VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA PODNIKATELSKÁ
FACULTY OF BUSINESS AND MANAGEMENT

ÚSTAV INFORMATIKY
INSTITUTE OF INFORMATICS

INOVACE BEZPLATNÉHO REGISTRU JÍZDNÍCH KOL
INNOVATION OF THE FREE BIKE REGISTRY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. Michal Petričko

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Jan Luhan, Ph.D., MSc

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA PODNIKATELSKÁ

Zadání diplomové práce

Ústav:	Ústav informatiky
Student:	**Bc. Michal Petričko**
Studijní program:	Systémové inženýrství a informatika
Studijní obor:	Informační management
Vedoucí práce:	**Ing. Jan Luhan, Ph.D., MSc**
Akademický rok:	2019/20

Ředitel ústavu Vám v souladu se zákonem č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů a se Studijním a zkušebním řádem VUT v Brně zadává diplomovou práci s názvem:

Inovace bezplatného registru jízdních kol

Charakteristika problematiky úkolu:

Úvod
Cíle práce, metody a postupy zpracování
Teoretická východiska práce
Analýza současného stavu
Vlastní návrhy řešení
Závěr
Seznam použité literatury
Přílohy

Cíle, kterých má být dosaženo:

Na základě analýzy současného stavu projektu navrhnout inovaci webového portálu i samotného projektu registru jízdních kol se zaměřením na oblast technického řešení platformy a s vazbou na marketingové aktivity.

Základní literární prameny:

BOAG, P. User Experience Revolution. 1st ed. Freiburg: Smashing Media AG, 2017. 219 p. ISBN 978-3-945749-54-8.

DUCKETT, J. HTML and CSS: Design and Build Websites. 1st ed. Indianapolis: WILEY, 2011. 512 s. ISBN 978-1-118-00818-8.

KUBÍČEK, M. 50 způsobů, jak získat zpětný odkaz. 2. vyd. Karviná: PRONETmedia, s.r.o., 2012. 90 s. ISBN 978-80-87721-01-8.

ŘEZÁČ, J. Web ostrý jako břitva. 1. vyd. Jihlava: BAROQUE PARTNERS s.r.o., 2014. 216 s. ISBN 978-80-87923-01-6.

Documentation | Nette Framework, 2019 [online]. Nette Foundation. [cit. 10. 11. 2019]. Dostupné z: https://doc.nette.org

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2019/20

V Brně dne 29.2.2020

L. S.

doc. RNDr. Bedřich Půža, CSc. ředitel	doc. Ing. et Ing. Stanislav Škapa, Ph.D. děkan

ABSTRAKT

Táto diplomová práca sa zaoberá vytvorením webovej aplikácie s využitím Nette frameworku pre projekt KradenaKola.cz. Teoretické východiská sú využité k popísaniu technológií použitých pri vývoji. Na základe analýzy požiadaviek je vytvorená celá webová aplikácia po funkčnej aj vizuálnej stránke.

ABSTRACT

This diploma thesis deals with the creating of a web apllication using the Nette framework for the KradenaKola.cz project. Theoretical background is utilized to describe the technologies used in development. Based on analysis of the request, the entire web application is created both functionally and visually.

KLÚČOVÉ SLOVÁ

Webová aplikácia, vývoj, PHP, HTML, CSS, Nette, Bootstrap, framework, register

KEYWORDS

Web application, development, PHP, HTML, CSS, Nette, Bootstrap, framework, register

BIBLIOGRAFICKÁ CITACE

PETRIČKO, Michal. *Inovace bezplatného registru jízdních kol* [online]. Brno, 2020 [cit. 2020-05-17]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/125605. Diplomová práce. Vysoké učení technické v Brně, Fakulta podnikatelská, Ústav informatiky. Vedoucí práce Jan Luhan.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že předložená diplomová práce je původní a zpracoval/a jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem ve své práci neporušil/a autorská práva (ve smyslu Zákona č. 121/2000 Sb., o právu autorském a o právech souvisejících s právem autorským).

V Brně dne 17. května 2020

……

podpis autora

POĎAKOVANIE

Týmto by som sa rád poďakoval Ing. Janovi Luhanovi, Ph.D., MSc za konzultácie a vedenie mojej diplomovej práce, Ing. Zbyňkovi Dufkovi za možnosť sa podieľať na vývoji webovej aplikácie pre projekt KradenaKola.cz a v neposlednom rade Ing. Vojtěchovi Sedláčkovi za neoceniteľné rady, čas a pripomienky v oblasti programovania.

OBSAH

ÚVOD

Webové aplikácie sa stali behom posledných rokov neoddeliteľnou súčasťou firiem alebo rôznych projektov. V dnešnom svete sa kladie veľký dôraz na ich rýchly a efektívny vývoj. V oblasti webových aplikácií neustále prichádzajú nové trendy, a preto je potrebné aby boli pripravené na zmeny, či rozšírenia. Vývoj webových aplikácii je mnohokrát nikdy nekončiaci proces tímovej práce, založený na inkrementálnych prírastkoch a iteráciách. K tomuto rýchlemu vývoji softvéru slúžia tzv. agilné metodiky.

Jednou z mnoho webových aplikácii vzniknutých pomocou agilnej metodiky je projekt KradenaKola.cz., ktorý vznikol na základe veľkého množstva krádeži bicyklov. Cieľom tohto projektu je, aby majitelia dobrovoľne a bezplatne evidovali svoje bicykle na serveri KradenaKola.cz. Zámerom je taktiež odhaľovanie predajcov kradnutých bicyklov a obmedzovanie trestnej činnosti. Vďaka registru bicyklov si jednoducho môže ktokoľvek overiť či v databáze je daný bicykel označený ako kradnutý. Ak je kradnutý, jednoduchou cestou je možné kontaktovať majiteľa bicykla a spraviť tak dobrú vec.

Keďže sa jedná o bezplatný register, firma uvažuje o zavedení platenej inzercie na webovú stránku a rozšíriť webovú aplikáciu o nové funkcionality v blízkej dobe. Súčasná webová aplikácia projektu od roku 2014 neprešla žiadnou technickou zmenou, používa zastaralé technológie a nesplňuje podmienky určené pre rozširovanie funkcionalít do budúcnosti. Preto sa firma rozhodla o inováciu projektu KradenaKola.cz, ktorej súčasťou je aj vytvorenie novej webovej aplikácie postavenej na najnovších technológiách.

CIELE PRÁCE, METÓDY A POSTUPY SPRACOVANIA

Táto diplomová práca je vytvorená v súlade s požiadavkami firmy DesignBeat s.r.o., ktorá sa rozhodla pre inováciu svojho projektu KradenaKola.cz.

Cieľom práce je na základe analýzy projektu navrhnúť inováciu webovej aplikácie pre projekt KradenaKola.cz so zameraním na oblasť technického riešenia platformy s väzbou na marketingové aktivity.

Na inovácii projektu sa podieľal 5 členný tím, ktorý sa riadil pri vývoji metodikou SCRUM. V tíme som zastával rolu junior programátora a mal som na starosti vytvorenie webovej aplikácie, to znamená programovanie back-endovej a front-endovej časti.

Pre funkčnú časť aplikácie bol využitý Nette framework a programovací jazyk PHP. Pre uschovávanie dát je použitá pôvodná MySQL databáza. Aby sa zachoval charakteristický dizajn projektu KradenaKola.cz bol použitý pôvodný grafický návrh prispôsobený Bootstrap frameworku, ktorý rieši hlavne responzivitu webovej aplikácie.

Prvá časť práce je zameraná na teoretické popísanie technológií použitých pri vývoji webovej aplikácie.

Nasleduje analytická časť, kde je predstavený samotný projekt KradenaKola.cz, analýza požiadavkou na aplikáciu a samotný výber použitých technológií.

Jadro celej práce predstavuje praktická časť, ktorá je venovaná celému vývoju aplikácie pomocou SCRUM metodiky. Praktická časť obsahuje prípravu webového serveru, databáze a konkrétne riešenia aplikácie popísané na určitých úlohách.

1 TEORETICKÉ VÝCHODISKA

V teoretickej časti sú popísané použité technológie, prístupy a metodiky pri tvorbe projektu a všetky pojmy potrebné k vypracovaniu práce.

1.1 HTML

Jazyk HTML je základom pre tvorbu webových stránok. Z anglického „Hypertext transfer markup laguage“, ktorý v preklade znamená hypertextový značkovací jazyk. Je to jeden zo spôsobov, ako definovať a interpretovať značky podľa pravidiel SGML (Standard Generalized Markup Language), ktoré sú definované normou ISO 8879 (1).

Každá z verzii HTML je určená hlavne pre tvorbu hypertextových dokumentov. Hypertextové súbory sú tvorené ako obyčajný text, preto nie je problém tieto dokumenty prečítať v akomkoľvek operačnom systéme. Jedná sa teda o univerzálny a platformovo nezávislý (1).

HTML sa skladá zo sady elementov, ktoré popisujú formátovanie web stránky, a z textu, ktorý tvorí obsah web stránky. Tieto elementy sa oddeľujú od bežného textu ostrými zátvorkami (<body>). Rozlišujeme párové a nepárové elementy. Párové elementy sú zložené z 2 častí, zo začiatočného ( ) a koncového ( ). Párové elementy hlavne ohraničujú text, ktorý bude formátovaný. Nepárové elementy tvoria iba jeden element (napr.
). Pre spresnenie vlastností elementov sa používajú atribúty. Atribúty sa zapisujú do začiatočných alebo nepárových elementov (napr.). Hodnoty v HTML sa môžu, ale aj nemusia zapisovať medzi úvodzovky (1).

HTML stránku môžeme rozdeliť na dve hlavné časti. Prvá časť je hlavička, ktorá začína elementom <head> a končí </head>. Do hlavičky sa zapisujú metadáta, informácie o stránke, titulok stránky, skripty atď. Informácie z hlavičky sa na stránke nezobrazujú. Telo stránky sa zapisuje do medzi párové elementy <body> a </body> a obsahuje text stránky spolu s elementmi, ktoré stránku formátujú (1).

Momentálne najnovšou verziou od W3C je HTML 5.3, ktoré vzniklo 18. októbra 2018. V tejto verzii sa naďalej objavujú nové funkcie, ktoré pomáhajú autorom webových aplikácii (2).

1.2 CSS

Kaskádové štýly

CSS (Cascading Style Sheets) vznikli ako nástroj pre formátovanie textu a vzhľadu pre HTML alebo XHTML dokumentov. Výhodou je oddelenie prezentačnej časti stránky od formátovacej časti, čo sprehľadní a zjednoduší údržbu. Nie je za potreby používať špecializovaný software, keďže CCS sú tvorené len obyčajným textom. Pomocou CSS môžeme presne nastaviť rozmery, farby pozadia alebo umiestnenie elementov, čo sa používa na vytvorenie rozloženia elementov na stránke bez používania tabuľkového web dizajnu. V CSS sa dá ďalej upravovať farba, typ poprípade veľkosť textu. Obsahuje aj dynamické vlastnosti, zmenu vzhľadu stránky pre rôzne výstupné zariadenia (3).

Rovnako ako pre HTML tak aj pre vývoj CSS sa stará spoločnosť W3C. Vývoj CSS prebieha na základe úrovní. To znamená, že každá ďalšia úroveň, respektíve „level“ pridáva a rozširuje nižšie úrovne (4).

Prvé CCS vzniklo v roku 1996 spoločnosťou W3C. Označuje sa ako CSS level 1, poprípade CSS 1. Táto verzia ponúka možnosť definície väčšej časti dnes často používaných a základných vlastností, do ktorých patrí nastavenie farby, textu, formátovania (5).

V roku 1998 vychádza CSS level 2 (CSS2), ktorý obsahuje naviac oproti predošlej úrovni podporu pre lepšiu prácu s formátovaním pomocou vlastností position a display (6). Umožňuje presnejší výber vďaka selektorom. Je teda možné upraviť vlastnosti elementu img, pokiaľ je potomkom elementu p a u ostatných elementov img ponechať ich pôvodné vlastnosti (7). O 4 roky neskôr vychádza revízia pre level 2, ktorá opravuje chyby v tejto verzii. Označuje sa ako CCS 2.1 (8).

V nasledujúcich verziách CSS sa spoločnosť W3C rozhodli ísť inou cestou. CSS sa začalo členiť na jednotlivé moduly. Pre každý modul je vývoj osobitný a je nezávislý na ostatných moduloch. Je preto možné, že niektoré moduly budú hotové skôr na ďalšom leveli ako ostatné moduly (4).

Preprocesory

Pre uľahčenie práce s kaskádovými štýlmi sa používajú preprocesory. CSS preprocesor je software, ktorý si za vstup berie CSS kód obohatený o syntax určitého preprocesoru,

ktorého výstupom je čistý CSS kód. Preprocesor nerozširuje CSS o ďalšie funkcie, neumožňuje formátovanie, či úpravu vzhľadu stránky. Toto všetko zvláda CSS. Jedná sa hlavne o zjednodušenie práce vývojárov (9).

CSS preprocesorov je mnoho. Každý obsahuje rôzne funkcie pre rôzne prípady. Je len na každom vývojárovi, ktorý si vyberie pre svoju prácu. Medzi najpoužívanejšie patrí napríklad Sass, Less alebo Styllus (9).

1.3 JavaScript

JavaScript je programovací jazyk založený na objektovo-orientovanej koncepcii. Jadro jazyka sa syntaxou podobá jazykom C, C++ poprípade Jave. Má potlačenú typovú kontrolu a mnohé myšlienky preberá z programovacieho jazyka Perl (10).

JavaScript pri svojom začiatku používal názov LiveScript. Zmena názvu prebehla tesne pred uvedením na trh. Dôvodom bol iba marketing. Za vývojom stáli spoločnosti Netscape a Sun Microsystems. Pod názvom JScript sa tento jazyk vyskytuje v produktoch od Microsoft (10).

V roku 1997 bol spoločnosti ECMA(European Computer Manufacturers Association) predložený JavaScript 1.1 ako návrh pre štandardizáciu. Po pár úpravách vznikol štandard ECMA – 262 definujúci nový skriptovací jazyk ECMAScript (11).

Implementáciou JavaScriptu do webového prehliadača vznikol takzvaný klientský JavaScript, ktorý používa okrem univerzálnych rysov jazyka aj (11) :

- **DOM(Document Object Model)** - poskytuje rozhranie pre prácu s obsahom dokumentov (11).
- **BOM(Browser Object Model)** - poskytuje rozhranie pre interakciu s prehliadačom (11).

Jadro jazyka obohatené o rôzne knižnice je možné používať aj mimo prehliadače, napríklad na desktopových aplikáciách, poprípade v PDF dokumentoch. JavaScript je možné využiť aj na strane serveru. LiveWire od spoločnosti Netscape uvedený v roku 1996 bol prvou implementáciou na strane serveru (11).

Pri prístupe na webovú stránku sa klientovi posiela v HTML okrem vlastného obsahu a kaskádových štýlov často aj zdrojový kód v JavaScripte, ktorý klient, teda prehliadač,

interpretuje a vykoná. JavaScript dodáva statickým stránkam určitú dynamiku. Dá sa meniť obsah stránky aj jej vzhľad, napríklad umiestnením kurzoru myši na menu sa zvýrazní alebo zobrazí určitá položka. Používa sa taktiež k validácii formulárov. Užívateľ sa po vyplnení poľa hneď dozvie, že napríklad zadal údaj v zlom formáte. Nemusí sa tak čakať na odpoveď zo strany serveru (11).

Problém môže nastať v prípade ak si užívateľ vypne JavaScript vo svojom prehliadači keďže beží na strane klienta. Tým pádom by sa nedostal k položkám v menu. Väčší problém by nastal ak by sa tvorcovia webu spoľahli iba na JavaScriptovú validáciu formulárov. V tomto prípade by boli v databáze nezmyselné údaje. Preto sa JavaScript berie hlavne ako doplnok (11).

1.4 JQuery

JQuery je open source JavaScriptová knižnica, ktorá výrazne zjednodušuje zápis JavaSkriptového kódu, funkcií, metód, prácu s elementmi na stránke a prácu s AJAX-om. Vo väčšine prehliadačoch pracuje rovnako. JQuery knižnicu vytvoril John Resig a verejnosti ju predstavil na počítačovej konferencií BarCamp v New Yorku v roku 2006. JQuery dokáže (12):

- vyberať a hľadať HTML elementy,
- manipulovať s HTML elementmi a ich atribútmi,
- manipulovať s CSS hodnotami elementov,
- spúšťať sa na základe udalostí,
- vytvárať efekty a animácie,
- presúvať sa medzi elementmi,
- AJAX,
- rozširovať sa pomocou doplnkov (12).

Pre využívanie JQuery knižnice je potrebné ju pripojiť k dokumentu. Pripája sa rovnako ako externé JavaScriptové kódy. JQuery knižnica sa môžem pridať ako JavaScriptový súbor, ktorý sa stiahne na stránke jquery.com alebo pomocou CND (Content delivery network), čo je externý server poskytujúci JQuery knižnicu (12).

Výhodou CND je, že pri návšteve stránky sa JQuery knižnica načíta do cache pamäte a pri ďalšej návšteve stránok sa tento súbor už znovu nemusí sťahovať. Nevýhodou je nutnosť pripojenia na internet. Preto je dobré mať uloženú JQuery knižnicu v súbore spolu so stránkami a v prípade nenačítania z externého servera sa použije (12).

1.5 PHP

PHP (Hypertext preprocessor) je open source interpretovaný, skriptovací, programovací jazyk, ktorý pracuje na strane servera a to tak, že prehliadač pošle požiadavku na server, na servery sa podľa PHP skriptov vygeneruje celá stránka a tá sa pošle klientovi. Klient dostane len statický HTML dokument bez scriptov a bez toho, aby vedel, aké scripty boli v PHP použité. PHP vytvoril v roku 1994 Rasmus Lerdorf ako jednoduchý systém na počítanie prístupov na stránku. (13).

Predchodca PHP, PHP Tools je v podstate sada CGI skriptov napísaných v jazyku C. V roku 1995 boli zverejnené zdrojové kódy PHP Tools. Od tejto doby bolo PHP niekoľkokrát úplne prepísané a premenované (14).

Vývoj išiel rýchlo dopredu, PHP 2.0 už malo v sebe zabudované prostriedky pre správu cookies a podporuje taktiež databázy. PHP 3.0, ktorá vyšla v roku 1998, pridáva podporu objektovo orientovaného programovania. Novinkou bola taktiež možnosť PHP hosťovať na platformách Windows a Macintosh. V roku 2000 vychádza PHP 4.0, ktoré ponúka lepší výkon, prácu s HTTP sessions a bufferovanie výstupov (14).

Výhody PHP sú, že sa dá spustiť na všetkých serverových platformách a väčšine operačných systémov, jednoduchá komunikácia s databázami, má vysokú podporu u poskytovateľov webhostingu, má širokú podporu komunity, a keďže je spustený len na servery užívateľ ho nemôže vypnúť (13).

Na to, aby PHP stránky sa zobrazovali v prehliadači správne, je potrebné ich umiestniť na web server, ktorý podporuje PHP alebo nainštalovať do vlastného operačného systému vlastný server s PHP, čo umožní spúšťanie PHP stránok z lokálneho disku. Ak by tento server nebol nainštalovaný, v prehliadači by boli vidieť len PHP scripty, prípadne aj spolu s HTML kódom (13).

Syntax PHP scriptov je podobná programovaciemu jazyku C. V PHP súbore sú zapísané PHP scripty samostatne alebo spolu s HTML kódom. Scripty sa od HTML kódu oddeľujú dvojicou značiek „ <? ” a „?>” alebo „<?php” a „?>” a jednotlivé inštrukcie scriptu sa oddeľujú dvojbodkou na konci. PHP používa aj premenné, pred každou premennou sa zapisuje znak „$“. Tiež umožňuje prácu s funkciami, s podmienkami, s cyklami, s objektmi, s poľami a databázami (13).

Pomocou PHP sa dajú vytvoriť rôzne dynamické stránky a aplikácie. Najčastejšie sú to internetové obchody, sociálne siete, diskusné fóra, vyhľadávače, spravodajské stránky, počítadlá prístupov, ankety a mnoho ďalších (13).

