Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Komunitně udržovaná dokumentace pro Common Lisp

2020

Ivo Šmerek

Vedoucí práce: Mgr. Jan Tříska, Ph.D.

Studijní obor: Aplikovaná informatika, prezenční forma

Bibliografické údaje

Autor:	Ivo Šmerek
Název práce:	Komunitně udržovaná dokumentace pro Common Lisp
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2020
Studijní obor:	Aplikovaná informatika, prezenční forma
Vedoucí práce:	Mgr. Jan Tříska, Ph.D.
Počet stran:	21
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Ivo Šmerek
Title:	Documentation for Common Lisp maintained by the community
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2020
Study field:	Applied Computer Science, full-time form
Supervisor:	Mgr. Jan Tříska, Ph.D.
Page count:	21
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Cílem práce je upravit Quickdocs server, který generuje dokumentaci pro Common Lisp knihovny ze zdrojových kódů. Je třeba analyzovat původní řešení, spustit ho na vlastním serveru, výslednou dokumentaci zpřehlednit a implementovat možnost přidávání příkladů použití či komentářů k jednotlivým funkcím či makrům. K přihlašování uživatelů poslouží GitHub OAuth API.

Synopsis

The main aim of this thesis is to edit Quickdocs server, which generates documentation for Common Lisp libraries from the source codes. It is necessary to analyze the original solution, run it on own server, streamline the documentation and implement the functionality of adding examples to functions or macros. GitHub OAuth API will be used to sign-in users.

Klíčová slova: Common Lisp, dokumentace, webový server

Keywords: Common Lisp, documentation, web server

Tímto bych rád upřímně poděkoval Mgr. Janu Třískovi, Ph.D. za odborné vedení, za jeho věcné a přínosné rady a za poznatky, které mi poskytl při zpracování této bakalářské práce.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam zdrojových kódů

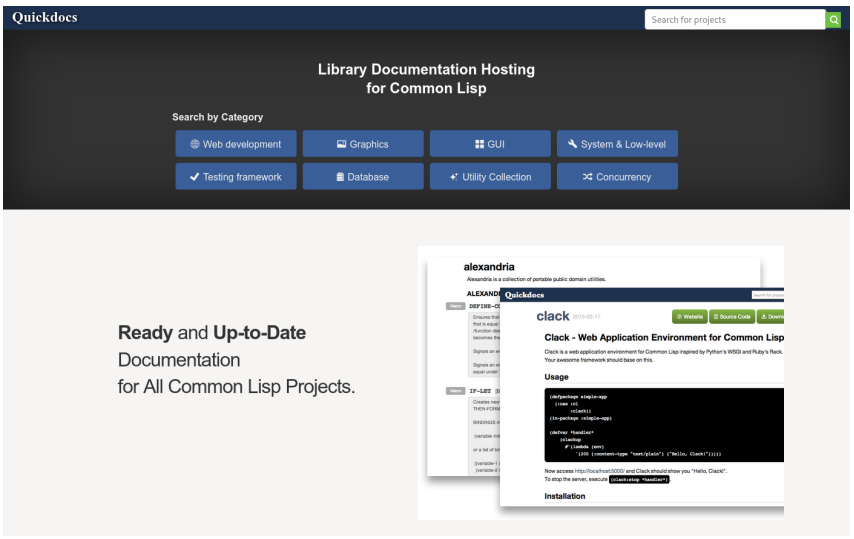
1 Úvod

Přehledná uživatelská dokumentace programovacího jazyka je jeden z důležitých faktorů, které hrají roli v jeho popularitě a použitelnosti v praxi, a je to místo, kde uživatelé hledají informace, jak se dané elementy jazyka chovají a používají. Common Lisp je dynamický jazyk, který klade důraz na modularitu. To znamená, že ve výchozím stavu jazyk obsahuje pouze nejdůležitější elementy pro práci v něm, a pokud uživatelé potřebují pokročilejší funkcionalitu, musejí si stáhnout knihovny z internetu a do jazyka je importovat. To se může jevit jako krajně nepraktické, zvlášť když jsou knihovny pro Common Lisp z velké části vytvořené jednotlivci z komunity, a ti je poskytují ke stažení na vlastních webech.

