VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

AUTOMATIZACE SPOLUPRACUJÍCÍCH MODULŮ PRO ONLINE PODNIKÁNÍ

AUTOMATION OF COOPERATING MODULES FOR E-BUSINESS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. PAVEL VAĎURA**
AUTHOR

VEDOUCÍ PRÁCE **doc. RNDr. JITKA KRESLÍKOVÁ, CSc.**
SUPERVISOR

BRNO 2017

Abstrakt

Tato práce pojednává o návrhu a implementaci API rozhraní pro aplikace společnosti IT STUDIO s.r.o. a následně návrhu a realizaci e-commerce kalendáře využívajícího API. V práci je popsán systém World Wide Web a jeho nejdůležitější části včetně možných protokolů pro realizaci rozhraní. Další část práce pojednává o specifikaci, analýze, návrhu rozhraní a návrhu aplikace. V části implementace jsou popsány použité knihovny a zvolená řešení včetně uvedení možných alternativ.

Abstract

This thesis discusses the design of API interface for applications for IT STUDIO s.r.o. and of the e-commerce calendar, using the API. In the introduction, there is described the system of the World Wide Web and its most important parts, including the possible protocols for implementing interfaces. The next part deals with the specification, analysis, interface design and application design. The implementation section describes used libraries and chosen solutions, including the description of possible alternatives.

Klíčová slova

rozhraní, aplikace, WWW, zdroj, HTTP, URL, XML-RPC, SOAP, REST, PHP, MySQL, Nette Framework, Symfony, Doctrine, Bootstrap, jQuery

Keywords

interface, application, WWW, zdroj, HTTP, URL, XML-RPC, SOAP, REST, PHP, MySQL, Nette Framework, Symfony, Doctrine, Bootstrap, jQuery

Citace

VAĎURA, Pavel. *Automatizace spolupracujících modulů pro online podnikání*. Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Kreslíková Jitka.

Automatizace spolupracujících modulů pro online podnikání

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením paní doc. RNDr. Jitky Kreslíkové, CSc. Další informace mi poskytli Václav Vlček a Pavel Kalina. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Pavel Vaďura
17. května 2017

Poděkování

Děkuji vedoucí práce paní doc. RNDr. Jitce Kreslíkové, CSc. za cenné rady a odbornou pomoc.

Obsah

Kapitola 1

Úvod

Tato práce se řeší ve spolupráci se společností IT STUDIO s.r.o., která je poskytovatelem komplexních internetových služeb. Specializací společnosti je vývoj internetových aplikací zaměřených na e-commerce a poskytování konzultací v oblasti internetového marketingu. V současné době společnost nabízí tři systémy pro online podnikání:

- EasyShop – e-shop poskytující specifické zákaznické služby
- WebGet – publikační systém (CMS - content management system) pro správu internetových stránek
- CentralNews – systém pro specializovaný e-mailing a SMS kampaně

Práce se zabývá využitím stávajících firemních systémů a vytvořením zastřešující aplikace, tzv. e-commerce kalendáře. Tento kalendář dokáže zobrazovat události plánovaných akcí stávajících systémů. Dle zákazníka může jít o spojení všech aplikací nebo propojení jen některých systémů, např. EasyShop ⇔ WebGet, EasyShop ⇔ CentralNews, WebGet ⇔ CentralNews. Aby tato aplikace mohla univerzálně pracovat s jakýmkoliv systémem, je nutné před vytvořením kalendáře navrhnout a implementovat API rozhraní (Application Programming Interface) mezi výše zmíněnými systémy, které následně bude kalendář využívat. V praxi by například mohl mít provozovatel na jediném místě kdykoliv přehled o tom, jaké slevové akce nyní v e-shopu probíhají, zda bude automaticky nově zobrazen či naopak skryt nějaký článek, zda je připraveno odesílání obchodních e-mailů ve vztahu k plánovanému zobrazení článků či zda plánované odeslání komerční SMS úzce kopíruje nastavení slev v e-shopu, kam je nakupující zákazník pomocí SMS veden.

Druhá kapitola se vzhledem k povaze systémů, se kterými se bude pracovat (tj. systémy veřejně přístupné na internetu), zaměří na World Wide Web a jeho nejdůležitější a nejpoužívanější části.

Výčet a popis technologií a protokolů vhodných pro realizaci API rozhraní v prostředí WWW bude uveden v kapitole třetí. Budou zde zmíněny protokoly RPC (Remote procedure call), XML-RPC (Extensible Markup Language), SOAP (Simple Object Access Protocol) a REST (Representational state transfer).

Kapitola čtvrtá se zabývá návrhem API rozhraní pro stávající systémy. Požadavkem společnosti je vytvoření rozhraní postaveného na RESTful API, které se liší i přístupem v návrhu. Na základě navrženého API bude v páté kapitole navržena aplikace e-commerce kalendář včetně neformální specifikace, analýzy a návrhu systému.

Před popisem implementace aplikace je popsán výběr vhodné platformy. Výběr je ovlivněn firemními požadavky. Je zde uveden skriptovací jazyk PHP a vybrané softwarové struktury, tzv. framework, vhodné k řešení, databázový systém MySQL, JavaScriptová knihovna jQuery a front-end framework Bootstrap. Framework je softwarová struktura, která slouží jako podpora při programování, vývoji a organizaci jiných softwarových projektů. Může obsahovat podpůrné programy, knihovny API, podporu pro návrhové vzory nebo doporučené postupy při vývoji.

Kapitola sedmá pojednává o implementaci navrženého kalendáře. Je zde zmíněna adresářová struktura respektující zvolený PHP framework, vytvořené komponenty, napojení na rozhraní a layout aplikace, který se dokáže přizpůsobit použitým zařízením pro zobrazení. Dále následuje popis implementace prototypů aplikací EasyShop, WebGet, CentralNews. Prototypy aplikací slouží jen pro demonstraci funkčnosti vytvořeného rozhraní. Důvodem pro nutnost využití prototypů a ne již stávajících aplikací je požadavek na zachování firemního tajemství. Hlavní částí popisovanou u prototypových aplikací je rozhraní REST API a problematika s ním spojená.

Kapitola osmá přibližuje problematiku testování z pohledu vývojáře, ale i postup a výsledky testování na 20 jedincích, kteří následně poskytli zpětnou vazbu.

V předposlední kapitole je popsán způsob instalace aplikací. Dané aplikace jsou zde popsány i z uživatelského hlediska.

Závěrečná desátá kapitola zhodnocuje výsledky aplikace a prototypů s nástinem možného rozšíření kalendáře pro budoucí vývoj.

Diplomová práce navazuje na semestrální projekt, který se zabýval popisem World Wide Web a jeho dílčích částí. Důležitou částí převzatou z projektu je zejména návrh REST API rozhraní a aplikace e-commerce kalendář. Výsledky semestrálního projektu byly využity zejména v kapitolách zabývajících se výběrem vhodné platformy (kapitola č. 6) a implementací (kapitola č. 7).

Kapitola 2

Webové technologie

Tato kapitola popisuje základy vzniku webu (WWW) a na něj navazující technologie zaměřené jak na adresování zdrojů, tak i na samotnou komunikaci účastníků (komunikace typu Request-response).

2.1 World Wide Web

World Wide Web (WWW, W3 nebo web) je systém zahrnující množinu internetových protokolů a dalších technologií. Činností systému WWW je prezentace hypertextových informací. Pomocí hypertextového protokolu zpřístupňují informační zdroje, konkrétně pomocí hypertextových odkazů uvedených v dokumentech (zdrojích). Odkazy, tzv. hyperlinky, mohou vést na další hypertextové dokumenty, prosté textové dokumenty, obrazové a zvukové zdroje atd. Zmíněné vlastnosti hypertextu umožňují rychlé a efektivní procházení velkého množství uložených zdrojů. Hierarchii dokumentů lze popsat pomocí orientovaného grafu. Jedním z důvodů, proč se toto řešení uchytilo i u široké veřejnosti, je fakt, že pro získávání nebo prohlížení informací není potřeba znát žádné příkazy, ale obvykle plně postačí jen klikání myší na odkazy, a tím se uživatelé dopracují k hledané informaci [13].

Historie WWW sahá do nedávné minulosti, kdy roku 1989 Tim Berners-Lee z výzkumného institutu pro jadernou fyziku CERN navrhl daný systém jako prostředek komunikace pro výzkumné pracovníky a pracovní skupiny ze vzájemně odlehlých pracovišť. Prvním webovým prohlížečem, který se rozšířil do celého světa, byl Mosaic [35]. Tento prohlížeč byl vyvinut v národním superpočítačovém centru NCSA (National Center for Supercomputing Applications) roku 1992. Tim Berners-Lee zároveň založil konsorcium World Wide Web Consortium (W3C). Toto mezinárodní konsorcium zajišťuje spolu s veřejností vývoj standardů pro World Wide Web.

Mezi základní prvky webu patří:

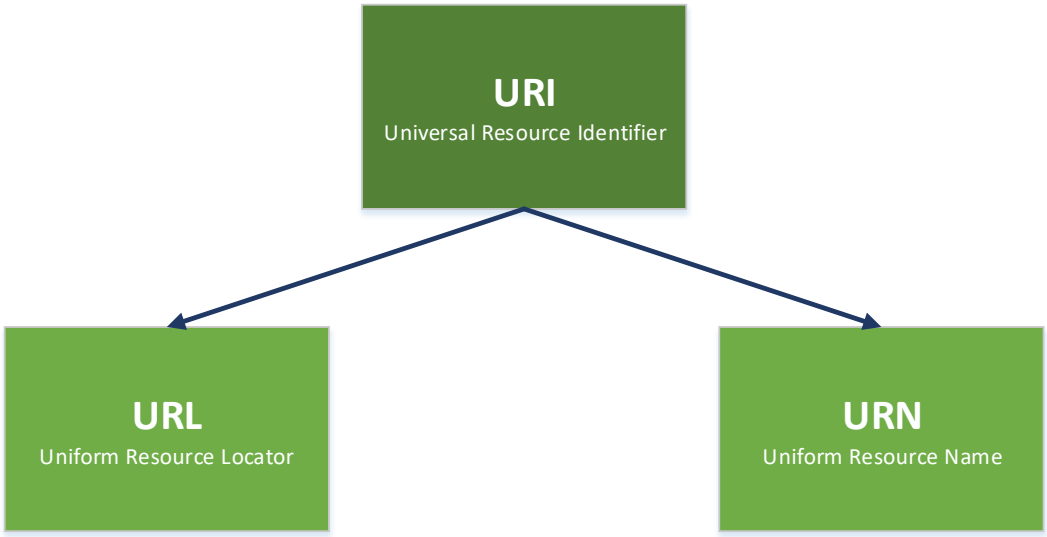
- URL (Uniform Resource Locator) – speciálně vyvinuté adresy pro identifikaci informačních zdrojů.
- HTTP (Hypertext Transfer Protocol) – základní protokol pro komunikaci webových serverů a klientů.
- HTML (Hypertext Markup Language) – jazyk pro sestavování hypertextových webových dokumentů.

2.2 Zdroje - URI, URL, URN

Jak už bylo zmíněno, jednotlivými místy v prostředí sítě internetu jsou zdroje dat. Aby bylo možné se zdroji pracovat, je potřeba je nějakým způsobem identifikovat. Web umožňuje dvě možnosti identifikace zdrojů [12]:

- URL (Uniform Resource Locator) - specifikuje místo v síti.
- URN (Uniform Resource Name) – pojmenování konkrétního zdroje, nezávislé na umístění.

Oba tyto způsoby se společně nazývají identifikace zdrojů, tj. URI (Universal Resource Identifier) [3]. URI tedy souhrnně pojmenovává zdroje, které mohou být identifikovány jako URL (častější způsob) nebo URN, viz obrázek 2.1.



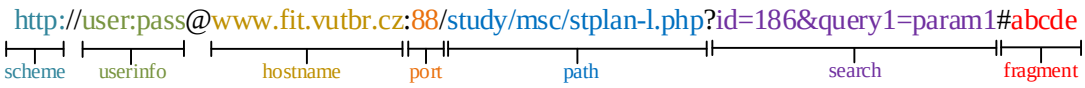
Obrázek 2.1: Vztah identifikace zdrojů URL, URN a URI [inspirováno 4]

2.2.1 URL

URL (Uniform Resource Locator) je adresa, která slouží k přesné specifikaci umístění síťového zdroje nebo jiného prostředku. URL představuje konec šipky orientovaného grafu hypertextového prostředí. Adresy slouží na webu pro propojení s ostatními dokumenty. Výhodou URL adres je možnost odkazovat i na mnohé služby, které existovaly před vytvořením webu, např. soubory načítané protokolem FTP, odkazy na diskusní skupiny Usenet a řadu dalších [2]. V obecném pohledu jsou části URL definovány na obrázku 2.2 následovně:

- **scheme** - Určuje jméno a strukturu protokolu, pro který je zadaná URL adresa. Na základě schématu je následně vybrán konkrétní protokol pro požadovanou komunikaci. Pro kontext této práce budou relevantní pouze protokoly HTTP a HTTPS.

- **Userinfo** - Jedná se o volitelnou část obsahující přihlašovací údaje, pomocí kterých je možné provést přihlášení na server. Tato část se skládá ze složeniny přihlašovacího jména uživatele a hesla odděleného pomocí dvojtečky, nebo lze heslo i vynechat. Z bezpečnostních důvodů není vhodné heslo uvádět, jelikož URL adresa se nijak nezabezpečuje (nedochází k šifrování).
- **hostname** - Jednoznačný identifikátor cílového serveru, na kterém se nachází požadovaný zdroj. Může být vyjádřen pomocí IP adresy nebo symbolicky pomocí doménového jména, kdy se následně očekává převod použitím DNS serveru.
- **Port** - Volitelné číselné označení portu je identifikátor aplikace či procesu na serveru. Nabývá hodnot v rozmezí 1 až 65535. U standardizovaných služeb v případě vynechání portu dochází k automatickému doplňování čísla portu podle uvedeného protokolu. Například protokol HTTP má číslo portu 80 a HTTPS 443.
- **Path** - Nepovinná cesta k souboru na serveru, případně k požadovanému zdroji. Cesta (hierarchická struktura) je oddělena znaky /.
- **Search** - Jedná se o volitelný řetězec parametrů uvedený za otazníkem. Parametry server využívá při dynamickém generování odpovědi, nejčastěji webové stránky. Parametry jsou uvedeny ve formátu nazev=hodnota. Je také možné uvést více parametrů tak, že se oddělí znakem &.
- **fragment** - Označuje identifikátor podřazeného zdroje, který je součástí hlavního zdroje. Identifikátor se neodesílá serveru, slouží nejčastěji jako odkaz na konkrétní prvek (místo) na webové stránce (tzv. kotva).



Obrázek 2.2: Schéma URL adresy [zdroj vlastní]

2.2.2 URN

URN (Uniform Resource Name) slouží k označení zdroje, jenž je nezávislý na umístění. URN popisuje zdroj jménem, nikoliv jeho umístěním (není zaručena exkluzivita). Často se označení používá pro vytváření jmenných prostorů v aplikacích, kde se pracuje s více zdroji [26].

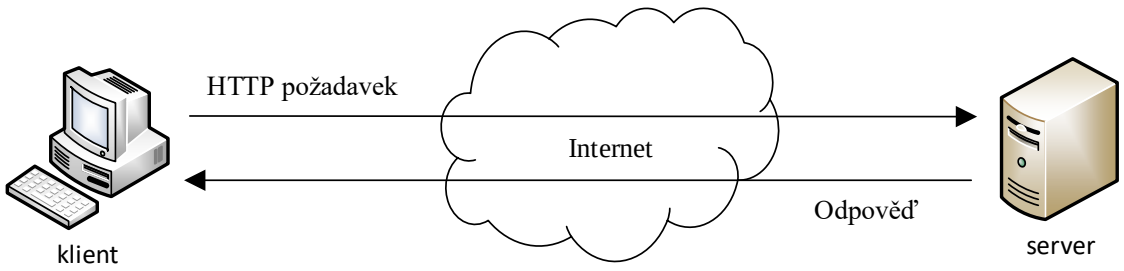
Syntaxe URN je následující: <URN> ::= „urn:“ <NID> „:“ <NSS> „urn:“ – povinný řetězec <NID> identifikátor jmenného prostoru <NSS> specifický řetězec jmenného prostoru.

Existují dvě skupiny identifikátorů, a to neformální a formální. Neformální identifikátor je ve tvaru urn-<číslo>, kdy je číslo přiděleno organizací IANA [9]. Oficiální identifikátory NID existují jako řetězce zapsané v seznamu dostupném na http://www.iana.org/assignments/urn-namespaces/urn-namespaces.xhtml a spravuje je skupina IANA.

2.3 HTTP

Protokol transferu hypertextových informací (HTTP) je aplikační protokol pracující s URL adresami, který je určen pro hypertextová prostředí. Protokol definuje soubor pravidel pro přístup k informacím různého charakteru (textové, zvukové, obrazové) a vytváří z nich pomocí odkazů orientovaný graf. Zásadní vlastností protokolu je jeho anonymita (bezstavovost) a pevně definované příkazy, tzv. metody, viz tabulka 2.2. Další nespornou výhodou protokolu je možnost rozšíření o další formáty dat, a to pomocí standardu MIME (Multipurpose Internet Mail Extensions).

HTTP pracuje na principu požadavek-odpověď (request-response), kdy klient naváže spojení se serverem a pošle mu požadavek. Následně čeká na přijetí odpovědi od serveru a ukončuje spojení, viz obrázek 2.3. Tím, že dojde k ukončení spojení, se ztrácí i stav spojení a další požadavek je novým spojením. Toto platí pro HTTP ve verzi 1.0. Obejít tuto vlastnost je možné pomocí hlavičky Connection a hodnoty Keep-Alive, která říká, po jaké době se spojení má ukončit. To umožní využít jedno spojení pro více požadavků a tím snížit síťový provoz. Od verze HTTP 1.1 je toto chování implicitní.



Obrázek 2.3: HTTP požadavek [zdroj vlastní]

Komunikace mezi klientem a serverem probíhá obvykle protokolem TCP/IP, a to na portu 80, který je v protokolu TCP implicitním protokolem pro aplikační protokol HTTP. Využití protokolu TCP/IP není podmínkou, ale v praxi se jiné řešení ve větší míře nepoužívá [32].

2.3.1 Syntaxe požadavku a odpovědi

Jelikož je HTTP textovým protokolem, tak i dané zprávy (požadavky/odpovědi) jsou v textové podobě, což klade požadavky na syntaktickou analýzu zpráv.

Požadavek (ukázka kódu č. 2.1) zaslaný na server se skládá z několika pevně daných částí. Na prvním řádku je uveden název HTTP metody, identifikace žádaného zdroje a verze protokolu. Protokol obsahuje definované metody uvedené v tabulce 2.2, které indikují sémantiku zpracování požadavku a případných dat. Identifikace zdroje je obsažena v URL adrese za částí port, viz obrázek 2.2. Na dalších řádcích požadavku bývají uvedeny parametry spojení, tzv. hlavičky, které specifikují vlastnosti připojení. Parametry jsou definovány ve formátu Název: hodnota, kdy každý řádek obsahuje právě jeden parametr (některé parametry je možné uvést vícekrát). Určité metody (např. POST) umožňují uvést do požadavku i data určená ke zpracování. Taková data musí být oddělena od parametrů pomocí mezery.

```
GET / HTTP/1.1
Host: www.fit.vutbr.cz
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Chrome/54.0.2840.99 Safari/537.36
Accept: text/html,application/xhtml+xml;q=0.9,*/*;q=0.8
DNT: 1
Accept-Encoding: gzip, deflate, sdch
Accept-Language: cs-CZ,cs;q=0.8
```

Ukázka kódu 2.1: HTTP požadavek

Po obdržení zprávy s daným požadavkem server vytváří zprávu s odpovědí, kterou zasílá zpět. Odpověď se podobně jako požadavek skládá z několika částí, viz ukázka kódu 2.2. Začátek odpovědi obsahuje řádek, tzv. Status-line. Za ním následují jednotlivá pole hlavičky ukončená prázdným řádkem. Za prázdným řádkem jsou již uvedena požadovaná data, tzv. tělo zprávy. Status-line se skládá z definice použitého protokolu a hodnoty stavového kódu [11].

```
HTTP/1.1 200 OK
Date: Mon, 21 Nov 2016 07:23:59 GMT
Server: Apache
Content-Location: index.php.cz
Vary: negotiate,accept-language
TCN: choice
Pragma: no-cache
X-Frame-Options: deny
Keep-Alive: timeout=60, max=85
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-2
Content-Language: cs

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
...
```

Ukázka kódu 2.2: HTTP odpověď

Vybrané ukázky konkrétních stavových kódů:

- **200 Ok** - Požadovaná operace byla v pořádku dokončena.
- **201 Created** – Požadavek typu POST byl zpracován bez chyb.
- **301 Moved Permanently** – Požadovaný zdroj na URL adrese byl přesunut na novou URL adresu (pole Location v odpovědi). Pro všechny další požadavky na daný zdroj je nutno použít novou URL adresu.
- **302 Moved Temporarily** – Požadovaný zdroj na URL adrese byl dočasně přesunut na novou URL adresu (pole Location v odpovědi). Pro všechny další požadavky je možno použít stále původní URL adresu.

Skupina kódů	Význam
1xx	Informační kódy. Jedná se o zprávy definované konkrétní aplikací.
2xx	Kódy označující úspěšné zpracování požadavku.
3xx	Odpověď obsahuje novou adresu, na kterou je potřeba přejít (tzv. přesměrování), aby se získal původní vyžadovaný zdroj. Tuto činnost uživatel často ani nezjistí.
4xx	Chybové kódy označující problém na straně klienta. Nejčastěji se jedná o nesprávně zaslaná data.
5xx	Chybové kódy označující problém na straně serveru.