1.6 PhpStorm

PhpStorm je komerčné multiplatformové IDE (integrované vývojové prostredie) pre PHP vytvorené českou spoločnosťou JetBrains. Slúži ako editor pre PHP, HTML a JavaScript, ktorý pomáha v analýze kódu, prevencií chýb. Podporuje PHP od verzie 5.3. Užívatelia môžu prostredie rozšíriť o rôzne pluginy (15).

1.7 Composer

Composer je programová utilita určená pre správu knižníc a iných zdrojov (závislostí) v PHP. Umožňuje užívateľovi deklarovať pre každý svoj PHP projekt, ktoré závislé knižnice chce využívať a tie potom za neho jednoduchým a zjednoteným spôsobom spravuje (16).

To zahrňuje napríklad:

- stiahnutie knižníc v prípade, že neexistujú,
- zistenie nových verzií knižníc a ich prípadná aktualizácia,
- kontrola požiadavkou pre každú knižnicu,
- riadenie automatického nahrávania tried spravovaných knižníc v PHP (class autoloading(16).

1.8 PHP frameworky

V súčasnosti webový vývojári potrebujú vytvárať komplexné webové stránky a aplikácie a nad určitou úrovňou zložitosti to môže spôsobiť príliš veľa ťažkostí ako aj časovú náročnosť. Preto vznikla potreba štruktúrovanejšieho a prirodzeného spôsobu rozvoja. Toto riešenie pre vývojárov poskytujú PHP frameworky, ktoré prácu uľahčia a zefektívnia proces vývoja webových aplikácií (14).

Výhody PHP frameworkov :

- rýchlosť,
- dobre organizovaný, opakovane použiteľný a udržiavateľný kód,
- bezpečnosť,
- oddelenie prezentácie od logiky (model MVC),
- podpora moderných postupov vývoja, ako sú napríklad objektovo orientované programovacie nástroje (14).

1.8.1 Laravel

Aj keď Laravel je relatívne nový framework (vydaný v roku 2011) pre PHP, patrí medzi najpopulárnejší medzi vývojármi. Laravel má obrovský ekosystém s okamžitou hostiteľskou a nasadzovanou platformou a jeho oficiálne webové stránky ponúkajú mnoho screencast tutoriálov nazývaných aj Laracast (17).

Laravel má mnoho funkcii, ktoré umožňujú rýchly vývoj aplikácií. Má aj svoj vlastný „light-weight teplating engine“ nazývaný „Blade“. Elegantná syntax, ktorá ulahčuje úlohy, ktoré často treba urobiť ako je napríklad autentifikácia, sessions, queueing, caching alebo restful routing. Taktiež obsahuje aj miestne rozvojové prostredie s názvom Homestead (17).

1.8.2 Symfony

Komponenty frameworku Symfony používajú mnohé pôsobivé projekty ako sú napríklad systém správy obsahu Drupal alebo softvér fóra phpBB. Symfony má širokú komunitu vývojárov a mnohých fanúšikov (17).

Symfony komponenty sú opakovateľne použiteľné knižnice PHP, s ktorými sa dajú plniť rôzne úlohy ako je tvorba formulárov, konfigurácia objektov, smerovanie, overovanie, templating alebo mnoho ďalších. Mnoho komponentov sa dá nainštalovať pomocou Composer PHP dependency manager. Webová stránka Symfony má sekciu, kde si môžeme pozrieť rôzne projekty od vývojárov postavené na základe tohto frameworku (17).

1.8.3 Nette

V Českej republike je najznámejším a jedným z najviac používaných PHP frameworkov Nette (18).

Vlastnosti :

- dokonalá bezpečnosť vďaka revolučnej technológií, ktorá eliminuje výskyt bezpečnostných dier,
- ladiaci nástroj Tracy,
- excelentný výkon,
- neustále sa rozširujúca ponuka pluginov a doplnkov,
- premyslený a čistý objektový návrh,
- vlastný šablónovací systém Latte (18).

História

Framework Nette začal David Grudl v roku 2004 vyvíjať ako vlastný projekt. Celý projekt aj vývoj viedol sám až do januára 2014, kde v rozhovore pre internetový server Zdroják oznámil, že s vývojom končí. Dôvodom bola slabá účasť komunity na vývoji (19).

V roku 2006 zverejnil Grundl niektoré časti Nette, ucelený framework bol ale predstavený až na konci roku 2007, pričom oficiálne vyšiel vo verzii 0.7 v januári nasledujúceho roku. Nikdy nevyšla verzia 1.0 a po verzii 0.9.6 mala ďalšia stabilná verzia označenie 2.0, ktorá vyšla v roku 2012 (19).

Vo verzii 2.0 pribudlo mnoho zmien ako sú napríklad nové makrá v šablónach, atribútové makrá, pre konfiguráciu sa začal po prvýkrát používať NEON miesto INI. Zmeny boli

taktiež v smerovaní a mapovaní URL a v nástroji „Tracy“, ktorá slúži na hľadanie chýb. Pribudla taktiež nová vrstva pre prácu s databázou (19).

Revolúcia v podobe modularizácie prišla vo verzii 2.2. Od tejto verzie sa framework rozdelil do samostatných komponentov. Sú v ňom napríklad komponenty pre formuláre, prácu s medzi pamäťou, databázou ako aj pre hľadanie chýb, ktoré je možné používať samostatne bez použitia Nette. V súčasnosti je nette framework vo verzii 3.0. využívajúci všetkých výhod PHP od verzie 7.1 (20).

Latte

Latte je šablonovací systém pre PHP, ktorí šetrí čas a zabezpešuje výstup. Vkladanie PHP úsekov medzi HTML elementy nepraktické a preto poskytuje Latte množstvo makier a filtrov, ktoré výrazne spríjemňujú prácu v šablóne (21).

V Latte existujú 2 druhy makier. Prvé sú makrá, ktoré sa vkladajú do zložených zátvoriek a dá sa pomocou nich urýchliť výpis premenných, cykly atď. Druhý druh makier sú tzv. n:makra. Sú to špeciálne kará, ktoré sa zapisujú do vnútra HTML elementu. Ďalej môžeme k vypísaniu premenných pripojiť filter. Vďaka filtrom je možné ľahko upraviť podobu, v akej bude premenná vypísaná (21).

Tracy

Tracy je debugovacia knižnica Nette, ktorá pomáha rýchlo odhaliť a opraviť chyby, zaznamenávať ich, prehľadne vypisovať premenné a merať čas operácií. Vypísanie chyby je značne prehľadnejšie a konkrétnejšie než chyba, ktorú vypísal interpret jazyka PHP (22).

Ďalším užitočným nástrojom, ktorý Tracy poskytuje je Debugger Bar, čo je plávajúci panel v spodnom rohu stránky. Poskytuje nám informácie o skripte, ako je čas a náročnosť prevedenia skriptu, systémové informácie, prevedené dotazy do databáze a ich časová náročnosť, aktuálnu routovaciu masku a zoznam všetkých definovaných v poradí podľa priority a informácie o práve prihlásenom používateľovi. Debugger bar je možné rozširovať o ďalšie funkcie (22).

1.9 MVC

Model-View-Controller je softwarová architektúra, ktorá vznikla z potreby oddeliť u aplikácií s grafickým rozhraním kód obsluhy (controller) od kódu aplikačnej logiky (model) a od kódu zobrazujúceho dáta (view). Nielenže aplikáciu sprehľadňuje, ale aj uľahčuje budúci vývoj a umožňuje testovanie jednotlivých častí zvlášť (22).

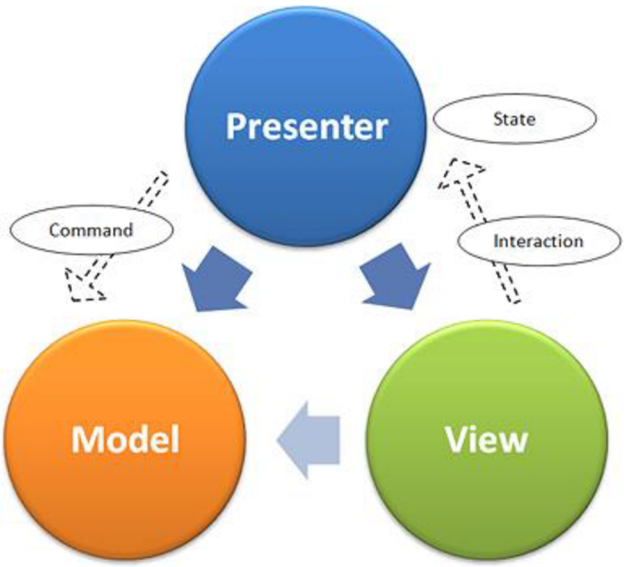
- **Model** – je dátový a hlavne funkčný základ celej aplikácie. Je v ňom obsiahnutá aplikačná logika. Akákoľvek akcia používateľa (prihlásenie, vloženie zbožia do košíka, zmena hodnoty v databázy) predstavuje akciu modelu. Model si spravuje svoj vnútorný stav a von ponúka pevne dané rozhranie. Volaním funkcie tohto rozhrania môžeme zisťovať či meniť jeho stav. Model o existencií view alebo controlleru nevie (22).
- **View** – je vrstva aplikácie, ktorá má na starosti zobrazenie výsledku požiadavkou. Obvykle používa šablónovací systém a vie ako sa má zobraziť potrebný komponent alebo výsledok získaný z modelu (22).
- **Controller** – radič, ktorý spracováva požiadavky používateľa a na ich základe následne volá patričnú aplikačnú logiku (model) a potom požiada view o vykreslenie dát. Obdobou kontrolórov v Nette Framework sú presentery (22).

MVP

Framework Nette je postavený na architektúre MVP (Model-View-Presenter), ktorá je založená na modelu MVC. Presenter v Nette frameworku má podobnú rolu ako controller v architektúre MVC. Vyberá náhlad (view) a predáva mu model alebo dáta z modelu. Udržuje jeho stav perzistentných premenných. A predovšetkým spracováva reakcie používateľa. Tie sa dajú rozdeliť na 3 typy (22):

- zmena pohľadu,
- zmena stavu,
- príkaz modelu (22).

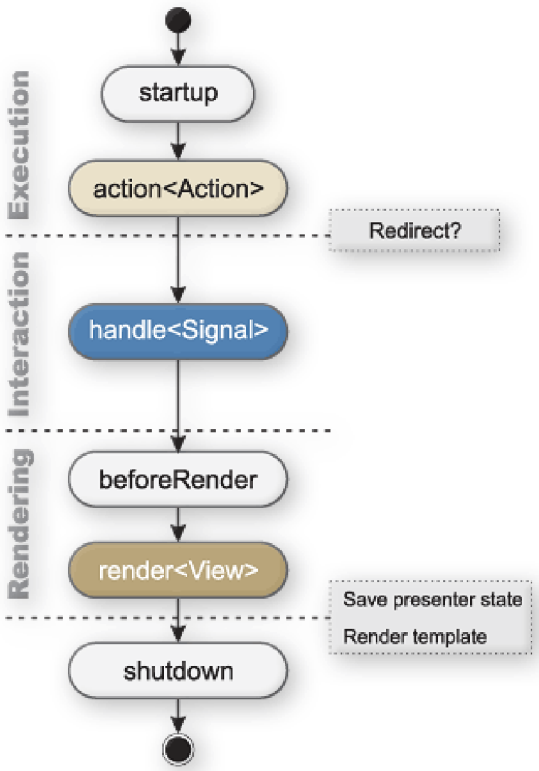
Každý presenter obsahuje sadu metód, ktoré sa volajú v určitom poradí pri každom načítaní stránky. Môžeme preto robiť akcie ešte predtým než sa stránka vykreslí (23).



Obr. 1: Architektúra Model-View-Presenter v Nette Framework
(Zdroj: 22)

Akcia presenteru spôsobí zavolanie metódy render < >, teda napríklad rederShow. To však nie je jediná metóda, ktorá sa volá. Pri písaní presentrov môžeme mimo iné vytvoriť aj nasledujúce metódy (23):

- **startup()** – ihneď po vytvorení presenteru sa zavolá metóda startup(). Tá inicializuje premenné alebo overí užívateľské oprávnenia,
- **action<Action>()** – podobný metóde render<View>(). Sú situácie kedy presenter prevedie určitý úkon a potom presmeruje inam. Dávať názov render metóde, ktorá nič nekreslí, je zbytočné, preto existuje alternatíva s názvom action,
- **handle<Signal>()** – metóda spracováva tzv. signály alebo subrequesty. Určené hlavne pre komponenty a spracovanie AJAXových požiadavkou. Na rovnakej úrovni tiež prevádza spracovanie formulárov,
- **beforeRender()** – volá sa pred známou metódou render<View>() a môže obsahovať napríklad nastavenie šablóny, predanie premenných spoločných pre viac view a podobne,
- **render<View>()** – obvykle nasype do šablóny dáta,
- **shutdown()** – je vyvolaná pre ukončenie životného cyklu presenteru (23).

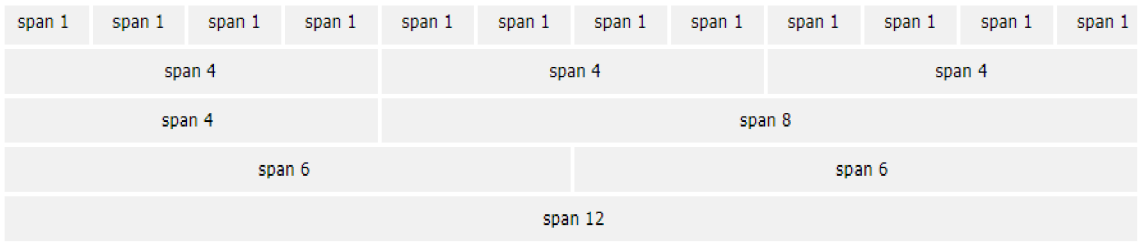


Obr. 2: Životný cyklus presenteru
(Zdroj: 23)

1.10 Bootstrap

Bootstrap je HTML, CSS a JavaScriptový framework pre ľahkú tvorbu vzhľadu pre web, ktorý sa dynamicky mení podľa aktuálnej veľkosti obrazovky a používaného zariadenia. Takémuto vzhľadu sa hovorí responzívny. Rovnako ako u Nette Frameworku sa jedná o open source software, ktorý môže ktokoľvek využívať a modifikovať. Obsahuje rozsiahlu zbierku HTML a CSS komponentov a radu jQuery pluginov, pripravených pre okamžité použitie (24).

Najvýznamnejším komponentom Bootstrapu pre tvorbu responzívneho webu je jeho Grid systém. Jeho hlavnou myšlienkou je rozloženie obsahu stránky na sériu stĺpcov a riadkov pomocou preddefinovaných tried (25).



Obr. 3: Bootstrap 4 Grid System
(Zdroj: 25)

Každý riadok sa rozdeľuje na 12 stĺpcov, ktoré bude element zaberať a s meniacou sa veľkosťou obrazovky zariadenia je veľkosť každého stĺpca zmenená o rovnakú hodnotu, tým pádom ostáva pomer medzi elementmi na riadku totožný. Pre docielenie úplnej responzivity, naberá trieda stĺpca ďalšieho atribútu, ktorý určuje, koľko bude na akom zariadení zaberať stĺpec. Každá trieda stĺpca má potom tvar „.col-xx-yy.“ , kde yy predstavuje počet stĺpcov s rozsahom 1-13 a xx predstavuje množinu hodnôt (25):

- **.col-** pre extra malé zariadenia so šírkou menšou ako 576px,
- **.col-sm-** pre malé zariadenia so šírkou obrazovky rovnou alebo väčšou ako 576px,
- **.col-md-** pre stredné zariadenia so šírkou obrazovky rovnou alebo väčšou ako 768px,
- **.col-lg-** pre veľké zariadenia so šírkou obrazovky rovnou alebo väčšou ako 992px,
- **.col-xl-** pre extra veľké zariadenia so šírkou obrazovky rovnou alebo väčšou ako 1200px (25).

1.11 XAMPP

XAMPP je v informatike označenie pre multiplatformový softwarový balíček vyvinutý firmou Apache Friends.. Obsahuje voľne dostupný otvorený software. XAMPP jednoducho pre vývojárov vytvorí lokálny webový server pre vývoj a testovanie. Inštalačný balíček obsahuje všetko potrebné – serverovú aplikáciu (Apache), databázu (MariaDB) a skriptovací jazyk (PHP). Vzhľadom na to, že väčšina implementácií webových serverov využíva rovnaké komponenty ako XAMPP, je prechod z lokálneho testovacieho serveru na živý server veľmi jednoduchý (26).

1.12 MySQL

MySQL je rýchla, ľahko použiteľná relačná DBMS, vyvíjaná pod open-source licenciou, takže ju každý môže používať bez platenia. Vyvinula ju a podporuje švédska spoločnosť MySQL AB. MySQL využíva mnoho malých, ale aj veľkých spoločností ako sú napríklad YouTube, PayPal, Google, Facebook atď (27).

Populárna sa stala hlavne kvôli dôvodom ako sú :

- Jedná sa o veľmi mocný nástroj, zvláda veľké množstvo funkcii najdrahších a najsinejších databázových balíčkov (27).
- Využíva štandardnú formu známeho SQL jazyka (27)
- Pracuje na mnoho operačných systémoch a s veľkým množstvom programovacích jazykov vrátane PHP, Java, C atď. (27).
- Podporuje veľké databáze, ktoré obsahujú až 50 miliónov záznamov (27).
- Ľahká správa databáze aj vďaka rôznym nástrojom (27).

1.13 Adminer

Adminer je kompletne vybavený nástroj pre správu databáze napísaný v PHP. Na rozdiel od nástroja phpMyAdmin obsahuje jediný súbor, ktorý je pripravený pre nahranie na cieľový server. Adminer je k dispozícií pre MySQL, MariaDB, PostgreSQL, SQLite, MS SQL, Oracle, Firebird, SimpleDB, Elasticsearch a MongoDB (28).

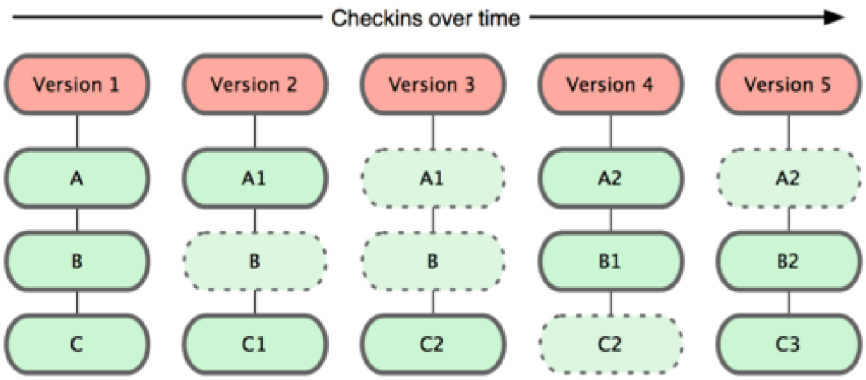
1.14 Git

Git je distribuovaný systém pre správu verzií. Bol navrhnutý pri vývoji Linuxového jadra, z dôvodu rozpadu firmy, ktorý prevádzkovala ich aktuálne používaný verzovací systém Bit-Keeper (29).

Pokiaľ pracujeme na väčšom projekte s tímom ľudí alebo potrebujeme iba pravidelne zálohovať svoju prácu, vyhľadanie nejakého systému pre správu verzií je inteligentná vec. Umožňuje zrkadliť Vašu pracovnú zložku a zálohovať ju na internet, vracať zmeny v súboroch do predošlého stavu, sledovať zmeny v súboroch, vrátane toho, kto zmenu

spravil atď. V prípade nejakých problémov preto nie je problém spätne dohľadať, kedy a kde sa stala chyba, vrátiť celý projekt do stavu kedy bol funkčný alebo v prípade straty lokálnych dát, stiahnuť všetky súbory projektu späť do stavu, v akom boli pri poslednom nahraní (29).

Na rozdiel od iných verziovacích systémov, ktoré ukladajú dáta ako zoznam zmien v súboroch, si Git pri každom nahraní vytvorí nový miniatúrny súborový systém. Aby bol takýto spôsob ukladania efektívny, tak súbory, ktoré neboli zmenené sa znovu nenahrajú, ale iba sa uloží odkaz na predošlí identický súbor (29).



