z: http://webml.deib.polimi.it/page1.do

[28] [Data model]. In: *WebML.org* [online]. [cit. 2020-03-18]. Dostupné z: http://webml.deib.polimi.it/upload/ent9/2/datamodel.png

[29] [Hypertext model]. In: *WebML.org* [online]. [cit. 2020-03-18]. Dostupné z: http://webml.deib.polimi.it/upload/ent9/1/HTexamples.png

[30] [Composition units]. In: *WebML.org* [online]. [cit. 2020-03-18]. Dostupné z: http://webml.deib.polimi.it/upload/ent9/1/CompositionUnit1.png

[31] [Links]. In: *WebML.org* [online]. [cit. 2020-03-18]. Dostupné z: http://webml.deib.polimi.it/upload/ent9/3/Links.png

[32] Why don't I see email addresses in the ORCID records I access? *ORCID* [online]. [cit. 2020-03-20]. Dostupné z: https://support.orcid.org/hc/en-us/articles/360006896114-Why-don-t-I-see-email-addresses-in-the-ORCID-records-I-access-

8 Seznam použitých zkratek

ACID	Atomicity, Consistency, Isolation, Durability
AJAX	Asynchronous JavaScript And XML
CEEOL	Central and Eastern European Online Library
CMS	Content Management System
DBMS	Database Management System
DDL	Data Definition Language
DML	Data Manipulation Language
HTML	HyperText Markup Language
JSON	JavaScript Object Notation
OID	Object Identifier
PHP	PHP: Hypertext Preprocessor
SQL	Structured Query Language
SŘBD	Systém řízení báze dat
UML	Unified Modeling Language
WCM	Web Content Management
WebML	Web Modeling Language
XSS	Cross Site Scripting