Tabulka 2.1: Tabulka kategorií stavových kódů HTTP protokolu

- **403 Forbidden** – Server na základě nesprávné autorizace nemůže dokončit požadavek.
- **404 Not Found** – Zadaná URL adresa zdroje je neplatná (na základě adresy nebyl nalezen žádný zdroj).
- **500 Internal Server Error** – Na serveru došlo k nepředpokládané chybě.
- **501 Not Implemented** – Daný požadavek server nepodporuje (není implementován).
- **503 Service Unavailable** – Požadavek není možné zpracovat. Nejčastěji se jedná o případy přetížení nebo údržby serveru.

Metoda	Význam
GET	Základní metoda, která musí být implementována ve všech implementacích protokolu. Metodu zasílá klient za účelem získání požadovaných zdrojů, které server zasílá nazpět.
POST	Metoda pro odesílání dat ke zpracování (data jsou vložena do těla požadavku). Tato metoda může být použita pro tvorbu nebo úpravu zdrojů.
HEAD	Příkaz je podobný příkazu GET. Server nezasílá nazpět data, ale jen metainformace o dané URL adrese. Použití pro testování dostupnosti nebo změn na adrese.
DELETE	Metoda určená k mazání zdrojů. S ohledem na bezpečnostní riziko tohoto příkazu bývá metoda často nepodporována.
PUT	Podobně jako POST slouží k odeslání dat ke zpracování.
OPTIONS	Tato metoda slouží ke zjištění ostatních metod, které daná URL adresa podporuje.
CONNECT	Rezervovaná metoda pro použití s proxy, která vytvoří tunelové spojení dvou bodů. Obvykle se používá pro SSL připojení.
PATCH	Slouží k částečné modifikaci zdroje. Odesílá pouze data, která je třeba změnit, nikoliv celou reprezentaci zdroje.
TRACE	Metoda slouží k testování a diagnostice spojení. Server na tuto metodu odpovídá obsahem obdrženého požadavku, tzn. že odesilatel požadavku zjistí, v jaké podobě server obdržel jeho požadavek.

Tabulka 2.2: Tabulka metod protokolu HTTP

Kapitola 3

Protokoly vhodné pro implementaci API rozhraní

V této kapitole jsou uvedeny tři protokoly, které by bylo možné použít pro implementaci API rozhraní využívající protokol HTTP. Konkrétně se jedná o protokoly XML-RPC, SOAP a REST. U každého protokolu je uvedena základní charakteristika a vlastnosti. Protokol REST je vysvětlen podrobněji, jelikož je zvolen pro následnou implementaci. Důvodem pro výběr daného protokolu jsou jeho vlastnosti, které zvyšují výkonnost systému jak z pohledu velikosti požadavků, tak i systému mezipamětí cache. Další výhodou je fakt, že sémantický význam požadavku obsahuje pouze hlavička HTTP požadavku.

3.1 XML-RPC

Jedná se o protokol, který kombinuje vzdálené volání procedur RPC (remote procedure call) a značkovací jazyk XML [7].

Vzdálené volání procedur je technika pro vytváření standardizované komunikace po síti aplikacím, které jsou rozdílné a mají i jiný účel. RPC bylo navrženo v 80. letech firmou Sun Microsystems. Aplikace pomocí RPC volá proceduru, která je implementována a vykonána na jiném počítači (server) než je volající proces (klient). Programování nevyžaduje znalosti síťové komunikace. Programátor pouze vytvoří klientský program obsahující volání této procedury. Pomocí RPC lokální klientská aplikace volá funkci na vzdáleném serveru podobným způsobem, jako se běžně volá funkce v programu. Serverová aplikace u RPC je program, který obsahuje jednu nebo více procedur. Komunikační protokol RPC přenáší volání těchto funkcí spolu s parametry na vzdálený počítač a nazpět vrací výsledek zpracování [37].

XML-RPC tedy využívá RPC volání, u kterých jsou data a parametry zaobaleny pomocí jazyka XML a dochází ke komunikaci na síti pomocí protokolu HTTP. Jelikož každé volání obsahuje i vrstvu s XML daty, je nutné pro požadavky použít metodu POST. V dnešní době se protokol už nepoužívá, jelikož posloužil jako základ pro protokol SOAP, který ho postupně nahrazuje [8].

3.2 SOAP

SOAP, dříve známý jako Simple Object Access Protocol, vznikl nedlouho po uvedení protokolu XML-RPC. První verze (1.0) protokolu SOAP vznikla na konci roku 1999 jako výsledek

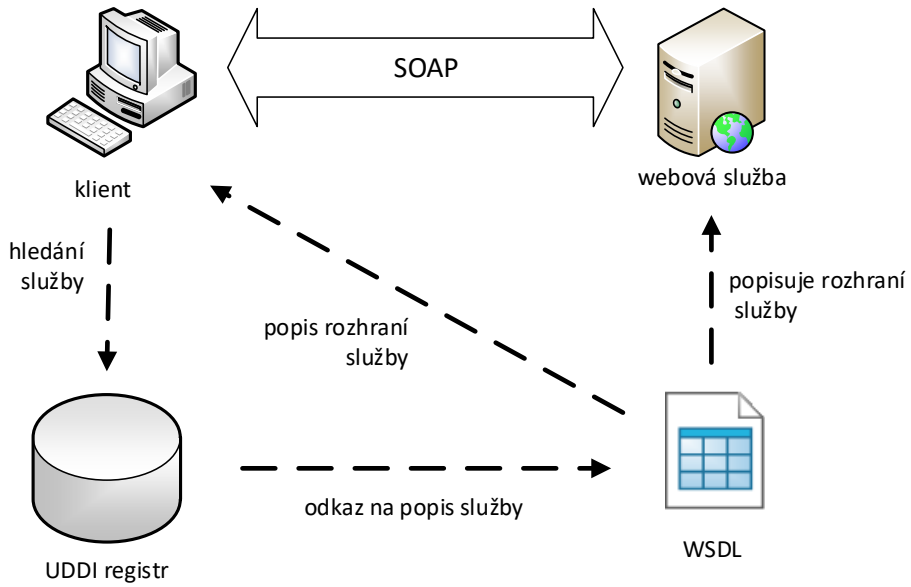
společné práce firem DevelopMentor, Microsoft a UserLand, které chtěly vytvořit protokol pro vzdálené volání procedur (RPC) založený na XML. Protokol navazoval na jednodušší a méně flexibilní protokol XML-RPC. V průběhu roku 2000 se k podpoře přihlásila i firma IBM a nová verze SOAP 1.1 byla zaslána W3C konsorciu [5].

Podobně jako XML-RPC, tak i SOAP využívá značkovací jazyk XML a HTTP. Na rozdíl od XML-RPC ale napravuje spoustu nedostatků. Mezi nedostatky se řadily absence uživatelských datových typů, nedostatečná podpora znakových sad a elementární zabezpečení.

Principem protokolu je zasílání XML zpráv mezi dvěma aplikacemi v modelu požadavek/odpověď. Inicializační aplikace zašle XML zprávu druhé aplikaci, která požadavek zpracuje, vytvoří odpověď a tu zasílá nazpět inicializační aplikaci. Obecně cílovou aplikací bývá webová služba, kterou vyvolá webový server při obdržení SOAP zprávy přes HTTP a předá jí požadavek. Služba se následně postará o vytvoření a zaslání odpovědi klientovi.

Celkově lze SOAP protokol, oproti XML-RPC, označit za flexibilnější protokol podporující mimo jiné jmenné prostory. Z toho plynou i jisté nedostatky v podobě větší složitosti na použití. Složitější jsou přenášené zprávy i následné úsilí na zpracování požadavku.

Největší předností protokolu SOAP bylo spojení se standardy WSDL (Web Services Description Language) a UDDI (Universal Description, Discovery and Integration). WSDL je standardizovaným formátem určeným pro popis rozhraní webové služby. UDDI je mechanismus pro registraci i vyhledávání webových služeb popsaných právě formátem WSDL. Na obrázku 3.1 je znázorněn princip použití těchto technologií. Webová služba se nejprve definuje pomocí popisu WSDL a následně dojde k zaregistrování služby do UDDI registru. Poté může jakýkoliv klient danou službu vyhledat dotazem na UDDI registr, který vrátí popis služby. Tento popis umožňuje automatické vygenerování požadavků SOAP určené ke komunikaci s webovou službou. To znamená, že je přesně dána struktura a cíl umístění odesílané zprávy [15].



Obrázek 3.1: UDDI - princip pro registraci i vyhledávání webových služeb [převzato 23]

3.3 REST

REST (Representational State Transfer) je architektonický styl návrhu rozhraní API určeného pro distribuované prostředí. Tento styl uvedl v roce 2000 Roy Fielding ve své disertační práci *Architectural Styles and Design of Network-base Software Architectures*. Podstatou rozhraní navrženého pomocí REST je jednoduchost, transparentnost a jednoduchá škálovatelnost. REST nezavádí žádné nové protokoly, ale využívá stávající protokol (HTTP) a jeho metod, které určují, co se má stát se zdroji informací.

Aplikační rozhraní se nazývají RESTové (tzv. „RESTful“) v případě, že splňují podmínky, které Fielding definoval následovně:

Architektura klient-server - První podmínkou je použití architektury klient-server. Oddělením uživatelské aplikace a ukládání/správy dat se získá mnoho výhod. Nespornou výhodou je zjednodušení jak uživatelské aplikace, tak i serverové části. Uživatelská aplikace nenese zodpovědnost za správu dat a na druhou stranu serverová část nemusí řešit způsob vykreslení dat. Toto oddělení dále napomáhá k samostatnému vývoji aplikací, který může probíhat paralelně či absolutně na sobě nezávisle. Z výše uvedených výhod plyne i potenciál pro snazší přenositelnost uživatelské aplikace na různé platformy.

Bezstavovost - Komunikace musí být bezstavová. Tuto vlastnost částečně zajišťuje použití architektury klient-server. Nicméně architektura umožňuje tuto vlastnost obejít pomocí SESSIONS, což jsou data uchovávaná na serveru. V REST je bezstavovost chápana striktně, tzn.: aplikační stav si musí udržovat klient na své straně. Tímto omezením ale získáme výhodné vlastnosti:

- **Viditelnost (visibility)** - Systémy, které pracují s příchozím požadavkem (zpracování nebo monitorování požadavku) nemusí řešit žádnou režii navíc. Pouze prozkoumají příchozí požadavek.
- **Spolehlivost (reliability)** - Jelikož požadavek vždy obsahuje všechny potřebné informace, tak se tím zjednodušuje proces zotavení po částečné chybě.
- **Škálovatelnost (scalability)** - Bez potřeby ukládání dat na serveru dochází k úspoře výpočetních prostředků a otevírá se možnost jednodušší paralelizace zařízení.

Tato vlastnost má i nevýhodu v podobě možného nárůstu síťového provozu, kdy se přenášejí pokaždé i informace, které by si mohl server pamatovat.

Cache - Pro podporu efektivního využití síťových prostředků se zavede cache na zdroje. Zdroje jsou označeny implicitně či explicitně pro použití pro cache. Rizikem zde je nevhodné označení zdroje, což může vést k práci s neaktuálními daty.

Jednotné rozhraní - Hlavním rysem architektonického stylu REST je důraz na jednotné rozhraní mezi komponentami. Pro získání jednotného rozhraní je zapotřebí splnit čtyři omezení:

1. **Identifikace zdrojů** - Každý zdroj je určen unikátním identifikátorem, pomocí kterého dochází k získání a následné manipulaci s daným zdrojem.
2. **Manipulace se zdroji** - Obdržení odpovědi, která obsahuje reprezentaci zdroje spolu s metadaty, plně dostačuje pro informování jak s daným zdrojem pracovat.

Element	Příklad použití ve webovém prostředí
zdroj	Zamýšlený konceptuální model hypertextového odkazu
identifikátor zdroje	URL, URN
reprezentace zdroje	HTML, XML, JSON, JPEG
Reprezentace metadat	media type, last-modified time
Zdroj metadat	source link, alternates, vary
control data	if-modified-since, cache-control

Tabulka 3.1: Datové elementy v REST

3. **Samopopisné zprávy (selfdescriptive messages)** - Zprávy, v tomto případě požadavky a odpovědi, musí samy o sobě být dostatečně jednoznačné, aby se dala určit jejich sémantika.
4. **Hypermédia jako stav aplikace** - Hypermédia se neuchovávají na serveru, ale jsou použita pro přenos aktuálního aplikačního stavu.

Vrstvený systém - Pro zlepšení chování při přijetí požadavků se do systému přidají další vrstvy, což umožní vytvořit hierarchickou strukturu vrstev. Tyto vrstvy nevidí za hranice nejbližších vrstev a interakce probíhá výhradně jen se sousedními vrstvami. Použití vrstev může sloužit například k zapouzdření starších služeb, a tím oddělit nové služby původních klientů. Takto škálovaný systém umožní snadnější zavedení tzv. load balancing, což je rozdělení zátěže na více systémů. Nevýhodou vrstveného systému je zvýšení latence při zpracování dat. Tuto nevýhodu se snaží odstraňovat zavedením cache.

Kód na vyžádání (Code on Demand) - Poslední podmínkou je „kód na vyžádání“. REST umožňuje rozšířit funkcionalitu klienta pomocí spustitelného kódu ve formě skriptů a malého programu, tzv. applet, které získá od serveru. Tímto lze snížit počet implementovaných funkcionalit na straně klienta, případně rozšířit klienta o novou funkcionalitu. Výhodou je větší flexibilita systému, nicméně nevýhody tohoto řešení jsou značné, a to v podobě snížení viditelnosti systému. Zejména zde figuruje značné bezpečnostní riziko, kdy není zaručeno, že spouštěný kód nebude útočné povahy. Z výše uvedených důvodů je tato podmínka volitelná a doporučuje se aplikovat jen v případě, kdy výhody značně převýší bezpečnostní rizika.

3.3.1 Reprezentace zdroje

V kapitole 2.1 byl informační zdroj definován jako libovolný logický objekt, se kterým je možné manipulovat pomocí zpřístupněných metod. Takový zdroj je základním elementem návrhového stylu REST. Reprezentací zdroje je myšlena transformace zdroje do zvoleného datového formátu, jenž dokáže uchovat všechny informace o zdroji. V prostředí WWW je každý zdroj identifikován pomocí URI adresy. Daná adresa vede pouze k jedinému zdroji, ale není vyloučen případ, kdy je jeden zdroj přístupný z více URI adres. Dále má každý zdroj svůj vlastní typ, který popisuje jeho sémantický význam (tzv. media type). Typ zdroje specifikuje HTTP hlavička `Content-Type`, která obsahuje MIME typ zaslané odpovědi. V Tabulce 3.1 je uveden souhrn datových elementů.

Kapitola 4

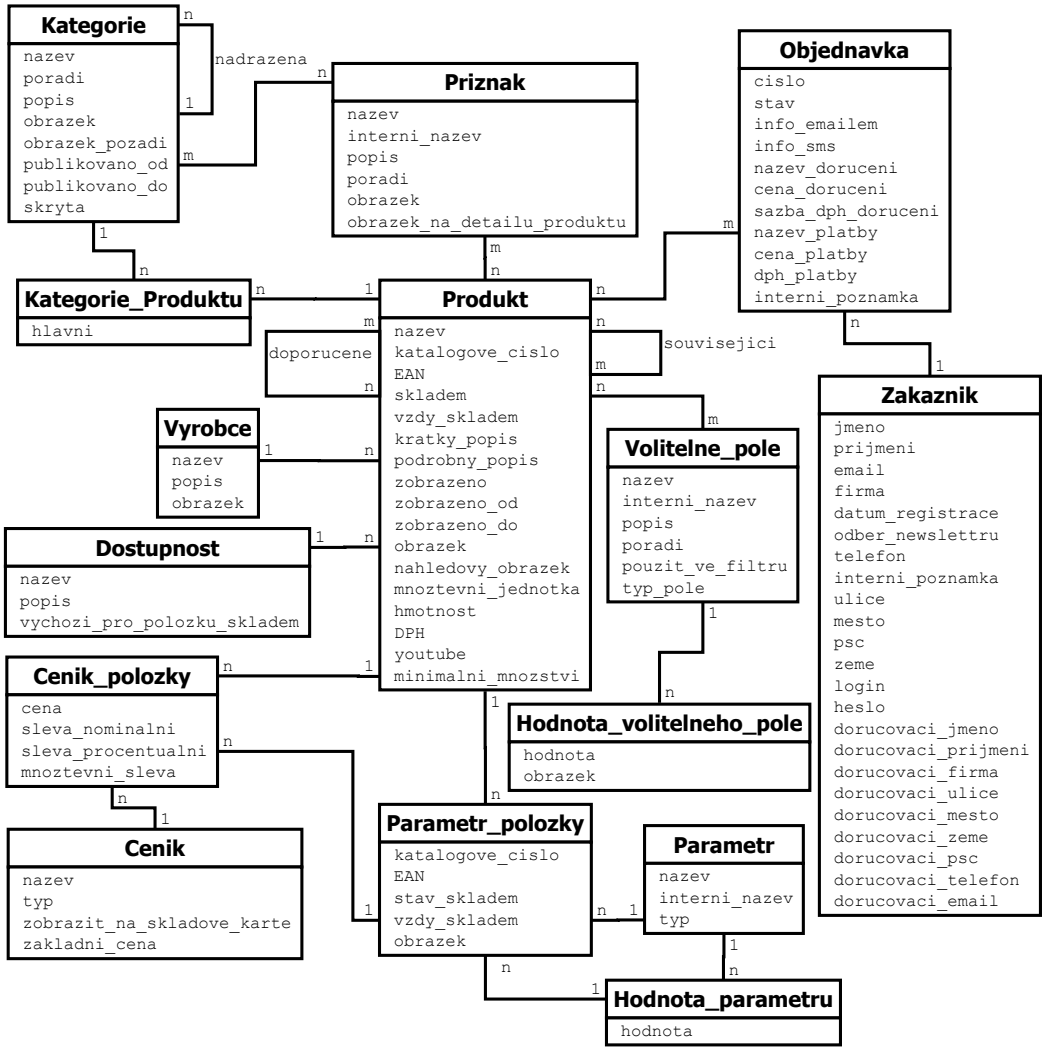
Návrh rozhraní API

Kapitola Návrh rozhraní API představuje podrobněji tři produkty společnosti IT STUDIO. U každého produktu je uveden zjednodušený diagram doménového modelu dané aplikace, který je následně použit pro identifikaci zdrojů a pro návrh unikátních URL adres pro přístup k daným zdrojům.

4.1 EasyShop

EasyShop je aplikace e-shopu, jejíž základ je upravován podle přání a potřeb zákazníků. Na obrázku 4.1 se nachází doménový model aplikace EasyShop, kde jsou uvedeny nejdůležitější komponenty relevantní z pohledu návrhu API rozhraní. Jednotlivé části jsou následující:

- **Produkt** - Nabízený produkt v e-shopu, u kterého jsou evidovány základní informace (název, katalogové číslo, EAN kód atd.). Produkty je možno shlukovat do množin reprezentujících související produkty (např. fotoaparát a jako související položka je výměnný objektiv daného fotoaparátu) a doporučené produkty.
- **Výrobce** - Každý produkt má přiřazeného konkrétního výrobce. Tím lze provádět filtraci produktů na základě vybraného výrobce.
- **Dostupnost** - U produktu je možno definovat dostupnost produktu (např. skladem, na objednání atd.). Tato vlastnost slouží nejen pro zákazníky e-shopu, ale využívá se i pro zdrojové soubory, ze kterých načítají informace porovnávače zboží.
- **Volitelné pole** - Definuje variantu produktu, která nemá vliv na cenu. Na základě volitelného pole je možno dané produkty třídit a filtrovat např. podle barvy nebo velikosti.
- **Parametr položky** - Obdobná vlastnost jako volitelné pole, ale s tou odlišností, že každý parametr disponuje vlastní cenou, která není závislá na ceně produktu, např. délka. Jedná se o spojení ceníku položky, produktu a konkrétního parametru.
- **Parametr** - Seskupuje možné hodnoty parametru, kterých položka nabývá. Obsahuje název parametru a typ, tj. jakou formou se daný parametr bude reprezentovat v e-shopu.
- **Hodnota parametru** - Určuje konkrétní hodnotu parametru.



Obrázek 4.1: Doménový model aplikace EasyShop [zdroj vlastní]

- **Ceník** - Soupis použitých ceníků zboží v e-shopu, např. velkoodběratelé používají jiný ceník než obyčejný zákazník.
- **Ceník položky** - Definice konkrétní ceny a případné slevy produktu/parametru produktu vztažené ke konkrétnímu ceníku. Dochází ke spojení produktu/parametru produktu a konkrétního ceníku.
- **Objednávka** - Každá objednávka v systému nese informace o zákazníkovi, zvoleném typu dopravy a platby, objednaných produktech, aktuálním stavu objednávky a interní poznámce.
- **Zákazník** - Každému zákazníkovi, který nakoupí nebo se zaregistruje v e-shopu, je vytvořeno konto. U tohoto konta jsou evidovány informace o zákazníkovi - dodací,

Entita	Šablona URL identifikující zdroj	Šablona URL kolekce
produkt	/products/{id}	/products/
kategorie	/categories/{id}	/categories/
výrobce	/manufacturers/{id}	/manufacturers/
volitelná pole	/optional-fields/{id}	/optional-fields/
hodnota volitelného pole	/optional-field-values/{id}	/optional-field-values/
parametr položky	/product-params/{id}	/product-params/
parametr	/params/{id}	/params/
hodnota parametru	/param-values/	/param-values/
příznak	/tags/{id}	/tags/
zákazník	/user/{id}	/users/
objednávka	/orders/{id}	/orders/
dostupnosti	/availabilities/{id}	/availabilities/
ceník produktu	/product-price-lists/{id}	/product-price-lists/
ceník	/price-lists/{id}	/price-lists/

Tabulka 4.1: Šablony URL adres rozhraní aplikace EasyShop

případně i fakturační adresa, přihlašovací údaje a souhlas s odběrem novinek z e-shopu.