Obr. 4: Ukladanie dát ako snímky projektu v Gite
(Zdroj: 30)

1.15 Dátové modelovanie

Cieľom dátového modelovania je navrhnúť kvalitnú dátovú štruktúru pre konkrétnu aplikáciu a databázový systém, ktorý bude táto aplikácia využívať. Pri dátovom modelovaní zvyčajne vytvárame najprv konceptuálny dátový model. Ten predstavuje určité zobecnenie oproti konkrétnej implementácii dátovej štruktúry v relačnej, objektovej, pripadne natívnej XML databáze. Zobecnením získame nezávislosť modelu na konkrétnom databázovom systéme, ale zároveň sme schopný tento model kedykoľvek previesť do konkrétneho implementačného prostredia (31).

Princíp troch architektúr

Niekedy označovaný ako P3A. Tento princíp zahrňuje tri vrstvy (návrhy), ktoré na seba nadväzujú, pričom každá z vrstiev zobrazenú realitu zachytáva na inej úrovni abstrakcie. Výhodou je, že ak sa rozhodneme po nejakom čase alebo sme nútený zmeniť platformu, nemusíme začínať znovu (32).

- **Konceptuálna úroveň** – vytvorený model, ktorý je nezávislý na počítačoch. Neberie sa v úvahu technologické, ani implementačné špecifiká konkrétneho riešenie. Je tu teda určené iba to čo bude obsahom systému (32).
- **Technologická úroveň** – zohľadňuje technologickú koncepciu riešenia. Spadá sem napríklad koncepcia organizácie dát, niekedy označovaná aj ako databázový model. Tie môžu byť rôzne, v súčasnosti je najpoužívanejší relačný model. Ďalšie modely sú sieťový, hierarchický, objektový alebo objektovo-relačný. Technologická úroveň nerieši obsah, iba to ako daný obsah má byť v konkrétnej technológií realizovaný (32).
- **Implementačná úroveň** – je už na platforme závislá. Nerieši obsah, ani technológie. Berie ohľad iba na špecifiká konkrétneho databázového systému a nie je teda abstrahované od žiadnych špecifík konečného riešenia. Tento model teda špecifikuje čím realizovať technické riešenia vychádzajúce z predošlého modelu (32).

Normalizácia

Cieľom normalizácie je organizovať dáta v databáze tak, aby nikde nemali duplicity a mohli s dátami lepšie pracovať. Pokiaľ máme databázu bez duplicít, docielime tým úspory miesta v databáze, ale hlavne môžeme potom dáta jednoduchšie upravovať. Pri normalizácií dát sa používajú tzv. normálne formy od nultej až po piatu normálnu formu, pričom môžeme povedať, že čím vyššia forma, tým sú dáta viac rozdelené a je tak menej možnosti pre výskyt aj na prvý pohlaď skrytých duplicít (33).

1.16 Agilný vývoj softwaru

Software je možné vyvíjať celou radou spôsobov. Či už sa jedná o tzv. Waterfall model, čo je klasický prístup, od analýzy cez návrh a beta verziu k finálnemu produktu alebo o agilnú metodiku, je nutné, aby manažér, tým analytikov, vývojárov a testerov koordinoval, bral v úvahu záujem klienta a zároveň kládol dôraz na kvalitu a produktivitu samotných pracovníkov (34).

Agilná metodika je spôsob rozvrhnutia a overovania práce. Cieľom agilnej metodiky býva lepšia organizácia práce. Odlišné agilné metodiky vedú k odlišným produktom,

pretože považujú iné aspekty produktu za kľúčové. Líši sa aj prístup ku zmene zadania (34).

Agilná technika je oproti tomu konkrétny postup, ktorý sa väčšinou týka samotných vývojárov. Cieľom týchto techník býva zvýšenie produktivity práce, odstránenie chýb, kvalitnejší software alebo presnejšie dodržanie špecifikácie. Agilné techniky sú od metodik oddeliteľné a cieľom agilnej techniky býva kvalitnejší produkt (34).

Fáze celého projektu v agilných metodikách

1. Nultá iterácia – prvá krátka analýza a naprogramovanie nejakej základnej činnosti. V agilnom tíme nejde príliš o to, čo sa v tejto časti bude implementovať. Podmienkou je, aby na konci bol hotový kúsok aplikácie, ktorý sa dá predviesť (aj klientovi) (34).
2. Analýza zmeny – výber toho, čo sa bude implementovať, analýza a designovanie zmien (34).
3. Samotná implementácia požadovanej vlastnosti (34).
4. Predvedenie klientovi (34).
5. Pokiaľ nie je produkt hotový, opakuje sa krok 2 (34).
6. Pokiaľ je produkt hotový, nasleduje údržba, rozvoj (rovnako v iteráciách) (34).

Body 2-4 sa označujú ako jedna iterácia. Tieto body sa opakujú tak dlho, pokiaľ nie je vývoj ukončený (úspechom, odložením projektu) (34).

1.17 SCRUM

Scrum je pravdepodobne najznámejšou agilnou metodikou, ktorá sa hodí do tímu pre menší počet ľudí. V ideálnom prípade by mali byť na jednom mieste (v jednej miestnosti), ale vyskytujú sa aj prípady, kedy sa Scrum prevádzkuje naprieč svetom. Samotná metodika sa rozšírila na začiatku deväťdesiatych rokov (35).

Role účastníkov Scrumu

V Scrume figurujú dve skupiny:

- Pigs – osoby, ktoré priamo súvisia s vývojom aplikácie (35).
- Chickens – užívatelia produktu, manažéri, ktorí priamo nezodpovedajú za výkon (35).

Medzi Pigs patria role :

- **Product Owner** je osoba, ktorá zodpovedá za priority, za to, čo sa bude v nasledujúcom šprinte implementovať a určuje implementačné detaily (35).
- **Scrum Master** pracuje ako ten, kto má programátorov odtieniť od okolitého sveta. Riadi vývojárov, ale zároveň sa stará o to, aby im fungovali počítače, mali dostupný software, ktorý potrebujú, rieši spory apod. Scrum Master a manažér sa môžu kryť, zároveň môže byť Scrum Master aj analytik. Je ale zakázané, aby bol Scrum Master zároveň programátor, v takejto chvíli by nebol schopný odfiltrovať programátorov od rušivých vplyvov. Pre túto rolu sa jednoznačne hodí Project Manager (35).

Medzi Chickens patria nasledujúce role:

- **Stakeholders** – zákazníci, testeri, pripomienky zvonku (35).
- **Managers** – osoby, ktoré pomáhajú nastaviť prostredie, ale niesú ani vývojári, ani Product Owneri, ani Scru Masteri (35).

Pri Scrumu je užitočná prítomnosť zákazníka pri vývoji. Zákazník sa teda účastí diskusie o vývoji, odpovedá priebežne na otázky a pomáha tak vyvíjať skutočne užitočný produkt (35).

Workflow celého projektu

Zahájenie nového projektu, vždy začína víziou výsledku. Hľadá sa výber vlastností, ktoré má systém obsahovať, dané vlastnosti sa odhadujú, spisujú sa takzvané User Stories (prípady použitia systému). Tejto fáze sa hovorí Release Planning (35).

Všetky User Stories sa spíšu do tzv. Product Backlogu (obsahuje všetky scenáre, ktoré systém zahrňuje). Product Backlog sa zoradí podľa priority (má na starosti Product Owner). Jednotlivým úlohám sa priradia časové náročnosti. Do prvého šprintu (iterácie)

sa pokúsia programátori odhadnúť, koľko úloh stihnú z product backlogu stihnúť behom jednej iterácie(šprint). Ešte pred zahájením normálneho vývoja prebehne tzv. nultá iterácia. V tej sa tým zameria na prvotnú implementáciu nejakej triviálnej úlohy, na ktorej bude ďalej pokračovať vývoj. Nultá iterácia nebýva podmienkou, vždy je ale nutná nejaká príprava tímu, prebiehajú prípadne školenia, volia sa ľudia do tímu, vyberá sa vhodná platforma (35).

Workflow jednej iterácie

Projekt bol rozčlenený do jednotlivých User Stroies, bolo zvolené, ktoré User Stories sú predmetom šprintu a bol stanovený termín dokončenia (Time Box). Do konca Time Boxu musia byť požadované vlastnosti implementované. Odhady sa stanovujú tak, aby bola práca dokončená presne v deň odovzdania (35).

Jednotlivé User Stories, vlastnosti, ktoré sa majú implementovať, sa presunuli z Project Backlogu do tzv. Sprint Backlogu (zoznamu krokov aktuálnej iterácie). Zákazníkovi nie je za žiadnu cenu umožnené zmeniť zadanie. Zmeny sú povolené iba v čase medzi iteráciami. Po dokončení šprintu sa predvádza produkt klientovi. Vlastnosti, ktoré sa nestihli dokončiť, sú skryté, klient vidí iba dokončenú prácu. Podľa nich sa rozhodne, čo sa bude ďalej diať. Klient vie, ktoré úlohy sa nestihli a môže sa rozhodnúť, či sa dokončia alebo nie. (35).

1.18 Marketing

Hard sell

Stratégie hard sell marketingu sú agresívne a zvyčajne kladú vysoký tlak na klienta. Medzi taktiky patria aj chladné hovory, násilné predajné lístky. Jedná sa priamo o predaj, vie to aj zákazník aj klient. Hlavnou výhodou hard sell marketingu je dostať sa priamo k veci. Toto je obzvlášť dôležité pre klientov, ktorí sú pripravený kúpiť a nepozerajú sa na ďalšie stretnutia. Problém s hard sell marketingom je v tom, že keď sa uskutoční príliš agresívne, pokus klienta o pomoc bude považovaný za nepríjemný. To hrá veľkú úlohu, najmä ak sa jedná o malé podniky, ktoré robia tieto predajné technik. Bez ohľadu na to, aká originálna je ponuka, môže to znieť ako podvod (39).

Soft sell

Stratégia soft sell marketingu sa zameriava na aspekt predaja pri budovaní vzťahov. Na potenciálnych kupujúcich sa nevyvíja psychologický tlak. Namiesto toho sa zameriava na pasívne spôsoby ako zákazníkom ukázať riešenia, ktoré potrebujú. Pri online marketingu to môže byť formou blogu, rôznych fór, poskytnutia bezplatnej knihy atď. Soft sell marketing môže v niektorých prípadoch fungovať, ale nemá zmysel uplatňovať tieto taktiky na všetkých klientov. Príliš jemný prístup môže vzbudiť dojem, že klient si nie je istý svojimi službami (39).

Earned media

Earned media sú definované ako nezaplatená zmienka o značke alebo organizácii subjektami tretích strán, ako sú napríklad mediálne publikácie, zákazníci alebo rôzny influenceri. Sú výsledkom so vzťahmi s verejnosťou. Eearned media sa môžu objavovať naprieč tradičnými, ale aj modernými kanálmi ako sú napríklad tlač, rozhlas, televízia, sociálne média, webové stránky, blogy atď.(40).

Direct mail

Pod pojmom direct mail alebo directmailová služba sa dá predstaviť spôsob reklamy alebo propagácie výrobku či služby. Sú úzko spojené s termínom direct marketing, ktorý predstavuje spôsob komunikácie so zákazníkom, a to preto, že tieto služby sú priamo jeho súčasťou. Jedná sa o reklamu priamo cielenú na stávajúceho alebo potencionálneho zákazníka. Jedná sa o akúkoľvek propagačnú záležitosť zaslanú pomocou pošty alebo distribučnej siete priamo osobe, ktorej chce predajca produkt ponúknuť (41).

Link building

Je jednou z najdôležitejších súčastí optimalizácie pre vyhľadávače (SEO). Dostatok kvalitných spetných odkazov zaručuje firme zvýšenie návštevnosti a zlepšenie pozície vo vyhľadávačoch. Ciele link buildingu sú napríklad nárast priamej návštevnosti prostredníctvom získania odkazov, zlepšenie pozície stránok vo vyhľadávačoch, zvýšenie návštevnosti z vyhľadávačov či zvýšenie obchodnej úspešnosti webu. Ďalšie metódy sú napríklad publikovanie na cudzích serveroch, účasť v diskusných fórach a konferenciách, vydávanie tlačených správ a PR článkov, návrh a realizácia microsites poprípade link baiting (vytváranie užívateľsky atraktívneho obsahu (42).

2 ANALYTICKÁ ČASŤ

V analytickej časti sa zameriavam na predstavenie projektu KradenaKola.cz. Opisujem prečo sa firma rozhodla pre inováciu webovej aplikácie a aké sú hlavné požiadavky. V tejto časti je popísaná aj marketingová stratégia projektu a analýza výberu vhodného technologického riešenia.

2.1 Predstavenie projektu

Obr. 5: Logo projektu KradenaKola.cz
(Zdroj: 44)

Projekt KradenaKola.cz vznikol v roku 2014 za účelom poukázať na problematiku množiacich sa krádeží bicyklov a následným obchodovaním s nimi. Aparátom pre možné odhaľovanie a obmedzovanie tejto trestnej činnosti je dobrovoľné registrovanie bicyklov ich majiteľmi na servery KradenaKola.cz, kde je možné si ho jednoducho a bezplatne zadaním identifikačného čísla overiť, či je alebo nie je v databáze označené ako kradnuté.

Ako to funguje ?

Obr. 6: Ako to funguje?
(Zdroj: 44)

Každý bicykel má svoje sériové číslo na spodnej časti rámu. Je viacero možností ako využiť projekt KradenaKola.cz :

- Overenie sériového čísla bez registrácie (či je bicykel kradnutý, vo vlastníctve majiteľa alebo nie je zaregistrované v databáze).
- Registrácia užívateľa pre pridanie vlastného bicykla do databáze.
- Nahlásenie krádeže majiteľovi.
- Odoberanie noviniek z oblasti cyklistiky.

2.2 Marketingová stratégia

Propagácia projektu je vzhľadom k jeho povahe a finančným možnostiam založená na oslovovaní komunity cyklistov a širokej verejnosti hlavne prostredníctvom internetu a predovšetkým pomocou sociálnej siete Facebook. Vďaka už od začiatku zvolenej stratégie v propagácii formou soft selling reklamy a earned media sa podarilo vybudovať relatívne slušné povedomie o projekte. O ďalšie šírenie dobrého mena sa starala už samotná komunita cyklistov, ktorá sympatizovala s myšlienkou projektu.

Diskusné fóra

Diskusné fóra s tematikou bicyklov sú dôležité pre šírenie povedomia o registru KradenaKola.cz a linkbaitingu. Preto bola nastavená periodická návšteva vybraných diskusných fór a v prípade relevantných dotazov nasleduje nevtieravou formou upozornenia na existenciu registru bicyklov KradenaKola.cz. Zbieranie týchto cenných spätných odkazov a privedenie návštevnosti na web má za následok nové registrácie užívateľov bicyklov, ale taktiež vylepšenie pozície z pohľadu internetových vyhľadávačov.

Facebook

Založením vierohodného Facebookového profilu s prvými príspevkami o myšlienke celého projektu registru bicyklov, vyhľadávaním cyklistických skupín a tém o bicykloch alebo krádežiach obecne, prišlo vplyvom ponúkania možností registrácie opäť formou soft selling reklamy k rozpútaniu záujmu o registrácie bicyklov do registru KradenaKola.cz.

V súčasnosti slúži Facebook ako pasívny príjem informácií a možností registrácií kradnutých bicyklov do registru od užívateľov, ktorým bol bicykel odcudzený a hľadajú rôzne cesty ako svoj bicykel opäť nájsť. Do projektového plánu bol zaradený návrh na

vylepšenie profilu bicyklov v registri KradenaKola.cz o funkciu automatického uverejňovania bicykla označeného ako kradnuté na Facebookovej stránke KradenaKola.cz v prípade, že túto voľbu užívateľ využije.

Fundraising a možnosti partnerskej spolupráce

Fundraising pre projekt je založený na získavaní partnerov, ktorý by mali záujem o propagáciu svojich produktov alebo značiek za pomoci možností, ktoré ponúka projekt KradenaKola.cz. Konkrétne sa jedná o publikovanie plateného obsahu na webovej stránke projektu, Facebookovom profile KradenaKola.cz, tlačených materiáloch a e-mailových správach.

Záujemcovia o propagáciu sú rozdelení do dvoch skupín. Na hlavného partnera projektu a ostatných partnerov projektu. Prostredníctvom presne definovaných pravidiel pre reklamu sa má administračný systém KradenaKola.cz starať o konfiguráciu poskytovaných služieb a vygenerovania ponuky. Má sa jednať o jednoduché CRM (customer relationship management), kde je do systému zavedená karta partnera ako zákazníka a k niekomu už môžu byť priradené ponuky či spustené konkrétne služby podľa ponuky na časovo obmedzenú dobu.

Samotná propagácia partnera musí rešpektovať víziu propagácie projektu KradenaKola.cz, tzn. Formou soft selling reklamy v kombinácii s earned media, teda v stručnosti formou nenásilnej a nevtieravej reklamy. Dôraz sa kladie n dôveryhodnosť sponzorovaného obsahu, previazanie firmy, značky či produktu s inou témou, ktoré upúta práve cielenú skupinu užívateľov partnera. Cieľom je dosiahnuť vyššieho efektu v budovaní dobrého mena a povedomia o firme, značke či produkte a navyšovanie lojality u zákazníkov. Súčasťou propagácie partnera sú také periodické reklamné zdieľania prostredníctvom direct mail marketingu, napríklad zaujímavé alebo špeciálne ponuky pre registrovaných užívateľov.

2.3 Inovácia

Keďže bol projekt vytvorený v roku 2014 a od tej doby webová aplikácia neprešla žiadnymi technickými zmenami, firma sa rozhodla pre inováciu.

Celá webová aplikácia je momentálne vytvorená na nette frameworku. Pre prezenčnú časť aplikácie bol vytvorený vlastný originálny grafický návrh, ktorý bol pre web naštýlovaný pomocou CSS. Web je postavený iba pre desktopové zariadenia, teda nie je responzívny.

Pre rozšírenie súčasnej webovej aplikácie o nové funkcionality však môžu nastať rôzne komplikácie. Keďže momentálne technologické riešenie je zastaralé a nesplňuje mnohé štandardy, ktoré sa v súčasnosti používajú, nebolo by výhodné či už z časovej alebo ekonomickej stránky rozširovať súčasný systém. Preto sa firma rozhodla o znovu vytvorenie webovej aplikácie, ktorá by využívala moderné technológie pre tvorbu webových stránok.

Požiadavky pre novú webovú aplikáciu :

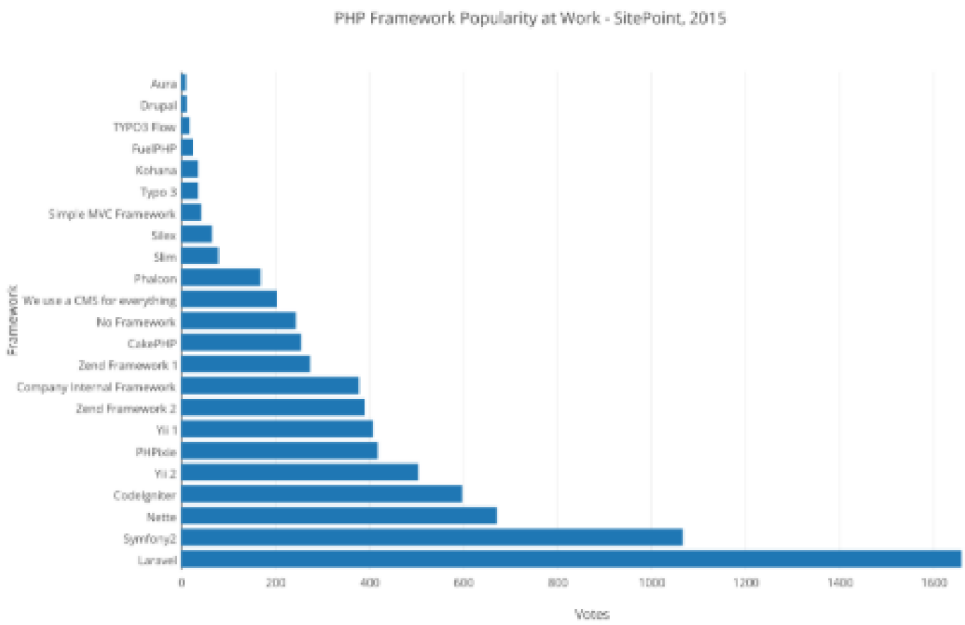
- Všetky funkcionality, ktoré obsahuje súčasná webová aplikácia.
- Aplikácia pripravená pre rozšírenia.
- Responzivita.
- Zvýšenie výkonu.
- Využitie open-source technológií.