9 Přílohy

Příloha A: Use Case scénáře

Use Case - registrace

- **Aktéři**

  Neregistrovaný uživatel

  Systém

- **Hlavní scénář**

  1.1 Systém zobrazí registrační formulář

  1.2 Uživatel vyplní alespoň požadované údaje a formulář odešle

  1.3 Systém data převezme a zvaliduje

  1.4 Systém uloží data o novém uživateli a zobrazí hlášení o úspěchu

- **Alternativní scénář**

  2.1 Uživatel zadal neplatný vstup

  2.2 Systém upozorní uživatele na chybu

  2.3 Uživatel opraví chybu a pokračuje dále od kroku 1.2

Use Case - přihlášení

- **Aktéři**

  Běžný uživatel

  Systém

- **Prerekvizity**

  Uživatel je registrován

- **Hlavní scénář**

  1.1 Systém zobrazí přihlašovací formulář

  1.2 Uživatel vyplní jméno a heslo a odešle formulář

  1.3 Systém data převezme a validuje

  1.4 Systém vytvoří pro uživatele sezení a přesměruje ho na hlavní stránku

- **Alternativní scénář**

  2.1 Uživatel zadal neplatný vstup

  2.2 Systém upozorní uživatele na chybu

  2.3 Uživatel chybu odstraní, pokračuje se od kroku 1.2

Use Case - přístup k pdf časopisu

- **Aktéři**

  Běžný uživatel

  Systém

- **Prerekvizity**

  Uživatel je přihlášen

- **Hlavní scénář**

  1.1 Systém zobrazí seznam vydání časopisu

  1.2 Uživatel vybere vydání ke stažení

  1.3 Systém poskytne PDF verzi

Use Case - uživatelská nastavení

- **Aktéři**

  Běžný uživatel

  Systém

- **Prerekvizity**

  Uživatel je přihlášen

- **Hlavní scénář**

  1.1 Uživatel přejde do nabídky nastavení

  1.2 Systém zobrazí přehled uživatelských nastavení

Use Case - změnit heslo

- **Aktéři**

  Běžný uživatel

  Systém

- **Prerekvizity**

  Uživatel je přihlášen

  Uživatel je v přehledu nastavení

- **Hlavní scénář**

  1.1 Uživatel vybere změnu hesla

  1.2 Systém zobrazí formulář pro změnu hesla

  1.3 Uživatel vyplní stávající a nové heslo a odešle formulář

  1.4 Systém data validuje a aktualizuje záznam

- **Alternativní scénář**

  2.1 Uživatel zadá neplatný vstup

  2.2 Systém upozorní uživatele

  2.3 Uživatel opraví chybu a pokračuje krokem 1.3

Use Case - změnit osobní údaje

- **Aktéři**

  Běžný uživatel

  Systém

- **Prerekvizity**

  Uživatel je přihlášen

  Uživatel je v přehledu nastavení

- **Hlavní scénář**

  1.1 Uživatel vybere změnu osobních údajů

  1.2 Systém zobrazí formulář pro změnu osobních údajů

  1.3 Uživatel vyplní změny a odešle formulář

  1.4 Systém data validuje a aktualizuje záznam

- **Alternativní scénář**

  2.1 Uživatel zadá neplatný vstup

  2.2 Systém upozorní uživatele

  2.3 Uživatel opraví chybu a pokračuje krokem 1.3

Use Case - odstranit účet

- **Aktéři**

  Běžný uživatel

  Systém

- **Prerekvizity**

  Uživatel je přihlášen

  Uživatel je v přehledu nastavení

- **Hlavní scénář**

  1.1 Uživatel vybere odstranění účtu

  1.2 Systém požádá o potvrzení

  1.3 Uživatel potvrdí odstranění

  1.4 Systém odstraní a odhlásí uživatele a přesměruje na index

- **Alternativní scénář**

  2.1 Uživatel nepotvrdí odstranění

  2.2 Systém přesměruje zpět na nastavení