- **Kategorie** - Pro organizaci produktu je použito kategorií, kdy produkt náleží do tzv. hlavní kategorie a další vazby na produkt jsou chápany jako zařazení do vedlejších kategorií.
- **Kategorie produktu** - Vytváří spojení mezi kategorií a produktem a zároveň určuje, zda dané spojení reprezentuje zařazení do hlavní nebo vedlejší kategorie.
- **Příznak** - U kategorií i produktů je možné uvést příznak, tj. krátká informace upozorňující na specifickou vlastnost, např. novinka.

Na základě analýzy doménového modelu, kdy se hledají hlavní entity, které mohou existovat samostatně, byly vybrány následující entity:

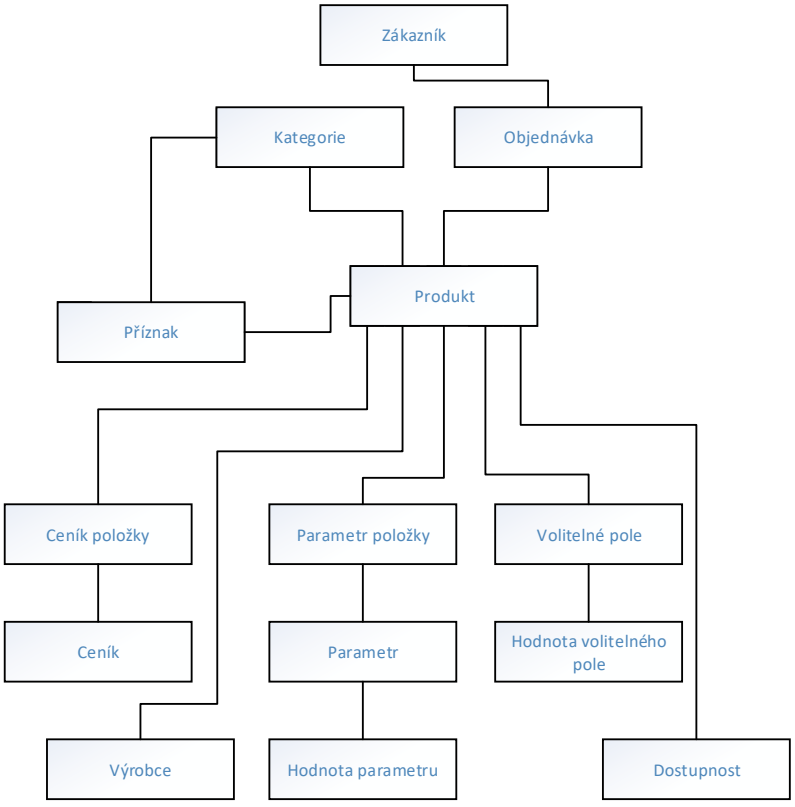
- produkt
- kategorie
- výrobce
- volitelné pole
- parametr položky
- parametr
- hodnota parametru
- příznak
- zákazník

- objednávka
- dostupnost
- ceník produktu
- ceník

Tyto entity lze unikátně identifikovat, a tudíž jim lze přiřadit konkrétní URL adresu. V tabulce 4.1 jsou uvedeny přiřazené URL šablony adres směřujících na konkrétní zdroj a URL šablony adres kolekcí. Pro pojmenování zdrojů byl zvolen systém použití anglických podstatných jmen v jednotném čísle. V případě, kdy se jedná o konkrétní zdroj upřesněný pomocí identifikátoru URL, končí adresa bez lomítka. Pro kolekce byla zvolena podstatná jména v množném čísle a URL adresy končí zpětným lomítkem (/), což jednoznačně značí, že se jedná o více položek [36].

Šablona URL je způsob jak vyjádřit obecný formát URL adresy s možností pozdějšího dosazení konkrétních hodnot. Místa v šabloně, která mají být nahrazena, budou uvedena ve složených závorkách, např. /product/{id}.

Kolekce jsou množiny zdrojů stejného typu tematicky sloučené pro hromadný přístup.



Obrázek 4.2: Hierarchická struktura zdrojů aplikace EasyShop [zdroj vlastní]

zdroj	Šablona URL kolekce	Šablona URL konkrétního zdroje
products	/categories/{id}/products/	/categories/{id}/products/{id}
products	/orders/{id}/products/	/orders/{id}/products/{id}
param	/products/{id}/param/	/products/{id}/parameters/{id}
param-values	/products/{id}/param/{id}/param-values	/products/{id}/param/{id}/param-values/{id}
optional-fields	/products/{id}/optional-fields	/products/{id}/optional-fields/{id}
optional-field-values	/products/{id}/optional-fields{id}/optional-field-values	/products/{id}/optional-fields/{id}/optional-field-values/{id}
tags	/products/{id}/tags/	/products/{id}/tags/{id}
tags	/categories/{id}/tags/	/categories/{id}/tags/{id}
availabilities	/products/{id}/availabilities	/products/{id}/availabilities/{id}
product-price-lists	/producst/{id}/product-price-lists	/products/{id}/product-price-lists/{id}
manufacturers	/products/{id}/manufacturers/	/products/{id}/manufacturers/{id}

Tabulka 4.2: Šablony URL adres rozhraní aplikace EasyShop

4.1.1 Hierarchie zdrojů

Zvolené zdroje aplikace utvářejí strukturu, která není plochá, jelikož její části mají mezi sebou vztahy. Tyto vztahy vyjadřují vlastnictví nebo náležitost k jinému zdroji. Popsaná struktura je znázorněna na obrázku 4.2. Z této struktury vyplývá, že produkt spadá pod kategorii nebo objednávku, kterou navíc vždy uskutečnil zákazník. Produkt dále zastřešuje informace o výrobci, parametrech, volitelných polích, dostupnosti a ceníku položky. Příznak dle struktury náleží kategorii i produktu. Zařazení zdroje na konkrétní úroveň hierarchické struktury neznamená, že se nemůže vyskytovat i v jiných úrovních. Každý zdroj se může nacházet na více logických úrovních podle daného kontextu, který je aktuálně používán, např. všechna volitelná pole konkrétního produktu vs. všechny produkty obsahující volitelné pole. Tabulka 4.2 obsahuje přehled URL, na kterých jsou dostupné jednotlivé kolekce a podřazené zdroje.

4.2 WebGet

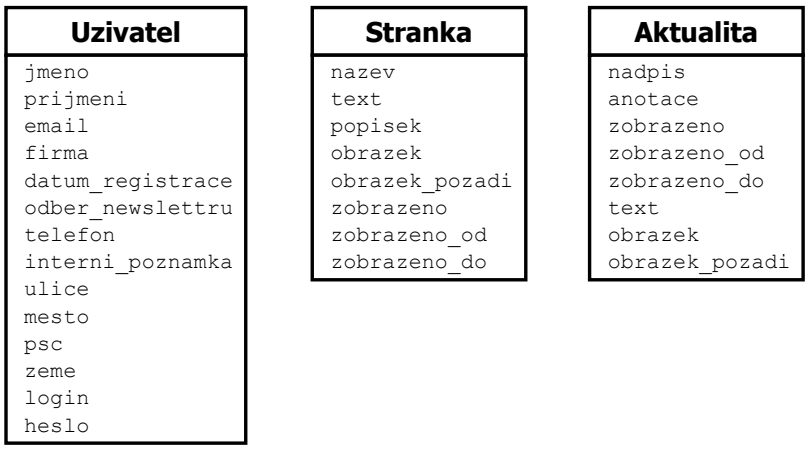
WebGet je publikační systém (CMS) pro jednoduchou správu WWW stránek a aktualit s možností individuální úpravy. Tato aplikace je zaměřena na prezentaci firmy. Na obrázku 4.4 je znázorněn doménový diagram. V tomto diagramu nejsou zobrazeny nepotřebné domény, které jsou z principu nevhodné pro zpřístupnění pomocí API, např. správci mající přístup do administrace aplikace.

Doménový model je složen ze tří částí:

Entita	Šablona URL identifikující zdroj	Šablona URL kolekce
stránky	/pages/id	/pages/
aktuality	/news/id	/news/
uživatelé	/users/id	/users/

Tabulka 4.3: Šablony URL adres rozhraní aplikace WebGet

- **Stránka** - Aplikace umožňuje definovat stránku včetně názvu, obsahu, obrázků, popisků a zobrazení.
- **Aktualita** - Zastupuje krátkou informativní zprávu o nějaké novince. Atributy aktuality jsou podobné jako u stránky, ale s tím rozdílem, že aktuality se navíc zobrazují na společném místě (v přehledu aktualit).
- **Uživatel** - Obsahuje informace o uživatelích systému, kteří po přihlášení v aplikaci mohou přistupovat k neveřejným stránkám a aktualitám, jenž by jinak byly pro veřejnost skryté.



Obrázek 4.3: Doménový model aplikace WebGet [zdroj vlastní]

4.3 CentralNews

Nástroj CentralNews umožňuje propojení a automatizaci jednotlivých obchodních e-mailů (Newsletter, Shopletter) a SMS kampaní.

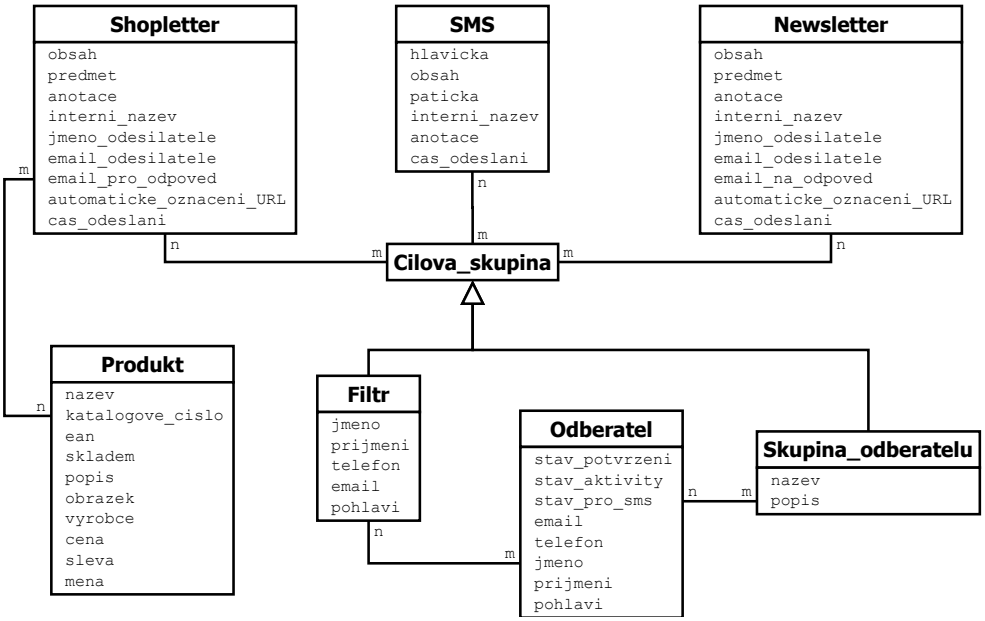
Doménový model se skládá z následujících částí:

- **Shopletter** - Zpravodaje a letáky zasílané v elektronické podobě.
- **Newsletter** - Specializované prodejní e-maily.

Entita	Šablona URL identifikující zdroj	Šablona URL kolekce
shopletter	/shopletters/id	/shopletters/
newsletter	/newsletters/id	/newsletters/
sms	/smss/id	/smss/
skupina	/subscriber-groups/id	/subscriber-groups/
odběratel	/subscribers/id	/subscribers/
filtr	/subscriber-filters/id	/subscriber-filters/

Tabulka 4.4: Šablony URL adres rozhraní aplikace CentralNews

- **SMS** - SMS zprávy s podobným zaměřením jako obchodní e-maily.
- **Filtr** - Množina odběratelů získaná nějakým filtrovacím pravidlem, např. podle pohlaví.
- **Skupina odběratelů** - Množina odběratelů vytvořená uživatelem, kterou není možné vytvořit pomocí žádného filtrovacího pravidla.



Obrázek 4.4: Doménový model aplikace CentralNews [zdroj vlastní]

Kapitola 5

Návrh e-commerce kalendáře

Tato kapitola se zabývá návrhem aplikace e-commerce kalendáře. Je zde uvedena neformální specifikace, ze které vychází diagram případů užití a jejich vybrané detaily. Závěrem je popsán vytvořený návrh databáze a systémový diagram komunikace jednotlivých aplikací na vybraný případ užití.

5.1 Neformální specifikace

Webová aplikace kalendáře dokáže zobrazovat události plánovaných akcí napojených systémů WebGet, EasyShop a CentralNews. Události budou obsahovat informace o produktech, kategoriích, stránkách, aktualitách, obchodních e-mailech (Shopletter a Newsletter) a SMS zprávách. Tyto události se budou z uvedených aplikací automaticky načítat nezávisle na běhu aplikace. Zároveň v případě, kdy dojde k úpravě nějaké události, dojde opět k automatickému přenesení změn do původní aplikace. Kalendář bude vyžadovat autentizaci z důvodu ochrany obchodního tajemství. Kalendář umožní přístup ke konkrétní události pomocí odkazu, který uživatele přesměruje do dané aplikace, a tím mu nabídne správu dané události. Aplikace dovolí vkládat do kalendáře úkoly. Tyto úkoly bude možno přidávat, upravovat, mazat a dokončovat. Vytvořený úkol bude možné přiřadit i jinému uživateli. Tím bude umožněno úkolovat ostatní uživatele (vztah nadřízený-podřízený).

5.2 Analýza

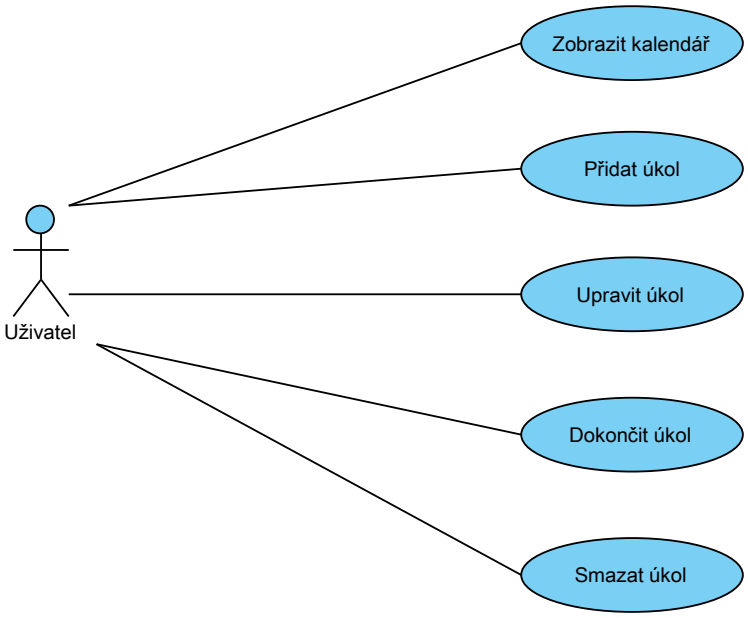
Analýza vychází z neformální specifikace, podle níž by měla navržená aplikace uvažovat jen jeden typ role uživatele. Navržený diagram případů použití je zobrazen na obrázku 5.1

5.2.1 Detaily případu užití

V následujících tabulkách 5.1 - 5.5 je uveden reprezentativní vzorek detailů případů použití.

5.3 Databáze

Návrh struktury databáze vychází z analýzy neformální specifikace. Závěrem této analýzy je nutnost ukládat informace o uživatelích a jejich úkolech. Dále bude nutno uchovávat data o zdrojích s významnou událostí. Z aplikace WebGet se budou ukládat stránky a aktuality.



Obrázek 5.1: Diagram případů použití [zdroj vlastní]

Aplikace CentralNews poskytuje zdroje SMS, Shopletter a Newsletter. Nakonec aplikace EasyShop poskytne informace o produktech a kategoriích.

Z výše uvedených požadavků je navrženo sedm databázových tabulek, které zajišťují uchování všech potřebných informací. Tyto tabulky jsou zobrazeny v návrhu schématu databáze, viz obrázek 5.3.

Na obrázku 5.2 je uvedený systémový sekvenční diagram popisující případ užití „zobrazit kalendář“ z pohledu komunikace aplikací. Kalendář v případě obdržení požadavku pro zobrazení kalendáře v daném časovém intervalu zahajuje proces získání dat od ostatních aplikací. Postupně se dotazuje všech napojených aplikací na potřebná data, která uloží do databáze. Po obdržení všech odpovědí aplikace kalendář zobrazí uživateli kalendář s událostmi v daném časovém intervalu.

Případ použití:	Zobrazit kalendář	**ID:**	1
Stručný popis:	Uživatel zobrazí události kalendáře v daném období.		
Primární aktéři:	Uživatel		
Předpoklady:	1. Uživatel je přihlášen do systému.		
Následné podmínky:	1. Systém zobrazí všechny události v daném období.		
Hlavní tok:	1. Uživatel přejde na hlavní obrazovku. 2. Uživatel vybere zobrazované období nebo ponechá přednastavené. 3. Systém zobrazí události v daném období.		
Výjimky:	Selhání systému, Storno		
Frekvence:	Několikrát za den.		

Tabulka 5.1: Případ použití zobrazit kalendář

Případ použití:	Přidat úkol	**ID:**	2
Stručný popis:	Uživatel vytvoří nový úkol.		
Primární aktéři:	Uživatel		
Předpoklady:	1. Uživatel je přihlášen do systému.		
Následné podmínky:	1. Systém uloží vytvořený úkol.		
Hlavní tok:	1. Uživatel vybere „Nový úkol“. 2. Uživatel vyplní formulář. 3. Uživatel potvrdí vstupní data tlačítkem „Vytvořit“. 4. Pokud vstupní data nejsou validní, systém na dané chyby upozorní a vyzve uživatele k jejich napravení. V případě, že je vše v pořádku, přejde se na krok 5. 5. Systém vytvoří nového uživatele spolu s přihlašovacími údaji.		
Výjimky:	Selhání systému, Storno		
Frekvence:	Několikrát do týdne.		

Tabulka 5.2: Případ použití přidat úkol

Případ použití:	Dokončit úkol	**ID:**	3
Stručný popis:	Uživatel označí úkol jako splněný.		
Primární aktéři:	Uživatel		
Předpoklady:	1. Uživatel je přihlášen do systému.		
Následné podmínky:	1. Systém zaznamená splnění úkolu.		
Hlavní tok:	1. Uživatel vybere konkrétní úkol. 2. Uživatel zvolí „Dokončit úkol“. 3. Systém zaznamená splnění úkolu.		
Výjimky:	Selhání systému, Storno		
Frekvence:	Několikrát do týdne.		