2.3.1 Výber vhodného PHP frameworku

PHP je najobľúbenejší skriptovací jazyk na strane serveru. Pri vývoji webových aplikácii zaberá mnoho času a úsilia vždy vytvárať všetko od začiatku. Adekvátne riešenie sú frameworky.

Dôvody prečo využívať PHP framework :

- Poskytujú organizovaný, znovu použiteľný a udržiavateľný kód (44).
- Rýchly vývoj (44).
- Umožňujú rast v priebehu času, pretože webové aplikácie bežiace na frameworkoch sú škálovateľné (44).
- Zabezpečenie webovej aplikácie (44).
- Dodržovanie vzoru MVC, ktorý zaisťuje separáciu prezentácie a logiky (44).
- Presadzujú moderné webové vývojové postupy, medzi ktoré patria aj nástroje objektovo orientovaného programovania (44).



Obr. 8: PHP framework popularity
(Zdroj: 45)

Keďže jednou s podmienok pre inováciu projektu je využitie open-source technológii , nasadenie PHP frameworku veľmi zjednoduší prácu na celkovom vývoji aplikácie. V teoretickej časti práce som popísal 3 PHP frameworky. ktoré sú vhodné pre použitie v našom projekte. Medzi najvyužívanejší v súčasnosti patrí framework Laravel.

Country	Total Votes	Work Favorite	Votes	Personal Favorite	Votes
United States	819	Laravel	219	Laravel	293
Czech Republic	770	Nette	611	Nette	639
United Kingdom	496	Laravel	138	Laravel	166
Germany	428	Symfony2	76	Laravel	100
France	343	Symfony2	149	Symfony2	136
Brazil	305	Laravel	100	Laravel	111

Obr. 7: PHP framework popularity by country
(Zdroj: 45)

V Českej republike je veľmi populárny nette framework, ktorý má veľmi širokú komunitu vývojárov.

Pre náš projekt sme vybrali nette framework. Jedným z dôvodov je ten, že pôvodný projekt je rovnako postavený na nette, a tým pádom, mnoho funkcionalít nie je potrebné od základu vymýšľať, ale použili by sme pôvodné riešenia.

Nette kladie veľký dôraz na zabezpečenie aplikácie, čo je veľmi dôležitá časť pri rozhodovaní vo výbere frameworku.

Naša aplikácia je teda pod záštitou nette frameworku chránená pred :

Cross-Site Scripting (XSS)

Čo je metóda narušovania webových stránok zneužívajúcich neošetrené výstupy. Útočník potom dokáže do stránky podstrčiť svoj vlastný kód a tým môže stránku pozmeniť alebo dokonca získať citlivé údaje o návštevníkoch (46).

Cross-Site Request Forgery (CSRF)

Je útok spočívajúci v tom, že prijmeme užívateľa navštíviť stránku, ktorá skryto vykonáva útok na webovú aplikáciu, kde je užívateľ zrovna prihlásený (46).

URL attack, control codes, invalid UTF-8

Rôzne pojmy súvisiace so snahou útočníka podstrčiť našej aplikácií škodlivý vstup. Následky môžu byť veľmi rôznorodé, od poškodenia XML výstupov, cez získanie citlivých informácií z databáze alebo hesiel (46).

Session hijacking, session stealing, session fixation

So správou sesion je spojené hneď niekoľko typov útokov. Útočník buď odcudzí alebo podstrčí svoje session ID vďaka tomu získa prístup do webovej aplikácie, bez toho aby poznal heslo užívateľa. Potom môže v aplikácii robiť čokoľvek, bez toho aby o tom užívateľ vedel (46).

Jedným z ďalších dôvodov rozhodnutia sú aj dlhodobé skúsenosti senior programátora s nette frameworkom, ktorý je členom tímu pri vývoji tejto aplikácie.

2.3.2 Prezenčná časť webovej aplikácie

Do pôvodného grafického dizajnu boli vložené nemalé finančné prostriedky a v súčasnosti vyjadruje dizajn charakteristické črty projektu KradenaKola.cz. Preto vytváranie nového grafického návrhu, nie je pre tento projekt vhodné. Avšak webová stránka bola naštýlovaná iba pomocou CSS.

Pre zmodernizovanie prezenčnej stránky využijeme pre našu webovú aplikáciu framework Bootstrap (popísaný v teoretickej časti práce), keďže spĺňa všetky požiadavky pre najmodernejšie trendy vo web dizajne.

Hlavné dôvody využitia Boostrapu:

Responzivita

Jedným z požiadavkou pre inováciu aplikácie je responzivita. Boostrap štýly sú dokonalé prispôsobené pre mobilné zariadenia. Je teda 100% responzívny. Práve mobilné zariadenia sú v dnešnej dobe na webe už častejšie ako klasické desktopové zariadenia. Preto je dôležité zameranie sa aj zákazníkov využívajúcich hlavne mobilné zariadenia.

Mobile – first

Framework je v súčasnej verzii kompletne pripravený aby podporoval mobile-first prístup. Jeho kód je kompaktný a podporuje potrebné praktiky.

Flat design

Web pomocou Bootstrapu vyzerá svieži a moderný. V prípade zmene trendov, Bootstrap reaguje na najnovšie požiadavky pre webové stránky.

Grid

Obsahuje dvanásť stĺpcový grid systém, ktorý vo väčšine prípadov nahradí pozíciovanie elementov na stránke, ktoré je naviac pravidelné a responzívne.

Open-source

Ďalšia požiadavka pre inováciu systému je využiteľnosť open-source technológií. Bootstrap je zadarmo aj pre komerčné účely, aj preto je vhodným frameworkom pre našu aplikáciu.

3 PRAKTICKÁ ČASŤ

V analytickej časti sme dospeli k tomu akým smerom sa bude projekt poberať a zvolili sme vhodné technológie, ktoré budeme používať. Táto kapitola bude venovaná konkrétnym metodikám pri vývoji webovej aplikácie, príprave webového serveru a databáze pre daný projekt a následne popísanie tvorby webovej aplikácie.

3.1 Metodika vývoja webovej aplikácie

Vo firme pri vývoji softwaru používame agilný prístup. Keďže je firma menšia a na projektoch pracuje malý tím pracovníkov používame osvedčenú metodiku SCRUM, ktorá sa výborne hodí aj pre náš projekt KradenaKola.cz.

3.1.1 Vízia

Vízia projektu je inovácia súčasného projektu KradenaKola.cz. Súčasťou bude vytvorenie webovej platformy postavenej na najnovšej verzii nette frameworku. Základom bude zachovanie súčasných dát. To znamená, že všetky dáta, ktoré momentálne slúžia pre projekt budú zahrnuté aj v novej platforme. Súčasťou výsledku bude spracovaná back-endová časť ako aj front-endová časť a rozšírenie o nové funkcionality.

3.1.2 Plán projektu

Plán rozdelíme na 3 hlavné časti, ktoré by mali byť súčasťou každého projektu. Jedná sa o časový plán, funkcionalitu a rozpočet. Avšak musíme rátať s tým, že projekt sa môže počas vývoja meniť a teda nemôžme mať pevne zafixované všetky atribúty. Zafixovať by sme si mali hlavne časový plán projektu, keďže pevne nastavené dĺžky iterácií nám umožnia pravidelne dodávať hodnotné prírastky.

Pri pevne stanovenom dátume a stabilnom tíme je teda jednoduché vypočítať náklady. Plán vydania v SCRUMe, nie je pevne určený predom v dokumentácií projektu, ale je postupne vytváraný a menený za spolupráce vývojárskeho tímu a zadávateľa. V prípade menších projektov ako je aj ten náš, zvyčajne stačí zoskupovať položky z Product backlogu.

Časový plán

Časový plán projektu pre vývoj webovej aplikácie je stanovený na 8 mesiacov. Projekt je rozdelený na 4 hlavné iterácie vývoju. Časový rámec pre jednotlivé iterácie bol rozdelený rovnomerne, to znamená, že pre každú iteráciu boli vyhradené 2 mesiace. Každú stredu v týždni prebiehali mítingy, kde sa predstavil progres vývoju a určili sa nasledovné úlohy.

Náklady

Pre reálny odhad časovej náročnosti a prevediteľnosti jednotlivých úloh, je nutné si určiť aj aké technológie a postupy bude tým dodržiavať. Dôraz je kladený na open-source technológie a minimalizáciu nákladov.

Výhodou SCRUMu je, že k riadeniu je potrebné úplné minimum nástrojov. Kvôli zefektívneniu procesov je potrebné mať nástroje, ktoré by umožnili strojové spracovanie dát, spoluprácu a export či import dát. Konkrétne nástroje použité v projekte budú uvedené v nasledujúcich častiach práce.

Technológie pre vývoj aplikácie boli pre náš projekt zvolené na základe predchádzajúcich skúseností členov tímu a sú popísane v teoretickej časti práce.

Jediným nákladom je práca na projekt, ktorá sa dá ľahko a transparentne vypočítať vďaka pevnému časovému rámcu. Konkrétne náklady na projekt sú uvedené v závere praktickej časti v ekonomickom zhodnotení.

Funkčné požiadavky

Vyššie spomenutá vízia stavia na reálnych podnetoch od majiteľa projektu, ktorý prejavil záujem o inováciu projektu. Aby bolo možné určiť veľkosť tímu pre vývoj aplikácie, bol v spolupráci s majiteľom zostavený zoznam základných funkčných požiadavkou na web. Web by mal byť rozdelený na dve hlavné časti, a to časť pre bežných užívateľov a časť pre administrátora.

Časť pre bežného užívateľa :

- registrácia,
- prihlásenie,
- pridanie bicykla,
- správa profilu (samotného užívateľa alebo bicykla),

- výpis bicyklov,
- odber noviniek z oblasti cyklistiky.

Časť pre administrátora :

- správa a riadenie prístupov,
- výpis všetkých užívateľov/bicyklov,
- správa všetkých užívateľov/bicyklov,
- správa inzercie,
- správa obsahu (cms).

Prezenčná časť, vrátane grafického návrhu stránky by mala byť podobná ako pri súčasnej webovej platforme. Hlavnou zmenou grafického návrhu by mala byť responzivita stránky.

3.1.3 Tím

Pre tento projekt bol zvolený 5 členný tím, ktorý sme podľa metodiky SCRUM rozdelili do rolí. Každá rola v tíme bude mať teda na starosti určité činnosti a zodpovednosť za konkrétne procesy.

Product owner

Keďže projekt patrí pod našu firmu, , rolu product ownera nesie majiteľ firmy. Zastupuje aj úlohu zákazníka a je teda zodpovedný za víziou projektu. Definuje smer tímu a oblasti, ktorým by sa mal vyhnúť.

Scrum master

Role Scrum mastra sa ujal senior programátor firmy Product ownera. Na základe požiadavkou vytvoril User Stories, ktoré následne dostali prioritu. Poskytoval potrebné znalosti pre vývoj projektu, keďže bol súčasťou tímu pôvodného projektu.

Vývojový tím

Pozostával zo senior programátora, mňa ako junior programátora, testera aplikácie a marketingového špecialistu. Na starosti som mal teda projekt pre časť programovania či už back-endu alebo front-endu.

3.1.4 Prispôsobenie metodiky

Keďže sa jedná o menší projekt a tím je zostavený iba z 5 členov a väčšinu času sme sa nachádzali spolu v jednej kancelárií, konkrétne postupy SCRUMu boli teda prispôsobené našej situácii. Zahájenie projektu teda prebiehalo v kancelárií. Kde Product Owner predstavil víziu projektu, spísali sa základné User Stories, určili priority a priradili časové náročnosti. Zvolila sa tzv. nultá iterácia, v našom prípade to bola celková analýza, zvolenie použitých technológií a príprava webového serveru a databáze.

Pri vývoji aplikácií v tíme potrebujeme mať nástroje kde môžeme spolupracovať na tvorbe kódu, testovaní a nasadzovaní, rozvrhnutie úloh pre jednotlivé iterácie, poprípade reporting Ako som vyššie spomínal pre tento projekt chceme využiť open-source nástroje. Pre náš projekt využijeme aplikácie od spoločnosti Atlassian.

Bitbucket

Obr. 9: Logo softwaru Bitbucket
(Zdroj: 36)

Bitbucket je webová služba podporujúca vývoj softwaru pri používaní verziovacích nástrojov Git a Mercurial. Bitbucket poskytuje bezplatný hosting pre open-source projekty a menšie tými do 5 ľudí. Ďalej poskytuje aj komerčné programy, ktoré po zaplatení mesačného poplatku umožňujú ukladať súkromné repozitáre. Služba je podobná GitHubu (36).

Bitbucket sa skvele hodí pre náš projekt a to hlavne pri spolupráci na tvorbe kódu, testovaní a nasadzovaní. Poskytuje nám kvalitné zabezpečenie nášho workflow, keďže si môžeme nastaviť prístup pre konkrétnych užívateľov a kontrolu ich akcií pomocou oprávnení. Slúži nám aj ako repozitár. Bibucket je webová platforma, čiže nám stačí pripojenie na internet aby sme ho mohli používať a v aplikácii môžem pracovať na viacerých projektov súčasne.

Scrum master teda na začiatku v tejto aplikácii vytvoril projekt kradenakolacz, pozval členov tímu k danému projektu a určil oprávnenia pre každého.

Softvér nám slúži hlavne pre postupné pridávanie spracovaných častí projektu (iterácií).

Obr. 10: Logo softwaru Sourcetree
(Zdroj: 37).

S touto webovou aplikáciou úzko súvisí software **Sourcetree**, ktorý rovnako používame v projekte. Sourcetree je desktopová aplikácia, ktorá nevyžaduje žiadne špeciálne požiadavky.

S aplikáciami pracujeme tak, že pre každú úlohu v daných iteráciách vytvárame vetvy(branches). Vetva pomenovaná Master vždy obsahuje kód, ktorý je už po revízii a po odsúhlasení Scrum Mastrom poprípade Product Ownerom bude kód použitý ako finálny do projektu.

Vždy keď začínam pracovať na novej úlohe, vytvorím si v aplikácií Sourcetree novú vetvu odvodenú od master vetvy a pomenujem ju feature. Môžem mať súčasne vytvorených viacero feature vetiev, a kedykoľvek pracovať na vetve, ktorej budem potrebovať pre danú úlohu. Vo vetvách si pomenovávam konkrétne časti, na ktorých pracujem názvom aby čo najviac vystihoval opis konkrétnej činnosti.

Po napísaní určitého kódu alebo teda spracovaní úlohy, ktorú som mal v danej úlohe jednoducho v sekcii commit nahrám dokončenú prácu. Program mi automaticky nahráva len kód rozdielny od master vetvy. Po nahrávaní súborov vždy napíšem jednoduchý popis k danému commitu.

Následne v aplikácii Bitbucket mi v sekcii commits pribudne teda mnou spracovaná časť kódu. Scrum Master má možnosť kód si otvoriť. Ak je v kóde nevyhovujúca časť, ktorú treba opraviť, aplikácia nám slúži aj na komentovanie. To znamená, že Scrum Master okomentuje, konkrétnu časť kódu. Ja si môžem komentár kedykoľvek pozrieť a podľa neho kód upraviť. Takto upravený kód znovu pridávam ako commit. Ak je všetko v poriadku a Scrum Master odsúhlasí správnosť kódu vytvorím pull request na finálnu časť kódu. Scrum Master takto vytvorený pull request potvrdí a tým pádom máme splnenú úlohu. Po skončení úlohy si zavriem vetvu, na ktorej som pracoval a pre ďalšiu úlohu vytváram rovnakým spôsobom novú vetvu.

Keďže v aplikácii vidíme časy pridania jednotlivých commitov, slúži nám výborne na kontrolu odvedenej práce. Ak by sme potrebovali stiahnuť celý kód odnova. Pomocou Bitbucket si ho vieme kedykoľvek stiahnuť ak máme prístup k danému projektu. Teda nám slúži aj ako záloha.

Trello

Obr. 11: Logo softwaru Trello
(Zdroj: 38)

Pre správu jednotlivých úloh sme využili software Trello. Sú v ňom zobrazené nástenky (boardy), ktoré obsahujú zoznam úloh. Úlohy je možné v rámci stĺpcov presúvať (pomocou drag and drop). K jednotlivým úlohám sa môžu priradiť, konkrétny členovia tímu, ktorí majú úlohu na starosti, dajú sa rozvrhnúť časové termíny poprípade nahrať súbory potrebné k splneniu úlohy. Úlohy sa dajú rozdeliť pomocou desiatich farebných štítkov, ktoré si môžeme pomenovať pre každý board osobitne. Úlohy sa dajú komentovať či pridávať kontrolné zoznamy menších činností. Software môžeme používať na webových prehliadačoch, ako desktopovú aplikáciu, poprípade sú dostupné natívne aplikácie pre Android či iPhone.

3.2 Príprava webového serveru a databáze

Pre vývoj webovej aplikácie potrebujeme webový server. Na výber sú dve možnosti. Buď to bude počítač, ktorý bude zodpovedný za vyriaďovanie http požiadavkou od klienta (webového prehliadača) tzn. odosielania cieľa špecifikovaného URL. Poprípade počítačový program, ktorý prevádza tieto činnosti.

Keďže prvá možnosť by bola ekonomicky a časovo zbytočne náročná, rozhodli sme sa pre voľbu počítačového programu.

Najpoužívanejším webovým serverom vôbec je Apache HTTP Server. Má otvorený kód pre GNU/Linux, BSD, Solaris, macOS, Microsoft Windows a ďalšie platformy. Apache podporuje veľké množstvo funkcii. Môžu to byť funkcie podpory programovacích

jazykov na strane serveru či rôzne autentizačné schémy. Pre nás je podstatné, aby podporoval jazyk PHP a databázu MySQL.

Pre správu webového serveru použijem XAMPP. Keďže je to znova open-source software výborne nám poslúži.

XAMPP je jednoduchá odľahčená distribúcia Apache, ktorou si vytvoríme lokálny server pre vývoj a testovanie. Inštalačný balíček obsahuje všetko čo budeme potrebovať

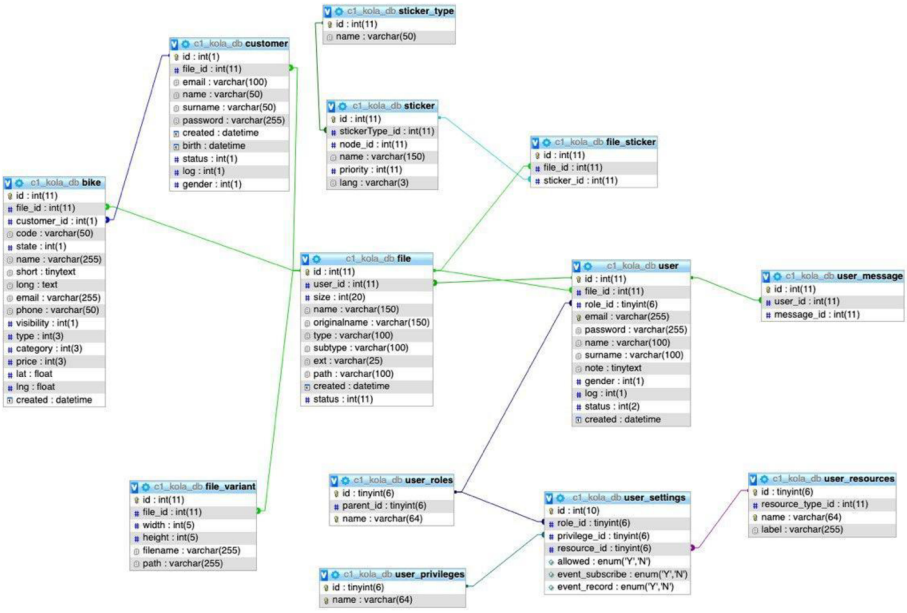
- serverovú aplikáciu Apache,
- databázu (MySQL),
- scriptovací jazyk (PHP).

V aplikácii jednoduchým stlačením tlačidla Start si spustíme lokálny server poprípade MySQL databázu. Jednotlivé moduly si môžeme nakonfigurovať podľa našich potrieb.