Use Case - přidat autora

- **Aktéři**

  Správce

  Systém

- **Prerekvizity**

  Uživatel je přihlášen

- **Hlavní scénář**

  1.1 Uživatel přejde na přidání autora

  1.2 Systém zobrazí formulář pro přidání autora

  1.3 Uživatel vyplní a odešle formulář

  1.4 Systém validuje data, vytvoří záznam a informuje o úspěchu

- **Alternativní scénář**

  2.1 Uživatel zadá neplatný vstup

  2.2 Systém upozorní uživatele

  2.3 Uživatel opraví chybu a pokračuje krokem 1.3

Use Case - upravit autora

- **Aktéři**

  Správce

  Systém

- **Prerekvizity**

  Uživatel je přihlášen

- **Hlavní scénář**

  1.1 Uživatel vybere autora k úpravě

  1.2 Systém zobrazí formulář pro úpravu autora

  1.3 Uživatel vyplní a odešle formulář

  1.4 Systém validuje data, aktualizuje záznam a informuje o úspěchu

- **Alternativní scénář**

  2.1 Uživatel zadá neplatný vstup

  2.2 Systém upozorní uživatele

  2.3 Uživatel opraví chybu a pokračuje od kroku 1.3

Use Case - odstranit autora

- **Aktéři**

  Správce

  Systém

- **Prerekvizity**

  Uživatel je přihlášen

- **Hlavní scénář**

  1.1 Uživatel vybere autora k odstranění

  1.2 Systém požádá o potvrzení

  1.3 Uživatel potvrdí odstranění

  1.4 Systém odstraní autora a informuje o úspěchu

- **Alternativní scénář**

  2.1 Uživatel nepotvrdí odstranění

  2.2 Systém vrátí uživatele na předchozí stránku

Use Case - přidat článek

- **Aktéři**

  Správce

  Systém

- **Prerekvizity**

  Uživatel je přihlášen

- **Hlavní scénář**

  1.1 Uživatel přejde na přidání článku

  1.2 Systém zobrazí formulář pro přidání článku

  1.3 Uživatel vyplní a odešle formulář

  1.4 Systém validuje data, vytvoří záznam a informuje o úspěchu

- **Alternativní scénář**

  2.1 Uživatel zadá neplatný vstup

  2.2 Systém upozorní uživatele

  2.3 Uživatel opraví chybu a pokračuje krokem 1.3

Use Case - upravit článek

- **Aktéři**

  Správce

  Systém

- **Prerekvizity**

  Uživatel je přihlášen

- **Hlavní scénář**

  1.1 Uživatel vybere článek k úpravě

  1.2 Systém zobrazí formulář pro úpravu článku

  1.3 Uživatel vyplní a odešle formulář

  1.4 Systém validuje data, aktualizuje záznam a informuje o úspěchu

- **Alternativní scénář**

  2.1 Uživatel zadá neplatný vstup

  2.2 Systém upozorní uživatele

  2.3 Uživatel opraví chybu a pokračuje od kroku 1.3

Use Case - odstranit článek

- **Aktéři**

  Správce

  Systém

- **Prerekvizity**

  Uživatel je přihlášen

- **Hlavní scénář**

  1.1 Uživatel vybere článek k odstranění

  1.2 Systém požádá o potvrzení

  1.3 Uživatel potvrdí odstranění

  1.4 Systém odstraní článek a informuje o úspěchu

- **Alternativní scénář**

  2.1 Uživatel nepotvrdí odstranění

  2.2 Systém vrátí uživatele na předchozí stránku

Use Case - změnit oprávnění uživatele

- **Aktéři**

  Správce

  Systém

- **Prerekvizity**

  Uživatel je přihlášen

- **Hlavní scénář**

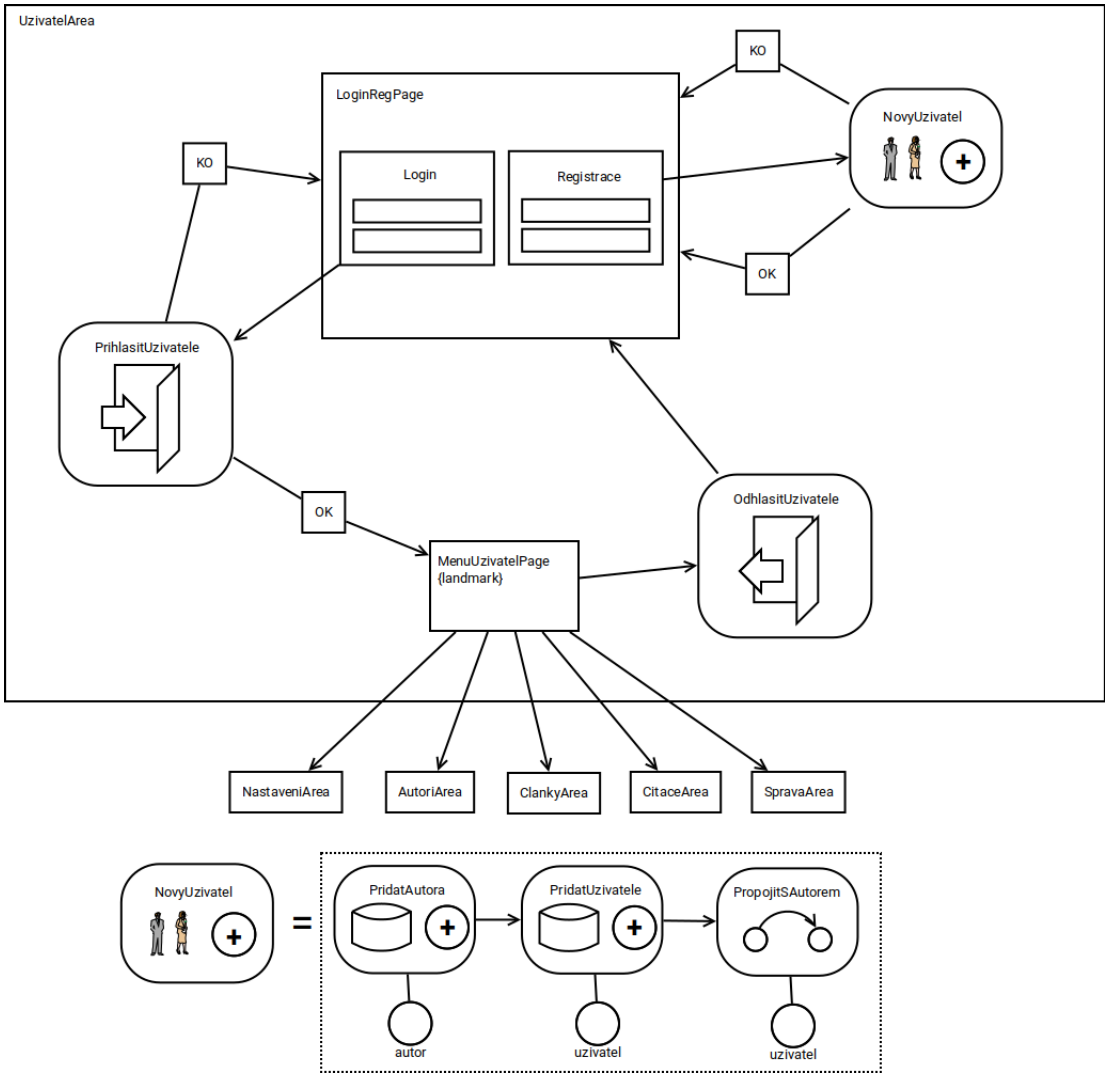
  1.1 Uživatel přejde na správu oprávnění

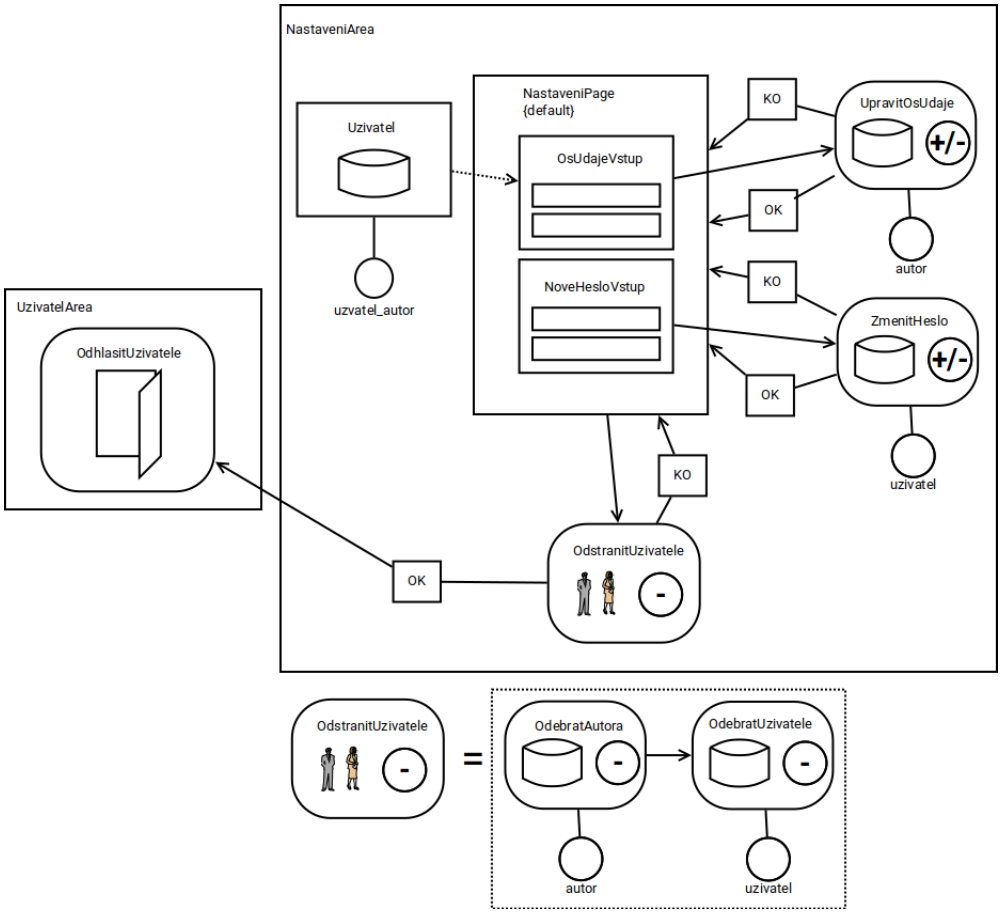
  1.2 Systém zobrazí seznam uživatelů a jejich oprávnění

