VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

PLATFORMA PRO INTERAKTIVNÍ VÝSTAVY ZALOŽENÉ NA AKTUÁLNÍ POLOZE

PLATFORM FOR INTERACTIVE EXHIBITIONS BASED ON CURRENT POSITION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **FILIP ČONKA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. VLADIMÍR BARTÍK, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Čonka Filip**
Program: Informační technologie
Název: **Platforma pro interaktivní výstavy založené na aktuální poloze**
Platform for Interactive Exhibitions Based on Current Position
Kategorie: Web

Zadání:

1. Seznamte se s principy tvorby webových aplikací, dostupnými prostředími a frameworky. Dále se seznamte s technologiemi pro určování polohy zařízení uvnitř budov.
2. Analyzujte požadavky na informační systém, který umožní spravovat údaje o jednotlivých výstavách, jejich částech a umístění exemplářů. Dále bude obsahovat single-page aplikaci, která bude na základě polohy zařízení zobrazovat informace k jednotlivým částem výstavy (text, obrázky, videa, kvízy apod.)
3. Navrhněte informační systém splňující zjištěné požadavky.
4. Navržený informační systém implementujte a otestujte jeho funkčnost.
5. Zhodnoťte dosažené výsledky a diskutujte další možné pokračování tohoto projektu.

Literatura:

- Dokumentace k technologii Sewio UWB-RTLS: https://www.sewio.net/rtls-tdoa/
- Gutmans, A., Rethans, D., Bakken, S.: Mistrovství v PHP 5, Computer Press, 2012

Pro udělení zápočtu za první semestr je požadováno:

- Body 1-3.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Bartík Vladimír, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 16. října 2018

Abstrakt

Táto práca sa zaoberá tvorbou webovej aplikácie na zobrazovanie interaktívnych výstav založených na aktuálnej polohe v interiéroch. V práci predstavujem aplikáciu neviazanú na konkrétnu implementáciu zdroju zobrazovaných dát. Aplikácia využíva moderné technológie pre tvorbu dynamických webových aplikácií. V práci sa zaoberám aj problematikou nasadenia aplikácie s využitím cloud computing. Samotná implementácia aplikácie zahŕňa využitie JavaScript frameworku Vue.js, ako zdroj dát sa využíva REST API naprogramované v PHP s využitím Laravel Framework a October CMS. Na polohovanie v interiéroch využívam RTLS platformu spoločnosti Sewio. Aplikácia beží v prostredí Docker kontajnerov s využitím technológie Kubernetes pre orchestráciu týchto kontajnerov. V práci som vytvoril platformu, ktorá prináša do prostredia výstav interaktivitu a možnosť zobrazenia dodatočných informácií k vystavovaným exponátom. Na základoch tejto práce je možné nasadiť aplikáciu vo vybranej výstave, galérii, či múzeu a tým zvýšiť atraktivitu daného miesta.

Abstract

This thesis focuses on the development of interactive exhibitions web application based on the current indoor position. The application is independent of a specific backend implementation and uses modern web development technologies. This thesis deals with deployment using cloud computing technologies. I implemented the application with the use of the frontend JavaScript framework Vue.js. As a data source, there is a PHP backend with the REST API based on the Laravel framework and the October CMS. I use the RTLS technology of the company Sewio. The application runs within Docker containers. I created a platform which brings interaction with the consumer and provides additional useful information about the presented exhibits. Galleries and museums can use enhance the attractivity of their exhibitions based on this application.

Kľúčové slová

webová aplikácia, SPA, JavaScript, Vue.js, Laravel, PHP, October CMS, MariaDB, pozicionovanie v interiéroch, RTLS, Sewio, výstavy, múzeá, galérie, Docker, kontajnerizácia aplikácií, Kubernetes, cloud, REST API

Keywords

web application, SPA, JavaScript, Vue.js, Laravel, PHP, October CMS, MariaDB, indoor positioning, RTLS, Sewio exhibitions, museum, galleries, Docker, application containerization, Kubernetes, cloud, REST API

Citácia

ČONKA, Filip. *Platforma pro interaktivní výstavy založené na aktuální poloze.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vladimír Bartík, Ph.D.

Platforma pro interaktivní výstavy založené na aktuální poloze

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Vladimíra Bartíka, Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Filip Čonka
15. mája 2019

Poďakovanie

Rád by som poďakoval Ing. Vladimírovi Bartíkovi, Ph.D. za jeho odborné rady pri príprave technickej časti a formálnej stránky tejto práce. Ďalej by som rád poďakoval Ing. Petrovi Passingerovi zo spoločnosti Actlocate s.r.o. za pripomienky a super nápady na funkčné časti aplikácie. Za trpezlivosť a podporu ďakujem svojej snúbenici Veronike Molnárovej. A nakoniec za precíznu jazykovú korektúru ďakujem Tomášovi Kortišovi.

Obsah

Kapitola 1

Úvod

Galérie a múzeá sú miesta, ktoré navštevujú milovníci umenia, architektúry, vedy, či histórie. Kurátori a zamestnanci múzeí sa prirodzene snažia ukázať pripravenú expozíciu čo najväčšiemu počtu návštevníkov. Pre návštevníka nie je vždy rentabilné objednať si živého sprievodcu expozíciou a hromadné prehliadky môžu zážitok skôr pokaziť. Od vzniku a rozvoja klasických audiosprievodcov prešla už dlhá doba a postupne vznikajú nové technológie a technologické vychytávky, ktoré je možné využiť aj v tomto odvetví. Aj preto mnoho známych múzeí využíva moderné technológie na umocnenie interaktívneho zážitku pre návštevníkov[1]. Zároveň moderné technológie a prenosné zariadenia, akými sú tablety či smartfóny, umožnia návštevníkom interagovať s vystavovaným dielom na vyššej úrovni a získavať množstvo dodatočných informácií zábavnou formou.

Motiváciou pre vznik tejto práce a jej prvotný nápad pochádza z idey vytvorenia platformy, ktorá umožní aj menším múzeám a galériám uskutočniť interaktívne výstavy so skvelým návštevníckym zážitkom. V práci sa zaoberám tvorbou platformy na zobrazovanie interaktívnych výstav založených na aktuálnej polohe v interiéroch.

V práci môže čitateľ nájsť teoretické informácie o tvorbe webových aplikácií, ich návrhu a rôznych druhoch systémov pre správu obsahu, čím sa zaoberá hlavne kapitola 2. Ďalej v kapitole 3 popisujem rôzne technológie určovania polohy v interiéroch a princípy ich fungovania, zvlášť potom technológie spoločnosti Sewio, ktoré aj prakticky využívam v tejto práci. V kapitole 4 analyzujem požiadavky na tvorenú aplikáciu a popisujem jej prípady použitia. V prípade webových aplikácií sa nesmie podceniť návrh ich dátového zdroja, ktorý rozoberám v kapitole 5. Rovnako dôležité je vybrať vhodné technológie na základe definovaných požiadaviek, čomu sa venujem v kapitole 6. Samotnú implementáciu platformy s využitím vybratých technológií popisujem v kapitole 7. Spôsobom priebežného testovania platformy a jej jednotlivých aplikácií sa venujem v kapitole 8. Posledným krokom pred nasadením aplikácie do produkcie je jej kontajnerizácia a nakonfigurovanie infraštruktúry, čím sa zoberám v poslednej kapitole 9.

[1] Príklady využitia moderných technológií svetovými múzeami: https://econsultancy.com/how-museums-are-using-immersive-digital-experiences/.

Kapitola 2

Princípy tvorby webových aplikácií s obsahom a interaktívnymi prvkami

Kapitola popisuje teoretické znalosti a prehľad rôznych technológií, ktoré som si musel naštudovať na to, aby som mohol navrhnúť webovú aplikáciu. Kapitola zároveň pojednáva o rôznych webových aplikáciách a systémoch pre tvorbu obsahu a spôsobu uloženia dát v týchto systémoch. V druhej časti sú popísané princípy objektovo orientovaného programovania a návrh aplikačných rozhraní webových aplikácií.

Webovú aplikáciu tvorí kolekcia statických a dynamických webstránok. Statická webstránka má nemenný obsah, a keď o ňu návštevník požiada, webový server mu ju zašle bez zmeny (obr. 2.1). V prípade dynamickej webovej stránky sa webová stránka generuje na základe vstupov od návštevníka a prechádza cez aplikačnú vrstvu (obr. 2.2). [1]

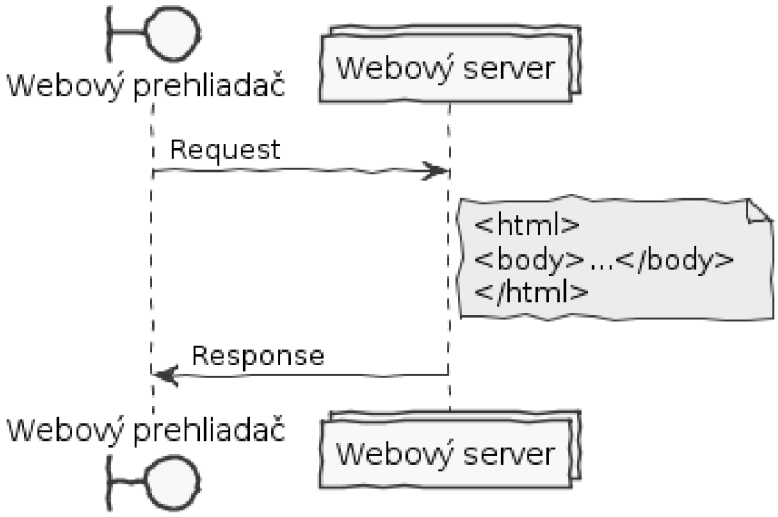
2.1 Systémy pre tvorbu a správu obsahu

Webový systém pre tvorbu a správu obsahu, ďalej len *CMS (Content Management System)*, je webová aplikácia, v ktorej je možná správa rôznych typov dokumentov a obsahu.

Jednými z najznámejších webových CMS v súčastnosti sú **Wordpress**, **Joomla!** a **Drupal**. [3] V nasledujúcich odsekoch predstavujem jednotlivé najznámejšie webové CMS za účelom výberu vhodného systému pre začlenenie do vytváranej platformy. V tomto systéme potom budú spravované údaje zobrazované k jednotlivými exponátom.