Databáza

Projekt KradenaKola.cz využíva MySQL databázu. Pôvodná databáza splňuje všetky požiadavky potrebné pre náš projekt, preto sme sa rozhodli využiť túto databázu aj pre novú aplikáciu.

Pri vývoji aplikácie sme si vytvorili vlastnú testovaciu databázu postavenú podľa E-R diagramu (príloha č.1, obr.č.42). Pre správu databáze využívame aplikáciu Adminer,



Obr. 12: Zjednodušený E-R diagram databáze
(Zdroj: Vlastné spracovanie)

v ktorej sa dá jednoducho vytvárať tabuľky, určovať dátové typy, priraďovať primárne či cudzie kľúče atď.

Pri začiatku projektu nevytváram kompletnú databázu, vždy vytváram iba tabuľky, ktoré sú potrebné pri danom progrese vývoja aplikácie aby sme vedeli otestovať funkčnosť vytvoreného kódu.

Najdôležitejšie tabuľky (obr.č.12), sú základom pre celú aplikáciu. Preto tieto tabuľky vytváram ako prvé. Najviac informácií budeme mať pre užívateľov a ich bicykle preto 2 hlavné tabuľky sú „bike“ a „user“. Od daných tabuliek sa odvíjajú všetky ostatné.

3.3 Vývoj

Keďže v súčasnosti na rôznych projektoch pracujú viacčlenné tímy vývojárov, je potrebné aby každý dodržiaval určitý štandard písania kódu a aby aplikácia spĺňala určité podmienky, ktoré bude každý programátor dodržiavať. Preto aj v našom projekte dodržiavam konkrétne štandard písania PSR-2 a celá aplikácia je postavená na SOLID princípoch.

SOLID:

- **Princíp jednej zodpovednosti** - Každá trieda nesie zodpovednosť za práve jednu vec, ktorá je vystihnutá jej názvom. Vďaka tomu pri úpravách softwaru jednoduchšie nájdeme miesta vyžadujúce zmenu.
- **Princíp otvorenosti** – Triedy sú písané tak aby ich funkčnosť išla rozšíriť bez nutnosti ich modifikovať. Vďaka tomu minimalizujeme možnosť, že pri úpravách neúmyselne poškodíme iné časti aplikácie spoliehajúce sa na pôvodný kód. Taktiež kladieme dôraz na abstrakciu a polymorfizmus.
- **Liskovov princíp zameniteľnosti** – Ak nejaký kód používa bázovú triedu, funguje aj v prípade, že miesto bázovej triedy použijeme ktorúkoľvek z jej podtried. Všetky podtriedy sú teda s bázovou triedou zameniteľné. V jednoduchosti , odvodené triedy nevyžadujú viac a neposkytujú menej ako bázová trieda.
- **Princíp oddelenia rozhrania** – Triedy závisia iba na rozhraniach, ktoré používajú.
- **Princíp obrátenia závislostí** – Triedy vyššej úrovne abstrakcie nezávisia na triedach nižšej úrovne abstrakcie. Všetky závislosti teda vedú od konkrétnych k abstraktným. Závislosti sú zamerané na rozhranie a abstraktné triedy a nie iba na konkrétnej implementácií.

Hlavným zmyslom týchto princípov je obmedzovanie závislostí medzi jednotlivými časťami softwaru a obmedzeniu sekundárnych úprav po každej zmene. Závislosti, a to predovšetkým tie skryté, sú hlavnou príčinou, prečo software prestáva byť po určitej dobe udržateľný a vyžaduje zásadné zmeny návrhu alebo rovno vytvorenie novej verzie od začiatku.

Celý vývoj bol rozdelený na niekoľko iterácií, kde každá pozostávala z určitých úloh, ktoré sa mali splniť v danom časovom rámci, po vypracovaní úloh nasledovalo testovanie podľa pripravených scenárov od senior programátora a prípadná korekcia.

Pri každej iterácií uvediem ako sa pracuje s nette frameworkom a predstavím konkrétne riešenia v našej aplikácií na pár základných úlohách, ktoré som v danej iterácii riešil.

3.3.1 Prvá iterácia

Základné úlohy:

- založenie nette projektu a očistenie od nepotrebných tried, presentrov, testov, šablón atď.,
- nastavenie configu,
- rozdelenie aplikácie pre admina a užívateľa,
- nastavenie route,
- príprava hlavného layoutu.

Keďže máme pripravený webový lokálny server a rovnako pripravenú aj databázu, nič nám nebráni vo vytvorení projektu. Keďže používame XAMPP náš projekt sme vytvorili do jeho podadresáru s názvom „htdocs“ cez príkazový riadok zadaním „composer create-project nette/web-project kradenakolacz“.

Náš webový projekt bude mať nasledujúcu štruktúru :

```
www/                ← koreňový adresár webu
└── kradenakolacz/
    ├ app/              ← adresár webovej aplikácie
    │   ├ config/       ← konfiguračné súbory
    │   ├ presenters/   ← triedy presentrov
    │   │   └── templates/ ← šablóny
    │   ├ router/       ← konfigurácie URL adries
    │   └── Bootstrap.php ← spúšťajúci súbor
    ├ log/              ← tu nájdeme chybové logy
```

├ temp/ ← miesto pre dočasné súbory (cache, sessiony, atď.)

├ vendor/ ← knižnice pre našu aplikáciu

│ └── nette/

└── www/ ← miestny koreňový adresár webu - iba tento adresár je prístupný z internetu

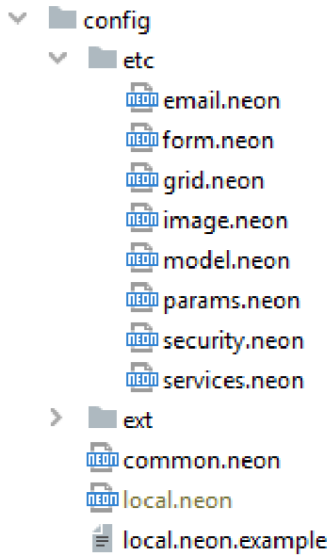
Vo webovom prehliadači po zadaní adresy localhost/kradenakolacz/www/ si priebežne kontrolujem v akom stave sa náš projekt nachádza.

Pre písanie kódu využívam vývojové prostredie cez aplikáciu PhpStorm. Projekt očistíme vymazaním zbytočných tried, presentrov, testov, šablón a popisov.

Konfigurácia

Nette umožňuje si nastaviť aplikáciu pomocou konfiguračných súborov. Tie sa zapisujú vo formáte NEON. Konfiguračný súbor je miesto, kde umiestňujeme definície vlastných služieb. Slúži nám pre to adresár /config.

Nastavíme si pripojenie do databáze, prístupové oprávnenia a mapovanie presentrov. Pre prehľadnosť každej zaregistrovanej služby som si adresár rozdelil na menšie podadresáre.



Obr. 13 : Konfiguračné adresáre
(Zdroj: Vlastné spracovanie)

V local.neon, nastavujem prihlasovacie údaje do databáze, ktoré potrebujem pre prihlásenie cez Adminera.

V comon.neon si includnem podadresáre, ktoré som si vytvoril, nastavím si mapovanie na presentre a pripojenie databáze (viď. príloha č.2, obr.č.43).

Rozdelenie aplikácie a routovanie

Keďže aplikácia má mať časť pre bežného užívateľa a časť pre administrátora. Všetky nové zložky a súbory vytváram do adresáru /app. Vytvorím si teda zložku FrontModule, ktorá bude slúžiť pre bežného užívateľa a zložku AdminModule, ktorá bude slúžiť pre administrátorov.

Aby som sa cez webový prehliadač dostal na konkrétny modul upravím si routovanie. Slúži nám na to trieda RouterFactory (viď. príloha č.2 , obr. č.44).

Po nastavení routov sa dostanem cez adresu localhost/kradenakolacz/www/ na užívateľské rozhranie a cez adresu localhost/kradenakolacz/www/admin na administrátorské rozhranie.

Nette pracuje na architektúre MVP. Oba moduly budú mať spoločnú logickú časť, ktorá bude spracovávať rôzne operácie, hlavne teda súvisiace s prácou s databázou. Časti „view“ a „presenter“(controller), potrebujeme pripraviť pre každý modul zvlášť.

Každý modul teda obsahuje adresár /presenters a v ňom podadresár /templates

Príprava hlavného layoutu

Pre správne zobrazenie vo webovom prehliadači sme si pripravili hlavný layout (viď. príloha č.2, obr.č.45). Layout sa nachádza v podadresári /templates a nesie názov „@layout.latte“.V ňom pomocou html a latte tagov si nachystáme základ pre našu webovú stránku.

Zo začiatku je pre nás podstatná časť <body>, do ktorej si pridám flash messages (upozornenia) a pomocou latte makra kód {include content}.

3.3.2 Druhá iterácia

Základné úlohy :

- modely pre prácu s databázou,
- formuláre pre prihlásenie a registráciu,
- presentre.

Modely

Pre logické operácie v našej aplikácii nám slúžia modely. V našom prípade modely budú spracovávať všetky operácie spojené s databázou. Model vôbec nevie o výstupe. Jeho funkcie spočívajú v prijatí parametrov zvonku a vydaní dát smerom von. Model nevie odkiaľ dáta v parametroch prišli a ani ako budú výstupné dáta sformátované alebo vypísané. Modely tvorím pre každú tabuľku v databáze.

Pre každý model budem potrebovať rovnaké funkcie. Pre tieto funkcie bude slúžiť jeden základný model, ktorý nesie názov BaseManager a nachádza sa v /app/model. Keďže modely budú spoločné pre užívateľskú aj administrátorskú časť nie je potrebné vytvárať ich do konkrétnych modulov.

Každý model určite potrebuje pripojenie do databáze, preto je to základná funkcia v BaseManagerovi.

```php
<?php declare(strict_types=1);

namespace App\Model;

use Nette\Database\Context;

class BaseManager
{
    /** @var Context */
    protected $database;

    public function __construct(Context $database)
    {
        $this->database = $database;
    }
```

Obr. 14: BaseManager model
(Zdroj: Vlastné spracovanie)

Pre pripojenie do databáze využijeme konštruktor a tým získame závislosť. Závislosti môžeme do aplikačných tried predávať štyrmi hlavnými spôsobmi :

- predávanie konštruktorom,
- predávanie settrom alebo nastavením členskej premennej,
- metódou inject*,
- anotáciou @inject u premennej s typom prístupu public.

Prvé dva spôsoby platia obecne vo všetkých objektovo orientovaných jazykoch, posledné dva spôsoby sú špecifické pre nette framework.

Na obr.č.14 je zobrazený spôsob predávania konštruktorom. Jedná sa teda o závislosť, ktorý sa predáva v okamžiku vytvárania objektu.

Trieda BaseManager takto deklaruje, že pri vytváraní objektu musí byť predaná instancia triedy Context, čo je v našom prípade prístup do databáze. Táto deklarácia je vhodná pre povinné závislosti, ktoré trieda potrebuje ku svojej funkcií, keďže bez nej nepôjde instanciu vytvoriť.

Pre každý model si vytváram osobitný adresár, v ktorom budú vždy modely s názvom končiacim slovom Manager.

Pre prihlásenie a registráciu užívateľa nám bude slúžiť model, ktorý bude spravovať tabuľku v databáze s názvom „user“. Model teda nesie názov „UserManager“ a bude obsahovať triedu rovnako pomenovanú.

UserManager bude potomok BaseManagera a ďalej sa už nebude rozširovať, preto triedu zapisujeme ako „final class UserManager extends BaseManager“.

```
public const
    TABLE_NAME = 'user',
    COLUMN_ID = 'id',
    COLUMN_EMAIL = 'email',
    COLUMN_PASSWORD_HASH = 'password',
    COLUMN_NAME = 'name',
    COLUMN_LASTNAME = 'lastname',
    COLUMN_GENDER = 'gender',
    COLUMN_STATUS = 'status',
    COLUMN_DATETIME_CREATED = 'datetime_created',
    COLUMN_DATETIME_UPDATED = 'datetime_updated',
    COLUMN_NOTE = 'note';
```

Obr. 15: UserManager konštanty
(Zdroj: Vlastné spracovanie)

Pre zjednodušenie práce s dátami, si vytváram konštanty pre každú položku v tabuľke (obr.č.15).

V triede je potrebné znova mať vytvorenú závislosť na pripojenie do databáze, keďže sa jedná o odvodenú triedu zavolám si tzv. „rodiča“ (obr.č.17).

```php
/** @var Passwords */
private $passwords;

public function __construct(Context $database, Passwords $passwords)
{
    parent::__construct($database);
    $this->passwords = $passwords;
}
```

Obr. 17: UserManager konštruktor
(Zdroj: Vlastné spracovanie)

Premennú $database si už nevytváram, trieda si ju automaticky prevezme od svojho predchodcu. Pre heslo využijem triedu Passwords od nette.

```php
/**
 * @param $data
 * @return bool|int|\Nette\Database\Table\ActiveRow
 * @throws \Exception
 */
public function add($data)
{
    $now = new DateTime();
    return $this->database->table( table: self::TABLE_NAME)->insert([
        self::COLUMN_EMAIL => $data['email'],
        self::COLUMN_PASSWORD_HASH => $this->passwords->hash($data['password']),
        self::COLUMN_NAME => $data['name'],
        self::COLUMN_LASTNAME => $data['lastname'],
        self::COLUMN_GENDER => $data['gender'],
        self::COLUMN_STATUS => $data['status'],
        self::COLUMN_DATETIME_CREATED => $now,
        self::COLUMN_DATETIME_UPDATED => $now,
        self::COLUMN_NOTE => $data['note'],
    ]);
}
```

Obr. 16: UserManager-funkcia add
(Zdroj: Vlastné spracovanie)

Pre uloženie dát do databáze som vytvoril funkciu „add“ (obr.č.16), do ktorej mi vstupuje iba jediný parameter a to je pole $data. Zápisom $this->database->table(self::TABLE_NAME) sa si určím, že sa jedná o tabuľku User, keďže som si ju nazačiatku triedy vložil do konštanty TABLE_NAME a príkaz self:: nám vyjadruje, že sa jedná o hodnoty v triede, ktorej sa nachádzame práve. Príkaz insert uloží dáta do tabuľky.

Pre editáciu záznamov v tabuľke som vytvoril funkciu edit (príloha č. 2, obr.č.46). Do tejto funkcie vstupujú dva parametre. Rovnako pole $data a $id konkrétneho užívateľa, ktorého budeme chcieť editovať. Funkcia edit je veľmi podobná funkcií add, avšak pridávame podmienku „where“ aby sme si pomocou id užívateľa nastavili, ktorý záznam v tabuľke sa bude upravovať. Pomocou príkazu update zmeníme pôvodné hodnoty v tabuľke.

Nette ma mnoho už vytvorených tried a funkcii. Keďže chcem do tabuliek vkladať aj dátum vloženia jednoducho použijem fciu DateTime().

Dôležité pre každú funkciu sú anotácie aby nette vedelo s týmito funkciami pracovať. O písanie anotácií sa stará vývojové prostredie, v našom prípade PhpStorm, stačí napísať pred funkciu „/**“ stlačiť enter a anotácia sa automaticky vytvorí. Anotácie si môžeme upravovať.

Rovnakým spôsobom si vytvorím modely pre ostatné tabuľky.

Formuláre

Trieda Nette Forms, výrazne uľahčuje vytváranie a spracovanie webových formulárov. Umožňuje:

- validovať odoslané dáta na strane serveru aj JavaScriptom,
- poskytuje zabezpečenie proti zraniteľnosti,
- zvláda niekoľko režimov vykresľovania,
- podporuje viacjazyčnosť.

Nette framework kladie veľký dôraz na bezpečnosť aplikácií, a preto dbá aj na dobre zabezpečenie formulárov. Nevyžaduje manuálne nastavovania a robí to úplne transparentne. Ochraňuje pred útokmi ako sú napríklad Cross Site Scripting (XSS) či Cross-Site Request Forgery (CSRF), odfiltruje zo vstupov kontrolné znaky, overuje validitu UTF-8 kódovania atď.

Použitím Nette Forms sa vyhneme celej rade rutinných úloh, ako sú napríklad písanie dvojitej validácie (či už na strane serveru alebo klienta) a minimalizujeme pravdepodobnosť vzniku chýb.

V našej aplikácií budú formuláre pre registráciu a prihlásenie odlišné pre bežného užívateľa a pre administrátorov. Registrácia pre administrátorov bude spočívať iba

v pridelení právomoci. Do vytvorených modulov si teda pripravíme zložky forms, ktoré budú obsahovať všetky formuláre.

V aplikácií budeme využívať rovnaké formuláre na viacerých miestach, preto si vytvoríme tzv. továreň na formuláre (FormFactory). Musíme správne pomenovať namespace aby nette vedelo, pre ktorý modul táto továreň pracuje. Továrne registrujeme do configu v adresári config/etc/form.neon v správnom tvare. Pre admina teda - App\AdminModule\Forms\FormFactory a pre bežného uživatela - App\FrontModule\Forms\FormFactory.

Pre užívateľa budeme mať na výber 2 formuláre a to pre registráciu a pre prihlásenie. Preto si vytvoríme do adresáru forms adresáre RegistrationForm a LoginForm.

Každý adresár obsahuje aj zložku templates, so súborom form.latte. V tomto súbore bude iba pomocou latte makra napísané {block content}{control form}{/block}.

```
interface IRegistrationControlFormFactory
{
    /**
     * @return RegistrationControlForm
     */
    function create();
}
```

Obr. 18: IRegistrationControlFormFactory
(Zdroj: Vlastné spracovanie)

Pre ušetrenie písania kódu využívame interface (obr.č.18.). Obsahuje iba jedinú funkciu a registrujeme ho rovnako do configu. V anotácii funkcie nastavíme formulár, pre ktorý je tento interface určený.

Samotný formulár bude mať vlastnú triedu s názvom RegistrationControlForm. Kedže chceme aby sa formulár renderoval špecifickým spôsobom a chceme ho opätovne využiť na viacerých miestach v aplikácii, je vhodné ho obaliť do UI\Control, ktorý si nesie vlastnú šablónu. Preto triedu napíšeme ako class RegistrationControlForm extends Control.

Znova pre získanie závislosti použijeme konštruktor a funkciou render si nastavíme do akej šablóny sa formulár bude vytvárať.

Formulár vytvárame ako komponentu. Komponenty predstavujú základný kameň znovu použiteľnosti kódu, zjednodušujú nám prácu a dovoľujú využívať práce komunity.

Komponenta predstavuje vykresliteľný objekt. Môžu to byť napríklad formuláre, menu, ankety atď. V rámci jednej stránky ich môže existovať ľubovolný počet. Komponenta býva z pravidla potomkom triedy Nette\Application\UI\Control. Je to aj jeden z dôvodov prečo v názvoch tried/súborov používam slovo „Control“.

```
protected function createComponentForm(): Form
{
    $form = $this->formFactory->create();
    $form->addEmail( name: 'email', label: 'E-mail:')
        ->setRequired( value: 'Položka „E-mail“ musí být vyplněna.')
        ->addRule( validator: Form::EMAIL, errorMessage: 'Vložte svůj e-mail v korektním tvaru.')
        ->addRule( validator: Form::MIN_LENGTH, errorMessage: 'E-mail musí mít alespoň %d znaky', arg: 3)
        ->addRule( validator: Form::MAX_LENGTH, errorMessage: 'E-mail musí mít nejvýš %d znaků', arg: 150);
    $form->addPassword( name: 'password', label: 'Heslo:')
        ->setRequired( value: 'Položka „Heslo“ musí být vyplněna.')
        ->addRule( validator: Form::MIN_LENGTH, errorMessage: 'Heslo musí mít alespoň %d znaky', arg: 3)
        ->addRule( validator: Form::MAX_LENGTH, errorMessage: 'Heslo musí mít nejvýš %d znaků', arg: 150);
    $form->addPassword( name: 'passwordVerify', label: 'Zadejte heslo znovu:')
        ->setRequired( value: 'Zadejte heslo znovu.')
        ->addRule( validator: Form::EQUAL, errorMessage: 'Heslá se nezhodují.', $form['password']);
    $form->addText( name: 'name', label: 'Jméno:')
        ->setRequired( value: 'Položka „Jméno“ musí být vyplněna.');
    $form->addText( name: 'lastname', label: 'Příjmení:')
        ->setRequired( value: 'Položka „Příjmení“ musí být vyplněna.');
    $form->addRadioList( name: 'gender', label: 'Pohlaví', ['F' => 'Žena', 'M' => 'Muž']);
    $form->addTextArea( name: 'note', label: 'Poznámka:');
    $form->addSubmit( name: 'add', caption: 'Uložit');
    $form->onValidate[] = [$this, 'registrationControlFormValidate'];

    return $form;
}
```

Obr. 19: RegistrationControlForm
(Zdroj: Vlastné spracovanie)

V prvom riadku funkcie (obr.č.19) si pomocou továrne vytvoríme formulár.