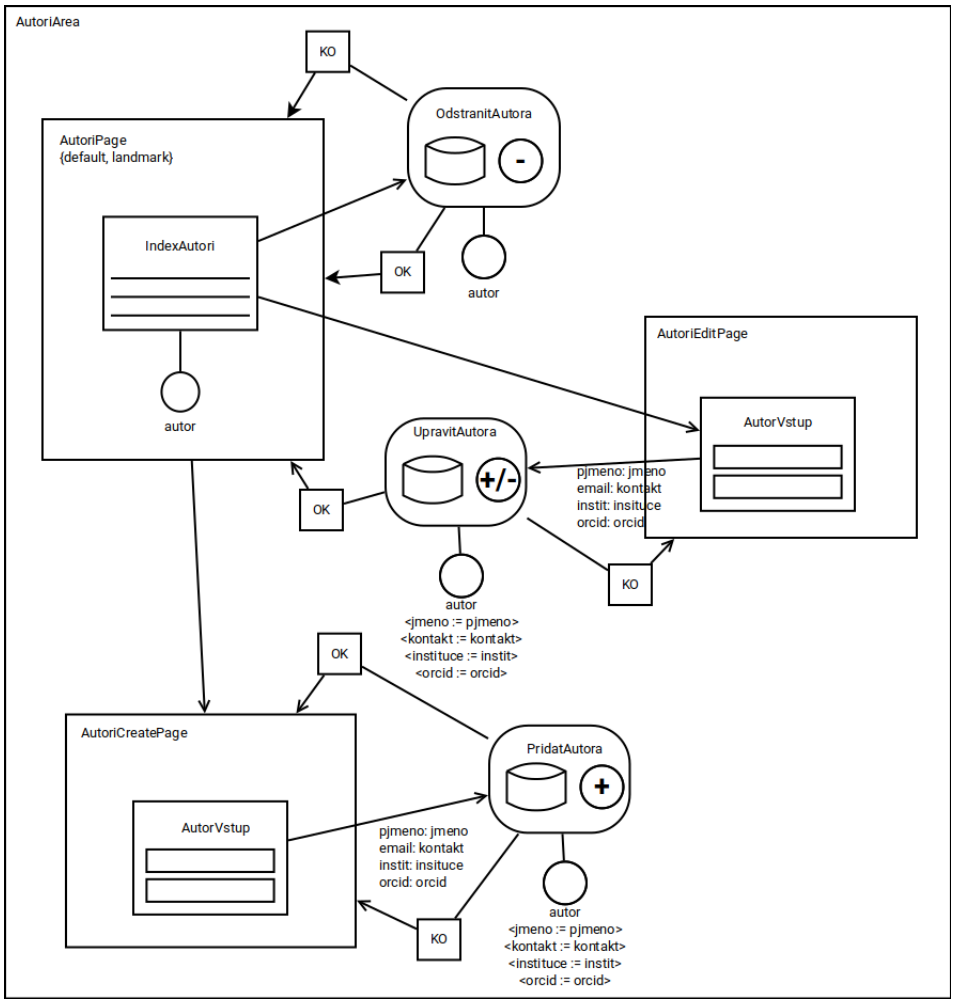
  1.3 Uživatel upraví oprávnění a odešle