1.1 Quicklisp a Quickdocs

Za účelem sjednocení těchto knihoven roztroušených různě po internetu vznikl projekt Quicklisp, který poskytuje uživatelům jednoduché rozhraní pro hledání knihoven a jejich následnou instalaci do systému. Quicklisp jako takový neposkytuje žádnou sjednocenou dokumentaci, proto následně vznikl projekt Quickdocs, který přichází s nástroji pro extrakci dokumentace ze zdrojových kódů projektů v Quicklispu a její následné zobrazení na webu.

Cílem mé bakalářské práce je vytvořit webovou aplikaci, která bude vycházet ze zdrojového kódu Quickdocs a přinese vylepšení jako zpřehlednění dokumentace knihoven a API referencí, novou funkcionalitu ve formě přihlašování přes GitHub účet a vytváření uživatelských příkladů k prvkům dané knihovny.



Obrázek 1: Quickdocs

2 Quickdocs nástroje

Následující kapitolu bych rád věnoval krátkému popisu nástrojů, které Quickdocs projekt [1] poskytuje, a stručně vysvětlil jejich účel v celém projektu. Všechny nástroje jsou napsané v Common Lispu a jejich zdrojové kódy jsou dostupné na GitHubu [2].

1. **Parser**

   Quickdocs-parser je nástroj, který umožňuje extrahovat dokumentaci ze systémů Common Lispu. Funguje na základě parsování zdrojového kódu a návratu množiny objektů reprezentující jednotlivé prvky jazyka v daném systému. Slouží jako pomocný nástroj pro quickdocs-extracter.

2. **Serializer**

   Quickdocs-serializer je opět pomocný nástroj pro quickdocs-extracter. Obstarává serializaci extrahovaných objektů, aby následně mohly být uloženy do databáze.

3. **Extracter**

   Quickdocs-extracter využívá parser i serializer a obstarává samotnou extrakci dokumentace z Quicklisp release (vydání). Je možné jej spustit paralelně.

4. **Database**

   Quickdocs-database poskytuje databázové modely a schéma, které Quickdocs využívá. Pomocí toho je třeba vytvořit databázi v systému, na kterém bude celý projekt běžet.

5. **Updater**

   Quickdocs-updater zastřešuje všechny výše zmíněné nástroje. Obstarává extrakci dat, jejich uložení v databázi, stažení metadat z internetu a aktualizace.

6. **Server**

   Quickdocs-server je nástroj, kterým se v této bakalářské práci budu zabývat nejvíce. Zjednodušeně řečeno bychom mohli říci, že se jedná o webový server se šablonovacím systémem a přístupem do databáze, který všechna nashromážděná data poskytuje uživateli skrze webovou aplikaci.

3 Příprava systému a projektu

Vývoj projektu i jeho hostování budu provádět na Linuxovém systému. Proto zde v několika odstavcích popíši, co vše bylo třeba udělat pro to, abych mohl projekt bez problému vyvíjet i spustit.

3.1 Volba implementace Common Lispu

Již delší dobu používám pro vývoj v Common Lispu „**Steel Bank Common Lisp**“ [3] implementaci (dále jen **SBCL**) a i v tomto případě vše funguje bez problémů. Dále je třeba nainstalovat **Quicklisp** [4], protože Quickdocs projekt, ze kterého budu vycházet, je na Quicklispu založený, a používá z něj spoustu knihoven.

3.2 Instalace potřebných programů

SBCL musíme nainstalovat jak na desktop, kde budu projekt vyvíjet, tak na server, kde bude dokončená aplikace později běžet. Ve většině Linuxových systémů je SBCL dostupný přímo v oficiálních repozitářích, takže vše, co je třeba udělat, je napsat jeden terminálový příkaz v závislosti na dané Linuxové distribuci.