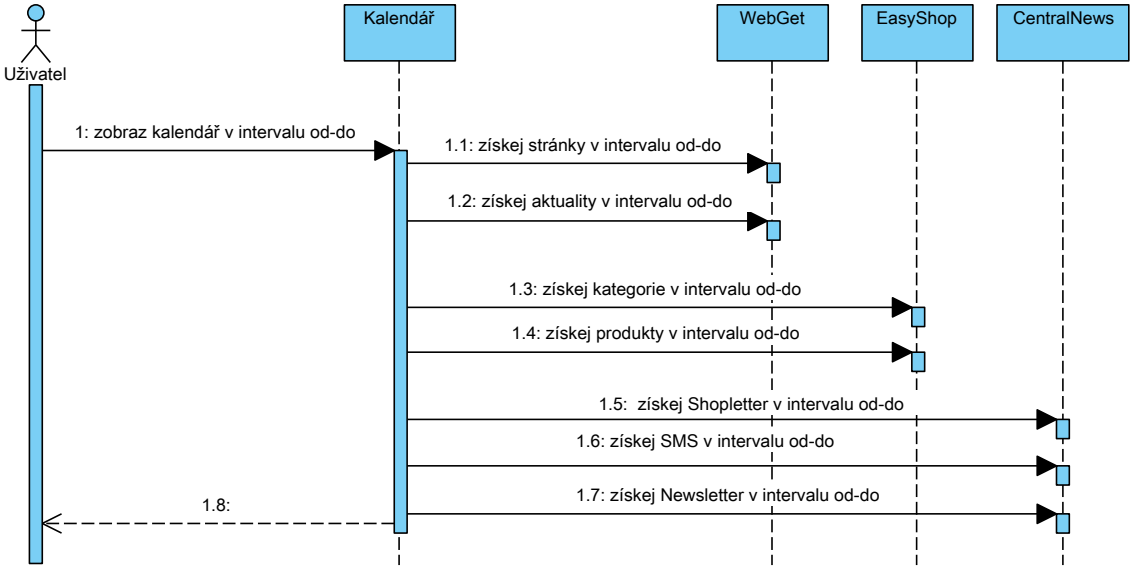
Tabulka 5.3: Případ použití dokončit úkol

Případ použití:	Selhání systému	**ID:**	E.1
Stručný popis:	V systému se vyskytne chyba a není možné dokončit operaci.		
Primární aktéři:	Dle daného případu použití.		
Předpoklady:	1. V systému se vyskytla chyba.		
Následné podmínky:	1. V systému nebyly provedeny změny.		
Tok:	1. Tok selhání systému je vyvolán během provádění hlavního toku. 2. Systém informuje uživatele o chybě.		
Frekvence:	Výjimečně		

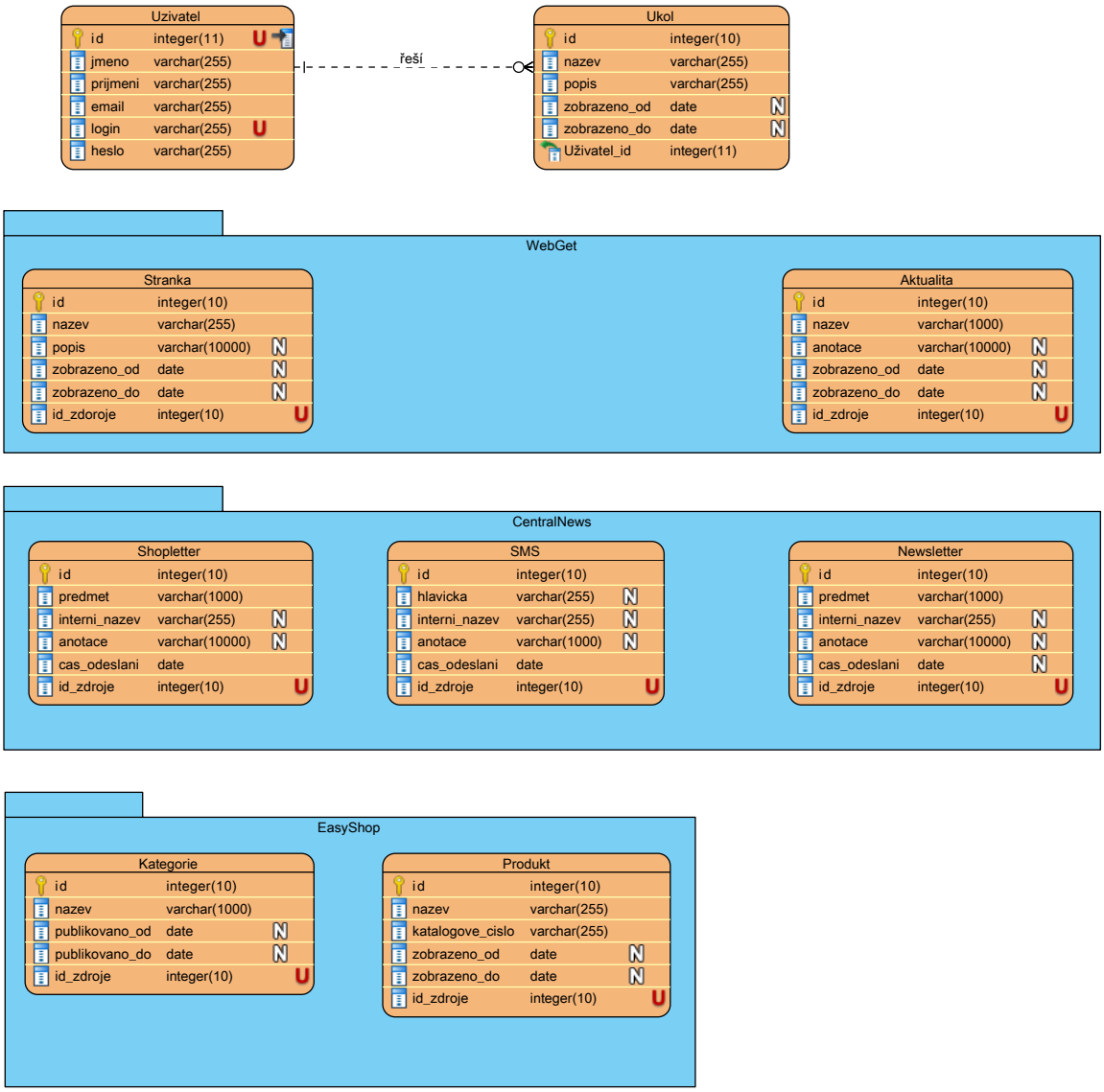
Tabulka 5.4: Případ použití selhání systému

Případ použití:	Storno	**ID:**	E.2
Stručný popis:	Přerušení operace.		
Primární aktéři:	Dle daného případu použití.		
Předpoklady:	1. Uživatel provedl storno operaci.		
Následné podmínky:	1. V systému nebyly provedeny změny.		
Tok:	1. Tok Storno může být vyvolán během provádění hlavního toku. 2. Systém informuje uživatele o chybě.		
Frekvence:	Výjimečně		

Tabulka 5.5: Případ použití storno



Obrázek 5.2: Sekvenční diagram [zdroj vlastní]



Obrázek 5.3: Návrh schématu databáze [zdroj vlastní]

Kapitola 6

Výběr vhodné platformy

Z důvodu zaměření uvedených aplikací na použití na webu bude výběr vhodné platformy směřovat na technologie používané ve webovém prostředí. Pokud uvažujeme pohled od klienta k serveru, bude jako první potřeba použít nějaký značkovací jazyk - HTML, případně XHTML. Pro ovlivnění vzhledu zobrazovaných informací se zde nabízí použití CSS (Cascading Style Sheets). Pro kvalitnější práci s aplikacemi je taktéž vhodné použít JavaScript, jenž umožňuje provádění skriptů v klientském prohlížeči. V konkrétním případě je použita JavaSriptová knihovna jQuery popsaná v kapitole 6.4. Celou klientskou část z pohledu výběru platformy zastřeší front-end framework Bootstrap popsaný v kapitole 6.3.

Serverovou část tvoří technologie PHP (Hypertext Preprocessor), respektive použitý PHP framework, ORM (Object-Relational Mapping) framework Doctrine, jenž mapuje objekty na relační databázi, a databáze MySQL (My Structured Query Language). Tyto technologie v současné době patří mezi nejpoužívanější pro tvorbu webových prezentací.

6.1 PHP

PHP [22] (Hypertext Preprocessor) je interpretovaný skriptovací jazyk zaměřený na tvorbu internetových stránek a dynamických internetových aplikací. První verzi PHP vytvořil v roce 1995 Rasmus Lerdorf. Nyní jazyk vyvíjí skupina zvaná PHP Group a je šířen pod open-source licencí. PHP umožňuje vytvářet jak skripty běžící na serveru, tak i konzolové aplikace a dokonce i desktopové aplikace, u kterých se použije kompilovaná forma jazyka. Jedná se o nejpoužívanější skriptovací jazyk určený pro webové aplikace [38].

Důvodů velké oblíbenosti je několik:

- Syntaxe je inspirována již známými jazyky (Perl, C, Java atd.).
- Nezávislost na platformě.
- Podpora mnoha rozšiřujících knihoven.
- Nativní podpora mnoha databázových systémů.
- Podpora řady internetových protokolů.
- Hostingové služby mají podporu PHP jako standard [19].

Webové aplikace (skripty) vytvořené v PHP jsou uloženy na serveru. Pro spuštění konkrétního skriptu je zapotřebí, aby klient provedl HTTP požadavek pro získání zdroje (v konkrétním případě dokument obsahující PHP kód) ze serveru. Server obdržený požadavek

zpracuje a před odesláním odpovědi vykoná daný PHP kód. Tento kód může např. zapsat/přečíst data do/z databáze, může vygenerovat nějaký výstup pro klienta apod. Odpověď od serveru je tedy výsledkem běhu PHP skriptu, např. v podobě HTML stránky. Používáním PHP jazyka nedochází k vytvoření žádných perzistentních proměnných, po vykonání požadavku si server neukládá žádné informace o proběhlé akci.

6.1.1 Nette Framework

Nette Framework (dále jen Nette) je rozsáhlý objektový PHP framework (aplikační rámec) určený pro tvorbu webových aplikací využívající jazyk PHP 5 nebo PHP 7. Nette byl vytvořen českým autorem Davidem Grudlem, který ho zveřejnil pod licencí GNU GPL (GNU's Not Unix General Public License). Nyní je vyvíjen společností Nette Foundations a zveřejňován pod licencemi GNU GPL a Nette (obdoba BSD licence). Nette se vyznačuje použitím architektury MVP (Model-View-Presenter), což je obdobou MVC (Model-View-Controler). Dalším výrazným rysem Nette je použití událostmi řízeného programování a rozčlenění na komponenty. Tyto zmíněné vlastnosti umožňují použít Nette přesně podle potřeb konkrétního programátora. To znamená, že je možno využít doporučenou strukturu, tzv. skeleton, jenž vede na využití většiny funkcí a vlastností, které framework nabízí. Případně je možné použít konkrétní vybranou komponentu (např. komponentu pracující s formuláři) samostatně v jakékoliv aplikaci bez potřeby načítat zbylé části aplikačního rámce. Nette si zakládá na vysoké úrovni bezpečnosti, a proto sám zajišťuje ochranu proti Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), URL attack, invalid UTF-8, Session hijacking, session stealing, session fixation. Jedním z dalších specifik, jenž framework nabízí, je vlastní šablonovací systém, který používá pohledy (šablony). Vlastní řešení spočívá například v podobě zápisu odkazů nebo možnosti použít makra. Tato makra umožňují zapisovat cykly, podmínky, vypisovat proměnné atd. [28][33][43].

Nette bude použito pro řešení projektu. Důvodem pro použití je, že se jedná o jeden z nejvýkonnějších dostupných nástrojů [17]. Dalším důvodem jsou pokročilé ladící nástroje, tzv. laděnka, popsané v kapitole 8, a mnoho dostupných rozšiřujících komponent od vývojářů třetích stran. V neposlední řadě se jedná o projekt těšící se největší české komunitě. Zároveň Nette neobsahuje mnoho funkcí navíc, které by byly nevyužité a zpomalovaly by tak výslednou aplikaci. Proto se framework řadí mezi jedny z nejrychlejších a je určen spíše pro menší projekty.

6.1.2 Symfony

Symfony [39] je další PHP framework zaměřeným na vývoj webových aplikací vycházející z návrhového vzoru MVC. Framework návrhem vychází z jiných aplikačních rámců, jako např. Ruby on Rails, Django a Spring. Symfony je zveřejněn jako open-source a je vydáván pod MIT licencí. Podobně jako Nette se jedná o framework s objektovým návrhem využívající vlastní šablonovací systém Twig a rozčlenění do balíčků, tzv. bundle. Za projektem Symfony stojí mnohonásobně více vývojářů než v případě Nette, což se projevuje v podobě rychlejšího vývoje, rozsáhlejší dokumentace a většího počtu rozšíření. Obě uvedené softwarové struktury používají podobné nástroje, např. Dependency Injection, šablonovací systémy, které mají funkce jako dědění šablon, automatický escape proměnných. V případě, kdy nějaká funkcionalita u Symfony chybí, je často k dohledání jako rozšíření pomocí již zmiňovaných balíčků bundle. Výhodou Symfony je přítomnost více funkcí a nástrojů podpořená velkou komunitou. Jako jedna z nevýhod je uváděna horší komponenta pro hledání chyb, jelikož obsahuje méně potřebných informací. Rozsáhlý framework s sebou nese i vyšší

nároky na výpočetní výkon ve vývojářské verzi. V produkčním prostředí již výkonnostní rozdíl není tolik patrný, ale jen v případě použití APC (Alternative PHP Cache). Poslední zmíněnou nevýhodou oproti Nette je složitější použití formulářů zejména z pohledu nastavení vlastností jednotlivých prvků. Obecně lze Symfony označit za framework určený pro větší a business projekty, kde je upřednostňována stabilita [33].

6.1.3 Doctrine

Doctrine, konkrétně Doctrine 2, je ORM (Object-Relational Mapping) framework pro jazyk PHP. Jedná se tedy o vrstvu, která mapuje objekty na relační databázi. To umožní, aby aplikace pracovala pouze s objekty. O správu (vytvoření databázového schématu), uložení a načítání se automaticky stará framework bez dalších zásahů vývojáře. Pro korektní chod tedy postačí definovat entity a připojení do databáze, o zbytek se stará ORM vrstva [41][42]. Doctrine 2 je rozdělen do tří vrstev:

- **Common** - Tato nejnižší vrstva definuje základní obecná rozhraní, třídy a knihovny. Například nástroje pro práci s kolekcemi, anotacemi, systém použití cache, událostmi apod. Nástroje jsou následně využívány vyššími vrstvami. Jelikož se jedná o nejnižší vrstvu, tak pro svůj běh nepotřebuje závislosti dalších vrstev.
- **DBAL (DataBase Abstraction Layer)** - Vrstva zajišťuje abstrakci aplikace od konkrétního typu databáze. Primárně rozšiřuje standardní PDO (PHP Data Objects) [14], ale je možno použít i jiný databázový driver. Tato vrstva zavádí notaci DQL (Doctrine Query Language), což je zcela nový jazyk. Podobnost s dotazovacím jazykem SQL je jen zdánlivá, jelikož SQL pracuje s konkrétními tabulkami a řádky v databázi, zatímco DQL pracuje s entitami a jejími atributy, tzn. vrstva databáze je zcela abstrahována.
- **ORM (Object-Relational Mapping)** - Nejvyšší vrstva, která zajišťuje mapování aplikačních objektů na relační databázi, jejich ukládání a načítání. ORM je závislá na DBAL i Common.

6.2 MySQL

MySQL (My Structured Query Language) je systém řízení báze dat používající relační databázový model. Databáze vznikla roku 1995 a jedná se o multiplatformní databázi s dotazovacím jazykem SQL (Structured Query Language). Systém je zveřejněn pod dvěma licencemi - pod bezplatnou licencí GPL a pod komerční licencí. MySQL klade důraz na rychlost a výkon, a to za cenu zjednodušení určitých částí systému, např. způsobů zálohování. Pro své výhody se jedná o nejrozšířenější databázový systém[34]. Velmi často se tedy využívá kombinace GNU/Linux (operační systém), Apache (HTTP Server), MySQL a PHP, jako základní software webového serveru pojmenovaný jako LAMP. Databáze umožňuje ukládat různá data (texty, obrázky atd.), s nimiž lze dále jednoduše pracovat (třídit, řadit, filtrovat apod.). Pro jednodušší správu databáze je dostupný například nástroj phpMyAdmin [21][1].

6.3 Bootstrap

Bootstrap je HTML, CSS a JS framework pro tvorbu responzivních webových stránek. Spíše než framework by se dal označit jako sada nástrojů usnadňující tvorbu webových aplikací. Framework původně vyvinuli vývojáři Mark Otto a Jacob Thornton pro interní potřeby sociální sítě Twitter. V roce 2011 byl vydán Twitter Bootstrap jako open source a záhy v roce 2012 se stal nejoblíbenějším developerským projektem na GitHub (služba podporující vývoj softwaru za pomoci verzovacího nástroje Git). Bootstrap obsahuje mnoho návrhářských šablon založených na HTML a CSS sloužících pro úpravu typografie, tlačítek, formulářů, navigačního menu a dalších komponent rozhraní. Součástí jsou i další volitelná rozšíření JavaScriptu. Framework se těší velké oblibě zejména díky podpoře responzivního designu. To znamená, že zobrazení stránky je ovlivněno použitým zařízením (stolní PC, tablet, mobilní telefon) a od toho se odvíjí rozložení stránky, umístění a velikost jednotlivých prvků, případně skrytí vybraných prvků [40].

6.4 jQuery

Multiplatformní JavaScriptová knihovna jQuery [6] je navržená pro spouštění na straně klienta spolu s HTML kódem. Knihovnu představil v roce 2006 její autor Johh Resig pod svobodnou licencí MIT.

Knihovna jQuery umožňuje:

- Procházet a upravovat DOM (Document Object Model).
- Vytvářet reakce na události.
- Manipulovat s CSS.
- Efekty a animace.
- AJAX (Asynchronous JavaScript and XML) - načítání obsahu serveru bez nutnosti obnovení stránky.
- Podporuje mnoho plug-inu.
- Obsahuje mnoho utilit – např. informace o prohlížeči.

Kapitola 7

Implementace

Tato kapitola se zabývá implementací e-commerce kalendáře a prototypových aplikací, zejména implementací REST API.

Základním prvkem všech aplikací je PHP framework Nette. Při implementaci byla snaha využít co nejvíce prostředků, které nám nabízí framework nebo rozšíření určená přímo pro Nette. Konkrétně je převzata adresářová struktura, tzv. sandbox. Jedná se o adresářovou strukturu ukázkové aplikace, kde je předpřipraveno doporučené rozdělení adresářů podle datových vrstev a návrhového modelu.

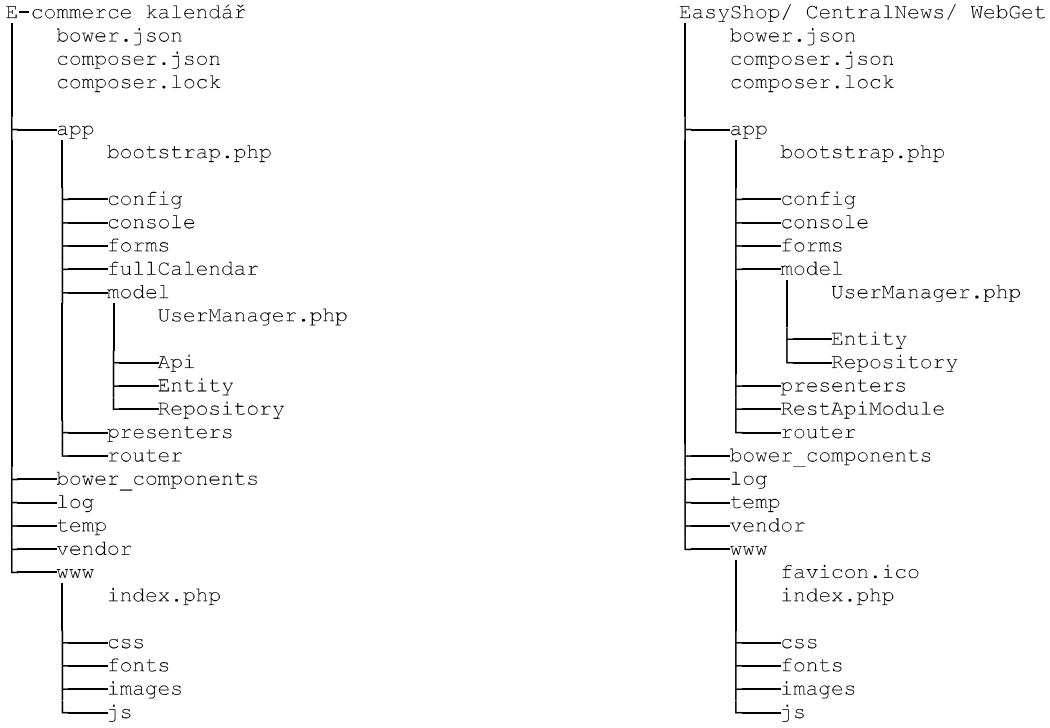
K implementaci databáze je použita databáze MySQL. Důvodem použití MySQL je její rozšíření, viz kapitola 6.2.

7.1 Adresářová struktura aplikací

Adresářová struktura všech aplikací je postavena na tzv. sandbox. Použití doporučené adresářové struktury vede k vhodnému použití Nette, dodržování zásad objektového programování a hlavně zajištění přehlednosti aplikace. Na obrázku č. 7.1 jsou zobrazeny dvě adresářové struktury, a to aplikace e-commerce kalendář a společná struktura pro EasyShop, WebGet a Central News, jelikož se jedná o podobné prototypy demonstrující funkcionalitu REST API.

V kořenovém adresáři se nachází šest adresářů aplikace a tři soubory `composer.json`, `composer.lock` a `bower.json`, ve kterých je zapsána konfigurace pro Composer (nástroj na správu závislostí v PHP) a pro Bower (správce front-end balíčků). Konkrétně jsou zde vypsány všechny použité závislosti včetně jejich verzí.

Adresář `app` obsahuje zdrojové kódy a konfigurace dané aplikace popsané dále. Je zde uložen důležitý soubor `bootstrap.php`, který má za úkol načíst framework společně s potřebnou konfigurací aplikace. Společný adresář `config` obsahuje nastavení aplikace - nalezneme zde přihlašovací údaje do databáze, jak se má chovat cache a session aplikace, jaké služby (service) jsou v aplikaci dostupné. Dále jsou zde údaje, jak se má aplikace zachovat při výskytu chyby, jakým způsobem má pracovat s adresami. Je zde i nastavení chování použitých komponent, např. nastavení pro komponentu pro práci s obrázky či komponentu pro REST API. Dalším společným adresářem je `console`. Tento adresář obsahuje skript spustitelný pomocí příkazové řádky (terminálu) a zajišťuje inicializaci databáze při prvním nastavení aplikace. Adresář `forms` obsahuje definici továren formulářů, které se používají v aplikaci. Definování formuláře pomocí továrny umožňuje opětovné použití formuláře kdekoliv v aplikaci. E-commerce kalendář zde navíc obsahuje adresář `fullCalendar`,



Obrázek 7.1: Adresářové struktury aplikací [zdroj vlastní]

ve kterém se nachází modul pro vykreslení a obsluhu kalendáře. Adresář `model` zastřešuje definici zdrojových kódů umožňujících komunikaci s databází (Repository) pomocí komponenty `kdyby/doctrine` - jedná se o databázovou vrstvu, která je volána z presenterů. Zároveň jsou zde definovány entity (objekty) reprezentující domény, které aplikace využívá. Na základě těchto entit se generuje i struktura databáze. Aplikace kalendáře zde navíc má umístěný adresář `Api`, který má na starost komunikaci s danými rozhraními. Předposledním adresářem je adresář `presenters`, jenž reprezentuje dvě vrstvy objektového modelu MVP, a to vrstvy Presenter a View. Vrstvu Presenter reprezentují tzv. Presentery a vrstva View se nachází ve složce `templates`, kde se nacházejí šablony jednotlivých pohledů seskupené podle příslušnosti k danému presenteru. Prototypové aplikace zde mají navíc umístěn adresář `RestApiModule` s modulem zajišťující obsluhu REST API rozhraní. Posledním uvedeným adresářem je `router`, jenž obsahuje třídu zajišťující překlad URL adres na adresy určující použitý presenter a pohled. V případě prototypových aplikací EasyShop, WebGet a CentralNews je zde definován i překlad URL adres směřujících na zdroje REST API.