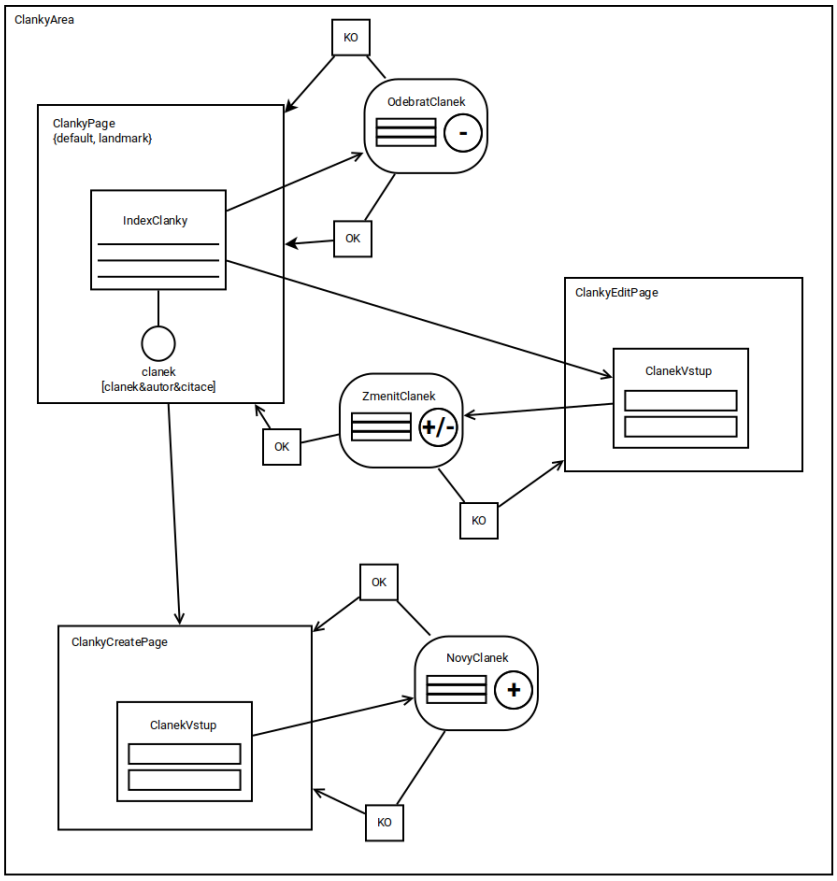
  1.4 Systém aktualizuje záznamy

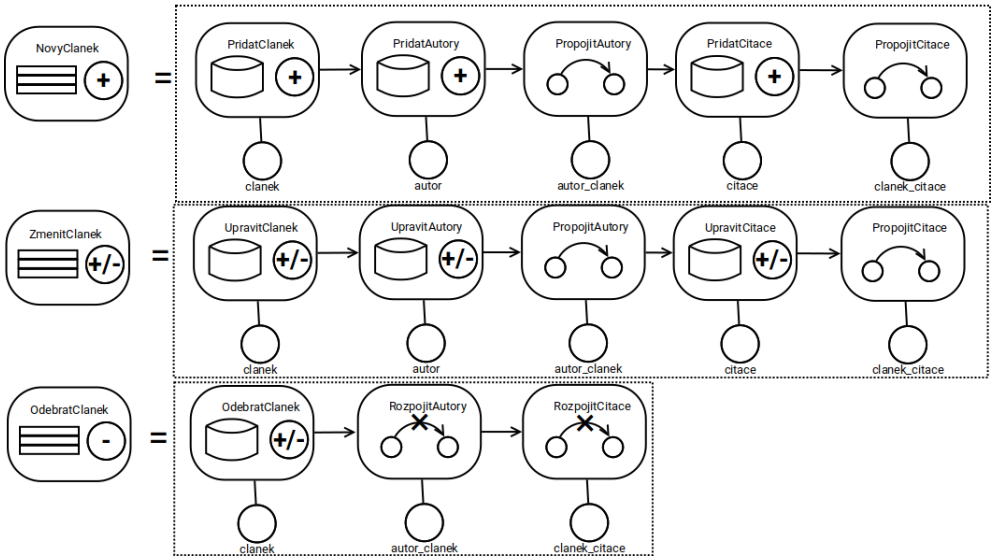
Příloha B: Hypertext model

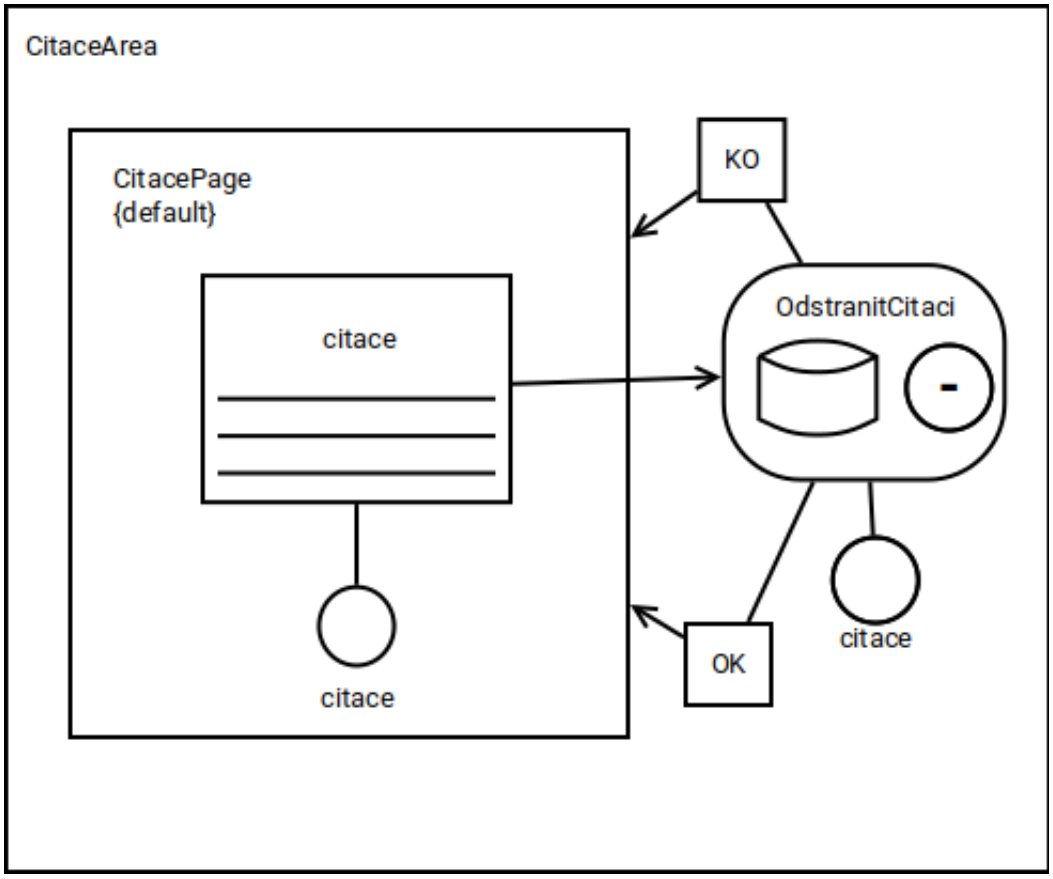


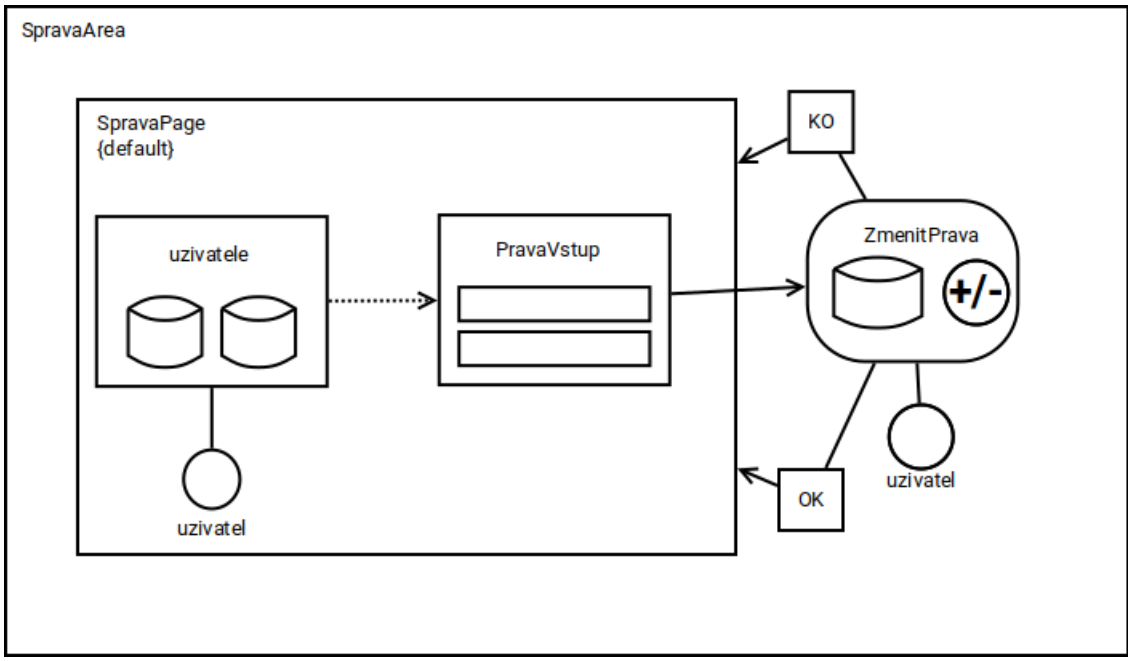

NastaveniArea
Uzivatel
uzvatel_autor
NastaveniPage
{default}
OsUdajeVstup
NoveHesloVstup
KO
OK
KO
OK
KO
UpravitOsUdaje
+/-
autor
ZmenitHeslo
+/-
uzivatel
UzivatelArea
OdhlasitUzivatele
OK
OdstranitUzivatele
-
OdstranitUzivatele
=
OdebratAutora
-
OdebratUzivatele
-
autor
uzivatel


AutoriArea
KO
OdstranitAutora
-
AutoriPage
{default, landmark}
IndexAutori
OK
autor
autor
AutoriEditPage
AutorVstup
UpravitAutora
+/-
pjmeno: jmeno
email: kontakt
institi: insituce
orcid: orcid
OK
KO
autor
<jmeno := pjmeno>
<kontakt := kontakt>
<instituce := instit>
<orcid := orcid>
OK
AutoriCreatePage
PridatAutora
+
AutorVstup
pjmeno: jmeno
email: kontakt
institi: insituce
orcid: orcid
KO
autor
<jmeno := pjmeno>
<kontakt := kontakt>
<instituce := instit>
<orcid := orcid>


ClankyArea
KO
OdebratClanek
-
ClankyPage
{default, landmark}
IndexClanky
OK
clanek
[clanek&autor&citace]
ClankyEditPage
ClanekVstup
ZmenitClanek
+/-
OK
KO
OK
ClankyCreatePage
NovyClanek
+
ClanekVstup
KO


NovyClanek
+
=
PridatClanek
+
PridatAutory
+
PropojitAutory
PridatCitace
+
PropojitCitace
clanek