Instalace **Quicklispu** se provádí stažením souboru „quicklisp.lisp“ z oficiálních webových stránek a jeho následným načtením v SBCL.

Dále je třeba nainstalovat databázový systém **MySQL**, protože na něm Quickdocs závisí, a ukládají se do něj všechna extrahovaná data, se kterými budeme dále pracovat. MySQL systém je opět dostupný z oficiálních repozitářů.

Nakonec nainstalujeme systém správy verzí **Git**, který nám v kombinaci s GitHubem umožní sledovat provedené změny a jednoduše je přenášet z desktopu na server.

V některých skriptech projekt využívá programy **Ros** a **Pandoc**, tak je třeba ověřit jejich přítomnost v systému a případně je doinstalovat.

```
1  emerge dev-lisp/sbcl
2  emerge dev-db/mysql
3  emerge dev-vcs/git
4  emerge dev-lisp/roswell
5  emerge app-text/pandoc
```

Zdrojový kód 1: Instalace potřebných programů v systému Gentoo

3.3 Vývojové prostředí

Ze všech vývojových prostředí, které jsem vyzkoušel, mi nejvíce vyhovuje textový editor **Atom**, který umožňuje doinstalovat rozšíření pro zvýrazňovaní syntaxe Common Lispu a automatické doplňování závorek a odsazování, dokonce i rozšíření pro vyhodnocování kódu přímo z editoru. Další z předností je integrace programu Git a GitHub API přímo v editoru. To mi umožňuje sledovat změny, které jsem v kódu provedl od poslední stage a umožňuje provádět commity přímo z textového editoru a jejich následný push na GitHub.

3.4 Příprava projektu

Abych mohl provádět změny v kódu a ukládat si je na GitHub, nejjednodušší řešení je vytvořit fork každého GitHub repozitáře daného Quickdocs nástroje, který budu potřebovat. To jsou následující:

1. quickdocs-parser
2. quickdocs-serializer
3. quickdocs-extracter
4. quickdocs-database
5. quickdocs-updater
6. quickdocs-server

Každý vytvořený fork je dále třeba naklonovat do složky, ze které Quicklisp načítá lokální projekty, kterou je v případě Linuxu ~/**quicklisp/local-projects/**.

3.5 Vytvoření databáze

MySQL databázi je třeba vytvořit podle souboru schema.sql přiloženého u Quickdocs-database.

```
1  mysql -u root -e 'CREATE DATABASE quickdocs DEFAULT CHARACTER SET
       utf8'
2  mysql -u root quickdocs < schema.sql
```

Zdrojový kód 2: Vytvoření databáze

4 Analýza a stanovení cílů

Ještě předtím, než jsem začal se samotným vývojem vylepšené webové aplikace, jsem provedl analýzu současné verze Quickdocs-serveru a vytvořil si seznam věcí, které bylo třeba upravit nebo přidat.

4.1 Návrh designu

Stávající vzhled webu je na některých místech nepřehledný a celkově na poměry dnešní doby nedostatečný, tudíž jsem se rozhodl vytvořit nový, moderní design, který bude přehlednější a responzivní. Měl by být příjemný na pohled a práce s ním by měla být intuitivní.

4.2 Zpřehlednění dokumentace

Většina dokumentací obsahuje ukázky kódu jako jsou příklady použití, konfigurace, výstupy a podobně. Současná verze Quickdocs-serveru neimplementuje žádné zvýrazňování syntaxe, což je vlastnost, která výrazně zlepšuje čitelnost a přehlednost kódu.

4.3 Vylepšení API referencí

API reference jsou v současné verzi vyřešeny poměrně nešťastně. Textové popisy jednotlivých symbolů nezobrazují správně nové řádky tak, jak jsou nastaveny ve zdrojovém textu. Dále jsou všechny systémy a balíky vybraného projektu zobrazeny na jedné dlouhé stránce, což prakticky znemožňuje cokoliv efektivně vyhledat. Jedním z cílů bylo přijít s novým, přehlednějším řešením, které by umožňovalo i vyhledávání symbolů v systémech. Inspirací mi byla především webová aplikace ClojureDocs [5].