Všechny zachycené chybové stavy a události se ukládají do adresáře `log`. Dočasné soubory `cache` slouží ke zrychlení načtení stránek aplikace tím, že výsledný vygenerovaný zdrojový kód odeslaný do prohlížeče uloží a při stejném požadavku systém odpoví obsahem

uloženého souboru cache. Tím odpadá režie běhu PHP skriptu a zároveň i režie dotazování dat na databázi. Do adresáře se ukládají nejen výstupy posílané klientům, ale také výsledky dotazů databáze a vygenerované nastavení aplikace. Správu obsahu dané složky plně zajišťuje framework, který určí, jaký obsah s jakou dobou expirace bude uchovávat.

Knihovny Nette jsou uloženy v adresáři `vendor`. Tento adresář obsahuje jak nativní knihovny, tak i rozšíření pro framework, které sem umístil již zmiňovaný správce závislostí Composer.

Posledním adresářem v kořenové složce je adresář `www`. Do tohoto adresáře mají přístup klienti, kteří se připojují na server a využívají aplikace. Důležitým souborem je `index.php`, který je spouštěn při každém požadavku. Soubor obsahuje kód pro vyvolání souboru `app/bootstrap.php` a následné spuštění aplikace. Dalšími veřejně přístupnými kódy jsou kódy zajišťující vzhled aplikace uložené v adresářích `css`. Tyto kódy obsahují kaskádové styly a adresář `font`, kde je uložen font písma použitý v projektu. Následující adresář `images` shromažďuje obrázky použité pro grafický návrh aplikace. Adresář `js` obsahuje JavaSriptové kódy a knihovny, které jsou interpretovány na straně uživatele. V projektu jsou použity knihovny třetích stran jQuery, Bootstrap, výběr času datetime picker, kalendář fullcalendar spolu se skriptem `moment.js` pro práci s datem a časem. Dále je využíván skript `nette.ajax.js` a `netteForms.js` napsaný přímo pro Nette Framework, čímž lze snadno dosáhnout interaktivního chování aplikace pomocí ajaxových volání.

7.2 E-commerce kalendář

V kapitole popisující implementaci e-commerce kalendáře budou uvedeny nejdůležitější části aplikace. Rozebrán zde bude proces přihlášení, který je stejný i pro aplikace EasyShop, WebGet a CentralNews. Dále zde bude popsána vytvořená komponenta FullCalendar a způsob aktualizace dat pomocí REST API rozhraní. Na závěr této kapitoly se popíše layout aplikace, který bude podobný pro všechny vytvořené aplikace.

7.2.1 Přihlašování

Každý uživatel aplikace e-commerce kalendář je reprezentován službou `user`, což je objekt třídy `Nette\Security\User`. Pro vstup do aplikace je zapotřebí být přihlášen. Zda je daný uživatel řádně přihlášen do systému je ověřeno zavoláním funkce `isLoggedIn()`. Presenter `HomepagePresenter` při vytvoření objektu v metodě `startup()` zkontroluje, zda je uživatel přihlášen. U nepřihlášeného uživatele dojde k přesměrování na přihlašovací formulář. Zároveň s přesměrováním se předává tzv. backlink, což je adresa, na kterou je po přihlášení uživatel přesměrován zpět. Po odeslání přihlašovacího formuláře dojde k pokusu o přihlášení. Třída `UserManager` implementující rozhraní `Nette\Security\IAuthenticator` provede přihlášení uživatele pomocí funkce `authenticate()`, která se pokusí za pomoci repositáře `UserRepository` nalézt uživatele na základě přihlašovacího jména. V případě nalezení entity provede verifikaci zadaného hesla pomocí `Passwords::verify()` a následně naplní službu `user` potřebnými daty. V případě negativní verifikace vygeneruje nový kontrolní součet hesla `Passwords::needsRehash`.

7.2.2 Komponenta FullCalendar

Vytvořená komponenta FullCalendar zajišťuje funkcionalitu celé aplikace, tj. vykreslení dvou kalendářů. První kalendář slouží pro rychlou navigaci a druhý kalendář zobrazuje

konkrétní informace. Oba kalendáře jsou provázané - výběrem data v navigačním kalendáři se překreslí data ve druhém kalendáři. Dále komponenta umožňuje kromě zobrazení informací i zadávat do kalendáře úkoly, případně zadané úkoly upravovat. Pro navigační kalendář byla zvolena JavaSriptová komponenta Bootstrap 3 Date/Time Picker [30]. Pro druhý kalendář pro zobrazení událostí byla využita opět JavaSriptová komponenta, a to FullCalendar. Důvodem pro volbu JavaSriptových komponent je komfort uživatele při použití daného kalendáře jako celku. Při listování v kalendáři není zapotřebí se stále dotazovat serveru na zobrazení kalendáře v konkrétním časovém úseku. Vše výše zmíněné je vykonáváno na straně klienta a jediná komunikace se serverem probíhá tehdy, když se načítají potřebné události, a je vykonána na pozadí. Komponenta se skládá ze tří souborů, a to šablony, třídy a rozhraní. Použití komponenty je následující: v presenteru se přichystá funkce `createComponentFullCalendar()`, jejímž úkolem je vytvořit instanci komponenty, která předá potřebné závislosti, např. třídy pracující s databází. Vygenerování kalendářů zajišťuje funkce `render()` třídy komponenty `/app/fullCalendar/FullCalendarControl`. Tato funkce pouze předá do šablony datum, které bylo obdrženo jako parametr URL adresy, v případě chybějícího parametru předává aktuální datum. Šablona vykreslí předpřipravenou strukturu kalendářů a JavaScriptový kód pro inicializaci a definici chování kalendářů. Jak už bylo zmíněno, veškerá komunikace kalendáře se serverem probíhá na pozadí pomocí volání událostí, tzv. handle. Komponenta FullCalendar má implementovanou obsluhu na následující události:

- `handleGetEvents ($start, $end)` - handle očekává jako parametr časové rozmezí, ve kterém chce uživatel vyhledat dostupné události uložené v databázi. Při zpracování se postupně vyhledávají entity vyhovující zvolenému časovému rozmezí. Po nalezení všech entit je vytvořena odpověď ve formátu JSON.
- `handleChangeEvent ($id, $start, $end)` - handle je vyvolán při změně události v kalendáři, např. posunutí začátku/konce události nebo přesunutí na jiné datum. Jako parametry očekává identifikátor události, počátek a konec trvání události. Identifikátor události je tvořen složením názvu entity a jejího interního identifikátoru, např. category. Tím se zabrání kolizi identifikátorů v kalendáři a zároveň se rozpozná, jaká entita má být upravena. O výsledku operace je klient informován zobrazením zprávy.
- `handleAddTask ($title, $start, $end, $description)` - handle je vyvolán po vyplnění a odeslání formuláře určeného pro vytvoření úkolu. Jako parametry přijímá název úkolu, datum vytvoření úkolu, nejzazší termín dokončení úkolu a nepovinný popis úkolu.

7.2.3 Napojení na API rozhraní

Komunikace s napojenými aplikacemi probíhá automaticky nezávisle na běhu kalendáře. Každou hodinu cron (Linux/Unix systémový nástroj, který spouští různé programy v předem definovanou dobu a intervalu) spustí skript na adrese `WEB-ROOT/api`, kde `WEB-ROOT` zastupuje základní adresu umístění aplikace, tj. dle obrázku č. 2.2 URL adresa po hostname, případně po port. Spuštěním uvedeného skriptu se v aplikaci vyvolá akce presenteru `ApiPresenter`. Tato akce do třídy `ApiContainer` zaregistruje entity pomocí metody `registerEntity`. Registrací je myšleno vytvoření třídy, která dokáže obsluhovat komunikaci dané entity a které je předána URL adresa, kde se nachází požadovaný zdroj. Po registraci všech entit akce volá metodu `load`. Metoda `load` projde všechny zaregistrované entity

a u každé provede metody `beforeLoad`, `loadEntity`, `afterLoad`. První metoda `beforeLoad` zajistí, aby se všechny změny provedené v kalendáři nejprve přenesly do napojené aplikace. Nepřenáší se všechny uložené entity, ale pouze změněné, což se pozná nastaveným příznakem `pushToServer`. Po přenesení dat z kalendáře do napojené aplikace dojde k nastavení dalšího příznaku `updated` na hodnotu `FALSE`. Po dokončení metody `beforeLoad` následuje metoda `loadEntity`. Tato metoda se dotáže na všechny entity konkrétního zdroje a získaná data uloží do databáze (stávající entity aktualizuje a nové vytvoří). V případě velkého množství dat je zapotřebí provést více dotazů, viz kapitola 7.3.5. Poslední metoda `afterLoad()` zkontroluje, zda u všech entit daného zdroje došlo k nastavení příznaku `updated` na hodnotu `TRUE` (nastavení na hodnotu `TRUE` se provádí při aktualizaci entity nebo při vytvoření nové entity). Entity, které stále obsahují hodnotu `FALSE`, byly ve zdrojové aplikaci smazány, a proto dojde ke smazání i v kalendáři.

Obrázek 7.2: Layout aplikace e-commerce kalendář na zařízení stolní počítač [zdroj vlastní]

7.2.4 Layout aplikace

Pro vytvoření grafického rozvržení stránek systému (layout) byl použit front-end framework Bootstrap. Snahou bylo navrhnout co nejjednodušší rozložení prvků na stránce tak, aby nejdůležitějším byl obsah (v konkrétním případě kalendář). Layout se skládá ze tří hlavních částí, viz obrázek č. 7.2 Horní okraj stránky je lemován úzkým pruhem, ve kterém se nachází název aplikace. Tento pruh zároveň slouží jako navigační menu. Druhou částí je prostor pro obsah stránek, jenž je umístěn na bílošedém podkladu a zarovnán na střed obrazovky. Poslední částí je patička, na které se nachází copyright. Bootstrap framework zajišťuje i přizpůsobení stránky podle použitého zobrazovacího zařízení. Toto chování umožňuje tzv. Grid systém. Tento systém pracuje s kaskádovými styly, které jsou rozděleny do tří kroků, tzv. breakpoint. Konkrétně jsou tyto kroky rozděleny podle šířky na mobilní zařízení (do 768 px), tabletová zařízení (do 992 px), stolní zařízení (do 1200px) a velká stolní zařízení (nad 1200px). Nastavování šíře jakéhokoliv HTML prvku se provádí pomocí určení třídy. Každý krok je rozdělen na dvanáct částí, nastavení vlastností je tedy dáno tím, o jaký krok se

jedná - `col-xs-` (extra small), `col-sm-` (small), `col-md-` (medium), `col-lg-` (large), a počtem částí. Nastavené vlastnosti zároveň respektují vlastnosti rodičovských prvků. Jestliže budeme mít prvek s vlastností `col-xs-6` a tento prvek bude obsahovat další prvek taktéž s vlastností `col-xs-6`, bude výsledná šíře potomka čtvrtina zobrazované plochy. Toto chování zajistí, že nebude docházet k nevhodnému překrývání prvků nebo k rozbití vzhledu aplikace jiným způsobem. Nastavení chování zobrazení prvků se pak provede pomocí zmíněných definic tříd. Například mějme tři HTML prvky, které budou mít definován atribut třídy následovně: `class="col-xs-12 col-md-6 col-lg-4"`. Tato definice tříd tedy říká:

- na stolních zařízeních zobraz prvky vedle sebe (prvek má šířku třetiny nadřazeného prvku)
- na tabletu zobraz dva prvky vedle sebe a třetí zalom pod ně (prvek má šířku poloviny nadřazeného prvku)
- na mobilním zařízení zobraz všechny prvky pod sebe (prvek má šířku nadřazeného prvku)

Obrázek 7.3: Layout aplikace e-commerce kalendář na zařízení mobilní telefon [zdroj vlastní]

Tato pravidla jsou použita i v aplikaci kalendáře, kdy na obrázku č. 7.2 je vidět rozložení pro stolní zařízení a na obrázku č. 7.3 je zobrazena stejná stránka v zobrazení pro mobilní telefony.

7.3 Aplikace EasyShop, WebGet a CentralNews

Tato kapitola popíše základní prvky prototypů aplikací EasyShop, CentralNews a WebGet. Z důvodu podobnosti uvedených prototypů využívajících stejné komponenty a funkce, nebudou tyto prototypy popisovány samostatně. Bude zde zmíněna komponenta DataGrid, která slouží jako přehled uložených entit. Jako nejdůležitější část zde bude uvedena implementace REST API rozhraní spolu s překladem adres na vytvořené obslužné třídy. Struktura aplikací vychází z definovaných entit. Skoro každá entita má vlastní presenter, který vykreslí komponentu Datagrid. Každý presenter umožňuje zobrazit detail entity, danou entitu vytvořit či ji upravit. Pro práci s entitou je použit předem definovaný formulář.

7.3.1 Komponenta Datagrid

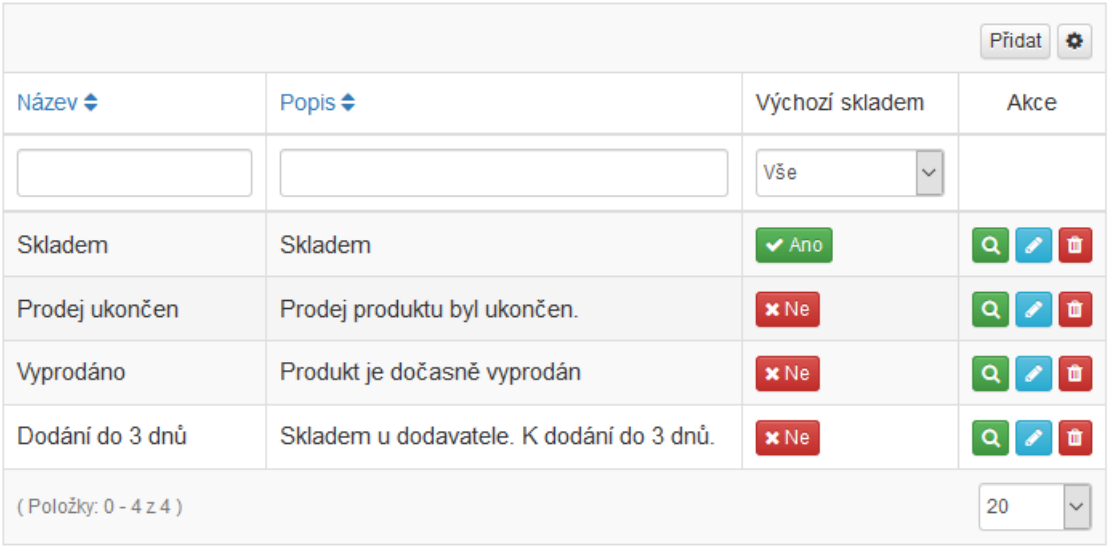
Datagrid [18] je komponenta třetí strany, která dokáže zobrazit data do tabulky a dále s tabulkou pracovat. Rozšiřujícími funkcemi je možnost řadit data podle zvoleného klíče, vyhledávat v daném sloupci, nastavovat zobrazení sloupců a stránkovat data. Na příkladu 7.1 je zobrazena definice Datagridu pro entitu dostupnost.

```
1  $grid = $this->getDataGrid($this, $name);
2  $grid->setDataSource(
3  $this->availabilityRepository->getAllAvailability());
4
5  $grid->addColumnText('name', 'Nazev')
6  ->setSortable()->setSortableResetPagination();
7
8  $grid->addColumnText('description', 'Popis')
9  ->setSortable()->setSortableResetPagination();
10
11 $grid->addColumnText('defaultStock', 'Vychozi skladem')
12 ->setSortable()->setSortableResetPagination();
13
14 $grid->setColumnsHideable();
15 $grid->addFilterText('name', 'Nazev');
16 $grid->addFilterText('description', 'Popis');
```

Ukázka kódu 7.1: Definice Datagridu pro entitu dostupnost

Instance třídy `Ublaboo\DataGrid` se získá pomocí metody `getDataGrid()`. Metoda vytvoří instanci Datagrid a provede počáteční nastavení, jako je definování počtu záznamů na stránku, definování tlačítek akce (detail, úprava a smazání) pro dané záznamy. Dále definuje tlačítko pro vytvoření nového záznamu. Naposledy jsou definovány překlady. Po získání inicializované instance Datagridu jsou instanci předána data `setDataSource($this->availabilityRepository->getAllAvailability())`. Data jsou instancí třídy `Doctrine\ORM\QueryBuilder`, která na požádání provede dotaz do databáze a poskytne kolekci entit dostupností. Dále pomocí metody `addColumnText($key, $name)` se DataGridu sdělí, jaké sloupce se mají zobrazit, a to určením názvu atributu entity a popisku, který se následně zobrazí. Metoda `setSortable()` umožní daný sloupec řadit dle abecedy a metoda `setSortableResetPagination()` zaručí, že při volbě řazení dojde k přechodu na první stránku záznamů. Povolení možnosti výběru a skrytí zobrazených sloupců se provede metodou `setColumnsHideable()`. Posledním krokem v popisovaném příkladu je nastavení filtrování, kdy se pomocí metody `addFilterText($key, $name)` určí název atributu entity, který se má filtrovat, a popisek filtrovacího pole. Při vykreslení komponenty

se do tabulky přidá pole pro vepsání hledaného textu. Příklad vykreslení popisovaného případu je na obrázku č. 7.4.



Obrázek 7.4: Vykreslena tabulka komponenty Datagrid [zdroj vlastní]

Použití této komponenty je u všech případů podobné. V některých případech je např. u atributu `showed` (zobrazeno Ano/Ne) možnost tuto volbu měnit přímo v tabulce, a to vyvoláním handle `statusChange($id, $status)`, kterému je předán identifikátor entity a zvolená možnost. V jiných případech se kromě textu zobrazuje i obrázek atd.

7.3.2 Rozhraní REST API

Pro implementaci rozhraní byla použita komponenta `Drahak\Restful` - Nette REST API bundle [24]. Toto rozšíření pokrývá problematiku zpracování požadavku a správného mapování na obsluhující metody. Zpracování využívá informací uvedených v hlavičce požadavku, např. u hlavičky `Accept` podporuje příjem různých formátů:

- `application/xml`
- `application/json`
- `application/x-data-url`
- `application/www-form-urlencoded`

Z pohledu vývojáře to znamená, že ke zpracování přijatých dat od komponenty obdrží vždy stejná data, konkrétně instanci objektu `Drahak\Restful\Http\InputFactory`. Obdobným způsobem se přistupuje i k odpovědím ze serveru klientovi, kdy se komponentě předají odesílané informace a metodou `sendResource($type)` se definuje formát odpovědi a odpověď se odešle v požadovaném formátu včetně korektně nastavených HTTP hlaviček. Výše popsaný proces odpovědi je za předpokladu absence výskytu chyb. V případě, kdy dojde k výskytu nějaké chyby, je možno zaznamenanou chybu předat komponentě a ta opět vytvoří a odešle příslušnou odpověď s patřičnými stavovými kódy. Implementace

uvažuje jako výchozí formát pro odpověď `application/json`, případně `application/xml`, pokud URL adresa zdroje obsahuje koncovku `.xml`. Komponenta Restful podporuje i různé způsoby zabezpečení:

Podepsaný požadavek

Klient a server sdílí tajemství v podobě tajného klíče. Tento klíč slouží jako jeden ze vstupů funkce `hash_hmac`. Spolu s dotazem na server se zasílá i hash posílané zprávy, který slouží ke kontrole oprávnění. Celý proces autentifikace je tedy následovný:

- klient do těla požadavku přidá `timestamp`
- klient vytvoří hash zprávy pomocí funkce `hash_hmac` (sha256 algoritmem) s tajným klíčem
- klient do hlavičky požadavku přidá `X-HTTP-AUTH-TOKEN` spolu s počítaným otiskem zprávy (hash) a požadavek odesílá serveru
- server spočítá hash příchozí zprávy stejně jako klient
- porovná vypočítaný otisk a hodnotu obdrženou v hlavičce `X-HTTP-AUTH-TOKEN`
- dále server porovná i klientem přidaný `timestamp` a aktuální `timestamp`, kdy rozdíl nesmí být větší než 5 minut (zabezpečení před Replay attack)

OAuth 2.0

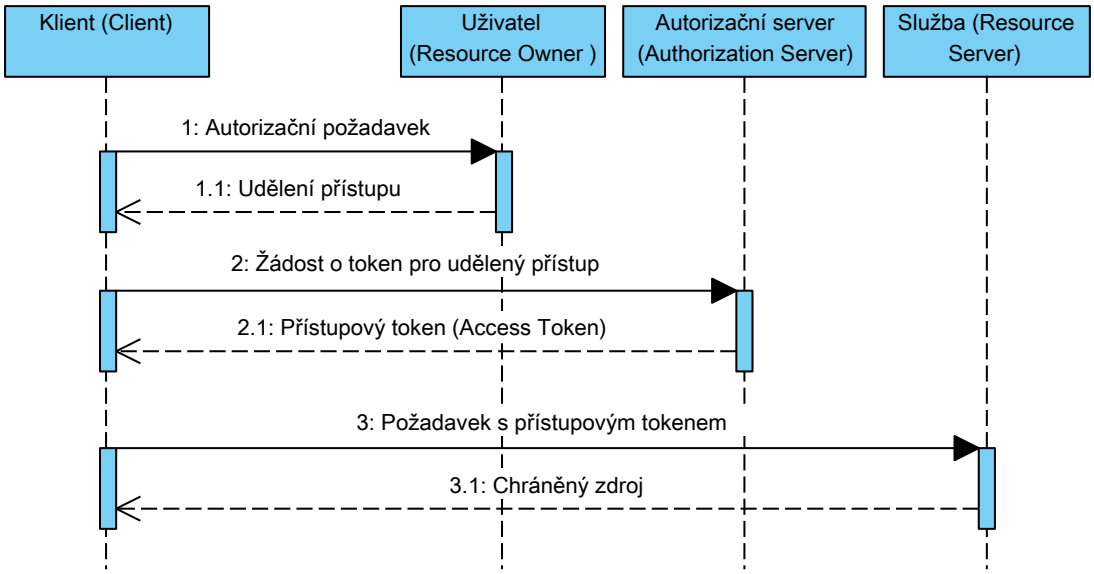
OAuth 2.0 [16] je autorizační protokol (resp. framework), používaný pro zabezpečení služeb využívající REST API rozhraní. Princip zabezpečení spočívá v přidání autorizační vrstvy mezi klientem a serverem poskytující přístup ke zdrojům. Zabezpečení poté spočívá v předávání tokenu, který slouží pro přístup do požadované aplikace.