2.1.1 Wordpress

Wordpress je najpoužívanejším open-source CMS vo svete s 18 miliónmi inštaláciami. Pôvodne bol určený len ako blogovací systém, ale vďaka množstvu rozšírení, ktoré sú dnes k Wordpressu dostupné, je z neho možné spraviť plne funkčný CMS. Komunita okolo Wordpressu vytvorila veľké množstvo tém, pluginov a rozšírení. Kritici Wordpressu často vidia negatívne jeho komplexitu, pre niektorých môže byť systém zmätočný. Wordpress zároveň často býva cieľom rôznych hackerských útokov, hlavne vďaka jeho silnému rozšíreniu. Používatelia tohto CMS tak musia pravidelne systém aktualizovať, aby zaistili jeho bezpečný chod. Bezpečnostné aktualizácie bývajú uvoľnené krátko po objavení bezpečnostnej diery, avšak nie vždy je aktualizácia kompatibilná so všetkými nainštalovanými témami,



Obr. 2.1: Diagram komunikácie medzi webovým prehliadačom a webovým serverom. Webový prehliadač zašle validný *Request* na webový server, a ten mu zašle *statický HTML kód* vo forme *Response*.[1]

či rozšíreniami. Vo výsledku tak môže zabrať aktualizovanie Wordpressu množstvo času a administrátorského úsilia, a to obzvlášť pri veľkých projektoch. [3]

2.1.2 Joomla!

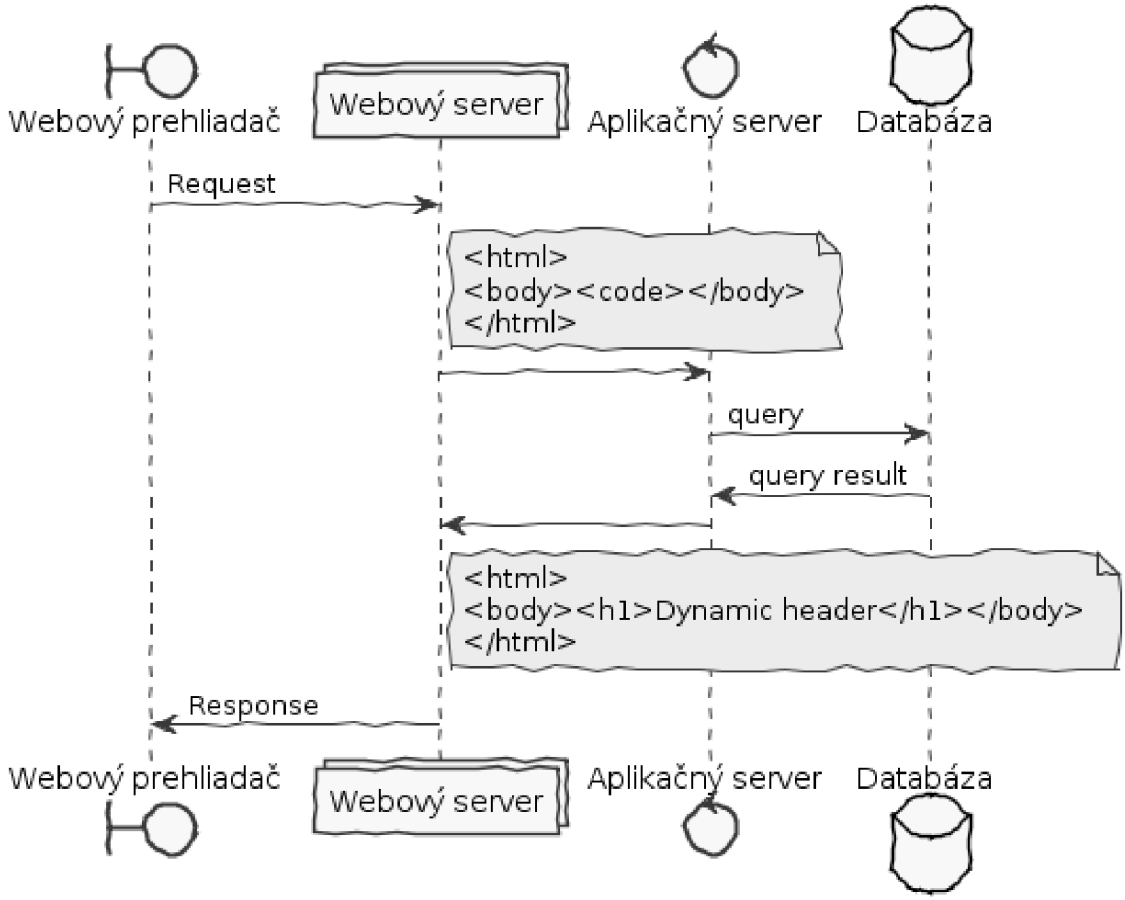
Joomla!, ktorá má celosvetovo 2,5 milónia inštalácií, je druhý najväčší hráč na trhu CMS. Joomla! cieli na začiatočníkov a pokročilých používateľov, napriek tomu je ťažšia na používanie než Wordpress. Na rozdiel od Wordpressu obsahuje Joomla! množstvo CMS funkcionalít bez nutnosti inštalovať rozšírenia. Výhodou tohto CMS je aj dobrá dokumentácia, ktorú využijú hlavne používatelia bez IT vzdelania. Joomla! je založená na **CMS Mambo** a je najviac populárna v USA. Jej softvérový návrh je kompletne objektovo orientovaný, založený na vlastnom MVC frameworku. S použitím tohto frameworku je možné vyvinúť vlastné rozšírenia a zdieľať ich v komunite ostatných vývojárov Joomla!. [3]

2.1.3 Drupal

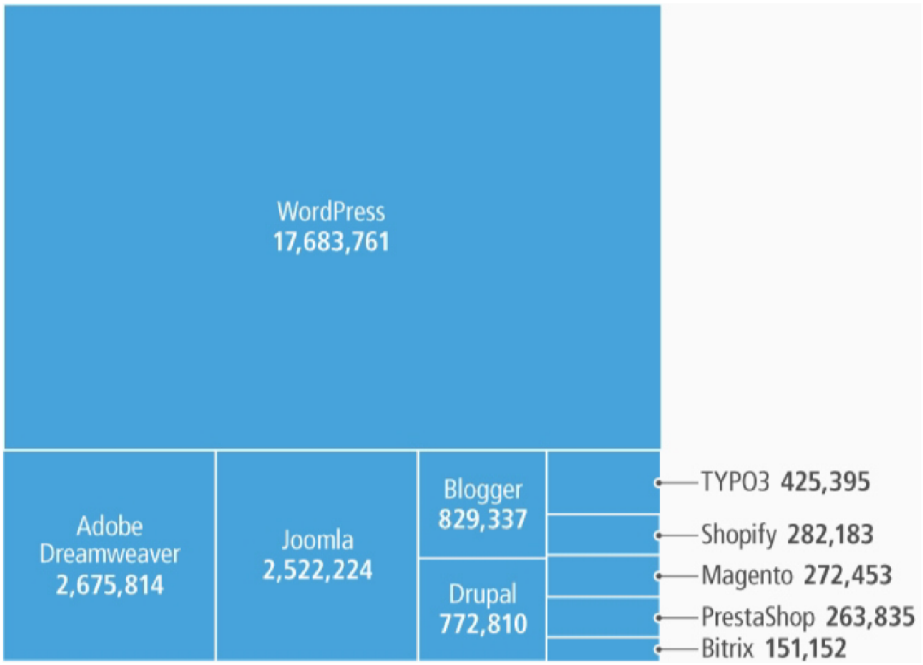
Modulárne CMS Drupal bolo pôvodne vyvinuté študentmi ako komunitné riešenie. V súčasnosti je však plne funkčným open-source CMS. Drupal je veľmi jednoduchý na inštaláciu a môže byť rozšírený množstvom modulov. Sila Drupalu tkvie v jeho komunite. Drupal je možné rozšíriť vďaka jeho modulárnemu návrhu. [3]

2.1.4 October CMS

October je open-source, bezplatná CMS platforma založená na **PHP frameworku Laravel**. V celosvetovom meradle nepatrí k lídrom tohto segmentu, avšak vďaka svojej jednoduchosti, flexibilite a modernému dizajnu je veľmi populárny medzi programátormi pracujúcimi na voľnej nohe a v digitálnych agentúrach. October CMS umožňuje bezbolestné vytváranie administračných rozhraní pre vlastné pluginy. Vytváranie administračných stránok pritom nevyžaduje množstvo programovania a používa jednoduché konfiguračné súbory.



Obr. 2.2: Diagram komunikácie medzi webovým prehliadačom a webovým serverom v prípade dynamických webových aplikácii. Webový prehliadač zašle validný *Request* na webový server, ten zavolá aplikačný server, ktorý vygeneruje *HTML kód* z predpripravenej šablóny. Tento *HTML kód* zašle webovému serveru, ktorý tento kód obalí do *Response* a zašle webovému prehliadaču.[1]



Obr. 2.3: Najpopulárnejšie CMS z hľadiska počtu inštancií. [3]

2.2 Dátové uložisko – Databáza

CMS potrebujú pre svoje fungovanie úložisko dát, ktoré sa v systéme spracúvajú. V prípade jednoduchých CMS môže ísť o úložisko formou statických textových, či špeciálne formátovaných súborov, ktoré sú ďalej spracované samotným CMS. Ako úložisko normalizovaných dát je možné využiť niektorú z relačných databáz. V prípade dát, ktoré nie je možné ľahko normalizovať a ich štruktúra je komplexná, pričom je v systéme málo položiek s rovnakou štruktúrou, je možné na ukladanie dát využiť aj objektovú databázu.