4.4 Přihlašování uživatelů a vytváření příkladů

Tyto funkce Quickdocs-server vůbec nemá, tudíž jsem si dal za cíl je implementovat. Pro přihlašování uživatelů jsem se rozhodl využít GitHub OAuth API, aby nebylo nutné implementovat registraci a uživatelé k přihlášení do aplikace mohli využít svůj GitHub účet. Toto řešení poskytuje všechny údaje, jejichž sdílení uživatel odsouhlasí. Dále by uživatelé mohli pod svým účtem vytvářet příklady použití k jednotlivým symbolům v API referencích vybrané projektové dokumentace.

5 Implementace

Nyní se budu věnovat samotnému dosažení cílů stanovených v předchozí kapitole. Pokud jsou v systému nainstalovány a nakonfigurovány všechny potřebné programy včetně databáze, jak jsem popsal ve 3. kapitole, měl by jít Quickdocs-server bez problému spustit. Změny provedené v kódu Common Lispu se pak projeví po restartu aplikace.

```
1   (ql:quickload :quickdocs-server)
2   (quickdocs-server:start :address "0.0.0.0")
```

Zdrojový kód 3: Spuštění Quickdocs-serveru v SBCL

Po provedení výše zmíněných příkazů se server spustí na lokální adrese, viditelný pro všechny zařízení v síti na portu, který má výchozí hodnotu 5000. Port lze specifikovat klíčovým slovem :port.

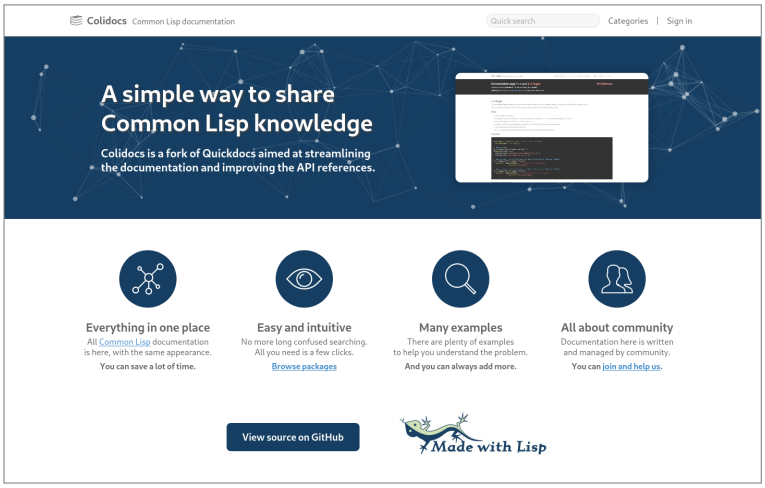
5.1 Design webových stránek

Návrh designu i jeho samotnou implementaci provádím přímo přes HTML a CSS. S tímto přístupem mám nad stránkami největší kontrolu a v kombinaci s frameworkem Djula [6] (Common Lispový port šablonovacího systému Jinja) nemá člověk ani moc na výběr. CSS needituji přímo, ale používám stylovací jazyk SASS [7], který využívá podobnou syntaxi, ale poskytuje uživateli větší svobodu a umožňuje využít prvky, které by v normálním CSS nebylo možné použít, jako jsou například cykly nebo proměnné. SASS kód se s každou změnou převádí na CSS za pomoci watcheru (skript, který hlídá změny v souborech a automaticky spouští náležité programy po každé úpravě).