OAuth 2.0 definuje čtyři druhy rolí:

- **resource owner** (uživatel) – vlastník chráněného zdroje, má právo přidělovat nebo odepírat přístup ke zdroji
- **resource server** (služba) – poskytovatel zdroje, očekává požadavky obsahující `access token`, na jehož základě obslouží požadavky
- **client** (klient) – aplikace, která přistupuje ke zdroji dotazem na resource server, který obsahuje `access token` omezený nastavenými oprávněními uživatele (resource owner)
- **authorization server** (autorizační server) – server, který klientovi vydává `access token` v případě jeho úspěšné autentizace od resource ownera (uživatele) a získání autorizace (autorizační server může být přímo součástí resource server nebo oddělený).

Na obrázku č. 7.5 je uveden sekvenční diagram OAuth 2.0 autorizace. Z důvodu, kdy navrhované rozhraní není určené pro širokou veřejnost, ale pro interní použití (komunikace mezi vlastními aplikacemi), byla zvolena varianta s podepsaným požadavkem.

Poslední důležitou otázkou zůstává, jak se vypořádat s problémem ohledně práce s multimédii, a to konkrétně s obrázky. I zde existuje několik přístupů závisejících hlavně na způsobu použití. Možné přístupy jsou:



Obrázek 7.5: Sekvenční diagram OAuth 2.0 autorizace [zdroj vlastní]

- **přistup k obrázku jako k samostatnému zdroji** - tento přístup umožní chovat se k obrázku stejně jako k jinému zdroji, tj. využívat HTTP metody GET, PUT, POST, DELETE. Výhodou této varianty je možnost přímého přístupu k obrázku/obrázkům daného zdroje. Nevýhodou je zde nutnost provádění nových požadavků vůči serveru. Pokud například vytváříme nový zdroj včetně obrázků, tak je nejprve zapotřebí vytvořit zdroj. Až poté je možné od serveru získat vytvořený identifikátor potřebný pro URL adresu sloužící k přístupu k obrázkům a následně nahrát obrázky na server. Dané řešení je vhodné pro zdroje s více obrázky, např. fotogalerie.
- **přistup k obrázku jako součást daného zdroje** - toto řešení uvažuje obrázek jako jeden z atributů daného zdroje. Pro čtení zdroje se do atributu vkládá URL adresa vedoucí přímo na daný obrázek (URL adresa nereprezentuje adresaci daného zdroje a nepodporuje HTTP metody určené k manipulaci, tak jako to chápeme u REST API rozhraní). Vytváření a úprava obrázku probíhá spolu s vytvořením/úpravou daného zdroje a obrázek je zde uveden v kódování Base64 popsaném níže. Výhodou tohoto řešení je snížení počtu požadavků při manipulaci se zdrojem.

Oba uvedené přístupy mají své výhody i nevýhody. Záleží jen na okolnostech použití, zda je obrázek hlavním zdrojem a nadřazený zdroj reprezentuje jen metadata daného obrázku, a nebo zda je důležitější částí zdroj a obrázek je jen jedním z jeho mnoha atributů. V projektu byla zvolena druhá varianta, kdy obrázky nejsou v daném kontextu tak důležité.

Base64

Base64 je kódování sloužící k převodu binárních dat na tisknutelné znaky. Umožní tedy přenos informací kanálem, který podporuje jen tisknutelné ASCII znaky (American Standard Code for Information Interchange). Toto kódování neobsahuje kontrolní mechanismy proti poškození. Zašifrovaná data bývají o cca 33% delší jako je tomu u originálu. Proces kódování probíhá takovým způsobem, kdy se skupiny vstupních bitů o velikosti 24 bitů převádí na výstupní řetězec o délce čtyř znaků [20][10].

7.3.3 REST API Modul

REST API Modul je modul implementující rozhraní. Každý zdroj je reprezentován vlastním presenterem. Tyto presentery obsahují metody pro obsluhu požadavků obsahujících HTTP metody GET, POST, PUT a DELETE.

Metoda `actionRead($id = null, $type = 'json')`

obsluhuje požadavky obsahující HTTP metodu GET. Nepovinnými parametry jsou `id` (identifikátor konkrétního zdroje) a `type` (formát odpovědi). Parametr `id` určuje, zda se bude vypisovat kolekce zdrojů či konkrétní zdroj. Výpis kolekce je ovlivněn tzv. stránkováním uvedeným v kapitole 7.3.5. V některých případech, např. zdroje reprezentujícího odběratele (aplikace WebGet), jsou uvedeny další volitelné parametry, a to identifikátory filtru odběratelů a skupiny odběratelů, které jsou použity při hierarchickém procházení zdrojů. V konkrétním příkladu se jedná o situaci, kdy URL adresa obsahuje dotaz na odběratele konkrétního filtru nebo skupiny odběratelů. Tzn., že i kdyby parametr `id` určoval existujícího odběratele, ale požadovaný filtr/skupina odběratelů by daného odběratele neobsahovala, došlo by k výpisu chyby s informací o neexistenci daného zdroje.

Metoda `actionDelete($id = null, $type = 'json')`

zpracovává požadavek s HTTP metodou DELETE a očekává zadaný identifikátor zdroje, který se má smazat z databáze.

Metoda `actionUpdate($id = null, $type = 'json')`

zpracovává požadavek s HTTP metodou PUT a následně aktualizuje stávající záznam v databázi. V případě, kdy je definována metoda `validateUpdate()`, provádí komponenta kontrolu přijatých dat a až po úspěšné kontrole dochází k volání metody `actionUpdate()`. Tato metoda nejprve nalezne záznam určený k aktualizaci, poté aktualizuje záznam přijatými daty a ukládá ho do databáze.

Metoda `actionCreate($type = 'json')`

je podobná jako `actionUpdate()`. Zpracovává požadavek s HTTP metodou POST. Po kontrole obdržených dat metodou `validateCreate()` vytvoří novou entitu zpracovávaného zdroje a tu naplní přijatými daty. Pokud je vytvářen zdroj, který v rámci hierarchického řazení může obsahovat potomky, je možno uvést identifikátory potomků, se kterými se má vytvořit vazba, a při plnění dat se tyto vazby vytvoří. Jelikož pro plnění a aktualizaci dat se používá stejná metoda, tak tato vlastnost platí i pro aktualizaci záznamu. Podobně jako u metody `actionRead()` i zde mohou být nepovinné parametry navíc, které budou

reprezentovat nadřazené zdroje a uvedením nadřazeného zdroje v URL adrese dojde při vytváření i k vytvoření vazby. Mějme například kategorii s identifikátorem 1. Vytvoření produktu zařazeného do dané kategorie se provede vytvořením požadavku s HTTP metodou POST směřující na URL adresu `.../categories/1/products/`.

7.3.4 Překlad URL adres

Pro správné zpracování požadavků na REST API rozhraní je důležitou částí definice překladu URL adres na presentery a jejich akce v třídě `app\router\RouterFactory`. Základní pravidlo pro přímý přístup ke zdrojům bez nadřazených zdrojů je reprezentováno třídou `Drahak\Restful\Application\Routes\CrudRoute`. Třída pracuje s adresovacím pravidlem `api/v1/<presenter> [/<id>][.<type xml|json>]`, kde jednotlivé části znamenají:

- `api/v1/` - část URL adresy směřující na REST API rozhraní
- `<presenter>/` - určení druhu zdroje a zároveň název voleného presenteru (názvy jsou stejné)
- `[/<id>]` - nepovinný identifikátor zdroje
- `[.<type xml|json>]` - nepovinná volba výstupního formátu

Příkladem takové adresy může být `/api/v1/shopletters/1.xml`, což znamená adresaci zdroje `shopletter` s identifikátorem `1` a požadovaný výstupní formát je `application/xml`. Na dalším příkladu 7.2 je zobrazeno pravidlo pro přístup ke zdroji `subscribers`, kdy se prochází hierarchická struktura. Pravidlo reprezentuje třída `Drahak\Restful\Application\Routes\ResourceRoute`. Tato třída přijímá adresovací pravidlo podobné jako v předchozím příkladu, kde jsou ale uvedeny konkrétní zdroje náležející cestě v hierarchické struktuře. Dalším parametrem třídy je pole s definicí modulu, presenteru a hlavně akcí na dané HTTP metody. Akce na dané metody jsou popsány v předchozí kapitole 7.3.3. Jedinou odlišností je zpracování čtení zdroje (HTTP metoda GET), kdy se v konkrétním případě volá metoda `actionReadNewsletterSubscriberGroup ($id = null, $newsletterId = null, $subscriberGroupId = null, $type = 'json')`, která zkontroluje, zda existuje hierarchická cesta a v kladném případě následně volá metodu `actionRead()`, která přečte požadovaný zdroj.

```
1  $router[] = new ResourceRoute('api/v1/newsletters/
2  <newsletterId>/subscriber-groups/<subscriberGroupId>/
3  subscribers [/<id>] [.<type xml|json>]', array(
4     'presenter' => 'Subscribers',
5     'module' => 'RestApi',
6     'action' => array(
7         IResourceRouter::GET => 'readNewsletterSubscriberGroup',
8         IResourceRouter::POST => 'create',
9         IResourceRouter::PUT => 'update',
10        IResourceRouter::DELETE => 'delete')),
11 IResourceRouter::GET | IResourceRouter::DELETE |
12 IResourceRouter::POST | IResourceRouter::PUT);
```

Ukázka kódu 7.2: Adresovací pravidlo pro přístup ke zdroji subscribers

7.3.5 Stránkování

V rámci přístupu ke kolekcím je nutné řešit otázku tzv. stránkování výsledků. V mnoha případech není možné klientovi poskytnout celou kolekci zdrojů najednou. Problémy mohou nastat jak na straně serveru (datová nebo výpočetní náročnost), tak na straně klienta, kdy nemusí být připraven na přijetí tak velkého výsledku. To by ve výsledku znamenalo výrazně delší dobu zpracování a doručení požadavků. Proto je přistoupeno k řešení výsledky rozdělit na menší části. Poté bude záležet jen na klientovi, zda bude postupně žádat o všechny části nebo se ve finále spokojí jen s částí, jelikož často ani celou kolekci nepotřebuje, ale vystačí si s několika prvními částmi.

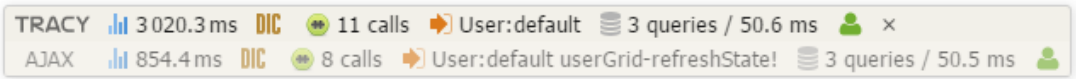
Pro stránkování výsledků neexistuje v rámci protokolu HTTP žádný standardní mechanismus. Zvoleno bylo řešení pomocí query parametru v URL adrese. Implementace tedy kontroluje query parametry a pro stránkování očekává uvedený parametr `offset`. Tento parametr udává, o kolik zdrojů se má posunout výpis kolekce. Maximální velikost kolekce je stanovena na hodnotu 30 prvků. Popsané řešení umožňuje klientovi vypisovat postupně celou kolekci, ale klient nepozná, že obdržená kolekce je již celá a musí provést o jeden dotaz navíc s offsetem nastaveným na větší hodnotu než je velikost kolekce. Nehledě na nutnost potřeby zajistit vytváření korektních URL adres pro další části na straně klienta. Proto pro omezení zbytečných dotazů od klienta a usnadnění práce klientovi server v případě rozdělení výsledku na části do odpovědi přidá hodnoty `next` a `prev`. Tyto hodnoty obsahují URL adresy na další či předchozí část kolekce. Klient se v odpovědi tedy dozví, že neobdržel celou dostupnou kolekci, ale jen část, kterou nalezne na uvedené adrese.

Kapitola 8

Testování

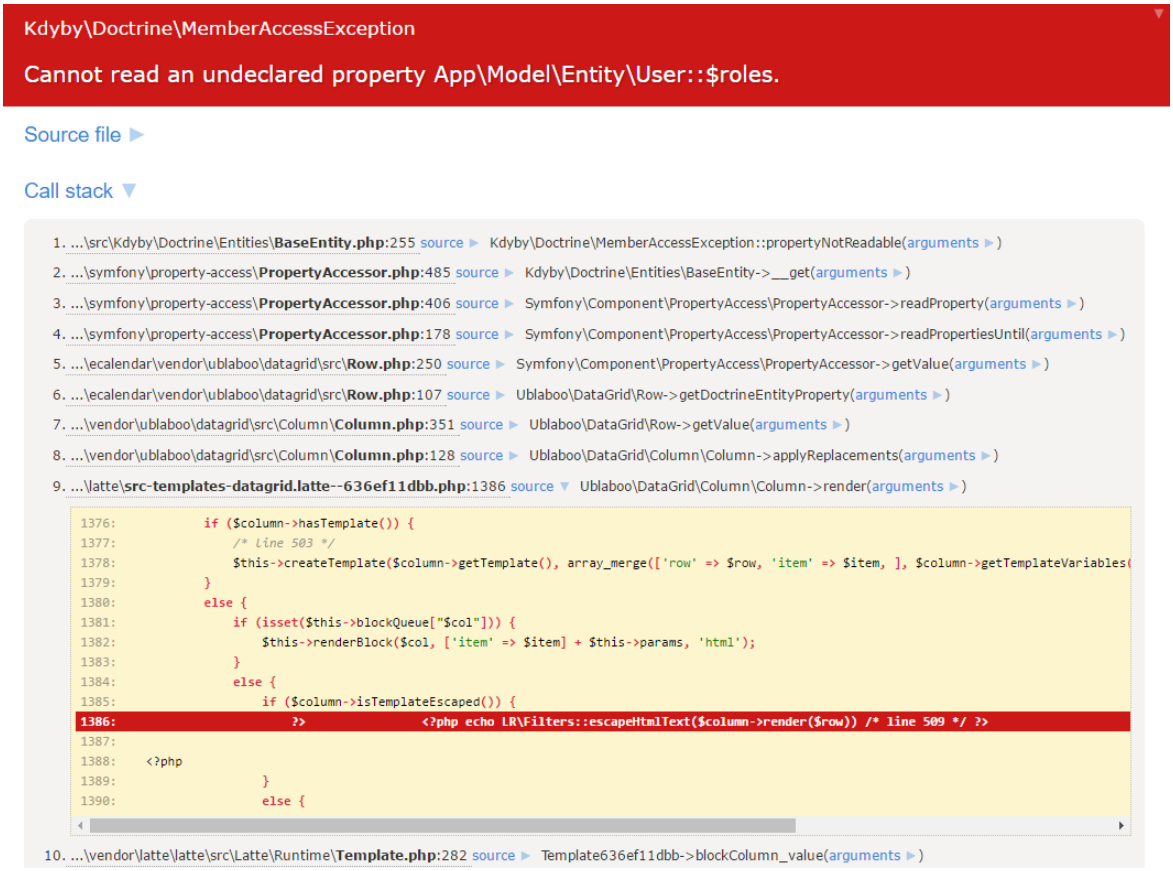
Testování a ladění aplikací probíhalo kontinuálně v průběhu implementace. Většinu chyb, převážně syntaktických, se podařilo odhalit již při psaní v editoru. Použitý editor NetBeans provádí syntaktickou analýzu kódu a dokáže upozornit na chybné konstrukce. Chyby, které neodhalil editor, hlásil interpret jazyka PHP ve spolupráci s knihovnou Nette `Tracy \Debugger` nazývanou laděnka [27]. Při výskytu chyby knihovna informuje o dané chybě a ukáže část zdrojového kódu, kde chyba vznikla. Spolu s chybou zobrazí definované proměnné včetně jejich obsahu, viz obrázek 8.2. Další část sémantických chyb týkajících se databázové vrstvy odhalil Doctrine framework po spuštění příkazu `php ./www/index.php orm:schema-tool:update -force` v terminálu. Úspěšné dokončení uvedeného příkazu je nezbytné pro dokončení procesu tvorby databázových tabulek a vazeb mezi tabulkami.

Dalším užitečným nástrojem laděnky je tzv. debugger bar. Tato plovoucí lišta je umístěna na všech stránkách v pravém dolním rohu. Lišta zobrazuje informace spojené s vykonáváním skriptu, jako je čas trvání běhu skriptu, vyžadované závislosti, přehled volání akcí, použité adresovací pravidlo, zobrazení provedených SQL dotazů do databáze a informace o přihlášeném uživateli. Debugger bar zobrazí nejen SQL dotazy, ale také časovou náročnost každého dotazu. Debugger bar není zapouzdřeným nástrojem a umožňuje využívat různých rozšíření podle potřeb vývojáře. Ukázka podoby lišty je na obrázku č. 8.1.



Obrázek 8.1: Debugger bar [zdroj vlastní]

Popsané ladící nástroje jsou určeny pro vývojové prostředí aplikací. Je značně nevhodné, aby byly aktivovány v produkčním prostředí, tzn., aby návštěvník měl možnost v případě výskytu chyby zjistit části zdrojového kódu, případně jaké SQL dotazy se provádějí při vykreslení stránky. Jelikož by se jednalo o bezpečnostní slabinu, tak problematiku řeší samotná knihovna. Knihovna sama zkontroluje, zda se aplikace spouští z veřejné IP adresy, a všechny chybové zprávy se zapíší do adresáře `log/`. V některých výjimečných případech je ale zapotřebí ladící nástroje spustit i v produkčním prostředí. Nette Framework, respektive knihovna, umožňuje i tuto variantu, kdy se v souboru `app/bootsrap.php` do konfigurace přidá výjimka pro konkrétní IP adresu `$configurator->setDebugMode($ipAdress)`.



Obrázek 8.2: Laděnka Nette framework [zdroj vlastní]

Pro testování REST API rozhraní během vývoje bylo použito rozšíření Advanced REST client [31] pro prohlížeč Google Chrome. Toto rozšíření umožňuje vytvářet dotazy na zvolenou URL adresu zdroje. Samozřejmostí je volba HTTP metody a možnost určit přenášená data. U obsahu požadavku je možno zvolit i formát přenášených dat, kdy se adekvátně ke zvolenému formátu nastaví i HTTP hlavička požadavku. Po odeslání požadavku zobrazuje rozšíření již obdrženou odpověď, kterou zformátuje pro lepší čitelnost. Pro ověření funkcionality rozhraní byl použit vytvořený e-commerce kalendář, který ale nepokryl všechny možnosti rozhraní. Rozšířením tedy byly ověřeny všechny případy včetně chování na chybně uvedené požadavky.

Ověření vzhledu aplikací bylo provedeno ve třech nejpoužívanějších prohlížečích. Stránky byly testovány v prohlížeči Mozilla Firefox verze 52.0.2, Microsoft Edge 38.14393.1066.0 a Google Chrome verze 57.0.2987.133. V těchto prohlížečích se stránky zobrazily korektně dle definovaného grafického návrhu. Zároveň byla kontrola provedena i na smartfonu a tabletu, kde se projevil responzivní layout.

Při ladění aplikace bylo zapotřebí vypisovat obsahy polí a proměnných. K této činnosti byla použita funkce `Debugger::dump()` vycházející z funkce `var_dump()`. Obě tyto funkce dokáží zobrazit vypsané pole včetně indexů pole. Důvodem pro použití funkce `dump()` je hlavně fakt, že bez dalšího zásahu dokáže zobrazit naformátované informace v HTML stránce, zatímco výstup funkce `var_dump()` se slije do nepřehledného bloku textu.