2.2.1 Relačná databáza

Relačné databázy sú štrukturované a už vyše 40 rokov je *SQL (Structure Query Language)* primárnym jazykom relačných databáz. S rastúcou popularitou webových aplikácií v 90. rokoch vznikli open-source riešenia pre relačné databázy ako sú **PostgreSQL**, **MySQL** či **SQLite**. Okrem open-source riešení existujú aj proprietárne SQL databázové riešenia, akými sú **Microsoft SQL Server**[1] či **Oracle Database**[2],

Aby boli relačné databázy efektívne, musia v nich byť dáta uložené štruktúrovane. SQL databázové riešenia majú zvyčajne dobrú komunitnú a komerčnú podporu, a tým pádom je riešenie problémov pri vývoji jednoduchšie. Avšak neexistuje jedno databázové riešenie pre všetky mysliteľné prípady použitia. Častokrát sa pri komerčných projektoch používa aj kombinácia relačných a nerelačných databáz pre rôzne úlohy v softvérových projektoch. Existuje množstvo prípadov, v ktorých je SQL štruktúrovaná databáza preferovaná pred neštruktúrovanou.

ACID (z anglických Atomicity, Consistency, Isolation, Durability – atomicita, konzistentnosť, izolovanosť a trvalosť) je hlavnou súbornou požiadavkou na transakcie v relač-

[1]Microsoft SQL Server – https://www.microsoft.com/en-us/sql-server
[2]Oracle Database – https://www.oracle.com/database/

ných databázach. Redukujú sa tým prípadné anomálie a chráni sa integrita ukladaných dát. Pri nerelačných databázach sa často od *ACID* upúšťa v prospech rýchlosti a flexibility. Jedným z hlavných problémov SQL databázových riešení je ich škálovateľnosť, keď požiadavky na databázový systém rastú. [12] Množstvo spoločností však poskytuje komerčné cloudové riešenia s automatickou škálovateľnosťou, čo robí tento problém menším. Napríklad **Azure Databases** od spoločnosti Microsoft[3] či **Managed Databases** od spoločnosti DigitalOcean[4]

2.2.2 NoSQL databáza

V prípade projektov, kde sa pracuje s množstvom neštrukturalizovaných dát, alebo dátový model nie je normalizovateľný, prichádzajú vhod objektové, tzv. *NoSQL databázové systémy*. Nerelačnú databázu si môžeme predstaviť ako priečinok súborov zahŕňajúci podobné informácie rôzneho druhu a štruktúry. Jedným z hlavných motivátorov pre NoSQL databázové systémy je v súčasnoti trend *BigData*. Typickými zástupcami NoSQL databázových riešení sú **MongoDB**, **CouchDB** a **Cassadra**. Cieľom NoSQL databáz je, aby sa nestali pri škálovaní aplikácie „úzkym hrdlom“ celého riešenia, keď bola aplikácia navrhnutá pre rýchlosť.

Výhodou NoSQL databáz je možnosť ukladať veľké množstvo dát bez nutnosti definovania ich štruktúry. NoSQL databázy nekladú limity na typy dát, ktoré sú v nej ukladané. Dáta je možné kedykoľvek vymeniť za iné, a to aj iného typu. Na druhú stranu NoSQL komunita nie je tak vyzretá ako komunita okolo SQL databázových riešení. Z pohľadu dĺžky histórie SQL sú NoSQL databázové systémy stále v plienkach. NoSQL riešeniam tiež chýbajú robustné nástroje na analýzu a reportovanie, či testovanie výkonu. Zároveň tiež NoSQL databázam chýba štandardizovanie na takej úrovni, na akej je SQL, čo môže spôsobiť problém pri migráciách z NoSQL systému v budúcnosti. [12]

2.3 Objektovo orientované programovanie a návrhové vzory

Objektovo orientované programovanie (ďalej len OOP) je prístup, kde sú vytvárané softvérové objekty korešpondujúce s fyzickými objektmi. OOP je spôsob programovania rôznych aplikácií, pri ktorom sú vytvárané objekty, ktoré do istej miery odpovedajú objektom z reálneho sveta, ktoré sú pre ne predlohou. Medzi reálnymi objektmi a tými softvérovými existuje určitá miera abstrakcie. Výhodou OOP je, že softvérový a reálny model sú analogické. Softvérové modely sú flexibilné a znovu použiteľné. Súčasťou OOP je aj analýza a špecifikácia požiadavkov, návrh, implementácia a testovanie. Základné koncepty OOP sú:

- objekt,
- abstrakcia,
- zapuzdrenie,
- dedičnosť,
- polymorfizmus.

[3] Azure Databases – https://azure.microsoft.com/en-us/product-categories/databases/
[4] Managed Databases od spoločnosti DigitalOcean – https://try.digitalocean.com/dbaas-beta/

Objekt je základným stavebným prvkom v OOP. Uchováva a zjednocuje dáta a funkcionalitu. To znamená, že je možné daný problém rozdeliť na menšie problémy, ktoré môžu byť reprezentované pomocou objektov.

Abstrakcia vyznačuje základné charakteristiky objektu, ktorými sa odlišuje od všetkých ostatných druhov objektov, a tak poskytuje presne definované hranice z pohľadu pozorovateľa.

Zapuzdrenie znamená, že skutočná implementácia objektu a jeho metód je plne skrytá a objekty medzi sebou komunikujú vždy prostredníctvom rozhrania. Rozhranie definuje, aké operácie je možne vykonávať nad daným objektom.

Novovytvorené objekty môžu zdieľať a rozširovať správanie už existujúcich objektov. Táto vlastnosť sa nazýva **dedičnosť** a slúži k zdieľaniu implementácie a k špecializácii niektorých objektov.

Polymorfizmus využíva princíp zasielania správ objektom v OOP. Rôznym objektom sú zasielané správy, pričom reakcia na tieto správy závisí na konkrétnom objekte.

Programovacie jazyky používané v tejto práci používajú OOP a jeho princípy.

2.3.1 Domain-driven design

Domain-driven design (DDD; voľne preložené ako návrh riadený doménou) je prístup pri návrhu a vývoji softvéru s komplexnými požiadavkami, pričom vzniká hlboký vzťah medzi implementáciou a samotnými biznis požiadavkami na softvér.

Každý softvér sa vzťahuje na niektorú z aktivít alebo záujmov používateľa. Predmetom oblasti, kde používateľ program aplikuje, je *doména* softvéru. Niektoré domény v sebe zahŕňajú fyzický svet: doména rezervačného systému aero-liniek zahŕňa fyzických ľudí nastupujúcich do fyzických lietadiel. Naproti tomu doména účtovného softvéru sú financie a peniaze. Doména systému pre verzovanie zdrojového kódu je samotný softvérový vývoj.

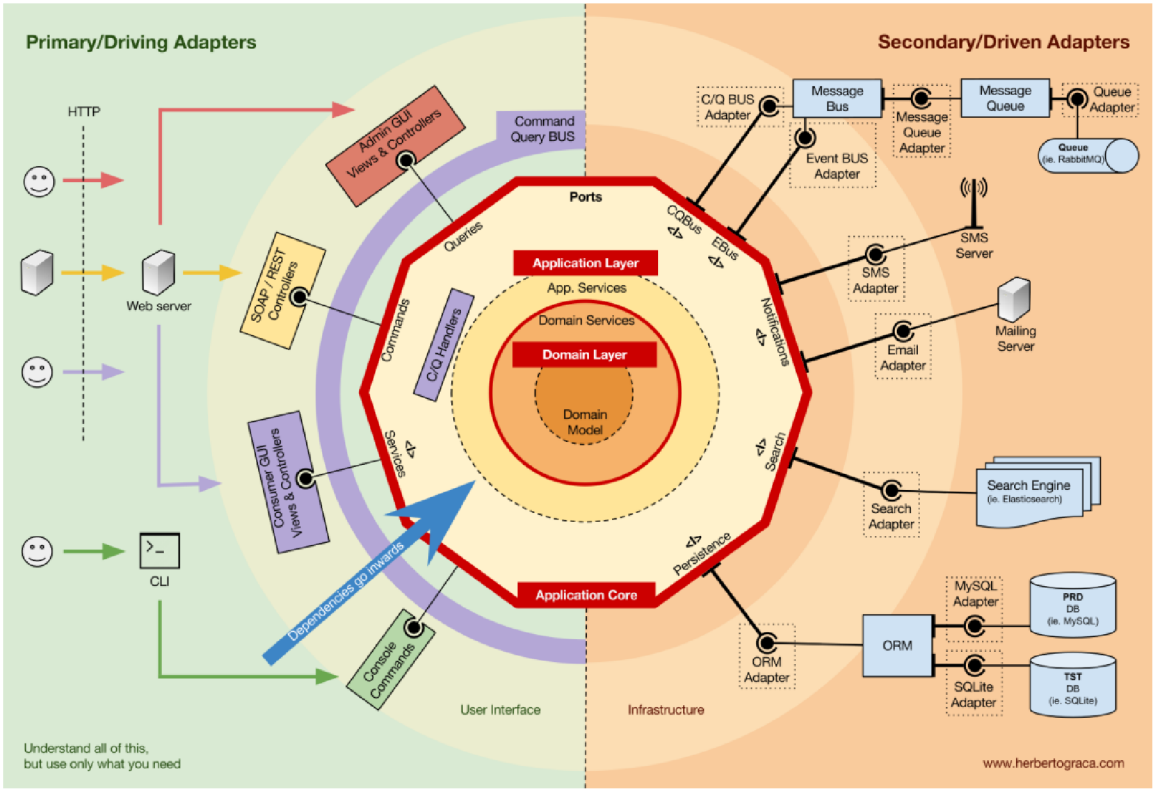
Vytvorenie softvéru, ktorý má byť pre používateľa hodnotný a zapojený do používateľských aktivít, vyžaduje od tímu vývojárov množstvo vedomostí o danej aktivite. Pričom množstvo týchto vedomostí môže byť skľučujúce. Obsah a komplexnosť informácií môže byť ohromujúca. Modely patria k nástrojom, ktoré nám uľahčujú získať kontrolu nad týmito informáciami. *Doménový model je selektívne a vedome zjednodušená štruktúrovaná forma vedomostí.* Vhodný model dáva informáciám pochopiteľnejší zmysel a zameriava sa na daný problém.