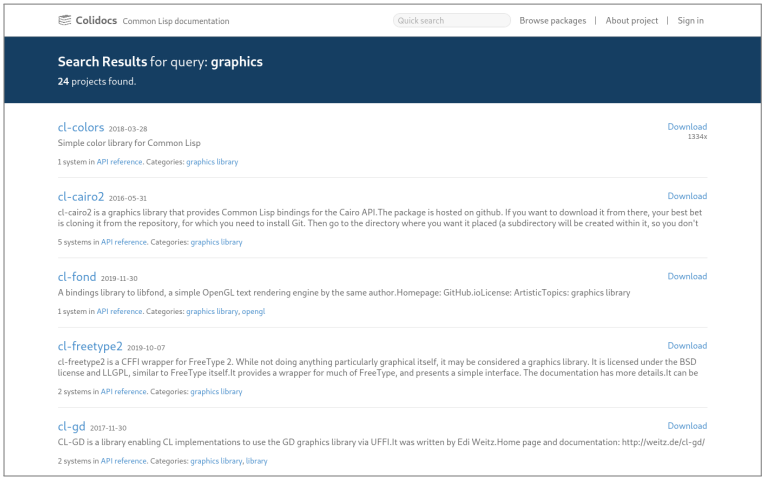
Rozhodl jsem se pro bílé pozadí s modrou a šedou výplňovou barvou. Horní panel, který byl v původním řešení velmi jednoduchý, jsem kompletně přepracoval a obsahuje kromě názvu stránky i vyhledávač a tlačítka rychlé navigace.

Hlavní stránka (obr. 2) se skládá ze dvou vodorovných pruhů. První, tmavý, obsahuje slogan projektu, stručný popis a náhled. Na pozadí jsem použil JavaScriptovou knihovnu particles.js, která vytváří pěkné dynamické pozadí. Ve druhém pruhu jsem vytyčil charakteristické body projektu s jejich krátkým popisem a některými důležitými odkazy.

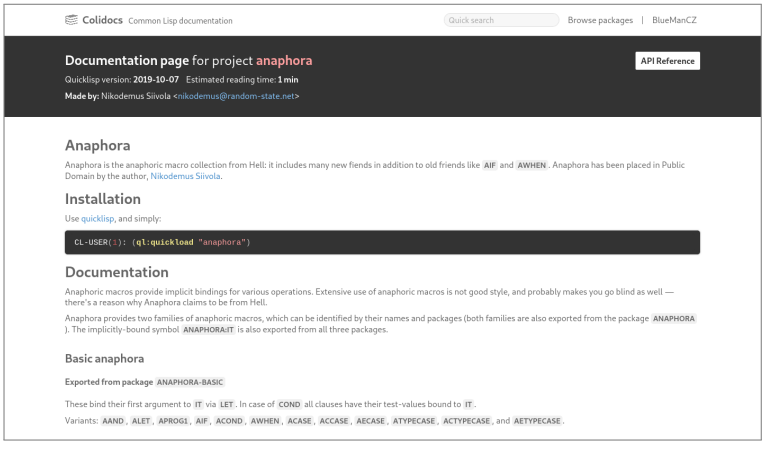
Po zadání výrazu do vyhledávače se uživatel dostane na stránku s výsledky (obr. 3), kde jsem také provedl optimalizace. Je zde přímo odkaz na API reference, jednotlivé kategorie a stažení dané knihovny. Konkrétně odkaz na API reference je v původním řešení až na konci stránky dokumentace, což je velmi neefektivní, pokud uživatel potřebuje něco rychle najít.



Obrázek 2: Hlavní stránka



Obrázek 3: Stránka vyhledávání



Obrázek 4: Stránka dokumentace

Stránku dokumentace (obr. 4) jsem také celou přestyloval. Zde jsem se rozhodl pro tmavě šedou, téměř černou barvu hlavičky stránky a pozadí zdrojových kódů. Vytvořil jsem JavaScriptový algoritmus pro vypočítání přibližné doby čtení na základě počtu slov na stránce a průměrné rychlosti čtení, a tuto informaci zobrazuji v hlavičce stránky. Ve spodní části stránky jsem vytvořil tabulku, kde se nachází informace jako autor, licence, kategorie nebo závislosti.