	Přihlášení do aplikace	Změna hesla pro přihlášení	Nalezení události určeného data	Vytvoření úkolu	Úprava úkolu	Zobrazení podrobnosti o události	Změna data události	Změna koncového data události
1	Ano (3)	Ano (5)	Ano (5)	Ano (6)	Ano (6)	Ano (6)	Ano (5)	Ano (6)
2	Ano (2)	Ano (5)	Ano (5)	Ano (5)	Ano (5)	Ano (5)	Ano (4)	Ano (5)
3	Ano (4)	Ano (6)	Ano (5)	Ano (5)	Ano (5)	Ano (5)	Ano (5)	Ano (5)
4	Ano (3)	Ano (5)	Ano (5)	Ano (6)	Ano (6)	Ano (6)	Ano (6)	Ne (6)
5	Ano (5)	Ano (5)	Ano (5)	Ano (6)	Ano (6)	Ano (6)	Ano (6)	Ano (6)
6	Ano (3)	Ano (7)	Ano (5)	Ano (5)	Ano (5)	Ano (5)	Ano (3)	Ano (5)
7	Ano (2)	Ano (5)	Ano (3)	Ano (5)	Ano (5)	Ano (5)	Ano (5)	Ano (5)
8	Ano (3)	Ano (6)	Ano (5)	Ano (5)	Ano (5)	Ano (5)	Ne (5)	Ne (5)
9	Ano (4)	Ano (7)	Ano (5)	Ano (6)	Ano (6)	Ano (6)	Ano (6)	Ano (6)
10	Ano (3)	Ano (5)	Ano (5)	Ano (5)	Ano (5)	Ano (5)	Ano (3)	Ano (5)
11	Ano (4)	Ano (8)	Ano (10)	Ano (5)	Ano (5)	Ano (5)	Ano (5)	Ne (15)
12	Ano (3)	Ano (5)	Ano (5)	Ano (5)	Ano (5)	Ano (5)	Ano (6)	Ano (5)
13	Ano (3)	Ano (9)	Ano (7)	Ano (5)	Ano (5)	Ano (5)	Ano (5)	Ano (5)
14	Ano (3)	Ano (5)	Ano (5)	Ano (5)	Ano (5)	Ano (5)	Ano (5)	Ano (5)
15	Ano (3)	Ano (6)	Ano (5)	Ano (6)	Ano (6)	Ano (6)	Ne (6)	Ne (15)
16	Ano (3)	Ano (8)	Ano (8)	Ano (5)	Ano (5)	Ano (5)	Ano (7)	Ano (5)
17	Ano (3)	Ano (5)	Ano (6)	Ano (5)	Ano (5)	Ano (5)	Ano (7)	Ano (5)
18	Ano (4)	Ano (7)	Ano (5)	Ano (5)	Ano (5)	Ano (5)	Ano (5)	Ano (5)
19	Ano (3)	Ano (10)	Ano (12)	Ano (15)	Ano (15)	Ano (15)	Ne (15)	Ne (20)
20	Ano (2)	Ano (4)	Ano (4)	Ano (5)	Ano (5)	Ano (5)	Ano (1)	Ano (5)

Tabulka 8.1: Tabulka zaznamenaného měření testování aplikace

Aplikace E-commerce kalendář byla podrobena testování na vzorku 20 lidí. Jako způsob testování bylo zvoleno pozorování daných uživatelů při práci s aplikací. Každý z uživatelů dostal sadu úkolů, kterou musel provést. Při provádění dílčích úkolů bylo sledováno, jak se uživatel orientuje v prostředí, kam přejíždí myší, kolikrát a na jaké prvky kliká. V tabulce 8.1 jsou uvedeny jednotlivé úkoly a uživatelé, kde v průniku úkolu a uživatele se vyskytuje informace, zda se daný úkol podařilo dokončit a v závorce počet kliknutí myši.

Po dokončení sady úkolů proběhla s uživateli diskuse. Cílem bylo zjistit, kde má aplikace nedostatky, zda má nějaké zásadní chyby, případně co by jako uživatel tohoto programu uvítal, aby aplikace uměla navíc. Nejčastějšími problémy, na které uživatelé poukazovali, byly:

- Bez předchozího představení není patrné, že je možno s událostmi v kalendáři manipulovat.
- Na první pohled není jasné, o jaký záznam se jedná.

- Barvy událostí EasyShop a WebGet jsou podobné a dochází k záměně.
- Chybí soupis zaznamenaných úkolů. Uvedení jen v kalendáři není dostačující.

Z uvedených podmětů ke zlepšení, které z velké části reflektují tabulku, byly učiněny změny v aplikaci, aby se zpříjemnilo používání aplikace. Konkrétně došlo k úpravě popisu událostí, nyní je před názvem události uveden i druh události. Barvy událostí byly upraveny pro lepší rozpoznání. Byla nově vytvořena stránka se zapsanými úkoly, kde zákazník zjistí všechny úkoly, které má přiřazeny.

Zajímavý fakt byl objeven při pozorování. Činnost vytvoření úkolu většina uživatelů prováděla pomocí odkazu v menu a ne kliknutím do kalendáře, kde se předvyplní zvolené datum. Po představení aplikace již tento problém nenastával. Jelikož budoucím zákazníkům bude aplikace vždy představena, tak nebylo přijato žádné nápravné opatření a nepředpokládá se výskyt podobného problému.

Kapitola 9

Demonstrační aplikace

V této kapitole je popsán postup pro instalaci aplikací na webový server. Pro chod webových aplikací je vyžadován interpret jazyka PHP minimálně verze 7.0. Pro databázový server MySQL je vyžadována minimálně verze 5.6. Nette Framework klade zároveň jisté požadavky na webový server a nastavení PHP. Kontrolu všech požadovaných vlastností lze provést spuštěním PHP skriptu `www/checker/index.php` na daném serveru, který je součástí Nette [29].

9.1 Instalace aplikací

Všechny zdrojové kódy všech aplikací jsou uvedeny na přiloženém CD. Instalace tedy poté spočívá v překopírování zdrojových kódů konkrétní aplikace na webový server. Podle použitého operačního systému na serveru je zapotřebí zkontrolovat a případně nastavit přístupová práva. Po nakopírování zdrojových kódů se založí MySQL databáze a získané přístupové údaje se nastaví v konfiguračním souboru `/app/config/config.neon`. Před prvním spuštěním aplikace je nutné stáhnout všechny php závislosti pomocí nástroje composer (pomocí terminálu v kořenovém adresáři spustit příkaz `composer install`). Pro získání všech front-end balíčků je zapotřebí dalšího nástroje a to Bower, kdy se podobně jako u PHP závislostí v terminálu spustí příkaz `bower install`.

V momentě, kdy jsou všechny závislosti staženy a připraveny, zbývá jen vytvořit tabulky a vazby mezi nimi. O vytvoření databázové struktury se postará framework Doctrine. Nejprve se v terminálu spustí příkaz pro kontrolu validity databázového schématu `php ./www/index.php orm:validate-schema`. Očekávaným výstupem je:

```
1 [Mapping]  OK - The mapping files are correct.
2 [Database] FAIL - The database schema is not in sync with the
                                    current mapping file.
```

Ukázka kódu 9.1: Očekávaný výstup validace schématu

Zkontrolované databázové schéma je nyní možno vytvořit v databázi pomocí příkazu `php ./www/index.php orm:schema-tool:update -force`. Alternativním způsobem je možné si nechat vypsat SQL příkazy do terminálu. To se provede příkazem `php ./www/index.php orm:schema-tool:update -dump-sql`. Tyto vygenerované SQL dotazy vytvářející databázové schéma lze provést v databázi ručně. Proces vytvoření databáze se ověří opětovným spuštěním příkazu `php ./www/index.php orm:validate-schema`, kde očekávaným výstupem je:

```
1 [Mapping]  OK - The mapping files are correct.
2 [Database] OK - The database schema is in sync with the mapping
                                    files
```

Ukázka kódu 9.2: Očekávaný výstup validace schématu po druhé

Posledním krokem pro úspěšné dokončení konfigurace databáze je spuštění příkazu `php ./www/index.php database:init`, který provede vložení výchozích hodnot do databáze.

REST API rozhraní vyžaduje nastavení jak na straně poskytovatelů zdrojů, tak na straně klienta, který by chtěl k daným zdrojům přistupovat. U aplikací EasyShop, WebGet a CentralNews je potřeba vygenerovat tajný klíč pro přístup k rozhraní. Tento klíč se následně nastaví do konfiguračního souboru `/app/config/config.neon`, konkrétně do parametru `Restful: security: privateKey`. Aplikace E-commerce kalendář očekává ve výše zmíněném konfiguračním souboru nastavené parametry s URL adresou na REST API rozhraní konkrétní aplikace a její tajný klíč, který bude sloužit k podpisu zpráv. Aby aplikace automaticky stahovala události a přenášela změny událostí do zdrojové aplikace, je potřeba nastavit opakované spouštění skriptu (jednou za hodinu), který nalezneme na adrese `http://adresa_serveru/ecalendar/www/api/`.

9.2 Popis aplikací

Tato kapitola popíše vytvořené aplikace z pohledu použití. Všechny aplikace jsou vizuálně i ovládajícími prvky podobné. Prototypy aplikací EasyShop, WebGet a CentralNews neobsahují veřejnou část, ale pouze administrativní. Všechny aplikace tedy na počátku při spuštění nejprve zobrazí přihlašovací obrazovku a teprve po úspěšném přihlášení se zobrazí dané aplikace popsané níže.

9.2.1 EasyShop, WebGet a CentralNews

Po přihlášení do aplikací se uživatel dostane na úvodní obrazovku, kde nalezne panel s odkazy na jednotlivé zdroje. Podobné odkazy jsou k nalezení v menu, které je integrováno do horního pruhu. V levé části pruhu se nachází název aplikace, který slouží jako odkaz na úvodní obrazovku. V levé části se pak nachází rozbalovací pole Menu, ve kterém je odkaz v podobě informací o přihlášeném uživateli. Tento odkaz vede na detail přihlášeného uživatele, kde je možno provést změnu informací o uživateli. Dále může pole Menu na základě role, kterou má uživatel přiřazenu, obsahovat odkazy na přehled všech uživatelů systému. Posledním polem je možnost odlišení se z aplikace. Jak už bylo uvedeno výše tak vedle pole Menu se nacházejí odkazy na jednotlivé zdroje aplikace.

Při přechodu na libovolný zdroj se zobrazí tabulka s výpisem prvních 20 záznamů v databázi. Tato tabulka disponuje mnohými funkcemi. Je možné procházet všechny záznamy pomocí stránkování. V případě hledání konkrétního záznamu je možné využít filtrovacích polí uvedených v daných sloupcích. Pro lepší čitelnost dat v tabulce je zde možnost výběru, jaké sloupce se mají zobrazit. Tato volba se provede pomocí tlačítka v pravém horním rohu aplikace. Vedle tohoto tlačítka se nachází další tlačítko po přechod na formulář sloužící k vytvoření nového záznamu. Každý záznam má v posledním sloupci nazvaném Akce umístěny tlačítka s odkazy na zobrazení detailu záznamu, upravení nebo smazání záznamu. Detail záznamu se zobrazí na nové stránce, kde jsou vypsány všechny dostupné informace. Pro lepší orientaci ve vypsaných informacích je v pravé části stránky umístěn výpis atributů

záznamu, které slouží jako rychlé odkazy na dané informace, ale i jako navigace, tzn. při posunutí stránky na danou informaci se odkaz zvýrazní. Nad výpisem atributů jsou umístěny odkazy na úpravu a případné smazání záznamu podobně jako v tabulce. Odkaz na úpravu záznamu vede na stejný formulář jako je odkaz pro vytvoření nového záznamu, s tím rozdílem, že nyní je formulář předvyplněn vybraným záznamem. U všech formulářů probíhá online validace zadaných validačních pravidel, tudíž po vyplnění daného pole se provede kontrola a uživatel je ihned informován o výsledku validace. V případě, kdy uživatel na pole s definovanými pravidly neklikne a rovnou odešle formulář (např. při odeslání prázdného formuláře) dojde ke kontrole všech prvků a v případě chyby se odesílání formuláře zablokuje a nalezené chyby jsou sděleny uživateli.

9.2.2 E-commerce kalendář

Při vstupu do aplikace se uživateli po přihlášení zobrazí v horní části lišta s polem Menu. Tato lišta obsahuje název aplikace, který slouží jako odkaz na výchozí obrazovku. Pole Menu obsahuje informace o přihlášeném uživateli s možností přejít na detail daného uživatele. V případě, že přihlášený uživatel disponuje oprávněními administrátora, se v poli zobrazí nové odkazy na přehled uživatelů aplikace a odkaz pro vytvoření nového uživatele. Předposledním odkazem je možnost zaznamenání nového úkolu do kalendáře. Posledním odkazem, který se zde nachází, je odkaz pro odhlášení z aplikace.

Na úvodní stránce se nachází kalendář, ve kterém jsou vypsány události platné pro zobrazovaný časový úsek. V levé části je zobrazen druhý kalendář sloužící pro rychlou navigaci, jelikož umožňuje listovat po měsících, letech, desetiletích i stoletích. Výběrem konkrétního data dojde k překreslení velkého kalendáře na požadované datum. Nad velkým kalendářem jsou umístěny ovládací prvky. Nalevo se nacházejí prvky pro procházení v kalendáři a rychlá volba pro přesun na aktuální datum. V pravé části jsou umístěny prvky měnící pohled kalendáře. Výchozím pohledem je zobrazení celého měsíce, nicméně je možno zobrazit události po konkrétních dnech nebo týdnech. Jako poslední možnost je zobrazení agendy, což je soupis všech událostí do seznamu. Na všechny zobrazené události je možno v kalendáři kliknout, což vyvolá zobrazení okna s podrobnějšími informacemi o konkrétní události. Pokud je zvolenou událostí úkol, zobrazí se kromě detailu i tlačítko vedoucí na úpravu daného úkolu. Nové úkoly lze vytvářet nejen pomocí odkazu v položce Menu, ale i kliknutím do kalendáře ve zvolený den. Po kliknutí se zobrazí formulář pro vytvoření úkolu náležejícímu přihlášenému uživateli. V případě, kdy je zapotřebí přidělit úkol někomu jinému, je třeba zvolit cestu pomocí odkazu v Menu, kde se vybírá i vlastník/osoba zodpovědná za úkol.

V pohledu kalendáře zobrazujícího celý měsíc je možno pracovat s časovými údaji událostí. Přesunutím události (drag and drop) se změní i počáteční a koncové datum události. V případě, kdy je zapotřebí upravit jen konec události, postačí najet myší na konec události tak, aby se zobrazil místo ikony myši posuvník, a danou událost podle potřeby roztáhnout. O veškerých zaznamenaných změnách je uživatel informován ve stavovém řádku, který se zobrazí nad kalendáři. O přenos změn z e-commerce kalendáře do aplikací vlastnících dané zdroje se již uživatel nestará, jelikož aplikace automaticky jednou za hodinu veškeré změny synchronizuje.

Kapitola 10

Závěr

Tato práce se zabývá návrhem a implementací API rozhraní pomocí stylu REST pro již stávající aplikace EasyShop, WebGet a CentralNews. Dále byla navržena a implementována aplikace e-commerce kalendář, která využívá navrženého rozhraní a díky němu automaticky získává z aplikací významné události, které pak následně přehledně zobrazí do kalendáře. Data těchto událostí je možné upravit a aplikace se sama automaticky postará o přenesení změn zpět do aplikací, kterým daná událost náleží. Z důvodu neporušení firemního tajemství zdrojových kódů aplikací EasyShop, WebGet a CentralNews byly vytvořeny prototypy sloužící pouze k demonstraci funkčního REST API rozhraní.

Na začátku práce je popsán World Wide Web a jeho nejdůležitější části. Zmíněn je způsob identifikace zdrojů (URI, URL, URN) a protokol HTTP zejména z pohledu syntaxe dotazů, které poslouží jako základ návrhu API rozhraní. Možných protokolů pro návrh a realizaci rozhraní je několik. V práci jsou uvedeny ty nejvýznamnější, a to XML-RPC, SOAP a REST. Největší prostor byl věnován požadovanému architektonickému stylu REST, jelikož byl vybrán pro návrh. Důvodem výběru daného protokolu byly jeho vlastnosti, které v případě dodržení požadavků dávají značný potenciál výkonnosti celého rozhraní. Návrh rozhraní pro dané aplikace probíhal tak, že byl nejprve uveden redukovaný doménový model, který posloužil jako základ pro návrh URL adres sloužících k identifikaci zdrojů a tvorbě prototypů aplikací.

Na základě požadavků firmy byla vytvořena neformální specifikace aplikace e-commerce kalendář, která posloužila jako základ pro analýzu aplikace a návrhu diagramu případů použití. V poslední části návrhu kalendáře je stanovena podoba schématu relační databáze a uveden sekvenční diagram popisující komunikaci jednotlivých aplikací.

Před částí zabývající se implementací aplikací je rozebrán výběr vhodné platformy. Výběr technologií závisel částečně na požadavku firmy, který rozhodl o použití programovacího jazyka PHP a databázový systém MySQL. Kromě použitých technologií jsou zde uvedeny i alternativy jako např. PHP framework Symfony.

Implementace aplikací byla uskutečněna pomocí PHP softwarové struktury Nette využívající návrhovou architekturu MVP, která vychází z architektury MVC. Framework ovlivnil i vlastnosti vyvíjených aplikací, které byly do jisté míry podobné, jako je např. adresářová struktura vycházející z doporučované podoby, tzv. sandbox. U aplikace e-commerce kalendář byl popsán proces přihlášení, který je identický i pro prototypy aplikací. Byla popsána vytvořená komponenta FullCalendar, která zajišťuje vykreslení kalendářů a událostí, zpracování akcí s danými událostmi atd. Je zde věnována kapitola i implementaci modulu, který zajišťuje napojení na vytvořené REST API rozhraní a následnou zprávu dat v databázi lokální aplikace. U prototypů aplikací EasyShop, WebGet a CentralNews byla popsána

komponenta pro zobrazení dat v tabulce spolu s možností filtrování, stránkování a práce se zobrazovanými daty. Hlavní část byla věnována rozhraní REST API. Rozebrány jsou zde možné varianty zabezpečení, kde byla pro implementaci zvolena varianta pomocí podepsaného požadavku. Dále je uvedena obecná struktura pro přístup ke konkrétnímu zdroji. V posledních částech je popsán způsob zpracování adresace zdrojů a stránkování poskytovaných kolekcí, které zabraňuje nevhodnému vytěžování výpočetních zdrojů a komunikační linky.

Proběhlo i testování aplikace na vzorku 20 dobrovolníků, u kterých byl měřen počet kliknutí myší a zda splnili předem uvedený úkol. Na základě naměřených výsledků a zejména ze zpětné vazby byly provedeny změny v aplikaci e-commerce kalendář, aby byla aplikace vůči uživatelům vstřícnější.

Výsledkem je tedy aplikace a REST API rozhraní. Výhodami aplikace jsou nízké požadavky na výpočetní výkon a na úložiště a snadná implementace dalších funkcí. Předností je také vyšší rychlost odezvy aplikace, která nemusí čekat na výsledek požadavku vzniklého při úpravě události. Veškeré změny událostí jsou totiž zaznamenávány lokálně a aplikace nezávazně na práci uživatele jednou za učitý čas (v řešeném projektu jednou za hodinu) tyto změny automaticky odešle do zdrojových aplikací. Přehlednost aplikace podporuje i objektový návrh včetně vrstvy Doctrine zajišťující ukládání objektů do databáze. Vzhled aplikace působí moderním minimalistickým dojmem, kdy byl kladen důraz na co největší přehlednost aplikace. Uvedené vlastnosti potvrzuje použitý front-end framework Bootstrap, který navíc umožňuje snadné zapracování změn, které mohou vést např. k úplnému předělání grafického rozvržení stránek aplikace, aniž by bylo zapotřebí zásadně měnit kód vykreslovaných komponent. Jelikož neexistuje přesný návod nebo předpis, jak takové rozhraní má vypadat, bylo rozhraní navrženo a implementováno takovým způsobem, aby využívalo co nejvíce doporučených rad [25][36]. Výhodami rozhraní je bezesporu možnost komunikovat pomocí více textových formátů a výpis všech nalezených chyb do odpovědi (ne jen první nalezené). Tou nejdůležitější výhodou je možnost snadného rozšíření rozhraní o další zdroje bez potřeby jakkoliv upravovat stávající řešení (tedy pokud se jedná o nezávislý zdroj).

Možnosti dalšího rozvoje aplikace e-commerce kalendář jsou rozsáhlé. O jaké konkrétní rozšíření se bude jednat rozhodnou zákazníci, kteří budou danou aplikaci používat, konkurence na trhu, kde je zapotřebí reagovat na nejnovější trendy a v neposlední řadě i legislativní stránka věci. Možným rozšířením by mohla být nová funkce kalendáře, která umožní seskupovat různé události do jednoho bloku. Poté by úprava bloku automaticky upravila i dílčí události. Zákazník by například v aplikaci EasyShop měl naplánované uvedení nového produktu na konkrétní datum. Zároveň na stejné datum by bylo nastaveno v aplikaci CentralNews odeslání e-mailových a SMS zpráv klientům, informujících o novém produktu. V kalendáři by uživatel tyto události poté sloučil do jednoho bloku. Tím by se zobrazované informace v kalendáři zpřehlednily a v případě, že by se dodavatel zpozdil s dodávkou produktu, tak by postačilo pouze upravit datum u vytvořeného bloku a aplikace by automaticky danou změnu distribuovala dále.

Literatura

[1] *MySQL profesionálně optimalizace pro vysoký výkon*. Brno: Zoner Press, vyd. 1. vydání, 2009, ISBN 978-80-7413-035-9.

[2] Berners-Lee, T.: *Uniform Resource Locators (URL)*. 1994, [Online; cit. 20.11.2016].
URL https://tools.ietf.org/html/rfc1738

[3] Berners-Lee, T.: *Uniform Resource Identifier (URI): Generic Syntax*. 2005, [Online; cit. 20.11.2016].
URL http://www.ietf.org/rfc/rfc3986.txt

[4] Bhattal, G.: *Difference between URL, URI and URN*. [Online; cit. 21.12.2016].
URL http://bitpoetry.io/difference-between-url-uri-and-urn/

[5] Box, D.: *Simple Object Access Protocol (SOAP) 1.2*. [Online; cit. 20.11.2016].
URL http://www.w3.org/TR/soap/

[6] Chaffer, J.; Swedberg, K.: *Mistrovství v jQuery [kompletní průvodce vývojáře]*. Brno: Computer Press, první vydání, 2013, ISBN 978-80-251-4103-8.

[7] Chow, S.-W.: *XML pohotová referenční příručka : referenční příručka programátora ke XML, XPath, XSLT, XML Schema, SOAP a dalším*. Grada, vyd. 1. vydání, 2006, ISBN 80-247-0972-4.

[8] Chow, S.-W.: *Programujeme Mashup aplikace pro Web 2.0 v PHP*. Kopp, vyd. 1. vydání, 2008, ISBN 978-80-251-2057-6.

[9] Daigle, L.: *Uniform Resource Names (URN) Namespace Definition Mechanisms*. 2002, [Online; cit. 20.11.2016].
URL https://tools.ietf.org/html/rfc3406

[10] Dostálek, L.: *Velký průvodce protokoly TCP/IP*. Praha: Computer Press, druhé vydání, 2003, ISBN 80-722-6849-X.