Modelovanie domény nie je vecou vytvorenia čo najrealistickejšieho modelu. Aj v prípade, ak sa model zaoberá fyzickými hmatateľnými predmetmi, je model stále umelo vytvoreným dielom. Doménový model nie je špecifický diagram. Nie je to ani súhrn všetkých poznatkov z hlavy experta v danej doméne. Doménový model je dôsledne organizovaná a selektívna abstrakcia daných znalostí. Diagram môže reprezentovať a odkomunikovať daný model rovnako dobre, ako precízne napísaný programový kód, rovnako dobre ako veta v hovorenom jazyku. [19]

Hexagonal Architecture

Hexagonal architecture (známe aj ako Ports and Adapters pattern[5]) je návrhový vzor striktne spájaný s DDD a je nezávislý na programovacom jazyku.

[5]Voľne preložené ako šesťuholníková architektúra; návrhový vzor porty a adaptéry.



Obr. 2.4: Znázornenie hexagonal architecture – návrhového vzoru porty a adaptéry. [28]

Port v taxonómii tohto návrhového vzoru môžeme chápať ako vstupnú bránu, ktorá umožňuje dáta vstupovať a vystupovať z aplikácie. Z hľadiska kódu ide o *rozhranie*. Existujú dva druhy portov: vstupné a výstupné.

Vstupný port otvára cestu k funkčným častiam jadra aplikácie, ktoré môžu byť použité vonkajším prostredím. **Výstupný port** je rozhranie, ktoré jadro aplikácie potrebuje na komunikáciu a použitie služieb z vonkajšieho prostredia.

Adaptér môže byť primárny a sekundárny. **Primárny adaptér** implementuje niektorý zo vstupných portov aplikácie, riadi aplikáciu a môže spúšťať akcie jadra aplikácie. **Sekundárny adaptér** predstavuje napojenie na služby z vonkajšieho prostredia, ktoré jadro aplikácie potrebuje pre svoj beh. Z hľadiska webovej aplikácie môže ísť o databázy, externé knižnice a pod. Sekundárny adaptér implementuje rozhranie výstupného portu a reaguje na akcie vyvolané primárnymi adaptérmi.

Cieľom hexagonal architecture je izolovať hlavnú logiku aplikácie[6] od vstupných a výstupných mechanizmov používaných aplikáciou. Pre dosiahnutie izolácie sa používajú rozhrania, pričom všetky závislosti idú v smere od jadra aplikácie. Samotné jadro aplikácie nie je závislé na žiadnej ďalšej súčasti. [28]

[6] Angl. core business logic.

2.4 Frontend systémov pre tvorbu obsahu

Frontend[7] je možné vo webovej aplikácii implementovať rôznymi spôsobmi. Zjednodušene to môžeme rozdeliť do dvoch hlavných prístupov:

1. generovanie HTML kódu priamo v *backend* aplikácii (vnútorná časť webovej aplikácie),
2. *backend* poskytujúci API a na to napojený *frontend* implementovaný v **JavaScript** a spúšťaný vo webovom prehliadači.

2.4.1 Generovane HTML v backend aplikácii

V tomto prípade webová aplikácia vygeneruje výstupný HTML kód, ktorý je webovým serverom priamo zaslaný na vykreslenie prehliadaču. Na vygenerovanie HTML kódu sa zvyčajne používa niektorý zo šablónovacích systémov[20]. Medzi známe šablónovacie systémy v PHP patria:

- Twig[8],
- Blade[9],
- Latte.[10]

Tieto šablónovacie systémy umožňujú spracúvať dáta získané z vyššej vrstvy aplikácie (napríklad z časti *controller*), vykonávať nad nimi jednoduchšie operácie a podmieniť vykreslenie jednotlivých častí. Vygenerované HTML sa môžu v aplikácií aj zapísať do vyrovnávacej pamäte, a tým skrátiť čas spracovania.

2.4.2 Backend aplikačné rozhranie a frontend aplikácia

V súčasnosti je čoraz populárnejšie vytvorenie univerzálneho aplikačného rozhrania (ďalej len API[11]), ktorým je backend z pohľadu okolitého prostredia ukončený. Z tohto API potom môže samostatná frontend aplikácia čerpať dáta, spracúvať ich a podmienečne vykresľovať.

Známe druhy API sú:

- REST API[12],
- SOAP[13],
- GraphQL.[14]

Známe JavaScript technólie umožňujúce vývoj dynamických frontend aplikácií:

- Vue.JS[15],

[7] Slovensky vonkajšia časť webovej aplikácie.
[8] PHP šablónovací systém Twig https://twig.symfony.com/.
[9] PHP šablónovací systém Blade https://laravel.com/docs/5.8/blade.
[10] PHP šablónovací systém Latte https://latte.nette.org/cs/.
[11] Z anglického Application Programming Interface.
[12] http://standards.rest/
[13] https://tools.ietf.org/html/rfc4227
[14] https://github.com/graphql/graphql-spec
[15] https://vuejs.org/

- React[16],
- Angular.[17]

2.5 Návrh webových aplikačných rozhraní

Vznik API podmienila potreba výmeny informácií s poskytovateľmi dát, ktorí majú know-how na riešenie špecifických problémov a potrieb, takže ostatné spoločnosti nemusia tráviť čas nad riešením týchto problémov znovu. Typickým príkladom je, ak je potrebné na stránku implementovať *interaktívnu mapu* bez nutnosti naprogramovať celé Google Maps alebo je potrebné poskytnúť používateľovi *možnosť platby* v elektronickom obchode bez vývoja platobnej brány.

V týchto prípadoch sú dodatočné funkcie vytvorené pomocou dát a interakcií použitím špecializovanej platformy. API umožňujú firmám vyvinúť jedinečné produkty rýchlejšie.

API založené na princípe *požiadavka – odpoveď (ďalej len angl. Request – response)* sú typicky otvorené vonkajšiemu prostrediu alebo internetu pomocou webového serveru na báze HTTP. API definujú zostavu *koncových bodov (angl. endpoints)*. Klienti API posielajú na webový server HTTP požiadavku s dátami a webový server im vracia HTTP odpoveď. Typickým formátom tela požiadavky a odpovede je JSON alebo XML. Najčastejšími paradigmami Request–response APIs v súčasnosti sú REST, RFC a GraphQL. [24]

2.5.1 REST API

Z anglického *Representational State Transfer* (ďalej len REST) je v súčasnosti najpopulárnejšia voľba pre vývoj API. REST API používajú pre vývoj spločnosti ako Google, Stripe, Twitter, Microsoft, či GitHub. REST sa točí okolo *zdroja dát (ďalej len anglicky resource)*. Resource je entita, ktorá je identifikovateľná a pomenovaná podoba dát, ktorú REST vystavuje. REST používa štandardné HTTP slovesá/metódy na reprezentáciu: vytvorenie (Create), načítanie (Read), aktualizácia (Update) a vymazanie (Delete) (ďalej len CRUD) transakcií zo zdrojov dát. [24]

Základné pravidlá, ktoré v štandardoch REST API dodržiavajú:

- Resource je zobrazený v časti adresy URL, napríklad `/users`.
- V základe sú implementované dve základné adresy URL: jedna pre súbor resources `/users` a druhá pre špecifický element `/users/42`.
- V adresách URL sa používajú podstatné mená miesto slovies, napr. namiesto `/getUserInfo/42` sa používa `/users/42`.
- Rôzne HTTP slovesá sa používajú na vyjadrenie požadovanej akcie. Používa sa `POST` pre vytváranie nových resources, `GET` pre načítanie resources, `PUT` pre nahradenie existujúceho resource, `PATCH` pre čiastočnú aktualizáciu existujúceho resource a `DELETE` pre vymazanie existujúceho resource.
- Webový server vracia štandardné stavové kódy v odpovedi. Kódy v rozpätí 2xx signalizujú úspech (napr. 200, 201), 3xx signalizujú, že resource bol presunutý, 5xx signalizujú serverovú chybu.

[16] https://reactjs.org/
[17] https://angularjs.org/

- REST API môžu vrátiť odpoveď vo formáte JSON alebo XML. Kvôli jednoduchej syntaxi a jednoduchosti použitia s JavaScript sa JSON stal štandardom pre moderné API. [24]

Tabuľka 2.1: Tabuľka operácií v REST API s príkladom adresy URL [24]

operácia	**HTTP sloveso**	**URL:**`/users`	**URL:**`/users/42`
Vytvorenie	`POST`	Vytvorí nový resource	N/A
Načítanie	`GET`	Zoznam používateľov	Načítať ID 42
Aktualizovanie	`PUT` alebo `PATCH`	Hromadná aktualizácia	Aktualizovať ID 42
Vymazanie	`DELETE`	Vymazanie používateľov	Vymazanie ID 42

Niekedy je v REST API potrebné reprezentovať aj non-CRUD operácie. Najčastejšie prípady sú:

- Vrátenie len časti zo súboru všetkých položiek. API služby GitHub využíva vstupný parameter `archived` (archivované) v API pre archiváciu vybraného repozitára.
- Použitie akcie ako podpoložky. API služby GitHub používa nasledovnú adresu URL s parametrami pre zamknutie a odomknutie vlákna v repozitári:
  `PUT /repos/:owner/:repo/issues/:number/lock` zamkne vlákno (`:owner`, `:repo`, `:number` predstavujú vstupné parametre a identifikujú vybrané vlákno pod jeho číslom v repozitári daného používateľa).
- Niektoré operácie, ako je napríklad vyhľadávanie, je ešte ťažšie vyjadriť v rozmedziach REST paradigmy. V takýchto prípadoch je typické použitie iba jedného slovesa vyjadrujúceho akciu a použitie doplňujúcich parametrov
  (napr. `GET /search/code?q=:query`: vyhľadá na službe GitHub súbory v kóde v repozitári podľa zadaných parametrov). [24]

2.5.2 Remote Procedure Call

Volanie vzdialenej procedúry (angl. Remote Procedure Call – RFC) je jedným z najjednoduchších API paradigiem. V ňom klient priamo vykonáva blok kódu na vzdialenom serveri. Keď je REST orientovaný okolo resources, RPC je orientované okolo akcií. Klient typicky predá serveru názov metódy a argumenty a zo serveru sa potom vráti odpoveď vo forme JSON alebo XML.