autor
autor_clanek
citace
clanek_citace
ZmenitClanek
+/-
=
UpravitClanek
+/-
UpravitAutory
+/-
PropojitAutory
UpravitCitace
+/-
PropojitCitace
clanek
autor
autor_clanek
citace
clanek_citace
OdebratClanek
-
=
OdebratClanek
+/-
RozpojitAutory
RozpojitCitace
clanek
autor_clanek
clanek_citace


CitaceArea
CitacePage
{default}
citace
citace
KO
OK
OdstranitCitaci
citace


SpravaArea
SpravaPage
{default}
uzivatele
uzivatel
PravaVstup
KO
ZmenitPrava
+/-
OK
uzivatel

Příloha C: Uživatelský manuál

SLOVJANI.info

časopis pisany v medžuslovjanskom jezyku

e-ISSN 2570-7116
p-ISSN 2570-7108

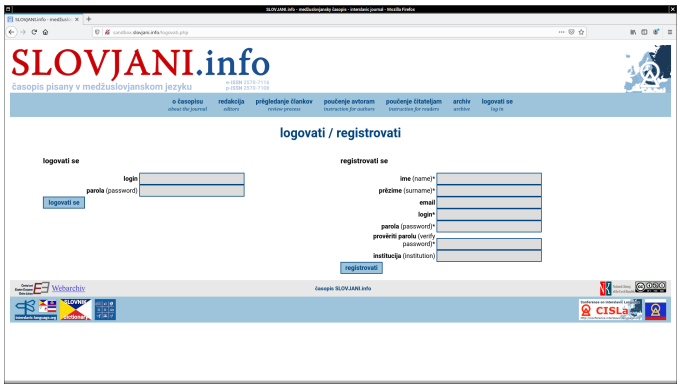
Content management interface
User manual

Contents

1

1 Login & Register

To start working with the system, you first have to log in. This can be done from the login prompt accessed by clicking the *"logovati" (login)* button in the main menu. To login, simply enter your login and password into the *log in* and *parola (password)* fields and click on *logovati se*

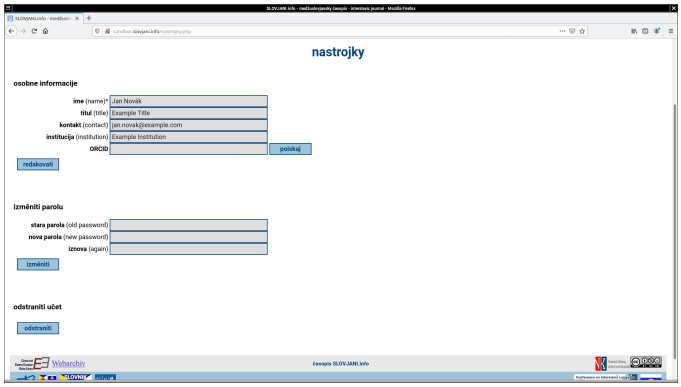


If you were succesfully logged in, and if you were granted administrator rights, you should see a secondary menu underneath the main one. In any way you should see a *nastrojky (user settings) and odlogovati se (log out)* buttons in the main menu.