Pre každý prvok vo formulári môžeme určite jeho pravidlá, ktoré budú platiť. Napríklad pre email si nastavíme, že musí byť vyplnený, musí byť v korektnom tvare. Korektný tvar nám automaticky kontroluje nette framework pomocou validačného pravidla, ktoré sme pridali (Form::EMAIL), automaticky teda vie, že sa jedná o email a v prípade zle zadaného emailu si môžeme nastaviť chybovú hlášku akú budeme chcieť. Ďalšie pravidlá sú úplne na nás či sa jedná o minimálne alebo maximálne dĺžky textov, ktoré budú vo formulári napísané.

Pomocou príkazu addPassword dáme nette vedieť že daný riadok vo formulári bude heslo a teda sa k nemu tak bude správať.

Nastavíme teda rôzne podmienky a pravidlá pre všetky položky, ktoré potrebujeme.

Môže nám nastať situácia, že sa bude chcieť niekto zaregistrovať, ale zadá email, ktorý už bol použitý. Pre tento prípad je pridaný posledný riadok $form->onValidate[] = [$this, 'registrationControlFormValidate']; Znamená to, že po validácii formuláru nám spustí funkciu (obr.č.20), ktorá bude kontrolovať či daný email sa už nenachádza v databáze.

```
public function registrationControlFormValidate(Form $form, $values)
{
    if ($this->userManager->isEmailExist($values->email)) {
        $form['email']->addError('E-mail je již použit.');
    }
}
```

Obr. 20: RegistrationControlFormValidate
(Zdroj: Vlastné spracovanie)

Funkcia si teda vezme hodnotu z riadku kde máme email a pošle túto hodnotu do nášho už pripraveného modelu.

```
public function isEmailExist($email, $exclude = [])
{
    return ($this->database
        ->table( table: self::TABLE_NAME)
        ->where( condition: self::COLUMN_EMAIL, $email)
        ->where( condition: 'email NOT', $exclude)
        ->count() !== 0);
}
```

Obr. 21: IsEmailExist
(Zdroj: Vlastné spracovanie)

V našom modely UserManager máme teda prichystanú funkciu (obr.č.21), ktorá poslanú hodnotu z formuláru skontroluje so záznamami v tabuľke. V prípade, že email existuje, užívateľovi vypíše chybovú hlášku.

Pre formuláre si vytvoríme traitu. Je to v jednoduchosti ďalšia trieda, ktorá v tomto prípade je napísaná tak aby nám pre daný formulár sprístupnila funkciu getForm(), ktorá obsahuje príkaz $form = $this->getComponent('form'); Pridáme ju jednoducho na začiatku napísaním use FormTrait;

Ostáva nám pre užívateľa vytvoriť formulár pre prihlásenie. Postup ostáva úplne rovnaký. A to vytvorenie si adresáru LoginForm, v ktorom sa bude nachádzať template, interface , a súbor pre vytvorenie komponenty formuláru, ktorý však bude mať závislosť

v konštruktore iba pre našu továreň pre formuláre. Bude obsahovať iba 2 riadky a to email a heslo + tlačidlo „Prihlásiť“.

Momentálne máme prichystané formuláre po logickej stránke. Potrebujem si ich teda previesť do viditeľnej podoby na čo nám slúžia presentre a templaty.

Presentre

Presenter je objekt, ktorý vezme požiadavku preložený routerom z http požiadavku a vygeneruje odpoveď. Odpoveďou môže byť HTML stránka, obrázok, XML dokument, súbor na disku, JSON, presmerovanie atď. Obvykle sa pod pojmom presenter myslí potomok triedy Nette\Application\UI\Presenter. Podľa prichádzajúcich požiadavkou spúšťa odpovedajúce akcie a vykresľuje šablóny.

Znova si vytvoríme do zložiek AdminModule aj FrontModule adresáre s názvom presenters a s podadresárom templates.

Adresáre templates vždy obsahujú zložku s rovnakým názvom aký bude mať presenter, chceme si teda v prvom rade zobraziť formulár pre registráciu. Teda mám adresár presenters/templates/Registration a v ňom iba súbor default.latte ktorý zatiaľ obsahuje {block content} {control registrationForm} {/block}

Na začiatku projektu som si v configu volil mapovanie mapping: *: App*Module\Presenters*Presenter, preto všetky presentre budú v jednom adresári a názov sa bude končiť vždy Presenter aby ho nette správne rozpoznalo.

Vytvoríme si svoj základný presenter s názvom BasePresenter.ktorý je potomok od Nette\Application\UI\Presenter a obsahuje všetko čo potrebujeme mať v každom odvodenom presentri.

Pre registráciu užívateľov nám slúži RegistrationPresenter (obr.č.22). Pri registračnom presentri potrebujem formulár a model, ktorý mi uloží dáta z formuláru do databáze.

Závislosti si vyžiadame pomocou anotácie @inject. Do presentru injektujem iba interface formuláru.

Funkcia vytvára komponentu na konkrétny formulár, ktorý sme si pripravili a nesie názov RegistrationControlForm.

Cez premennú $control si pomocou interface vytvorím nový formulár, a do $form priradím daný formulár cez príkaz getForm(), ktorý je dostupný vďaka použitej traite v RegistrationControlForm.

```
class RegistrationPresenter extends BasePresenter
{
    /** @var IRegistrationControlFormFactory @inject */
    public $registrationControlFormFactory;

    /** @var UserManager @inject */
    public $userManager;

    public function createComponentRegistrationForm(): RegistrationControlForm
    {
        $control = $this->registrationControlFormFactory->create();
        $form = $control->getForm();

        $form->onSuccess[] = function (Form $form, $values) {
            try {
                $this->userManager->add($values);
            } catch (UniqueConstraintViolationException $e) {
                throw new \Exception($e->getMessage());
            }

            $this->flashMessage( message: 'Úspěšná registrace.');
            $this->redirect( destination: 'Homepage:');
        };

        return $control;
    }
}
```

Obr. 22: RegistrationPresenter
(Zdroj: Vlastné spracovanie)

Po vyplnení formulára, validácií a úspešnom odoslaní využijeme znova náš model userManager a jeho funkciu add(), ktorá nám uloží dáta do databáze. Vo formulári každý riadok nesie rovnaký názov ako položka v tabuľke, do ktorého má byť daná hodnota uložená. Funkciu píšem vo forme try/catch. Do try píšem všetko čo požadujem aby sa spravilo a catch mi zachytáva výnimky.

Ak všetko prejde bez chyby, užívateľovi sa zobrazí hláška o úspešnej registrácií a presmerujeme ho poprípade na inú časť webu.

Pre prihlasovanie užívateľa si vytvoríme LoginPresenter, rovnako aj do templates adresár Login, kde bude súbor default.latte, ktorý bude obsahovať iba {block content} {control loginForm} {/block}

Keďže sa nám jedná o bezpečnosť užívateľov a teda nechceme aby sa do účtu dostala neoprávnená osoba, nette framework kladie veľký dôraz na zabezpečenie. Preto pri prihlásení budeme overovať či sa jedná o oprávnenú osobu pre daný účet.

Vytvoríme si pre FrontModule aj AdminModule vlastný autentikátor, ktorý je potomok od triedy Nette\Security\IAuthenticator. Upravíme si ho teda aby nám správne komunikoval s databázou, overoval správnosť hesla, poprípade či má účet stále oprávnenie k prihláseniu. Pre každý module bude mierne odlišný autentikátor..

V LoginPresenter si už len vytvorím závislosť na vlastný autentikátor a na interface pre formulár rovnako ako pre RegistrationPresenter.

Nastavenie autentikátoru a následné prihlásenie mi prevedú funkcie

```
$this->getUser()->setAuthenticator($this->authenticator);
```

```
$this->getUser()->login($values->email, $values->password);
```

Nastavíme si aj časový limit za aký užívateľa automaticky odhlási napr.

```
$this->getUser()->setExpiration('+ 24 hour');
```

Pre manuálne odhlásenie nám slúžia akcia (obr.č.23).

```
public function actionOut()
{
    $this->getUser()->logout( clearIdentity: True);
    $this->flashMessage( message: 'Byl jste odhlášen.');
    $this->redirect( destination: 'Login:');
}
```

Obr. 23: LoginPresenter - actionOut
(Zdroj: Vlastné spracovanie)

Rovnakým spôsobom postupujem pri všetkých častiach aplikácie. To znamená, že na logické operácie si vytváram vlastné modely, v ktorých riešim prácu s databázou. Následne si vytvorím template a príslušný presenter, ktorý bude slúžiť ako prostredník medzi ostatnými časťami aplikácie.

3.3.3 Tretia iterácia

Základné úlohy:

- entity,
- nasadenie nette balíčkov,
- výpisy a správa užívateľov,
- výpisy a správa bicyklov.

Entity

Entita môžeme chápať ako objekt nesúci dáta. Nemá prístup k databáze ani k iným dátovým úložiskám. Jej úlohou je uschovávať a predávať dáta. Entita by nemala byť iba obyčajnou „prepravkou“, môže a aj by mala obsahovať napríklad validáciu vstupu, ako je napríklad ošetrenie dĺžky predávanej hodnoty atribútu. Taktiež môže obsahovať ľubovoľné metódy pracujúce s dátami, ktoré jej náležia. Existujú rôzny entity manageri, ktorý nám pomáhajú v práci s nimi. V našom prípade však nebudeme potrebovať žiadneho.

Pre užívateľa si vytvoríme entitu s názvom UserEntity. a bude sa nachádzať v rovnakom adresári ako UserManager. Entita bude obsahovať premenné pre každú položku v databáze. V anotácií premenných sú uvedené dátové typy.

```
/**
 * @return string
 */
public function getName(): string
{
    return $this->name;
}

/**
 * @param string $name
 */
public function setName(string $name): void
{
    $this->name = $name;
}
```

Obr. 24: UserEntity - getter a setter
(Zdroj: Vlastné spracovanie)

Entita pozostáva hlavne z gettrov a settrov (obr.18). Jedná sa o funkcie, ktoré nám pomôžu pri práci s entitami zvonku. Gettrom sa dostanem k danému záznamu a settrom si nastavím záznam. Dôležité sú znova anotácie, hlavne pri určovaní dátových typov.

V entitách môžem rovnako pracovať s konštantami. Napríklad v tabuľke mám položku gender a do nej ukladám iba hodnoty 1 alebo 0. Tie si jednoducho v entite priradím ku konštante.

Zápis je:

public const GENDER_MAN = 1;

public const GENDER_WOMAN = 0;

Následne jednoduchou funkciou (obr.č.25) si zvolím ako bude užívateľ vidieť tieto položky napríklad vo formulároch. To znamená, že on si vyberie typ Muž, ale do databáze sa uloží iba hodnota 1. Aby entita obsahovala nejaké hodnoty potrebujem si ich

```php
/**
 * @param $row
 * @return UserEntity
 */
public function mapToUser($row)
{
    $user = new UserEntity();
    $user->setId($row['id']);
    $user->setEmail($row['email']);
    $user->setPassword($row['password']);
    $user->setName($row['name']);
    $user->setLastname($row['lastname']);
    $user->setGender($row['gender']);
    $user->setStatus($row['status']);
    $user->setDatetimeCreated($row['datetime_created']);
    $user->setDatetimeUpdated($row['datetime_updated']);
    $user->setNote($row['note']);

    return $user;
}
```

Obr. 26: UserManager - mapToUser
(Zdroj: Vlastné spracovanie)

namapovať. Preto do UserManager vytvorím funkcie (obr.č.26 a obr.č.27), ktoré budú dáta predávať do entít v momente ak budú potrebné.

Vždy keď potrebujem naplniť entitu teda vytváram novú. Pomocou setrov nastavím hodnoty, ktoré sú v databáze na danú entitu.

Funkcia mapToEntity (obr.č.27) nám umožní aby nám vyplnená entita zobrazila napríklad v formulároch, ktoré sa majú editovať dané hodnoty z databáze.

```php
/**
 * @param array $rows
 * @return array
 */
public function mapToEntity(array $rows)
{
    $output = [];
    foreach ($rows as $row) {
        $output[] = $this->mapToUser($row);
    }

    return $output;
}
```

Obr. 27: UserManager - mapToEntity
(Zdroj: Vlastné spracovanie)

Entity vo formulároch

Ak by si napríklad chcel užívateľ upraviť svoj profil, formulár bude veľmi podobný registračnému ale je potrebné ho mať pred vyplnený dátami, ktoré zadával pri registrácií.

Pre tento účel máme UserControlForm.

```php
interface IUserControlFormFactory
{
    /**
     * @param UserEntity|null $userEntity
     * @return UserControlForm
     */
    function create(UserEntity $userEntity = null);
}
```

Obr. 28: IUserControlFormFactory
(Zdroj: Vlastné spracovanie)

Do interface (obr.č.28) pridávame ako parameter entitu užívateľa. V konštruktore (príloha č.2 obr.47) si potrebujeme vytvoriť závislosť na entite a nastaviť defaultne hodnoty v prípade, že sú nejaké údaje v databáze.

Pre nastavenie hodnôt do formuláru nám slúži funkcia setDefaults (príloha č. 2, obr.48). Zistí či je v danom riadku nejaká hodnota a ak je, tak nám vráti formulár s už prichystanými dátami, ktoré môžeme editovať.

Problém nám môže nastať pri hesle. Keďže užívateľ pri registrácií zadával svoje heslo do databáze ukladáme iba jeho hash, to znamená, že jeho reálne heslo nevieme. Pomocou entity vieme zistiť, že v databáze zrovna v položke heslo je určitá hodnota, teda táto položka nie je prázdna.

Heslu nedáme podmienku, že musí byť vyplnené, keďže rátame, že pri registrácii užívateľ vyplňoval povinné heslo. Pridáme teda k heslu v tomto formuláre iba podmienku if (empty($this->userEntity)) {$password->setRequired('Položka "Heslo" musí být vyplňena');} Teda nám chybovú hlášku vypíše iba v prípade ak by v databáze pre položku heslo nebola žiadna hodnota.

```
$form->addSelect( name: 'gender', label: 'Pohlaví', UserEntity::getGenderList())
    ->setPrompt( prompt: 'Vyberte hodnotu')
    ->setRequired( value: 'Položka "Pohlaví" musí být vyplňena');
```

Obr. 29: UserControlForm - gender
(Zdroj: Vlastné spracovanie)

Entity využijeme vo formulári aj pre výberový zoznam (obr.č.29). Užívateľovi po rozbalení zoznamu ukáže v tomto prípade iba údaje „Muž“, „žena“ či popis „Vyberte hodnotu“, ktoré sme si už predtým vytvorili v UserEntity.

```
public function userControlFormValidate(Form $form, $values)
{
    $exclude = empty($this->userEntity) ? [] : [$this->userEntity->getEmail()];

    if ($this->userManager->isEmailExist($values->email, $exclude)) {
        $form['email']->addError('E-mail je již použit.');
    }
}
```

Obr. 30: UserControlForm- výnimka ukladania emailu
(Zdroj: Vlastné spracovanie)

Podobný problém ako pri hesle nastane aj pri emaily, teda že nechceme zmeniť email, ktorý užívateľ zadával pri registrácii. Vytvorím si teda výnimku (obr.č.30) ukladania emailu.

Nette balíčky

Mnoho problémov, s ktorými sa pri písaní kódu stretávame pri tvorení aplikácie, určite riešime pri viacerých projektoch poprípade tieto problémy už niekto mal. Preto je zbytočné zbytočne rozmýšľať ako by sme napísali kód pre určité veci, keďže už to niekto raz vyriešil a daný kód skvele funguje. Preto sú tu aj nette balíčky, ktorými si vieme

zjednodušiť prácu. Ktokoľvek môže na internet zavesiť svoje riešenie problému a jednoducho vieme daný kód použiť aj v našom projekte.

Dávam do popredia spoločnosť Contributte, ktorá sa venuje vytváraniu týchto balíčkov pre nette framework a teda si u nich môžeme nájsť zrovna to čo potrebujeme .

Datagrid

Administrátor stránky potrebuje mať prehľad v databáze, či sa jedná už o užívateľov alebo bicykle. Pre prehľadný výpis a správu dát využijem balíček datagrid. Inštalácia je jednoduchá. V príkazovom riadku sa dostanem do adresára projektu a napíšem „composer require ublaboo/datagrid“. Na stránke pre balíček vždy nájdeme návod, čo daný balíček obsahuje, a ako sa sním pracuje

S datagridom sa pracuje veľmi podobne ako s formulármi. Pre všetky gridy mi bude slúžiť adresár /grids pod AdminModule Vytvorím si v ňom zložku bikeGrid (cesta AdminModule/grids/bikeGrid) a v nej zložku templates kde budem mať súbor default.latte a príkaz {control grid}

```php
trait GridTrait
{
    /**
     * @param string $name
     */
    protected function resolveGridRequest($name = 'grid'): void
    {
        if ($this->getPresenter()->isAjax()) {
            $this->redrawControl( snippet: 'flashes');
            $this->getGrid($name)->reload();
        } else {
            $this->redirect( destination: 'this');
        }
    }

    /**
     * @param string $name
     * @return DataGrid
     */
    public function getGrid($name = 'grid')
    {
        return $this->getComponent($name);
    }
}
```

Obr. 31: GridTrait
(Zdroj: Vlastné spracovanie)

Vo všetkých gridoch využívam traitu (obr.č.31), ktorá rieši napríklad asynchronné spracovanie. Zjednodušene to znamená, že mení obsah stránky bez kompletného znovu načítania.

Rovnako ako pre formuláre aj pre gridy vytváram interface. Pre bicykle mám interface s názvom IBikeGridFactory, ktorý obsahuje iba funkciu create. Trieda BikeGrid bude rovnako rozširovať Nette\Application\UI\Control, obsahuje traitu, konštruktor so závislosťou na model pre správu bicyklov (BikeManager), renderovaciu funkciu.

```
public function createComponentGrid()
{
    $grid = new DataGrid();
    $grid->setDataSource($this->bikeManager->getDataSource());
    $grid->setColumnsHideable();
    $grid->addColumnText( key: 'id', name: 'ID')
        ->setSortable();
    $grid->addColumnText( key: 'code', name: 'Kód')
        ->setSortable()
        ->setFilterText( columns: 'code')
        ->setPlaceholder( placeholder: 'Hledat');
```

Obr. 32: BikeGrid - createComponentGrid
(Zdroj: Vlastné spracovanie)

Vytváram si teda nový DataGrid, do ktorého načítam dáta cez bikeManager, ktorý obsahuje funkciu getDataSource. Funkcia nám jednoducho z danej tabuľky poskytne všetky dáta. Príkaz vo funkcii je return $this->database->table(self::TABLE_NAME);

Môžem každému stĺpci určiť vlastné pravidlá, pridať rôzne filtrovania atď.

```
    $grid->addColumnStatus( key: 'state', name: 'Status')
        ->addOption( value: 0, text: 'Kradené"')
        ->endOption()
        ->addOption( value: 1, text: 'V oprávněném vlastnictví"')
        ->endOption()
        ->setSortable()
        ->onChange[] = [$this, 'statusChange'];
    $grid->addAction( key: 'delete', name: 'Zmazať', href: 'delete!');
    $grid->addAction( key: 'edit', name: 'Editovať', href: 'edit!');

    return $grid;
}
```

Obr. 33: BikeGrid - addColumnStatus
(Zdroj: Vlastné spracovanie)

Pri prepínaní medzi rôznymi stavmi, napríklad či je bicykel kradnutý alebo je v oprávnenom vlastníctve si jednoducho pridám rôzne možnosti (obr.č.33) a na záver sa

odkážem na funkciu, ktorá mi tieto zmeny bude automaticky ukladať do databáze bez zbytočného znovu načítania stránky.

```
public function statusChange($id, $value)
{
    if (in_array($value, [0, 1])) {
        $this->bikeManager->updateState($id, $value);
    }

    $this->resolveGridRequest();
}
```

Obr. 34: BikeGrid - statusChange
Zdroj: Vlastné spracovanie)

Pri funkcii statusChange (obr.č.34) si posielam hodnoty cez bikeManager na update v databáze. Funkcia resolveGridRequest, ktorá pochádza z GridTrait nám zabezpečuje plynulé prepnutie stavu.