5.2 Zvýrazňování syntaxe

Ve většině dokumentací projektů se nachází ukázky zdrojových kódů. Ty v tuto chvíli působí docela fádně. Aby se v nich dalo lépe orientovat, rozhodl jsem se implementovat zvýrazňování syntaxe. Musel jsem ale čelit hned několika výzvám. Bylo třeba najít JavaScriptovou knihovnu pro zvýrazňování syntaxe, která podporuje syntaxi Common Lispu. Nakonec jsem narazil na knihovnu code-prettify [8], která má určitou podporu pro Common Lisp, ale jak se později ukázalo, zvýrazňování nebylo tak bohaté, jak bych si představoval. Nicméně mohl jsem vyjít z tohoto řešení a zdrojový kód rozšířit o další pravidla a symboly tak, jak jsem si představoval. Zvýrazňování probíhá na základě vyhledávání výrazů podle regexových vzorů a jejich následné obalení HTML tagem s nastaveným CSS stylem.

Další věcí, kterou bylo třeba vyřešit, je odlišení kódů, které nejsou napsané v Common Lispu, ale jiných jazycích. Například v některých dokumentacích byly ukázky kódu z jazyka C, HTML nebo kódu v shellu. Markdown, kterým je většina dokumentací napsaná, umožňuje specifikovat jazyk daného bloku kódu tak, že se název jazyka napíše za úvodní tři apostrofy. Napsal jsem jednoduchý JavaScriptový kód který, pokud je název jazyka uveden, nastavuje syntaxi podle něj. Pokud není, automaticky předpokládá, že se jedná o Common Lisp.

```
(defclass foo-class ()
  ((a :accessor a :initarg :a)
   (b :accessor b-accessor :initarg :b)))

(let+ (((&slots a (my-b b)) (make-instance 'foo-class :a 1 :b 2)))
  (list a my-b))  ; => (1 2)
```

Obrázek 5: Ukázka zvýrazňování kódu

5.3 Vylepšení API referencí

Zásadní změna, kterou jsem v API referencích provedl, je úprava původního konceptu tak, že se v levém panelu musí vybrat konkrétní balík, který chceme procházet. Tím se podstatně zvýšila přehlednost celé stránky. Dále jsem přidal možnost rozkliknutí daného symbolu, u kterého se následně zobrazí podrobnější informace a uživatelské příklady.

5.4 Přihlašování uživatelů

Pro přihlašování jsem využil GitHub OAuth API [9], které umožňuje autentizaci uživatelů pomocí HTML požadavků a následně poskytne všechny informace, s jejichž sdílením uživatel souhlasí.

V první řadě bylo třeba vytvořit nový záznam OAuth aplikace přímo na GitHubu. Zde se vyplňuje název, domovská URL, popis a callback URL pro přesměrování po přihlášení. Při vytvoření se vygenerují řetězce Client ID a Client Secret, které se dále používají při práci s API.

Dále jsem implementoval komunikaci s GitHub API v programu webového serveru. Ta probíhá ve třech krocích. Nejprve je nutné uživatele přesměrovat na oficiální GitHub OAuth autentizaci s Client ID v HTML parametru a po schválení autentizace GitHub uživatele přesměruje zpět na náš server s dočasným řetězcem „code“ v parametru. Ve druhém kroku je třeba odeslat HTML POST požadavek zpět do GitHub OAuth s tímto dočasným „code“ řetězcem, Client ID a Client Secret v parametrech. Jako odpověď dostaneme uživatelský token, což je opět řetězec, ale tento již můžeme použít přímo pro čtení uživatelských dat. Takže ve třetím kroku odešleme HTML GET požadavek, tentokrát do GitHub API, s upravenou hlavičkou, která obsahuje tento token. Jako odpověď dostaneme data konkrétního uživatele ve formátu JSON, která následně ukládám nebo aktualizuji v databázi a uživatelské session. Pro práci s HTML požadavky používám knihovnu Dexador.