[11] Fielding, R.: *Hypertext Transfer Protocol – HTTP/1.1*.

[12] Fielding, R. T.: *Architectural styles and the design of network-based software architectures*. Dizertační práce, University of California,Irvine, 2000, vedoucí práce Richard Taylor.

[13] Frystyk, H.: *The World-Wide Web*. 1994, [Online; cit. 20.11.2016].
URL https://www.w3.org/People/Frystyk/thesis/WWW.html/

[14] Group, T. P.: PHP Data Objects. [Online; cit. 23.04.2017].
URL http://php.net/manual/en/book.pdo.php

[15] Gudgin, M.: *SOAP Version 1.2 Part 1: Messaging Framework*. [Online; cit. 20.11.2016].
URL http://www.w3.org/TR/soap12-part1/

[16] Hardt, D.: The OAuth 2.0 Authorization Framework. 2012, [Online; cit. 23.04.2017].
URL https://tools.ietf.org/html/rfc6749

[17] Jakoubě, J.: Seriál: ORM test PHP frameworků (9 dílů). 2013, [Online; cit. 11.04.2017].
URL https://www.zdrojak.cz/serialy/test-php-frameworku/

[18] Janda, P.: DataGrid. 2015, [Online; cit. 23.04.2017].
URL https://ublaboo.org/datagrid/

[19] Janovský, D.: Hosting s PHP. 2017, [Online; cit. 11.04.2017].
URL https://www.jakpsatweb.cz/katalog/hosting-php.html

[20] Josefsson, S.: *The Base16, Base32, and Base64 Data Encodings*. 2003, [Online; cit. 23.04.2017].
URL https://tools.ietf.org/html/rfc3548.html

[21] Kofler, M.: *Mistrovství v MySQL 5 [kompletní průvodce webového vývojáře]*. Brno: Computer Press, vyd. 1. vydání, 2007, ISBN 978-80-251-1502-2.

[22] Kofler, M.; Öggl, B.: *PHP 5 a MySQL 5 - průvodce webového programátora*. Brno: Computer Press, vyd. 1. vydání, 2007, ISBN 978-80-251-1813-9.

[23] Kosek, J.: *Využití webových služeb a protokolu SOAP při komunikaci*. [Online; cit. 21.12.2016].
URL http://www.kosek.cz/diplomka/html/websluzby.html

[24] Malek, M.: Restful. [Online; cit. 23.04.2017].
URL https://github.com/drahak/Restful

[25] Massé, M.: *REST API design rulebook*. Sebastopol, CA: O'Reilly, první vydání, 2012, ISBN 978-144-9310-509.

[26] Moats, R.: *URN Syntax*. 1997, [Online; cit. 20.11.2016].
URL https://tools.ietf.org/html/rfc2141

[27] Nette Foundation: Debugování a zpracování chyb. [Online; cit. 23.04.2017].
URL https://tracy.nette.org/cs/

[28] Nette Foundation: Nette Framework - Rychlý a pohodlný vývoj webových aplikací v PHP. [Online; cit. 11.04.2017].
URL http://nette.org/cs

[29] Nette Foundation: Požadavky Nette Framework. [Online; cit. 23.04.2017].
URL https://doc.nette.org/cs/2.4/requirements

[30] Peterson, J.: Bootstrap 3 Date/Time Picker. 2015, [Online; cit. 23.04.2017].
URL https://github.com/Eonasdan/bootstrap-datetimepicker/

[31] Pszty ć, P.: Advanced REST client. 2017, [Online; cit. 23.04.2017].
URL https://chrome.google.com/webstore/detail/advanced-rest-client/hgmloofddffdnphfgcellkdfbfbjeloo?hl=en-US&utm_source=ARC

[32] Pužmanová, R.: *TCP/IP v kostce*. Computer Press, druhé vydání, 2009, ISBN 978-80-7232-388-3.

[33] Sharkie, C.; Fisher, A.: *Responzivní webdesign - okamžitě*. Brno: Computer Press, první vydání, 2015, ISBN 978-80-251-4384-1.

[34] solid IT: DB-Engines Ranking - popularity ranking of database management systems. 2017, [Online; cit. 11.04.2017].
URL https://db-engines.com/en/ranking

[35] Stewart, W.: *Mosaic - The First Global Web Browser*. [Online; cit. 19.11.2016].
URL http://www.livinginternet.com/w/wi_mosaic.htm

[36] Sturgeon, P.: *Build APIs you won't hate: everyone and their dog wants an API, so you should probably learn how to build then.* [s.l.], první vydání, 2015, ISBN 978-0692232699.

[37] Sun Microsystems, I.: *RPC: Remote Procedure Call*. [Online; cit. 20.11.2016].
URL https://www.ietf.org/rfc/rfc1057.txt

[38] Surveys, W. W. W. W. T.: Usage of server-side programming languages for websites. [Online; cit. 11.04.2017].
URL https://w3techs.com/technologies/overview/programming_language/all

[39] Symfony: Symfony - High Performance PHP Framework for Web Development. [Online; cit. 11.04.2017].
URL https://symfony.com/

[40] team, C.: Bootstrap - The world's most popular mobile-first and responsive front-end framework. 2011, [Online; cit. 11.04.2017].
URL http://getbootstrap.com/

[41] Team, D.: Doctrine Project. 2006, [Online; cit. 11.04.2017].
URL http://www.doctrine-project.org/

[42] Tichý, J.: Seriál: Doctrine 2 (12 dílů). 2010, [Online; cit. 11.04.2017].
URL https://www.zdrojak.cz/serialy/doctrine-2/

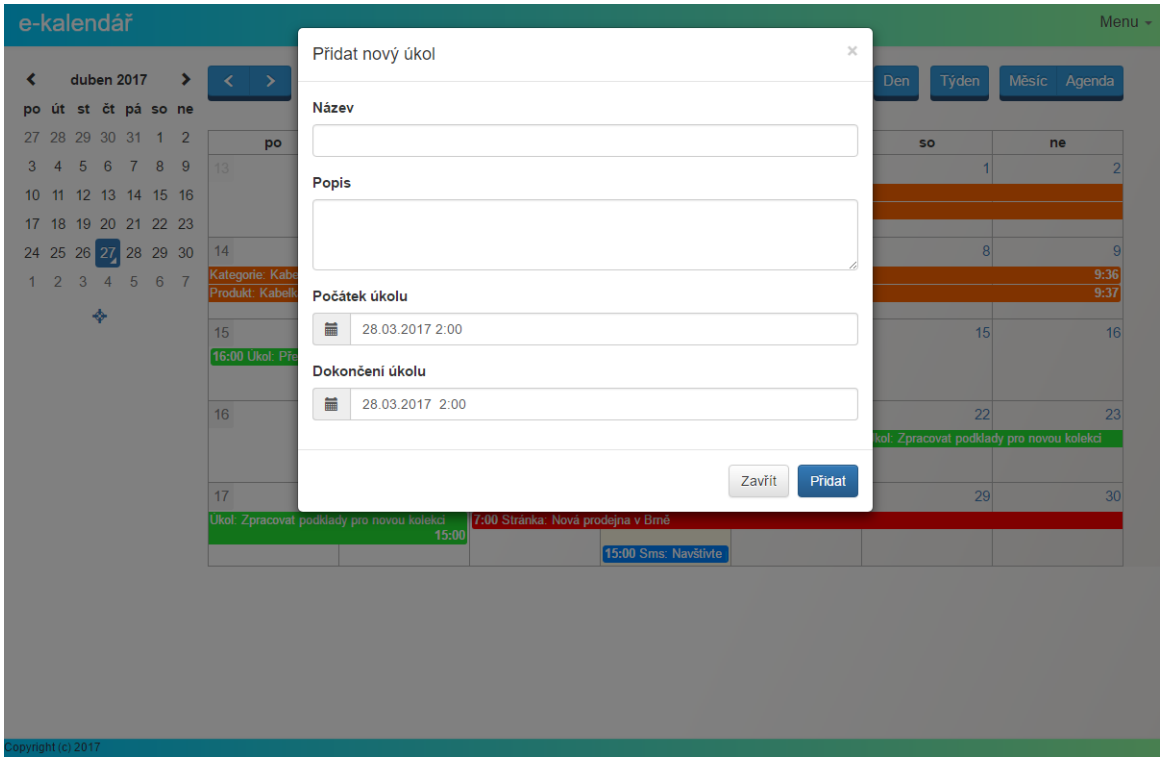
[43] Vrána, J.: *1001 tipů a triků pro PHP*. Brno: Computer Press, vyd. 1. vydání, 2010, ISBN 978-80-251-2940-1.

Přílohy

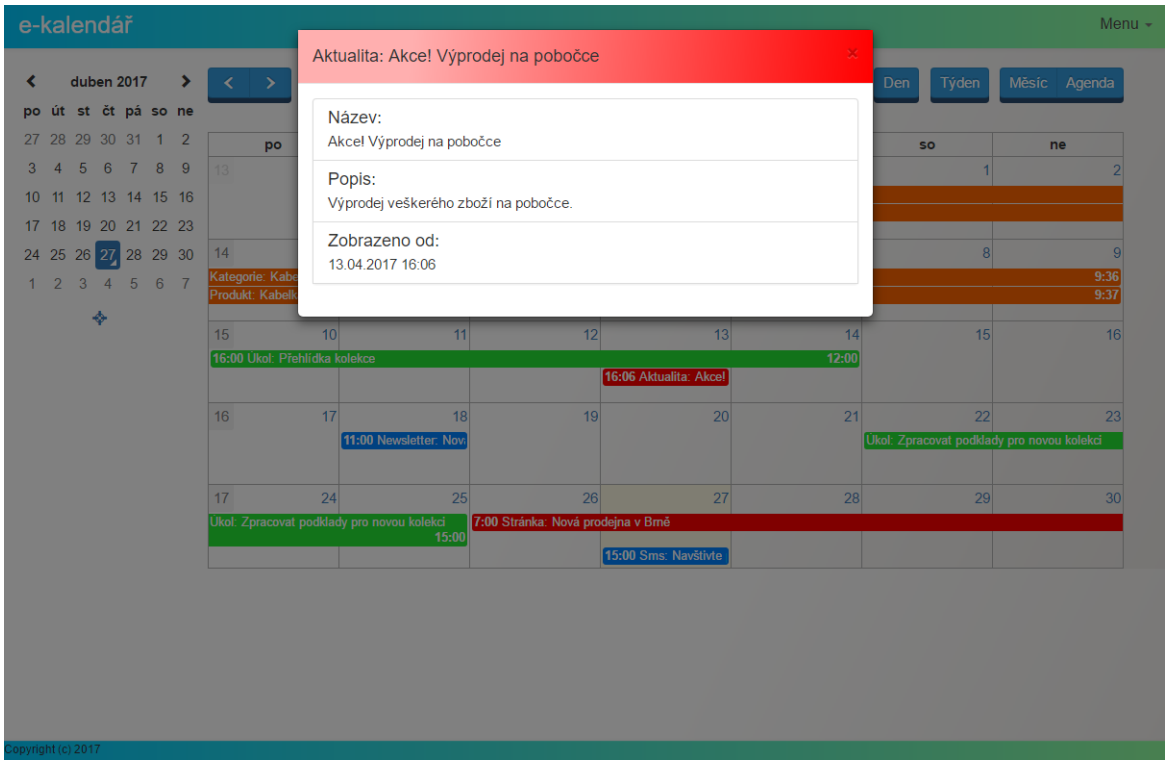
Příloha A

Ukázky vzhledu aplikace E-commerce kalendář

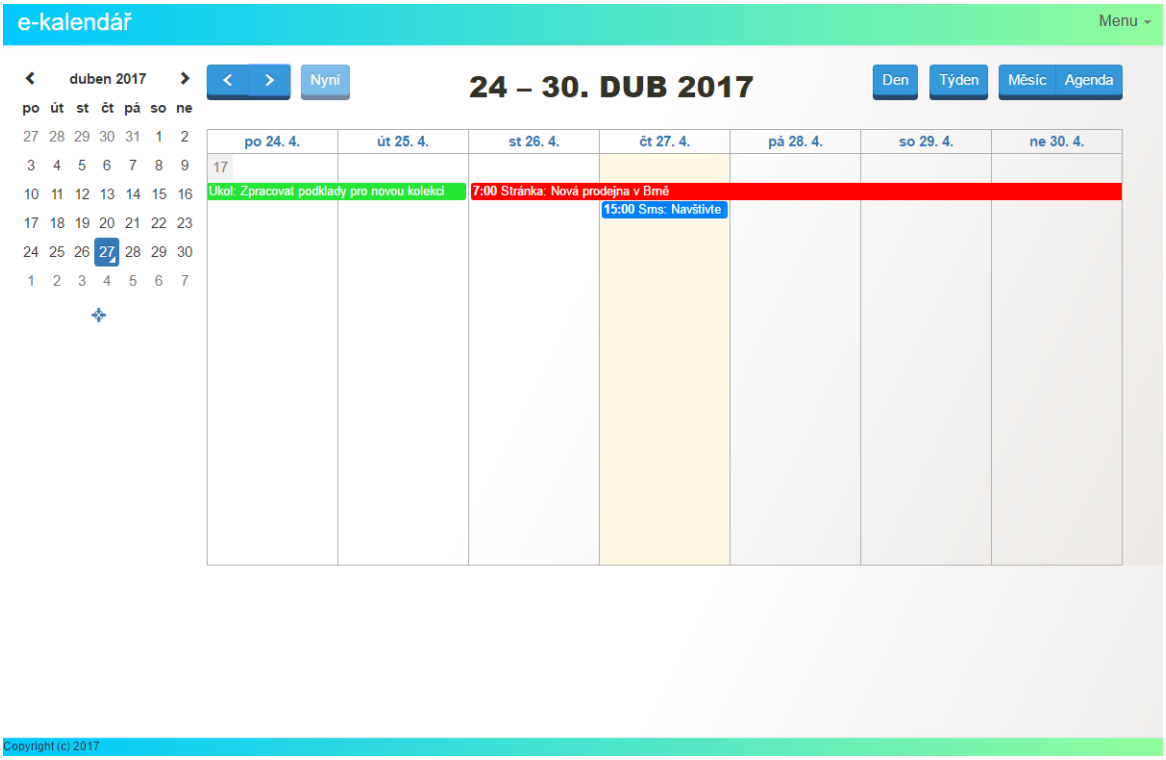
Obrázek A.1: Úvodní stránka [zdroj vlastní]



Obrázek A.2: Rychlé vytvoření úkolu [zdroj vlastní]

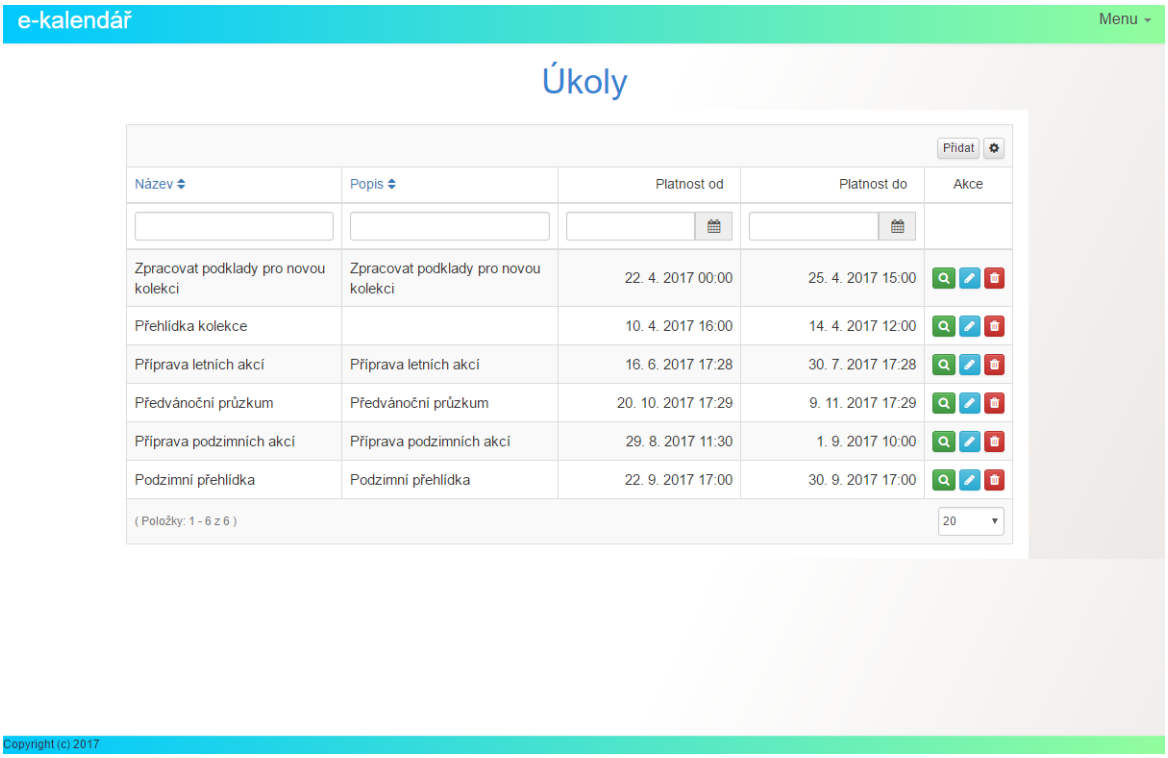


Obrázek A.3: Detail vybrané události [zdroj vlastní]

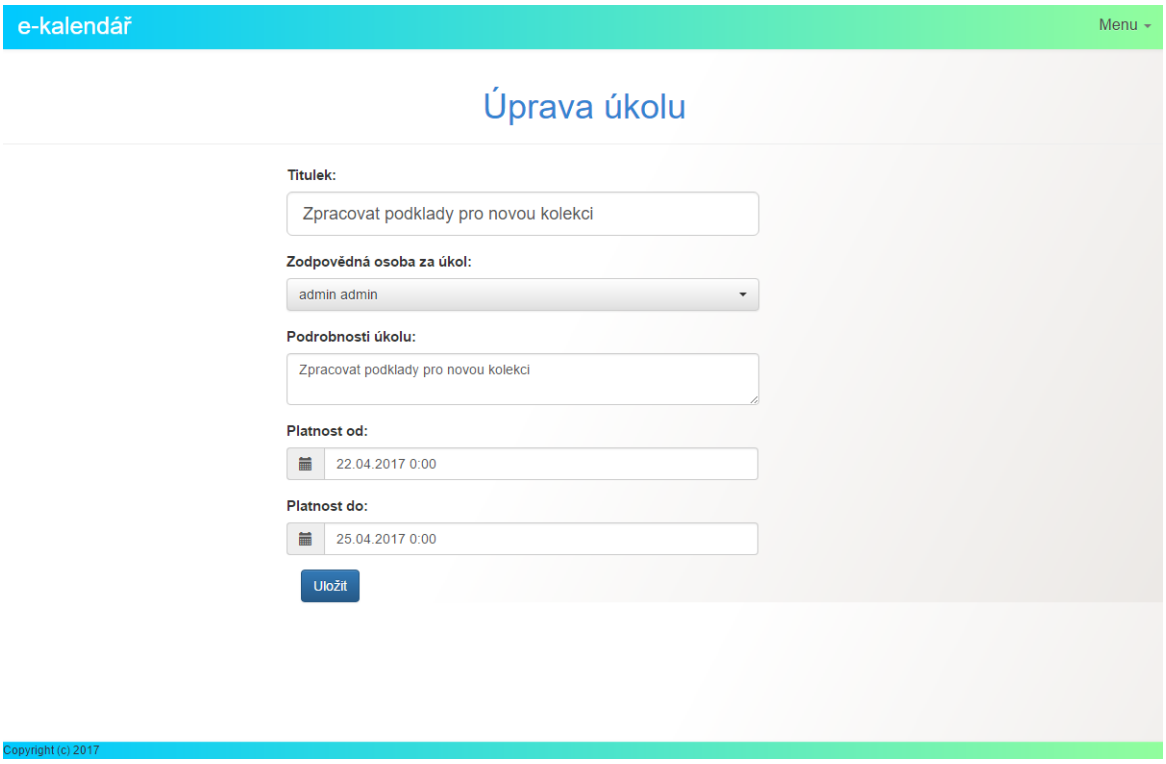


Obrázek A.4: Pohled kalendáře týden [zdroj vlastní]

Obrázek A.5: Pohled kalendáře agenda [zdroj vlastní]



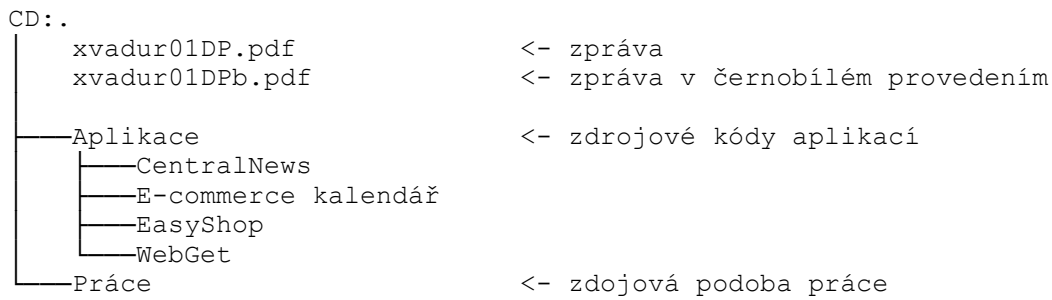
Obrázek A.6: Soupis všech úkolů [zdroj vlastní]



Obrázek A.7: Úprava úkolu [zdroj vlastní]

Příloha B

Obsah přiloženého CD



Obrázek B.1: Obsah přiloženého CD [zdroj vlastní]