RPC API všeobecne dodržiavajú tieto dve pravidlá:

- Adresa URL obsahuje názov operácie, ktorá sa má vykonať.
- API volanie sa vykonáva s použitím HTTP slovesa, ktoré je k akcii najviac výstižné. `GET` pre operácie len pre načítanie a `POST` pre ostatné operácie.

RPC štýl je najvhodnejší pre API, ktoré otvárajú rozličné nejednoliate akcie, ktoré môžu mať rôzne špecifické prípady a rozličné vstupné parametre. RPC sa používa vtedy, keď nie je možné na dané akcie použiť CRUD princíp. RPC je tiež možné použiť pre prípady, keď sa akciou ovplyňujú viaceré nejednoznačné resources. Známym príkladom RPC API je API služby Slack[18]. [24]

[18] https://api.slack.com/web – dokumentácie RPC API četovacej služby Slack

2.5.3 GraphQL

GraphQL je dotazovací jazyk pre API, ktorý získava v poslednej dobe popularitu. Pôvodne bol interne vyvinutý spoločnosťou Facebook v roku 2012. V roku 2015 ho vydali verejne a bol rýchlo adaptovaný službami ako sú GitHub, Yelp, či Pinterest. GraphQL umožňuje klientovi nadefinovať štruktúru dát, aké potrebuje, aby server vrátil. Príklad:

Požiadavka na GraphQL server:

```
{
  user(login: "xconka00") {
    id
    name
    company
    createdAt
    }
}
```

Odpoveď GraphQL servera

```
{
    "data": {
        "user": {
          "id": 42,
          "name": "Filip Conka,
          "company": "N/A",
          "createdAt": "2019-03-09T21:00:06+01:00"
        }
    }
}
```

Na rozdiel od REST a RPC API GraphQL vyžaduje len jeden jediný URL koncový bod, tiež nie je potrebné použiť rôzne HTTP slovesá pre popis vykonávanej operácie. Miesto toho v tele JSON request zadefinujeme či požadujeme dáta načítať alebo modifikovať.

GraphQL má niekoľko výhod oproti REST a RPC:

- GraphQL umožňuje klientom vytovriť dotaz nad viacerými `resources` v jedinej požiadavke. Pri REST API by to vyžadovalo viaceré HTTP volania.
- Zjednodušuje verzovanie API. Do GraphQL môžeme jednoducho vkladať nové parametre a polia bez ovplyvnenia ostatných. Zároveň na označenie parametrov ako zastaraných a určených na odstránenie je jednoduchšie. Jednoduchou analýzou záznamov používania GraphQL servera môžeme vyhodnotiť, ktorí klienti používajú dané pole. S REST a RPC je ťažšie vyhodnotiť, ktoré polia sú používané, a ktoré nie.
- REST a RPC API zvyčajne vracajú aj dáta, ktoré klienti nepožadovali a nepotrebujú. S GraphQL sa to nemôže stať, pretože klienti si presne zadefinujú, aké dáta požadujú a tie im aj budú vrátené v odpovedi servera.
- GraphQL server má vstavaný nástroj **GraphiQL**[19], ktorý umožnuje prehľadávanie serveru, jeho koncových bodov a testovanie dotazov rovno voči danému GraphQL serveru. [24]

[19] Webová stránka popisúca GraphiQL – https://github.com/graphql/graphiql

Napriek mnohým výhodám má GraphQL aj niektoré nevýhody. GraphQL pridá do API komplexnosť: server musí spracúvať často komplexné dotazy a verifikovať parametre. Optimalizácia dotazov v GraphQL je tiež komplikovaná. [24]

2.5.4 API Blueprint

Webové API sa stávajú novým štandardizovaným jazykom, ktorým technologické spoločnosti spolu komunikujú. Spolu s ich narastajúcou dôležitosťou narastá aj zodpovednosť architektov a manažérov API. API Blueprint je open-source formát pre definovanie webových API s ľahko čitateľnou syntaxou. [10]

Popisovanie koncových bodov API v API Blueprint sa skladá z niekoľkých základných prvkov:

- `Resource` API sa skladá z viacerých `resources`, ktoré sú definované ich URL adresou.
- `Actions` – Akcia. Podnadpisom sa špecifikuje akcia, ktorú je možné nad daným zdrojom vykonávať.
- `URI Parameters` – Parametre adresy URL. [2]

Syntax API Blueprint sa podobá syntaxi textového jazyku Markdown.

```
FORMAT: 1A

# Users

Users is~a~simple API allowing to~list users of~the system.

## Users Collection [/users]

### List All Users [GET]

+ Response 200 (application/json)

        [
            {
                "id": "42",
                "email": "john.doe@example.com"
            },
            {
                "id": "43",
                "email": "john.doe.jr@example.com"
            }
        ]
```

Výpis 2.1: Príklad dokumentácie jednoduchého API v API Blueprint

2.6 Používateľské rozhranie a interakcie

Webové stránky často plnia funkciu zobrazovania informácií, často však aj majú za úlohu od používateľa získať istý vstup. Výsledkom tejto práce je webová stránka, ktorá má byť

hlavne používaná na mobilných zariadeniach, ako sú telefóny či tablety. Práve na týchto zariadeniach, ktoré majú typicky menšie obrazovky než notebooky a stolné počítače, je dôležitý správny návrh používateľského rozhrania pre formuláre.

2.6.1 Formulárové prvky optimalizované pre mobilné zariadenia

Väčšina ľudí drží telefón pri vypĺňovaní formulárov na výšku. Šírka formulárového poľa sa nám tak skracuje na maximálne 300 pixelov. Keďže telefón ovládame prstami, musíme počítať pri návrhu formulárov s väčšími rozostupmi medzi jednotlivými prvkami. Na telefónoch tiež nie je možné prakticky dosiahnuť hover efektu[20] rovnako ako na stolnom počítači, či notebooku. [23]

Známy český UI/UX profesionál Ondrej Ilinčev spísal dobré rady pri návrhu formulárov pre telefóny:

- Je dobré predvyplniť obvyklé voľby a nepýtať sa používateľa na zbytočné veci, ktoré si môže aplikácia zistiť sama. (typicky krajina – podľa IP adresy, typ platobnej karty – podľa prvých štyroch číslic a podobne).

- Steppery a segmenty miesto rozbaľovacieho menu. Rozbaľovacie menu na telefónoch sa nepoužíva dobre. Výber trvá dlho a často je potrebné dlho skrolovať. Lepšie než vypisovať počet osôb na klávesnici je použiť tlačidlá s plus a mínus. Lepšie než rozbaľovacie menu pre 2-5 položiek je použiť tlačidlá pre výber.

- Posuvník namiesto polí s hodnotami. Výber hodnôt by mal byť v rozumných intervaloch a zmysluplne rozdelený na časti. Maximálne a minimálne hodnoty by mali byť vidieť vedľa posuvníka.

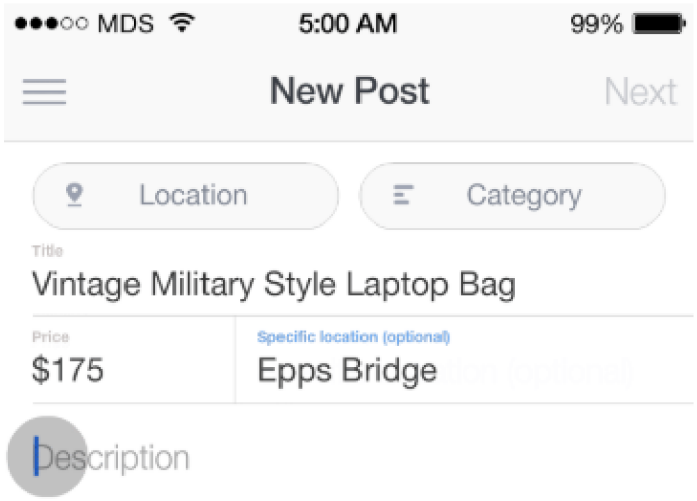
- Použitie zástupného textu (anglicky placeholder) vo formulárovom prvku bez popisu je nesprávne, pretože formulár vyzerá byť vyplnený. Nie je možné ukázať formát vstupu a po vyplnení celého formuláru ho nie je možné skontrolovať, pretože už nie je zrejmé, čo do ktorého políčka patrí. Nesprávne je umiestniť popis polí naľavo od poľa (pri používaní telefónu na výšku), pretože je potom potrebné zmenšiť samotné formulárové pole. Správne je zobraziť popis poľa nad formulárovým poľom pri používaní telefónu na výšku, vľavo od poľa pri používaní na šírku alebo použiť tzv. plávajúci popis, ktorý šetrí miesto (obr. 2.5).

- Je dôležité, aby používateľ videl na to, čo píše. Softvérová klávesnica na telefóne zaberie pri používaní na výšku cca 30 – 50 % displeja a niekedy môže zakryť aj vypĺňané pole. Autofocus – ohraničenie aktuálne vypĺňovaného poľa by sa nemalo explicitne vypínať, pretože znižuje orientáciu používateľov. Používateľovi sa ušetrí množstvo práce, ak sa mu na telefóne zobrazí klávesnica správneho typu. Toto zobrazenie je možné ovplyvniť HTML `type` atribútom formulárového prvku.[21]

- Validácia formuláru na telefóne by mala byť pre používateľa čo najpríjemnejšia. Väčšinou nie je na telefóne vidieť celý formulár, je preto dôležité, aby sa nemusel k chybne vyplneným poliam vracať. Najlepšou variantou je automaticky kontrolovať každé políčko po vyplnení, cca 0,5-1 sekundu po dopísaní. [23]

[20] Efekt pri umiestnení kurzora nad element.