From the same page new users can also register. This can be done by entering your details into the registration form (at least *ime (name)*, *login* and *password*) and clicking on *registrovati se*. If everything was okay and your chosen login wasn't already taken, a new account will be created for you.

2 User settings

Once you're logged in, you can manage your user settings. This can be done by clicking the *"nastrojky" (user settings)* button in the main menu. From the user settings interface you can change your personal details and the password to your user account. You can also delete your account, if you have any works published under your name in the journal however, your personal details will stay in the system.



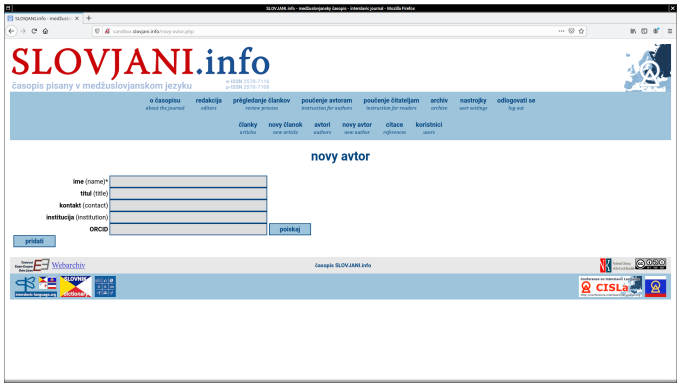
In the *osobne informacije* section you will see a form with your personal details pre-filled. If you at any time wish to change these, simply type in the changes and click on *redakovati* button. You should also notice the *poiskaj* button. If you have an ORCID, you can type it into the corresponding field and click this button to auto-fill the form.

If you want to change your password, you can do so in the *izměniti parolu* section. You need enter your old password, new password and, for confirmation, retype the new password. When you click on *izměniti*, the system will change your password. You can use the new password the next time you log in.

Should you want to delete your account, click on *odstraniti* in the *odstraniti učet* section. You will be asked for confirmation, if you confirm the deletion, you will be logged out and your account will be deleted.

3 Adding an author

To add a new author to the database, select *novy avtor (new author)* from the secondary menu. You will be shown a form simmilar to the one in user settings. Here you can enter information about the author you want to add. Minimum information you must fill in to add a new author is their name.

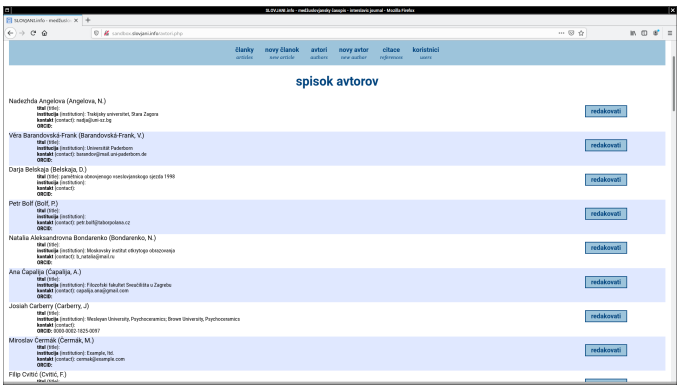


If you know the author's ORCID, you can enter it in the ORCID input field. By clicking the *poiskati* button, the system will attemp to to find and automatically fill in information for you. Note that some information, such as author's e-mail address, are not available.

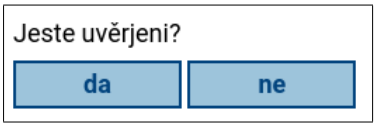
When you're done filling in the info, click *pridati* to add the new author to the database.

4 Modifying an author

To modify info on an existing author, you first need to select which author you want to modify. This is done from the *avtori (authors)* page, which can be accessed from the secondary menu. After locating the author you want to modify, you can either click *redakovati* to modify the info or *odstraniti* to delete the record. *Odstraniti* button will only show up if the author is unused.



After clicking on *odstraniti* you will be asked if you're sure you want to delete the record. Clicking on *da* (yes) will delete the record, while clicking on *ne* (no) will simply return you to the list of authors.



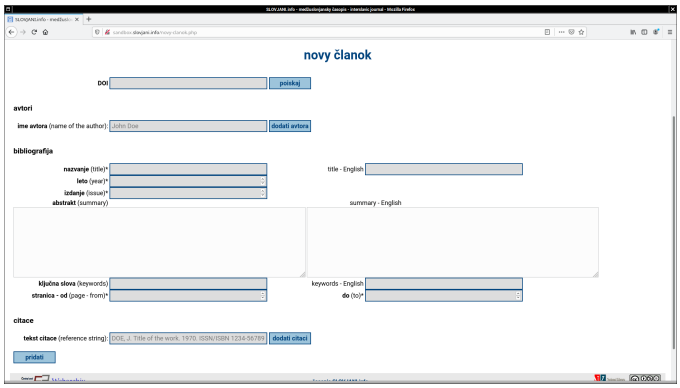
If you selected *redakovati*, you will be presented with the same interface you would be when adding a new author, the form will be pre-filled with information on the author you want to modify. From here you can modify the information. When you're done, click on *redakovati* to store the changes.