5.5 Vytváření příkladů

Vytváření příkladů je funkcionalita, kterou jsem musel implementovat. Vytvořit nový příklad může pouze přihlášený uživatel a ten zároveň může svůj příklad také odebrat. Použil jsem SimpleMDE Markdown Editor [10], což je jeden z nejpřehlednějších markdown editorů, na které jsem na internetu narazil. Umožňuje základní formátování textu a vkládání zdrojových kódů. Příklady ukládám v databázi jednak v původní markdown podobě, ale zároveň i převedené na HTML spolu s ID uživatele, který je vytvořil a údaji o symbolu, ke kterému se vztahují. Pro uživatele i příklady bylo třeba vytvořit nové tabulky v databázi.

6 Nasazení aplikace na webový server

Spuštění na linuxovém serveru je poslední krok, který je třeba udělat, aby byla aplikace plně použitelná. Do této chvíle jsem aplikaci spouštěl a ladil pouze lokálně na svém počítači.

6.1 Příprava

Nejprve je třeba opět nainstalovat všechny potřebné programy a knihovny, naklonovat repozitáře a vytvořit databázi, jak jsem popsal ve 3. kapitole. Pro vytvoření databáze ale použiji nové schéma, které zahrnuje i dvě nové tabulky pro uživatele a příklady, které jsem musel v rámci implementace přidat. Po instalaci programu Roswell je třeba pro správnou funkčnost vytvořit následující symbolický odkaz:

```
1   ln -s ~/quicklisp/local-projects/ ~/.roswell/local-projects/
```

Zdrojový kód 4: Vytvoření symbolického odkazu na local-projects

6.2 Naplnění databáze

K tomuto účelu slouží právě program **quickdocs-updater**, ve kterém se nachází skript **update-to-latest**, který zajišťuje stažení dokumentace všech projektů, extrakci referencí, stažení informací z CLiki [11] a naplnění databáze těmito daty.

```
1   cd ~/quicklisp/local-projects/quickdocs-updater/scripts/
2   ./update-to-latest
```

Zdrojový kód 5: Spuštění skriptu update-to-latest

6.3 Spuštění aplikace

Aplikaci spouštím v programu screen, který umožňuje funkcionalitu běhu na pozadí i po jeho ukončení a je tak možné se kdykoliv k běžící aplikaci a jejímu výstupu vrátit. Ke spuštění aplikace využívám již dříve zmíněný kód z kapitoly 5, který mám uložený v samostatném souboru, a načítám jej přímo do SBCL.

```
1   screen -dm sbcl --load ~/quicklisp/local-projects/quickdocs-server
        /start.lisp
```

Zdrojový kód 6: Spuštění webového serveru v aplikaci screen

6.4 Aktualizace aplikace

Pokud bych dodatečně provedl změny ve zdrojovém kódu některého z programů nebo upravil webové stránky, je třeba tyto změny reflektovat do dílčích zasažených projektů. K tomu poslouží program Git, který tuto funkcionalitu umožňuje, a aktualizace se provede následujícím příkazem:

```
1  cd ~/quicklisp/local-projects/quickdocs-server/
2  git pull
```

Zdrojový kód 7: Aktualizace projektu quickdocs-server

Závěr

Práce splnila zadaná očekávání. Byla provedena analýza projektu Quickdocs, která byla následně využita jako základ této bakalářské práce a podařilo se vytvořit plně funkční webovou aplikaci, která je přehledná, uživatelsky příjemná a jsou v ní implementovány všechny požadavky ze zadání. Zdrojové kódy se na webu zobrazují se zvýrazněnou syntaxí. Uživatelé se mohou přihlásit přes svůj GitHub účet a následně přidávat a editovat příklady u jednotlivých symbolů v daném balíku.

Práce by šla rozšířit o administrační systém, který by umožňoval nastavit některým uživatelům oprávnění správce, a ti by mohli upravovat dokumentaci nebo revidovat a mazat příklady ostatních uživatelů. Zároveň by bylo možné přidat systém hodnocení nebo hlášení jednotlivých příkladů, a ty posléze revidovat a řadit podle kvality.

Možnost přidávat další knihovny, jak bylo zmíněné v zadání, jsme po konzultaci s vedoucím práce vyloučili, protože řešení by bylo zbytečně komplikované a proti původnímu konceptu. Pro přidání nové knihovny je tedy nutné ji přidat do databáze Quicklispu, a pak bude po aktualizaci dat dostupná.