[21] Typy formulárových prvkov a ich vzťah k zobrazenej softvérovej klávesnici telefónu – http://mobileinputtypes.com/



Obr. 2.5: Znázornenie plávajúceho popisu formulárových prvkov. [23]

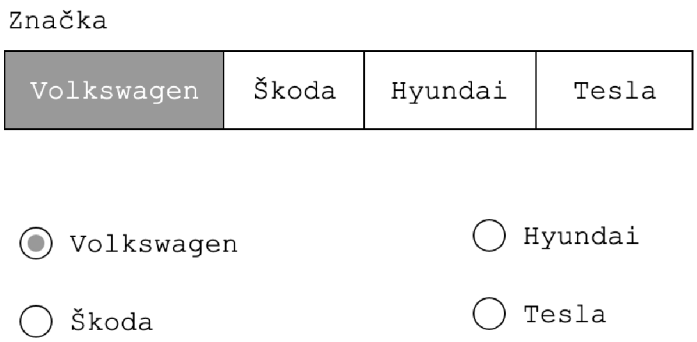
2.6.2 Alternatívy použitia rozbaľovacieho menu na mobilných zariadeniach

Rozbaľovacie menu (ďalej len anglicky dropdown) nie je vhodné na použitie na mobilných zariadeniach. Často sú v dropdown menu hodnoty zoradené nelogicky, a používateľ musí pri výbere vhodnej hodnoty pracne prechádzať zoznamom hore a dole.

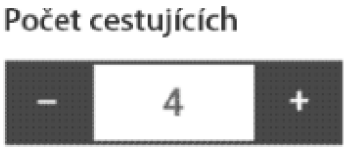
Niekoľko možností, ako sa vyhnúť použitiu rozbaľovacieho menu:

- Pri voľbe áno/nie je vhodnejšie použiť checkbox alebo posuvník (ďalej len anglicky toggle box) než dropdown (obr. 2.6).
- Pri 2 až 5 možnostiach je najvhodnejšie použiť formulárový prvok výberu z viacerých možností (ďalej len anglicky radio button) alebo výber z dlaždíc (obr. 2.7).
- Pri výbere počtu alebo množstva je ideálny číslený vstup s možnosťami plus a mínus (obr. 2.8).
- Pri výbere na škále hodnôt je vhodný box s možnosťou posúvania. V tomto prípade je dôležité, aby sa aktuálna hodnota zobrazovala nad samotným prvkom, ktorým používateľ hýbe (fyzicky nad prstom používateľa) a zároveň by mali byť zobrazené maximálne a minimálne hodnoty. Tento prvok však nie je vhodný na zadávanie presných hodnôt (obr. 2.9).
- Dátum – na jeho výber je jednoznačne vhodný kalendár (obr. 2.10). [22]

Obr. 2.6: Checkbox a toggle box. [22]



Obr. 2.7: Dlaždice, výber z možností. [22]



Obr. 2.8: Stepper – krokovač. [22]

Obr. 2.9: Škála hodnôt. [22]

Obr. 2.10: Kalendár. [22]

Kapitola 3

Technológie určovania polohy v interiéroch

Interiérový navigačný systém (anglicky Indoor Positioning System; ďalej len IPS) je systém určený na určenie polohy objektov a ľudí v interiéroch s použitím svetla, rádiových vĺn, magnetcikého poľa, akustický signálov alebo iných informácií. Na trhu existuje množstvo systémov, ktoré IPS komerčne integrujú, ale zatiaľ neexistuje jednotný štandard. [17]

3.1 Určovanie polohy bez použitia bezdrôtovej infraštruktúry

V nasledujúcich odsekoch popisujem rôzne spôsoby určovania polohy bez použitia bezdrôtovej infraštruktúry, napríklad s využitím magnetizmu či vizuálnych značiek.

3.1.1 Použitím pricípov magnetizmu

Navigovanie na základe magnetizmu ponúka presnosť na 1 – 2 metre s 90 % hodnotou istoty a bez použitia dodatočnej bezdrôtovej infraštruktúry. Navigovanie na základe magnetizmu je založené na železe v základoch budov, ktoré vytvára lokálne variácie v pôsobení magnetického poľa našej planéty. Kompas integrovaný na čipe v mobilných telefónoch môže tieto variácie zmerať a tak zmapovať interiér. [18]

3.1.2 Použitím vizuálnych značiek

Vizuálny systém na navigovanie určuje aktuálnu polohu na základe zariadenia vybaveného kamerovou technikou, ktoré dekóduje koordinácie z vizuálnych značiek. V takomto systéme sú značky umiestnené na predom určených miestach v budove, pričom každá značka určuje koordináty: šírku, dĺžku a poschodie. Meranie vizuálneho uhla k jednotlivým značkám umožňuje zariadeniu určiť vlastnú polohu. [27]

3.2 Určovanie polohy s použitím bezdrôtovej infraštruktúry

Každá bezdrôtová technológia môže byť použitá na určenie polohy. Na trhu sú rôzne systémy využívajúce existujúce bezdrôtové infraštruktúry, ale aj také, ktoré predpokladajú zavedenie novej špecifickej infraštruktúry.

3.2.1 Pomocou Wi-Fi signálu

Určenie polohy na základe Wi-Fi signálu (anglicky Wi-Fi positioning system; ďalej len WPS) je technika, ktorá využíva bezdrôtové prístupové body, pričom polohu určuje na základe intenzity prijatého signálu (Received Signal Strength; RSS) a metódy tzv. fingerprinting. Presnosť závisí na počte prístupových bodov, na základe ktorých sa poloha určuje. Často však dochádza ku skresleniu signálu a zníženej presnosti určenia polohy. [16]

3.2.2 Pomocou technológie bluetooth

Určovanie polohy pomocou technológie Bluetooth ja založené na vymedzení blízkej oblasti a nie presného bodu v priestore. Mikromapovanie vnútorného priestoru pomocou Bluetooth Low Energy technológie bolo prezetované firmou Apple pod názvom iBeacon. Zároveň boli implementované rozsiahle riešenia založené na tejto technológii v praxi. [26]

3.2.3 Určovanie polohy pomocou Ultra-Wideband

V tomto dokumente je navrhnutý asynchrónny systém na meranie polohy. Preukázaný systém sa skladá z vysielača UWB a niekoľkých prijímačov UWB signálu, ktorých polohy sú známe. Proces merania polohy začína lokátorom odosielajúcim UWB impulz. Po príchode je pulz zosilnený a vysielaný ďalej, pričom nie je implementovaný žiadny synchronizačný mechanizmus. Namiesto toho tento systém meria rozdiel času odoslania a prijatia signálu. Spolu so znalosťami polohy viacerých vysielačov a prijímača lokátora je možné vypočítať absolútnu polohu. Poloha lokátora je na priesečníku viacerých elíps vytvorených signálom medzi lokátorom a vysielačom. [30]

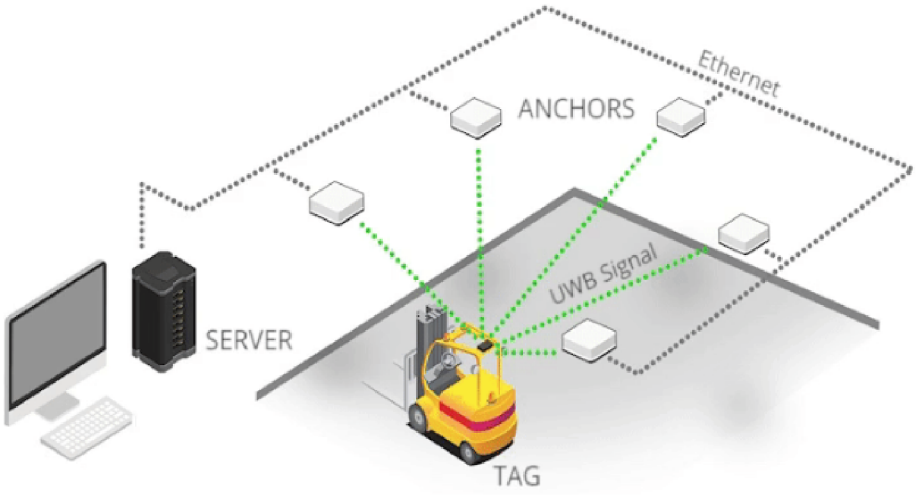
3.3 Technológie spoločnosti Sewio

Na určenie polohy objektu používa Sewio *Real-Time Location System*[1]. (ďalej len RTLS) zostavu hardvérových zariadení, ktoré zbierajú *Ultra-wideband*[2] *signál (ďalej len UWB)* a prenášajú ho ďalej na *RTLS server*, kde je výsledná poloha dopočítaná. RTLS pozostáva z *značiek (ďalej len anglicky Tag)*, *kotiev (ďalej len anglicky anchor)* a softvéru *RTLS Studio*. Kotvy sú nainštalované na strope alebo stenách miestnosti. Mobilné značky v pravidelných intervaloch vysielajú veľmi krátky UWB signál, ktorý je zachytený kotvami. Kotvy prenášajú údaje z UWB signálu prostredníctvom ethernetu alebo Wi-Fi na server, kde beží softvér RTLS Studio. RTLS Studio následne vypočíta presnú polohu a uloží tieto dáta do databázy. Dáta o aktuálnej pozícii sú následne dostupné cez API softvéru RTLS Studio. Sewio RTLS Studio umožňuje vizualizáciu a analýzu získaných dát v reálnom čase.

Medzi hlavné benefity Sewio RTLS patrí vynikajúci ovládací systém, jednoduché škálovanie a vysoká presnosť. Hlavné funkčné bloky RTLS platformy sú definované vo vrstvách. Prenášané dáta sú generované na hardvérovej vrstve. UWB signál z hardvérovej vrstvy je spracovaný vrstvou určenia polohy, kde je poloha vypočítaná a odfiltrovaná. Dátová vrstva je zodpovedná za uloženie dát o polohe a ich reprezentáciu pre následnú vizualizáciu a analýzu. Na vrstve služby RTLS platforma ponúka nástroje na kontrolu a optimalizáciu výkonu celého systému. Dáta o polohe v reálnom čase a analytické dáta sú následne vizualizované vo vizualizačnej vrstve (obr. 3.1). RTLS platforma môže byť napojená na dodatočné aplikácie cez jej **REST API**, **Websocket** alebo cez **dátový tok na UDP**. [7]

[1] Slovensky systém pre určenie polohy v reálnom čase

[2] Slovensky ultra-širokopásmový.