5 Adding an article

To add a new article, click on *novy clanok (new article)* in the secondary menu. A form will appear allowing you to add an article. After you've filled in the form, click on *pridati* to add the article to the database.

You can auto-fill this form if you know DOI of the article. Simply enter it into the DOI field (first on the page) and click on *poiskaj*. System will try to find any data on the article and fill the form in for you. Note that some

information, such as a summary or keywords, cannot be filled in this way, as they are not contained in the metadata stored in the DOI public repository.



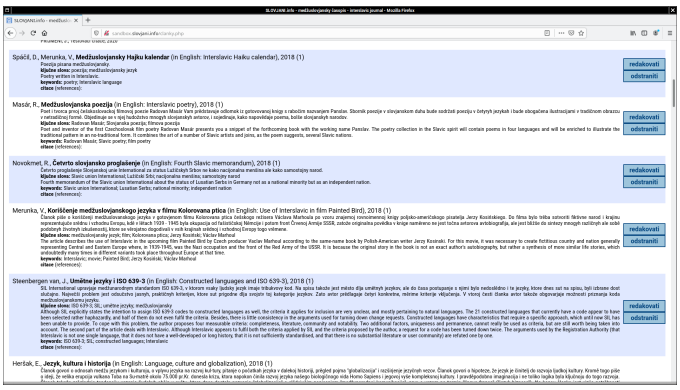
The next section is *autori* (authors). Here you can add the authors of the article. Enter the author's name in the *ime autora* field and click on *dodati avtora*. This will add a new entry to a list of authors. To remove an author, click on *odstraniti* next to the name of the author you want to remove.

The *bibliografija* (bibliography) section serves to enter basic bibliographical information. You can enter title of the article, it's summary, keywords, year of publication, magazine issue and page range. All fields marked with an asterisk are required.

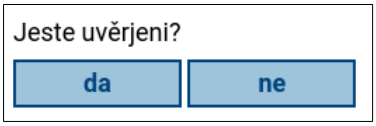
In the *citace* (references) section you can add new references. To add a reference, enter it into the *tekst citace (reference string)* field and click on *dodati citaci*. This will add a new entry to a list of references. To remove a reference, click on *odstraniti* next to the reference you want to remove.

6 Modifying an article

To modify information about an existing article, first go to the article list by clicking on *članky (articles)* in the secondary menu. There you can find the article you want and either edit the bibliography by clicking *redakovati* or delete it by clicking *odstraniti*.



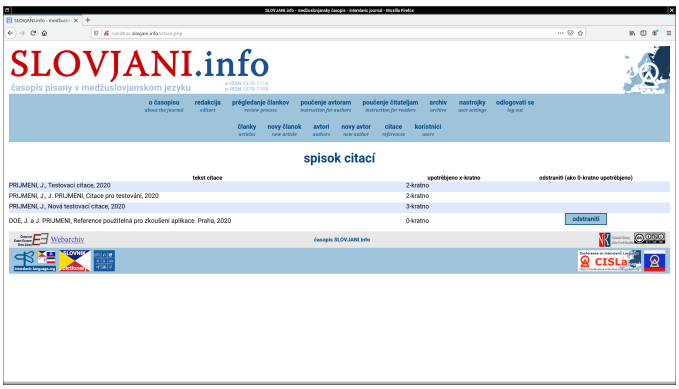
If you click on *odstraniti* you will be asked for confirmation. Clicking on *da* will remove the article from the database, clicking on *ne* will take you back to the list of articles.



Clicking on *redakovati* will take you to edit page. Here you will be presented with a pre-filled version of the form used for adding articles. You can make changes in the same way you would when adding a new article and then save them by clicking the *redakovati* button.

7 Managing references

By clicking on *citace (references)* in the secondary menu you will be taken to the list of references. Here you will be shown a list of all references used in the articles and how many times have they been used (*upotrěbjeno x-kratno*).

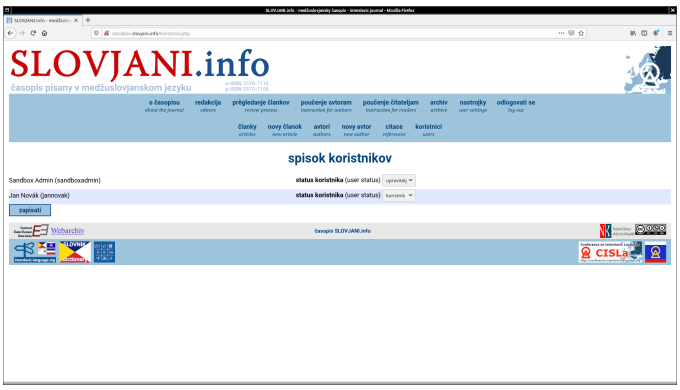


If a reference is currently unused, you can delete it by clicking the *odstraniti* button next to it.

8 Setting user permissions

To set user permissions, first click on the *koristnici (users)* button in the secondary menu. You will be taken to the list of users. Here you will see all users (their names and logins) and their current user status. The user status defines what permissions the user has:

koristnik	regular user, has permissions to download PDF versions of electronic issues of the magazine and to change their user settings
upravitelj	administrator, has the same permissions as regular user and in addition to those permissions to use the management interface to add, modify and delete records



To change status of a user, select new user status from the dropdown and click on the *zapisati* button.