Conclusions

The thesis has met the expectations. We analyzed the Quickdocs project, used it as the basis of this thesis and managed to create a fully functional web application that is clear, user-friendly, and it implements all the requirements. The source code is displayed on the web with highlighted syntax. Users can add or edit examples of individual symbols in the package after they sign in with their GitHub account.

The work could be extended by an administration system that would allow setting administrator privileges to some users so they could edit or delete documentation or examples. At the same time, it would be possible to add a system for evaluating or reporting individual examples and then revise and sort them according to quality.

After consultation with the supervisor, we decided not to implement the functionality of adding other libraries, since it would be unnecessarily complicated and against the original concept. To add a new library, it is necessary to add it to the Quicklisp database and it would be available in the proposed system after update.

7 Obsah přiloženého CD/DVD

Na samotném konci textu práce je uveden stručný popis obsahu přiloženého CD.

`doc/`
: Text práce ve formátu PDF vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu (v ZIP archivu), tj. zdrojový kód textu, vložené obrázky, apod.

`src/`
: Kompletní zdrojové kódy všech forků Quickdocs nástrojů a webové aplikace. Jejich stručný popis naleznete ve 2. kapitole tohoto textu.

`readme.txt`
: Soubor obsahuje informace o instalaci a spuštění projektu, odkazy na webové adresy, kde je dispozici zdrojový kód i odkaz na server, kde aplikace v současnosti běží. Zároveň obsahuje uživatelský návod, jak webovou aplikaci používat.

Literatura

[1] FUKAMACHI, Eitaro.
Quickdocs, Library Documentation Hosting for Common Lisp [online].
2015 [cit. 2020-5-24]. Dostupný z: ⟨http://quickdocs.org/⟩.

[2] FUKAMACHI, Eitaro. *Quickdocs, Library Documentation Hosting for Common Lisp. GitHub repository.*
Dostupný z: ⟨https://github.com/quickdocs⟩.

[3] AUTOŘI SBCL. *Steel Bank Common Lisp* [online]. [cit. 2020-5-20].
Dostupný z: ⟨http://www.sbcl.org/index.html⟩.

[4] BEANE, Zach. *Quicklisp beta* [online]. 2015-04-30 [cit. 2020-5-24].
Dostupný z: ⟨https://www.quicklisp.org/beta/⟩.

[5] ZACHARY, Kim. *Community-powered documentation and examples repository for the Clojure programming language* [online]. [cit. 2020-5-28].
Dostupný z: ⟨https://clojuredocs.org/⟩.

[6] MONTONE, Mariano.
Djula, port of Python's Django template engine to Common Lisp [online].
[cit. 2020-5-24]. Dostupný z: ⟨https://mmontone.github.io/djula/⟩.

[7] CATLIN, Hampton; WEIZENBAUM, Natalie; EPPSTEIN, Chris;
ANNE, Jina; AND OTHERS. *CSS with superpowers* [online]. [cit. 2020-5-24].
Dostupný z: ⟨https://sass-lang.com/⟩.

[8] VARIOUS CONTRIBUTORS. *Code-pretify, an embeddable script that makes source-code snippets in HTML prettier. GitHub repository.*
Dostupný z: ⟨https://github.com/googlearchive/code-prettify⟩.

[9] GITHUB INC. *Authorizing OAuth Apps* [online]. [cit. 2020-5-20].
Dostupný z: ⟨https://developer.github.com/apps/building-oauth-apps/authorizing-oauth-apps/⟩.

[10] SPARKSUITE. *SimpleMDE Markdown Editor, simple, embeddable, and beautiful JS markdown editor* [online]. [cit. 2020-5-28].
Dostupný z: ⟨https://simplemde.com/⟩.

[11] THE COMMON LISP FOUNDATION. *CLiki, the common lisp wiki* [online].
[cit. 2020-5-24]. Dostupný z: ⟨https://www.cliki.net/⟩.