Obr. 3.1: Schéma fungovania Sewio UWB RTLS platformy. [7]

3.3.1 Anchor

Anchor je základný hardvérový komponent platformy. Ide o statické zariadenie, ktoré zachytáva UWB signál zo značiek. Zostava kotiev vytvára polohovaciu infraštruktúru daného priestoru. RTLS platforma je plne škálovateľná a umožňuje neobmedzené zväčšovanie priestoru formou pridávania ďalších kotiev do siete. Na demonštratívne účely je možné použiť minimum 5 kotiev a potom jednoducho škálovať na reálne použitie. Kotva má HTTP rozhranie pre komunikáciu a ovládanie, konfigurácia je robená vzdialene cez RTLS manažér. [7]

3.3.2 Tag

Sewio RTLS platforma poskytuje niekoľko druhov tagov:

- **Tag pre osobné použitie** obsahuje tzv. Piccolino Tag pre dosiahnutie minimalistického dizajnu. Tento tag je napájaný malou gombíkovou baterkou s výdržou cez 1 kalendárny rok. Pokiaľ sa osoba nosiaca tento tag nepohybuje, tag prechádza do módu šetrenia energie. Osobný tag je dodávaný s náramkom na ruku, príveskom okolo krku alebo s klipom.
- **Tag pre priemyselné použitie** je vysokoodolný tag pre náročné premyselné prostredie. Sewio vyvinulo špeciálny obal, ktorý umožňuje umiestniť tag do kovových predmetov bez straty signálu. Je vhodný na sledovanie paliet, predmetov a iných potrebných vecí.
- **Tag pre použitie na vozidlách** je vylepšený tag, ktorý dokáže rozoznať aj malé pohyby. Obsahuje plno senzorov: akcelerometer, gyroskop, magnetometer, barometer a termometer. Tento tag je napájaný 600 mAh Li-ion batériou s predĺženou životnosťou až 5 rokov. Je vhodný na použitie vo vysokozdvižných vozíkoch, malých osobných vozíkoch alebo na sledovanie hráčov športov. [7]

Kapitola 4

Analýza požiadaviek na platformu interaktívnych výstav

Tvorím platfomu pre interaktívne výstavy založenú na aktuálnej polohe pozostávajúcu z viacerých vrstiev. V časti správy obsahu je požiadavkou vytvoriť plugin do October CMS, ktorý umožní administrátorom výstav definovať v systéme *exponáty*, ktoré majú ako vlastnosť fyzickú *lokalitu* nadefinovanú v systéme RTLS Manager spoločnosti Sewio a zároveň k *exponátom* je možné nadefinovať viac *obsahových blokov*. *Obsahový blok* môže byť rôzneho druhu: textový, obrázkový, video, audio, *anketa* či *kvíz*. *Anketa* je zložená z viacerých *otázok*, ktoré majú tiež dva typy: výber jednej z možností a výber viacerých možností. *Kvíz* je zložený z viacerých *kvízových otázok*, ktoré majú štyri možné *odpovede*, z toho je jedna *správna*.

Druhou časťou je HTTP REST API naprogramované v PHP a frameworku Laravel, ktoré poskytuje všetky vyššie uvedené dáta na čítanie.

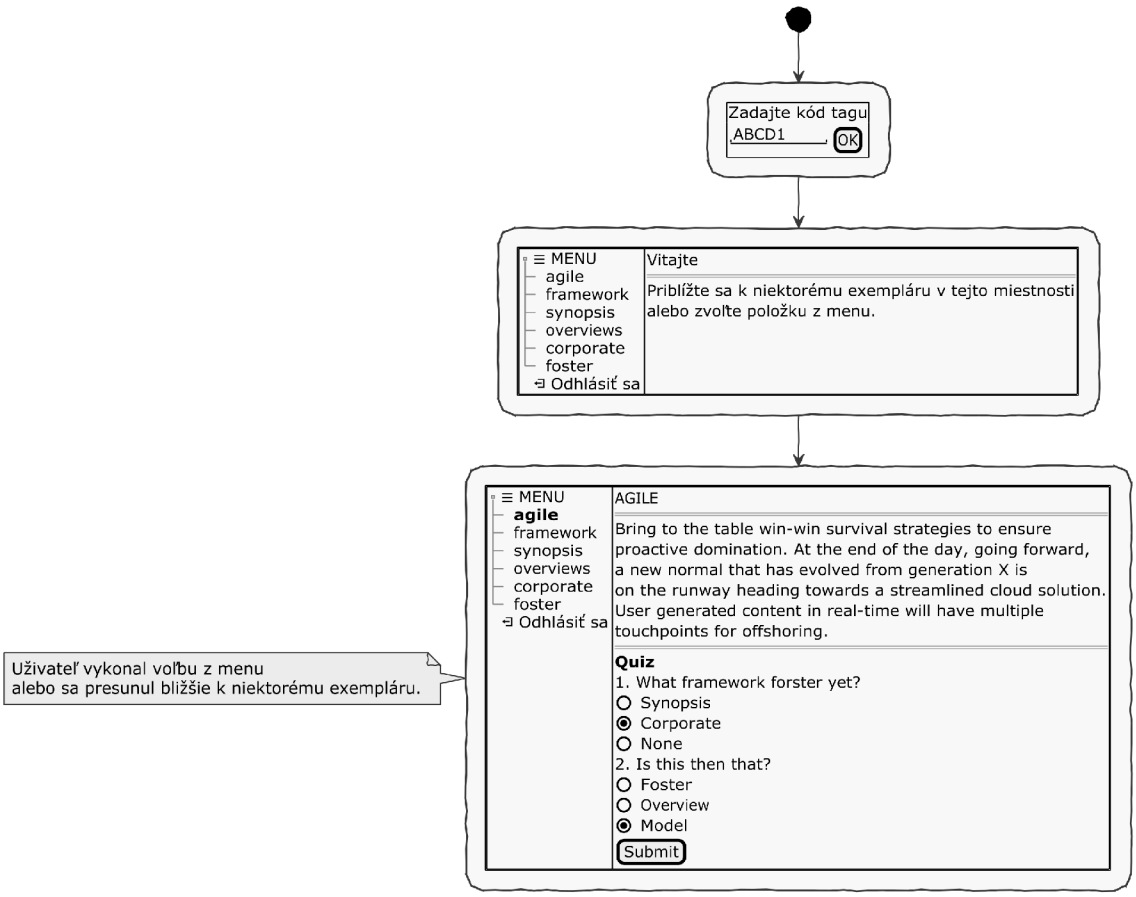
Treťou časťou je Vue.js single-page aplikácia ktorá má dve základné obrazovky. *Obrazovka pre prihlásenie sa* s kódom tagu (týmto tagom sa určuje aktuálna poloha používateľa) a po zadaní správneho kódu tagu sa načíta druhá obrazovka, ktorá je rozdelená na dve časti: menu s ponukou exemplárov a priestor pre vykreslenie obsahu pridelenému konkrétnemu exemplári (obr. 4.1).

4.1 Diagram prípadov použitia

Jednotlivé prípady použitia platformy zachytávam pomocou diagramu prípadov použitia, ktorý je súčasťou jazyka UML (obr. 4.2). V diagrame účinkujú dvaja aktéri: Administrátor a Používateľ, pričom každý má vlastnú oddelenú sadu prípadov použitia.

Aktér *administrátor* vykoná ako prvé **prihlásenie sa do CMS**, pričom túto činnosť vykonáva *administrátor* vždy po dlhšej nečinnosti v aplikácii, pretože ho po uplynutí určitého času automaticky odhlási. *Administrátor* ďalej môže v CMS **spravovať exemplár**e, s ktorými úzko súvisí **správa obsahových blokov**. K jednotlivým obsahovým blokom je možné v systéme namiesto statického obsahu priradiť anketu či kvíz, pričom **ankety** a **kvízy spravuje** *administrátor* samostatne. Aktér *používateľ* sa najskôr **prihlási do aplikácie zadaním kódu z tagu**, prípadne vyplní doplňujúce informácie.[1] *Používateľ* má potom možnosť **vybrať obsah manuálne** alebo v prípade, že sa priblíží k niektorému z vystavených exemplárov, aplikácia mu **obsah zobrazí automaticky na základe zistenej**

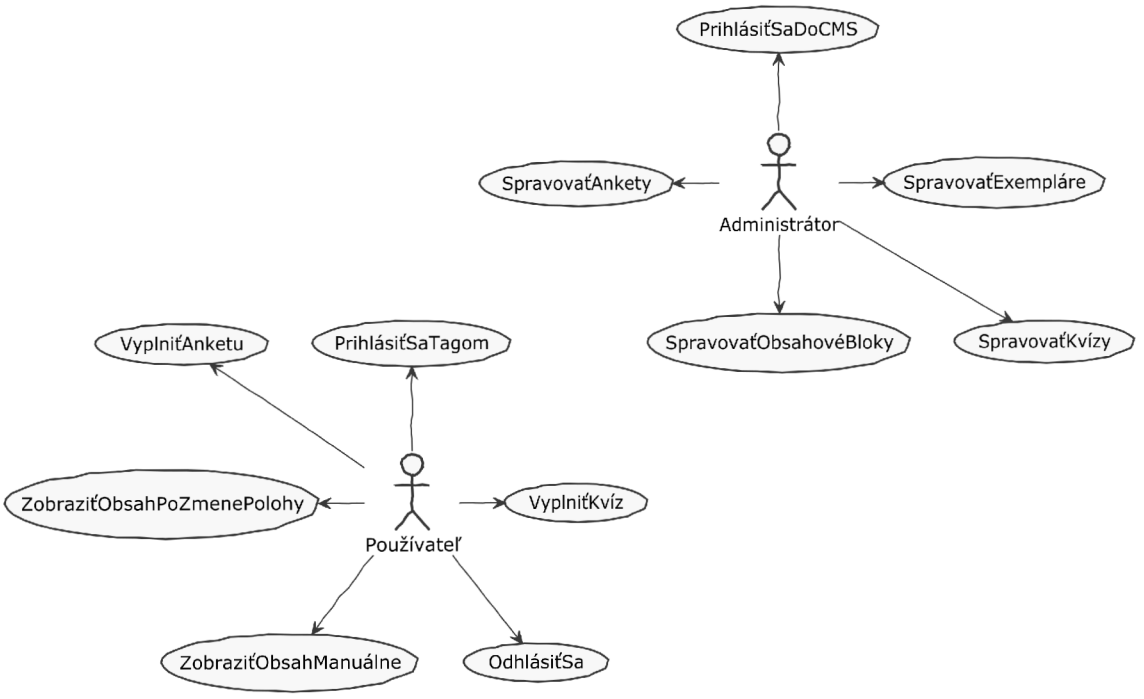
[1] Podrobnosti sú uvedené v prílohe C v detaile prípadu použitia UC1 (C.1).