Posledné pridané akcie, sú tlačidlá, ktoré nám po kliknutí buď daný záznam zmažú poprípade nás prepnú na editáciu záznamu.

Poslednú položku v akcii píšeme s výkričníkom. Následne funkciu nazveme handle + názov s výkričníkom aby sa funkcia automaticky priradila k danému tlačidlu.

```
public function handleDelete($id): void
{
    $this->bikeManager->delete($id);
    $this->getPresenter()->flashMessage( message: 'Operace proběhla úspěšne', type: 'info');
    $this->resolveGridRequest();
}

public function handleEdit($id): void
{
    $this->getPresenter()->redirect( destination: 'Bike:edit', ['id' => $id]);
}
```

Obr. 35: BikeGrid - handle funkcie
(Zdroj : Vlastné spracovanie)

Pri funkcii handleDelete (obr.č.35) jednoducho pošleme id bicykla do funkcie v bikeManagerovi, ktorý celý záznam vymaže. Funkcia handleEdit (obr.č.35) nás presmeruje na BikePresenter (priloha č.2 obr.č.49) a jeho akciu edit, rovnako odosiela aj id bicykla. Táto akcia si v presentri vezme id bicykla a pomocou bikeManagera namapuje dáta na entitu. Formulár automaticky rozozná či sa jedná o editáciu alebo o nové vytvorenie. Gridy vytvárame v presentroch podobným spôsobom ako formuláre.

ImageStorage

V aplikácii je možné nahrať obrázky bicyklov. Pre nahrávanie súborov slúži trieda Nette\Http\FileUpload. Pre prácu s obrázkami využijeme balíček ImageStorage ktorý dokáže ukladať, transformovať, mazať obrázky a mnoho ďalších funkcii. Inštalácia je znova jednoduchá, stačí do príkazového riadka sa nastaviť na adresár s projektom a napísať „composer require contributte/image-storage“.

Prácu s obrázkami využijeme ako pre užívateľa tak aj pre administrátora nie je teda potrebné rozdeľovať na 2 časti. Vytvoríme si adresáru /app zložku services. Môže sa stať, že obrázky nebudeme v aplikácii ukladať presne podľa nastavení balíčka a preto si vytvoríme aj vlastné. Preto si do adresáru /services vytvoríme funkcie pre oba spôsoby. V konfiguračnom súbore môžeme kedykoľvek zmeniť spôsob spracovania obrázkov. Napríklad v ext/image.neon mám zaregistrovanú službu pre prácu s balíčkom a v etc/image.neon mám zaregistrovanú službu pre vlastné nastavenia. Na základe toho, ktorú službu chcem používať si includnem daný súbor do common.neon.

Pre vlastné ukladanie obrázkov, poprípade súborov som si vytvoril zložku common kde mám triedy Files, Parameters a String. V ktorých sú funkcie na generovanie názvu súborov, vytváranie prípon atď. Súbory sa ukladajú ako do databáze pod určitým popisom, tak aj server kde budú k dispozícii.

```
/**
 * @param FileUpload $fileUpload
 * @return mixed
 */
public function upload(FileUpload $fileUpload)
{
    $fileName = Files::generateName($fileUpload);
    $image = $this->imageStorage->saveContent(
        file_get_contents($fileUpload->getTemporaryFile()),
        $fileName,
        namespace: self::DIR
    );

    return $image->identifier;
```

Obr. 36: ImageStorage - upload
(Zdroj: Vlastné spracovanie)

Príklad uploadu súboru (obr.31), ktorý využíva balíček a aj našu triedu „Files“, ktorá mi generuje meno pre súbor, ktorý sa ukladá.

```php
/**
 * @param FileUpload $fileUpload
 * @return string
 * @throws \Nette\Utils\ImageException
 */
public function upload(FileUpload $fileUpload)
{
    $image = $this->toImage($fileUpload);
    $absolutePath = $this->getAbsolutePath();
    $fileName = Files::generateName($fileUpload);

    $path = $absolutePath . "/" . $fileName;
    $image->save($path);

    return self::DIR . '/' . $fileName;
}
```

Obr. 37: ImageService - upload
(Zdroj: Vlastné spracovanie)

Pre porovnanie funkcie upload (obr.č.37). Mnou vytvorená trieda ImageService nevyužíva balíček na prácu so súbormi a teda pracujem len s vlastnými triedami, poprípade defaultnými od nette framerworku.

3.3.4 Štvrtá iterácia

Základné úlohy :

- nasadenie Bootstrapu,
- buildovanie Front-endu,
- navigačná lišta.

Bootstrap

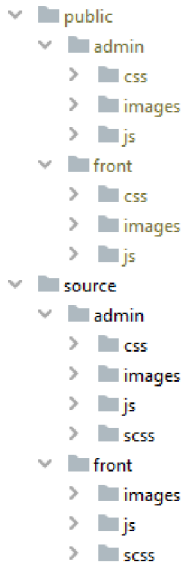
Pre nasadenie Bootstrap frameworku existuje viacero možností. Napríklad inštalácia cez composer, vložení linkov dostupných z webovej stránky bootstrapu do hlavného layoutu či už pre CSS, JavaScript alebo jQuery.

V našom prípade som využil predpripravený tzv. „front-end“ stack od nášho senior programátora, ktorý som si pullnul pomocou vyššie spomínaného softwaru Sourcetree do svojho projektu. Tento stack mi do projektu nahráva všetky časti Bootstrapu.

Buildovanie Front-endu

Pre prácu s použitým „front-stackom“ si nainštalujem Node.js, čo j vlastne prostredie umožňujúce mi spúšťať JavaScriptový kód mimo webový prehliadač. Jeho návrh a primárny účel je tvorba serverovej časti webových aplikácii.

Pri buildovaní Front-endu využijeme adresár /www. V adresári budeme mať časti public a source (obr.č.38).



Obr. 38: Adresáre public a source
(Zdroj: Vlastné spracovanie)

V oboch adresároch sú nahraté všetky časti Bootstrapu. Pre nás pri budovaní Front-endu je dôležitá časť source. Keďže chceme grafický dizajn mať vlastný nie ten predpripravený od Bootstrapu musíme určité jeho časti zmeniť, poprípade vytvoriť vlastné. Pred samotnou zmenou, sa v príkazovom riadku vždy nastavím na môj projekt a spustím príkaz „grunt dev“. Vďaka nášmu „front-stacku“ automaticky každú zmenu, ktorú spravím v adresári source sa mi táto zmena prejaví do public. Public zložka zjednodušene predstavuje finálny vzhľad stránky.

V našom hlavnom layoute si vložíme do hlavičky <link rel="stylesheet" href="{$basePath}/public/front/css/main.min.css"> , čo predstavuje hlavný zdroj csska použitého pre web a na koniec tela layoutu <script src="{$basePath}/public/front/js/main.js"></script> , čo predstavuje hlavný zdroj JavaScriptu pre web.

Pri úprave stránky využívame kompilovaný jazyk Sass, ktorý rozširuje jazyk CSS o premenné, cykly, podmienky, mixiny, funkcie atď. Šetrí nám čas, množstvo napísaného kódu, je prehľadnejší a ľahšie sa udržuje. Kód zapisujeme pomocou syntaxe SCSS (Sassy CSS), ktorá rozširuje CSS syntax, to znamená, akýkoľvek CSS kód je kompatibilný s SCSS nie však naopak.

Pokaždé keď vytvoríme nejaký SCSS súbor (do priečinkov scss -viď.obr.35), musíme ho skompilovať do CSS súboru pomocou kompilátoru Sass, ktorý je obsiahnutý v našom „front-end stacku“. To znamená, že každý kód, ktorý vytvoríme do adresáru scss sa automaticky skompiluje na zápis vo forme CSS.

Využívam pre zápis tzv. „Nesting“, čo je štýl zápisu, ktorý mi umožňuje vnorovať CSS štýly do seba. Potomka vždy zapíšem ako každú jeho vlastnosť. Táto funkcia preprocesoru je veľmi užitočná, keďže ušetrí mnoho nepríjemnosti s vypisovaním desiatok selektorov.

Obr. 39: Vyhľadávací formulár
(Zdroj: Vlastné spracovanie)

Napríklad pre vyhľadávací formulár použijem vnorenie (príloha č.2, obr.50). Keďže sa formulár nachádza v sekcii header, upravujem pozadia samotného headru, pozadie orámovania, inputy, tlačidlo „submit“ , poprípade samotný element <a>.

Pre samotné obrázky použité napriklad pri backgroundoch, používam pôvodnú grafiku (príloha č.3, obr. č. 51), ktorú si pomocou PhotoShopu upravujem, na kontrétne časti, ktoré potrebujem

Dôležitá časť pri úpravách je pozíciovanie. Máme na výber pozíciu absolútnu, ktorý umiestni objekt do strunky na udané súradnice bez ohľadu na okolitý text alebo pozíciu relatívnu, ktorý určuje iba o koľko sa má objekt posunúť oproti svojej normálnej polohe.

Aby som nemusel ručne štýlovať každý jeden formulár použitý v projekte využijem renderer (obr. 40) pre formuláre. Ktorý zakonponujem do našich továrni na formuláre pomocou príkazu použitého vo funkcii create() :

```
$form->onRender[] = [$this, 'makeBootstrap4'];
```

```php
public function makeBootstrap4(Form $form): void
{
    $renderer = $form->getRenderer();

    $renderer->wrappers['controls']['container'] = null;
    $renderer->wrappers['pair']['container'] = 'div class="form-group row"';
    $renderer->wrappers['pair']['.error'] = 'has-danger';
    $renderer->wrappers['control']['container'] = 'div class=col-sm-9';
    $renderer->wrappers['label']['container'] = 'div class="col-sm-3 col-form-label"';
    $renderer->wrappers['control']['description'] = 'span class=form-text';
    $renderer->wrappers['control']['errorcontainer'] = 'span class=form-control-feedback';
    $renderer->wrappers['control']['.error'] = 'is-invalid';

    foreach ($form->getControls() as $control) {

        $type = $control->getOption('type');

        if ($type === 'button') {
            $control->getControlPrototype()->addClass(empty($usedPrimary) ? 'btn btn-primary' : 'btn btn-secondary');
            $usedPrimary = true;

        } elseif (in_array($type, ['text', 'textarea', 'select'], strict: true)) {
            $control->getControlPrototype()->addClass('form-control');

        } elseif ($type === 'file') {
            $control->getControlPrototype()->addClass('form-control-file');

        } elseif (in_array($type, ['checkbox', 'radio'], strict: true)) {
            if ($control instanceof Nette\Forms\Controls\Checkbox) {
                $control->getLabelPrototype()->addClass('form-check-label');
            } else {
                $control->getItemLabelPrototype()->addClass('form-check-label');
            }
            $control->getControlPrototype()->addClass('form-check-input');
            $control->getSeparatorPrototype()->setName('div')->addClass('form-check');
        }
    }
}
```

Obr. 40: Bootstrap renderer
(Zdroj: Vlastné spracovanie)

Navigačná lišta

Framework Bootstrap obsahuje veľké množstvo komponent, ktoré pomáhajú vo výslednom vzhľade stránky. Jednou z najdôležitejších je aj komponenta navbar.

Jedná sa o responzívnu navigačnú lištu, ktorá môže obsahovať najrôznejšie elementy a na mobilných zariadeniach sa zmenší do podoby tlačidla, otvárajúceho zvislé menu.

Navigačná lišta (obr.č.41) sa tvorí elementom <nav> a triedou „.navbar“. Pre dosiahnutie responzivity sa používa trieda „.navbar-expand-md“. Breakpointy, teda ako presne veľké je ktoré zariadenie som popisoval v teoretickej časti práce.

```
{block}
    <header>
        <nav class="navbar navbar-expand-md navbar-dark bg-nav-black">
            <a></a>
            <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent"
                    aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarSupportedContent">
                <ul class="navbar-nav">
                    {if !$user->isLoggedIn()}
                        <li n:class="$presenter->isLinkCurrent('Login:') ? active, 'nav-item'">
                            <a n:href="Login:" n:class="nav-link">Prihlášení</a>
                        </li>
                        <li n:class="$presenter->isLinkCurrent('Registration:') ? active, 'nav-item'">
                            <a n:href="Registration:" n:class="nav-link, 'sketch-register'">Registrace</a>
                        </li>
                    {else}ho
                        <li n:class="nav-item">
                            <a n:href="Homepage:" n:class="nav-link">{$user->getIdentity()->email}</a>
                        </li>
                        <li n:class="nav-item">
                            <a n:href="Login:out" n:class="nav-link">Odhlásiť</a>
                        </li>
                    {/if}
                </ul>
            </div>
        </nav>
    </header>
{/block}
```

Obr. 41: Navigačná lišta
(Zdroj: Vlastné spracovanie)

Ako ďalší prvok viditeľný iba na mobilných zariadeniach je tlačidlo, ktoré slúži k rozbaleniu zvislého menu. Jedná sa o element <button> s triedou „.navbar-toggler“. Do data-target uvádzame selektor pre element reprezentujúci ďalší obsah navigácie, ktorý má byť na mobilných zariadeniach práve vo výsuvnom menu namiesto v navigačnej lište. Na väčších zariadeniach bude tento obsah prítomný priamo v navigačnej lište.

Jednotlivé časti navigácie vkladáme ako prvky zoznamu <ul> s triedou „navbar-nav“, ktorý sa rozťahuje do šírky tak ako je to možné. K odkazu na stránku, na ktorej sa práve nachádzame priradzujeme triedu „.active“. Navigačná lišta sa zobrazuje inak pre neprihláseného užívateľa a inak pre prihláseného.

ZÁVER

Táto diplomová práca bola predovšetkým zameraná na vytvorenie webovej aplikácie pre projekt KradenaKola.cz.

V prvej časti boli teoreticky popísané programovacie jazyky, technológie a metodiky použité pri vývoji či analýze samotného projektu.

Nasledujúca časť bola zameraná na predstavenie projektu KradenaKola.cz, v stručnosti opísané ako projekt funguje a analýza marketingovej stratégie do budúcnosti, s požiadavkou firmy pre rozšírenie v blízkej dobe o platenú inzerciu. Súčasťou analýzy je aj výber vhodného PHP frameworku.

Keďže súčasná aplikácia nesplňovala podmienky pre budúce rozšírenie, firma rozhodla o vytvorení novej webovej aplikácie, ktorá bude obsahovať všetky funkcionality ako aj pôvodná aplikácia, ktorá bude pripravená na možnosti rozširovania hlavne pre oblasť marketingu.

Pre inováciu projektu bol založený 5 členný tím, ktorý postupoval podľa metodiky SCRUM. Mojou úlohou v tíme bolo vytvorenie webovej aplikácie. Celý tento postup je popísaný v praktickej časti práce, kde som sa zameral hlavne na technologickú časť projektu, to znamená programovanie back-endovej a front-endovej časti webovej aplikácie. Na základných úlohách som vysvetlil prácu s Nette frameworkom a predstavil, konkrétne riešenia použité v aplikácií.

Novo vytvorená webová aplikácia pre projekt KradenaKola.cz je postavená na moderných technológiách, ktoré sa v súčasnosti používajú pri vývoji softvéru. Vizuálna časť je spracovaná podľa pôvodného grafického návrhu stránky avšak momentálne je plne responzívna vďaka využitiu Bootstrap frameworku.

ZOZNAM POUŽITEJ LITERATÚRY

[1] DUCKETT, J. *HTML and CSS: Design and Build Websites*. 1st ed. Indianapolis: WILEY, 2011. 512 s. ISBN 978-1-118-00818-8.

[2] HTML 5.3. *World Wide Web Consortium (W3C)* [online]. Copyright © 2018 [cit. 10.11.2019]. Dostupné z: https://www.w3.org/TR/html53/

[3] Web Tutorial CSS. *Web Tutorial Domov* [online]. Copyright © 2012 [cit. 10.11.2019]. Dostupné z: http://www.webtutorial.cekuj.net/css/uvoddocss.html

[4] Cascading Style Sheets (CSS) Snapshot 2010. *World Wide Web Consortium (W3C)* [online]. Copyright © 2011 [cit. 10.11.2019]. Dostupné z: https://www.w3.org/TR/css-2010/

[5] Cascading Style Sheets Level 1.*World Wide Web Consortium (W3C)* [online]. Copyright © 2011 [cit. 10.11.2019]. Dostupné z: https://www.w3.org/TR/REC-CSS1/

[6] Cascading Style Sheets Level 2 Revision 2 (CSS 2.2) Specification. *World Wide Web Consortium (W3C)* [online]. Copyright © 2016 [cit. 10.11.2019]. Dostupné z: https://www.w3.org/TR/CSS22/

[7] Selectors. *World Wide Web Consortium (W3C)* [online]. Copyright © 2016 [cit. 10.11.2019]. Dostupné z: https://www.w3.org/TR/1998/REC-CSS2-19980512/selector.html

[8] Cascading Style Sheets Level 2 Revision 2 (CSS 2.2) Specification. *World Wide Web Consortium (W3C)* [online]. Copyright © 2016 [cit. 13.11.2019]. Dostupné z: https://www.w3.org/TR/CSS22/

[9] CSS preprocesory: méně psaní, vyšší efektivita - Zdroják. *Zdroják - o tvorbě webových stránek a aplikací* [online]. Copyright © 2011 [cit. 13.11.2019] Dostupné z: https://www.zdrojak.cz/clanky/css-preprocesory-mene-psani-vyssi-efektivita/

[10] JavaScript. *JavaScript | MDN*. [online]. Copyright © 2005 [cit. 13.11.2019]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[11] ZAKAS, N. *JavaScript pro webové vývojáře*. Computer Press, Brno. 2009. ISBN 987-80-251-2509-0.

[12] Web Tutorial JQuery. *Web Tutorial Domov* [online]. Copyright © 2012 [cit. 13.11.2019]. Dostupné z: http://www.webtutorial.cekuj.net/javascript/jquery.html

[13] Web Tutorial PHP. *Web Tutorial Domov* [online]. Copyright © 2012 [cit. 13.11.2019]. Dostupné z: http://www.webtutorial.cekuj.net/php/php.html

[14] PHP: History of PHP - Manual . *PHP: Hypertext Preprocessor* [online]. Copyright © 2001 [cit. 20.11.2019]. Dostupné z: https://www.php.net/manual/en/history.php.php

[15] What is PhpStorm?. VPS Hosting, Instant VPS Server Activation, *100% SSD Servers MonoVM* [online]. Copyright ©2020 [cit. 20.11.2019]. Dostupné z: https://monovm.com/blog/what-is-phpstorm/

[16] Introduction - Composer. *Composer* [online]. Copyright ©2020 [cit. 20.11.2019]. Dostupné z: https://getcomposer.org/doc/00-intro.md

[17] 10 PHP Frameworks For Developers - Best of - Hongkiat. *Hongkiat* [online]. Copyright ©2020 [cit. 20.11.2019]. Dostupné z: https://www.hongkiat.com/blog/best-php-frameworks/

[18] Nette – Pohodlný a bezpečný vývoj webových aplikací v PHP. *Nette – Comfortable and Safe Web Development in PHP* [online]. Copyright © 2008, 2020 Nette Foundation. All rights reserved. [cit. 10.02.2020]. Dostupné z: https://nette.org/cs/

[19] David Grudl: Nette Framework čeká zlomový rok - Zdroják. *Zdroják - o tvorbě webových stránek a aplikací* [online]. Copyright © 2014 [cit. 10.02.2020] Dostupné z: https://www.zdrojak.cz/clanky/david-grudl-nette-ceka-zlomovy-rok/

[20] Nette Revolution 2.2 » phpFashion. *phpFashion* [online]. Copyright © 2004, 2020 [cit. 10.02.2020]. Dostupné z: https://phpfashion.com/nette-revolution-2-2

[21] Průvodce | Latte Templating Engine. *Latte – The Safest & Truly Intuitive Templates for PHP* [online]. Copyright © 2008, 2020 Nette Foundation. All rights reserved. [cit. 10.02.2020]. Dostupné z: https://latte.nette.org/cs/guide

[22] Průvodce | Tracy debugging tool. *Tracy – A Must-Have Debugging Tool for All PHP Developers* [online]. Copyright © 2008, 2020 Nette Foundation. All rights reserved. [cit. 10.02.2020]. Dostupné z: https://tracy.nette.org/cs/guide

[22] Nette Framework: MVC & MVP - Zdroják. *Zdroják - o tvorbě webových stránek a aplikací* [online]. Copyright © 2009 [cit. 10.02.2020]. Dostupné z: https://www.zdrojak.cz/clanky/nette-framework-mvc-mvp/

[23] Presentery | Nette Docs. [online]. Copyright © 2008, 2020 Nette Foundation. All rights reserved. [cit. 13.02.2020]. Dostupné z: https://doc.nette.org/cs/3.0/presenters

[24] Introduction · Bootstrap. *Bootstrap The most popular HTML, CSS, and JS library in the world.* [online]. Copyright © 2019 [cit. 13.02.2020]. Dostupné z: https://getbootstrap.com/docs/4.4/getting-started/introduction/

[25] Bootstrap 4 Grid Basic. *W3Schools Online Web Tutorials* [online]. Copyright © 1999 - 2020 [cit. 13.02.2020]. Dostupné z: https://www.w3schools.com/bootstrap4/bootstrap_grid_basic.asp

[26] What is XAMPP and How to Install XAMPP on your Local Computer?. WPBlogX - [online]. [cit. 15.02.2020]. Dostupné z: https://www.wpblogx.com/what-is-xampp/

[27] MySQL - Introduction - Tutorialspoint. *Apache HTTP Server Test Page powered by CentOS* [online]. Copyright © Copyright 2020. All Rights Reserved. [cit. 15.02.2020]. Dostupné z: https://www.tutorialspoint.com/mysql/mysql-introduction.htm

[28] Adminer - Správa databáze v jednom PHP souboru. *Adminer - Database management in a single PHP file* [online]. Copyright © 2019 [cit. 15.02.2020]. Dostupné z: https://www.adminer.org/cs/

[29] CHACON, S.; STRAUB, B. *Pro Git*: 2nd ed. Berkeley, CA New York, NY: Apress, 2014. ISBN 978-1484200773.

[30] Git - Základy systému Git. *Git* [online] Copyright © 2019. [cit. 15.02.2020]. Dostupné z: https://git-scm.com/book/cs/v2/%C3%9Avod-Z%C3%A1klady-syst%C3%A9mu-Git

[31] WebML – datové modelování | Interval.cz. Interval.cz | *Svět Internetu, Technologií a Bezpečnosti* [online]. Copyright © 2019 [cit. 15.02.2020]. Dostupné z: https://www.interval.cz/clanky/webml-datove-modelovani/

[32] CHLAPEK, Dušan, ŘEPA, Václav a STANOVSKÁ, Iva. Analýza a návrh informačních systémů. 1. vyd. Praha: Oeconomica, 2011. 157 s. ISBN 978-80- 245-1782-7

[33] Relační databázové systémy. *posterus.sk, portál pre odborné publikovanie* [online]. Copyright © 2008 POSTERUS.sk [cit. 15.02.2020]. Dostupné z: http://www.posterus.sk/?p=13526

[34] Agilní vývoj: Úvod - Zdroják. *Zdroják - o tvorbě webových stránek a aplikací* [online] Copyright © 2020 [cit. 11.03.2020]. Dostupné z: https://www.zdrojak.cz/clanky/agilni-vyvoj-uvod/

[35] Agilní vývoj: Scrum - Zdroják. *Zdroják - o tvorbě webových stránek a aplikací* [online]. Copyright © 2020 [cit. 11.03.2020]. Dostupné z: https://www.zdrojak.cz/clanky/agilni-vyvoj-scrum/

[36] Bitbucket | The Git solution for professional teams. *Bitbucket | The Git solution for professional teams* [online]. Copyright © 2020 [cit. 11.03.2020]. Dostupné z: https://bitbucket.org/product/

[37] Sourcetree | Free Git GUI for Mac and Windows. *Sourcetree | Free Git GUI for Mac and Windows* [online]. Copyright © 2020 [cit. 11.03.2020]. Dostupné z: https://www.sourcetreeapp.com/

[38] Trello | Atlassian. *Atlassian | Software Development and Collaboration Tools* [online]. Copyright © 2020 Atlassian [cit. 11.03.2020]. Dostupné z: https://www.atlassian.com/cs/software/trello

[39] Hard Selling vs. Soft Selling: Which Approach Do You Use With Clients? – Gigaom. *Gigaom – Your industry partner in emerging technology research* [online]. [cit. 11.03.2020]. Dostupné z: https://gigaom.com/2009/02/25/hard-selling-vs-soft-selling-which-approach-do-you-use-with-clients/

[40] What is Earned Media? Definition, Attribution, Best Practices with Examples | MarTech Advisor. *Marketing Technology News, Trends, Strategy, Best Practices & Insights | MarTech Advisor* [online]. Copyright © 2020 MarTech Advisor is among the trademarks of [cit. 11.03.2020]. Dostupné z: https://www.martechadvisor.com/articles/ads/what-is-earned-media/

[41] NASH, Edward L. Direct marketing. Praha: Computer Press, 2003. Praxe manažera (Computer Press). ISBN 80-7226-838-4.

[42] KUBÍČEK, M. *50 způsobů, jak získat zpětný odkaz*. 2. vyd. Karviná: PRONETmedia, s.r.o., 2012. 90 s. ISBN 978-80-87721-01-8

[43] Kradená Kola | bezplatný registr jízdních kol. *Kradená Kola | bezplatný registr jízdních kol* [online]. Copyright © 2014 GeoBasis [cit. 15.05.2020]. Dostupné z: http://www.kradenakola.cz/

[44] 10 nejlepších PHP frameworků pro vývojáře | *Interval.cz. Interval.cz | Svět Internetu, Technologií a Bezpečnosti* [online]. Copyright © 2015 [cit. 15.03.2020]. Dostupné z: https://www.interval.cz/clanky/10-nejlepsich-php-frameworku-pro-vyvojare/

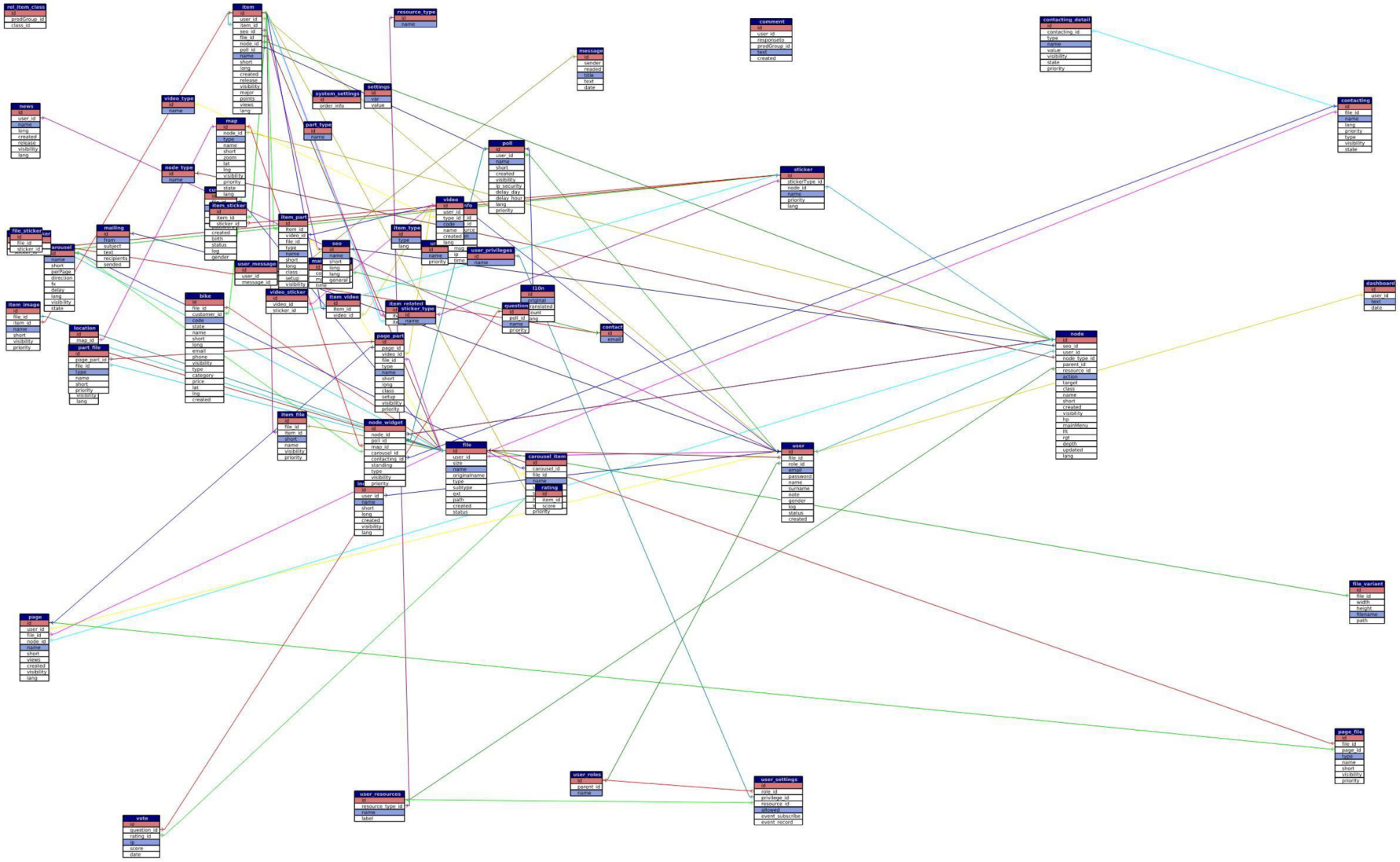
[45] The Best PHP Framework for 2015: SitePoint Survey Results — *SitePoint. SitePoint – Learn HTML, CSS, JavaScript, PHP, Ruby & Responsive Design* [online]. Copyright © 2000 [cit. 15.03.2020]. Dostupné z: https://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/

[46] Zabezpečení před zranitelnostmi | *Nette Docs*. [online]. Copyright © 2008, 2020 Nette Foundation. All rights reserved. [cit. 15.03.2020]. Dostupné z: https://doc.nette.org/cs/3.0/vulnerability-protection

ZOZNAM OBRÁZKOV

ZOZNAM PRÍLOH

PRÍLOHA Č.1: DATABÁZA



Obr. 42: E-R diagram databáze
(Zdroj: Vlastné spracovanie)

PRÍLOHA Č.2: TECHNICKÁ DOKUMENTÁCIA

```
application:
    errorPresenter: Error
    mapping:
        *: App\*Module\Presenters\*Presenter

session:
    expiration: 14 days

database:
    dsn: 'mysql:host=%db.host%;dbname=%db.name%'
    user: %db.user%
    password: %db.password%
    options:
        lazy: yes

services:
    router: App\Router\RouterFactory::createRouter
    parameters: App\Common\Parameters(@container::getParameters())

includes:
    - etc/params.neon
    - etc/model.neon
    - etc/email.neon
    - etc/form.neon
    - etc/grid.neon
    - etc/security.neon
    - etc/services.neon
    - ext/image.neon
```

Obr. 43: Konfiguračný súbor common.neon
(Zdroj: Vlastné spracovanie)

```php
<?php declare(strict_types=1);

namespace App\Router;

use Nette;
use Nette\Application\Routers\Route;
use Nette\Application\Routers\RouteList;


final class RouterFactory
{
	use Nette\StaticClass;

	public static function createRouter(): RouteList
	{
		$router = new RouteList;

		$admin = new RouteList( module: 'Admin');
		$admin[] = new Route( mask: 'admin/<presenter>/<action>', metadata: 'Dashboard:default');

		$front = new RouteList( module: 'Front');
		$front[] = new Route( mask: '<presenter>/<action>', metadata: 'Homepage:default');

		$router[] = $admin;
		$router[] = $front;

		return $router;
	}
}
```

Obr. 44: RouterFactory
(Zdroj: Vlastné spracovanie)

```
{**
 * @param string   $basePath web base path
 * @param array    $flashes  flash messages
 *}

{import 'components/form.latte'}

<!DOCTYPE html>
<html lang="cs">
<head>
    <meta charset="utf-8">

    <title>{ifset title}{include title|stripHtml} | {/ifset}Kradená Kola | bezplatný registr jízdních kol</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <link rel="stylesheet" href="{$basePath}/public/front/css/main.min.css">
</head>

<body>
{include 'parts/header.latte'}

{ifset hero}{include hero}{/ifset}

<div class=container>
    <div n:foreach="$flashes as $flash" n:class="alert, alert-dismissible,'alert-' . $flash->type">
        <a href="#" class="close" data-dismiss="alert" aria-label="close">×</a>
        {$flash->message}
    </div>
    {include content}
</div>

{include 'parts/footer.latte'}

{block scripts}
    <script src="https://code.jquery.com/jquery-3.4.1.slim.min.js" integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0i
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-Q6E9RHvbIyZFJc
    <script src="{$basePath}/public/front/js/main.js"></script>
{/block}
</body>
</html>
```

Obr. 45: Hlavný layout pre užívateľa
(Zdroj: Vlastné spracovanie)

```php
/**
 * @param $id
 * @param $data
 * @return int
 * @throws \Exception
 */
public function edit($id, $data)
{
    $now = new DateTime();
    return $this->database->table( table: self::TABLE_NAME)
        ->where( condition: self::COLUMN_ID, $id)
        ->update([
            self::COLUMN_EMAIL => $data['email'],
            self::COLUMN_NAME => $data['name'],
            self::COLUMN_LASTNAME => $data['lastname'],
            self::COLUMN_GENDER => $data['gender'],
            self::COLUMN_STATUS => $data['status'],
            self::COLUMN_DATETIME_UPDATED => $now,
            self::COLUMN_NOTE => $data['note'],
        ]);
}
```

Obr. 47: UserManager - funkcia edit
(Zdroj: Vlastné spracovanie)

```php
/** @var FormFactory */
public $formFactory;

/** @var UserEntity */
public $userEntity;

/** @var UserManager */
public $userManager;

public function __construct(FormFactory $formFactory, UserManager $userManager, UserEntity $userEntity = null)
{
    $this->formFactory = $formFactory;
    $this->userManager = $userManager;
    $this->userEntity = $userEntity;
    $this->setDefaults($userEntity);
}
```

Obr. 46: UserControlForm – konštruktor
(Zdroj: Vlastné spracovanie)

```php
/**
 * @param UserEntity $userEntity
 */
public function setDefaults(UserEntity $userEntity = null)
{
    if (empty($userEntity)) {
        return;
    }

    $defaults = [
        'email' => $userEntity->getEmail(),
        'name' => $userEntity->getName(),
        'lastname' => $userEntity->getLastname(),
        'gender' => $userEntity->getGender(),
        'status' => $userEntity->getStatus(),
        'note' => $userEntity->getNote(),
    ];

    $form = $this->getForm();
    $form->setDefaults($defaults);
}
```

Obr. 48:UserControlForm - funkcia setDefaults
(Zdroj: Vlastné spracovanie)

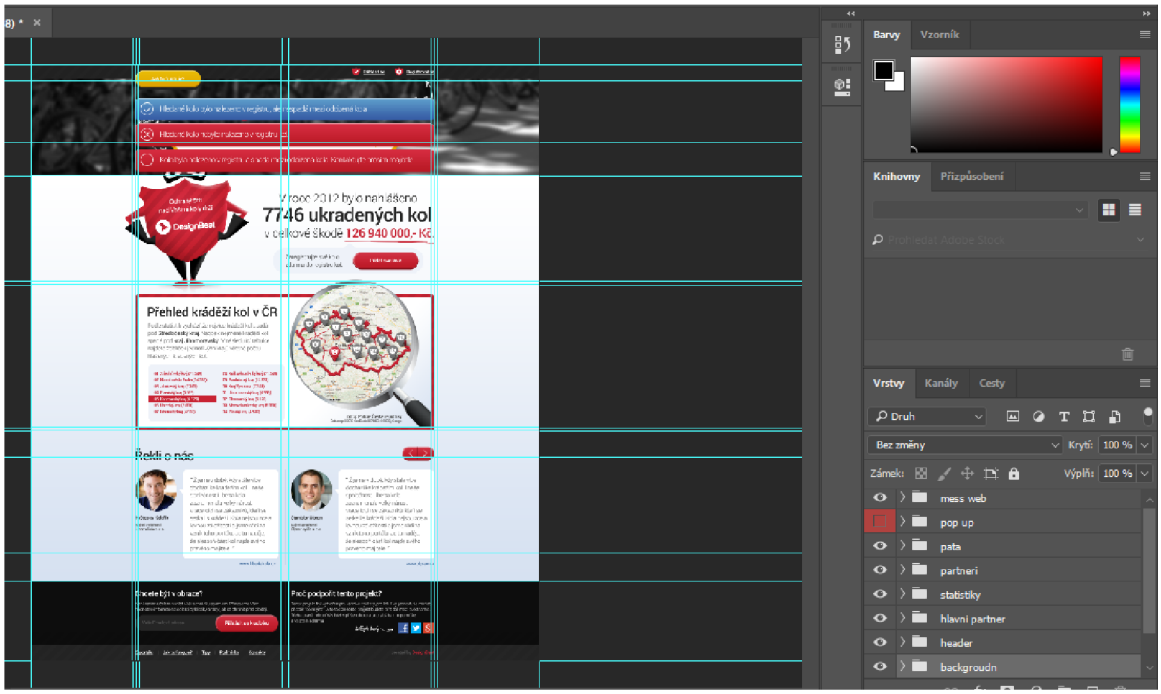
```php
public function actionEdit($id)
{
    $row = $this->bikeManager->load($id);
    $this->bike = $this->bikeManager->mapToBike($row);
}

public function createComponentBikeForm(): BikeControlForm
{
    $control = $this->bikeControlFormFactory->create($this->bike);
    $form = $control->getForm();

    $form->onSuccess[] = function (Form $form, $values) {
        try {
            if (empty($this->bike)) {
                $id = $this->bikeManager->add($values);
            } else {
                $id = $this->bike->getId();
                $this->bikeManager->edit($id, $values);
            }
        } catch (UniqueConstraintViolationException $e) {
            throw new \Exception($e->getMessage());
        }

        $this->flashMessage( message: 'Operace proběhla úspěšne');
        $this->redirect( destination: 'Bike:edit', ['id' => $id]);
    };

    return $control;
}

public function createComponentBikeGrid(): BikeGrid
{
    $control = $this->bikeGrid->create();

    return $control;
}
```

Obr. 49: BikePresenter
(Zdroj: Vlastné spracovanie)

```scss
.header {
  background: url("./../images/bg-header.png") no-repeat scroll 50% #212121;
  -webkit-background-size: cover;
  -moz-background-size: cover;
  -o-background-size: cover;
  background-size: cover;

  .search-bike {
    display: inline-block;
    padding: 10px;
    background: url("./../images/bg-striped.png") repeat scroll 0 0 transparent;
    border-radius: 40px;
    width: 600px;
    margin: 20px 110px 0 110px;
    position: relative;
    top: -60px;
    text-align: center;

    form, input {
      margin: 0;
      border: none;
    }

    input {
      height: 28px;
      float: left;
      text-align: center;
      font-size: 24px;
      color: #555;
      padding: 15px;
      font-family: "Roboto Light", sans-serif;
      width: 100px;
      border-radius: 30px 0 0 30px;
    }

    input[type="submit"] {
      padding: 0;
      height: 30px;
      position: relative;
      background: url("./../images/search-btn.png") no-repeat scroll 50% 50% #fff;
      width: 68px;
      border-radius: 0 30px 30px 0;
    }
  }

  a {
    position: relative;
    top: 40px;
  }
}
```

Obr. 50: Nesting - header
(Zdroj: Vlastné spracovanie)

PRÍLOHA Č.3: GRAFIKA



Obr. 51: Grafický návrh vo Photoshope
(Zdroj: Vlastné spracovanie)