Obr. 4.1: Náčrt podoby obrazoviek vo forme wireframe.

zmeny polohy. V obsahových blokoch, ktoré majú priradený **kvíz** či **anketu**, môže tieto používateľ **vyplniť** a odoslať. Po dokončení exkurzie **sa používateľ z aplikácie odhlási**.

Detaily vybraných prípadov použitia je možné nájsť v prílohe C.



Obr. 4.2: Diagram prípadov použitia.

Kapitola 5

Návrh štruktúry dátového zdroja webovej aplikácie

Medzi základné kroky, ktorými prechádzame pri vývoji softvérového systému patrí analýza požiadaviek, návrh, programovanie, testovanie a predanie do užívania používateľom. Konceptuálny návrh patrí do etapy analýzy požiadaviek. Jeho cieľom je analyzovať požiadavky na dáta, ktoré budú uložené v databáze. Je založený na objektoch aplikačnej domény, pre ktorú sa softvérový systém vyvíja. [29]

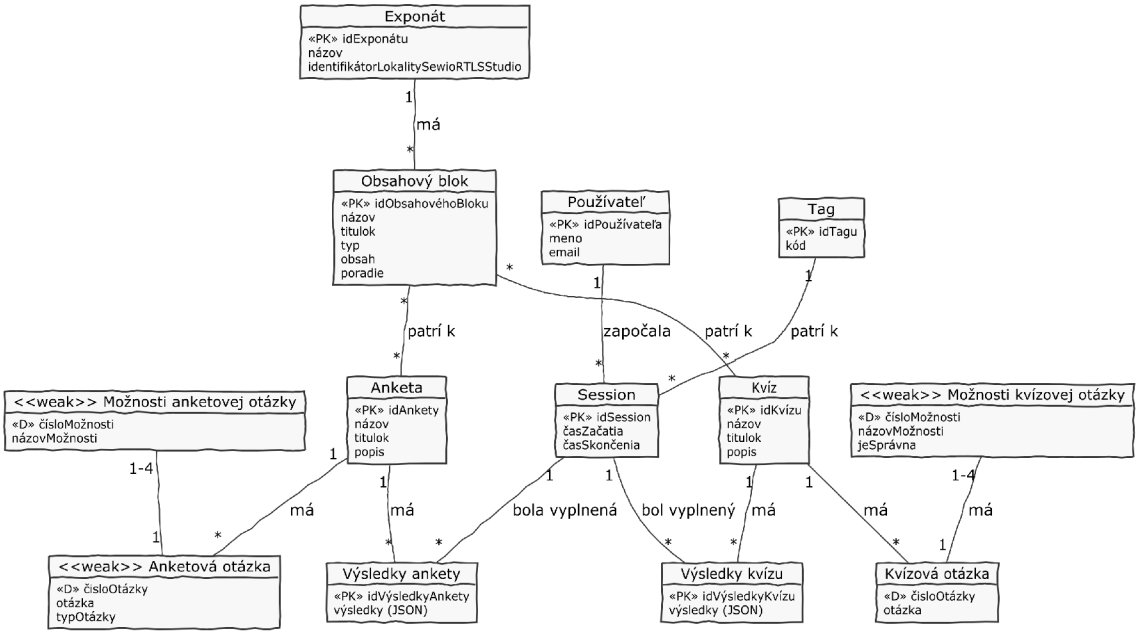
Keďže v tejto práci vyvíjam softvér pre zobrazovanie dynamického obsahu na základe aktuálnej polohy budem modelovať dáta reprezentujúce:

- bloky dát,
- exponáty,
- používateľov systému,
- tagy,
- sedenia (ďalej len anglicky sessions),
- dotazníky,
- kvízy a vzťahy medzi nimi.

Výsledkom konceptuálneho modelovania je konceptuálny model, ktorý reprezentujeme v podobe ER (z angličtiny Entity Relationship; slovensky vzťahy entít; ďalej len ER) modelu a prezentujeme formou ER diagramu.

Ďalším krokom je logický návrh. Jeho cieľom je navrhnúť štruktúru databázy (čiže štruktúru jednotlivých tabuliek) tak, aby bolo v databáze možné reprezentovať všetky požadované informácie, neexistovala redundancia a bola zabezpečená kontrola integritných obmedzení vyplývajúcich zo závislostí medzi hodnotami uloženými v databáze. Výsledkom logického návrhu je schéma relačnej databázy – logická schéma databázy.

Posledným krokom je fyzický návrh. Jeho výsledkom je fyzická schéma. Cieľom je navrhnúť fyzické uloženie tabuliek, ktoré sú výsledkom logického návrhu využitím prostriedkov konkrétneho databázového systému tak, aby bolo dosiahnuté čo najlepších výkonnostných parametrov. Na fyzickej úrovni majú relačné databázy zložitejšiu štruktúru než na úrovni logickej, kde sú jedinou štruktúrou podľa relačného modelu dát tabuľky. Organizáciu na fyzickej úrovni relačný model neurčuje, to je už záležitosť jeho konkrétnej implementácie.



Obr. 5.1: ER Diagram platformy.

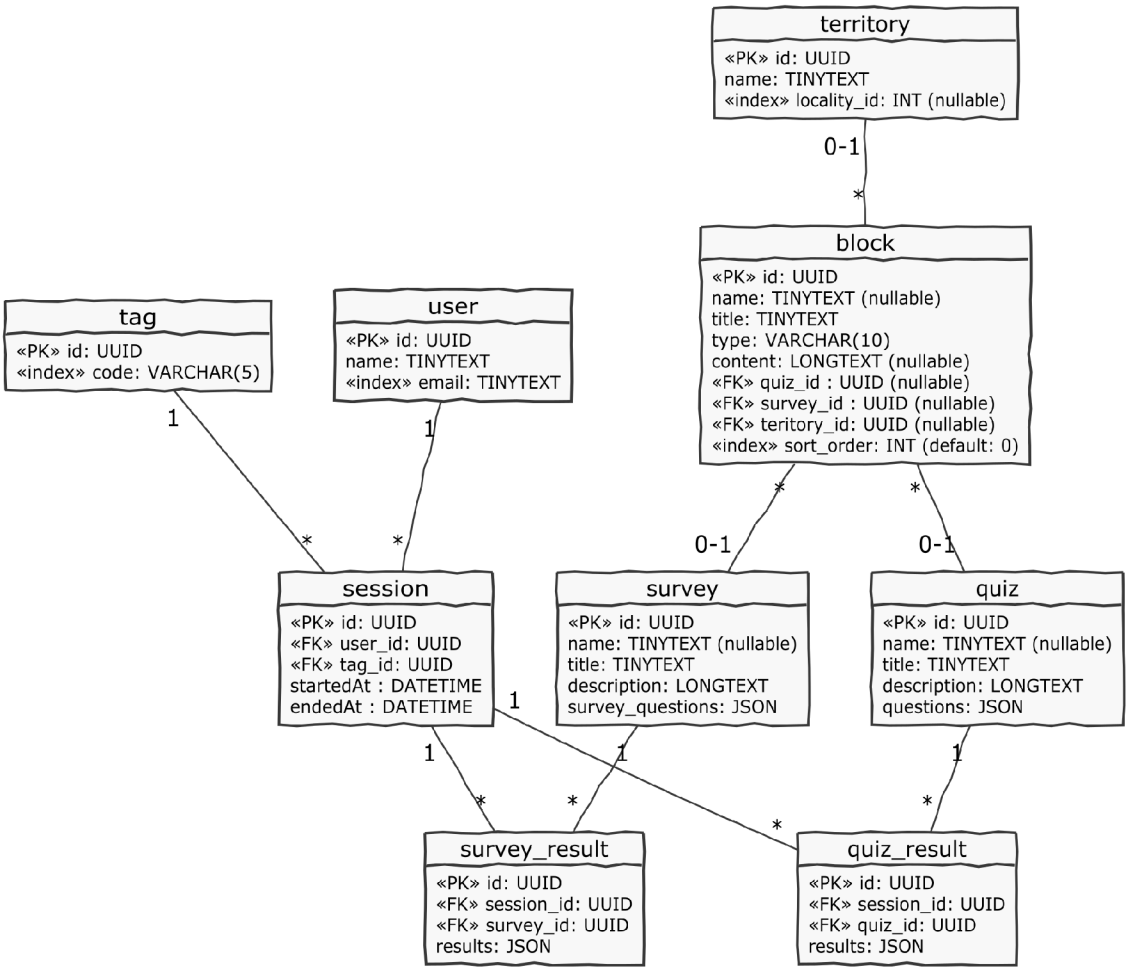
Pre zefektívnenie práce s dátami v databázom systéme sa zavádzajú prístupové metódy, ako je indexovanie a hešovanie. [29]

5.1 ER Diagram

ER diagram (obr. 5.1) zobrazuje entitné množiny a ich vzťahy v systéme.

5.2 Implementácia v MySQL

Na diagrame (obr. 5.2) je možné vidieť diagram tabuliek implementovaných na základe ER diagramu v databázovom systéme MySQL.



Obr. 5.2: Diagram návrhu tabuliek v databázovom systéme MySQL.

Kapitola 6

Výber vhodných technológií

Na získanie konkurenčnej ale aj všeobecnej výhody nového softvéru na mieru hrá dôležitú rolu správny výber technológií. V tejto kapitole popisujem rôzne technológie a nástroje určené na vývoj frontendu a backendu webovej aplikácie.

6.1 Jednostránková webová aplikácia

Webové stránky sú čoraz komplikovanejšie a interaktívnejšie, čo kladie tlak na vývojárov frontendu, aby spolu so zvyšujúcou sa komplexitou webových stránok používali silnejšie nástroje. Použitím jQuery je jednoduché upraviť časť textu na stránke. Avšak keď je potrebné vykonať úprav viac, pracovať s interaktívnymi časťami stránky, či s vytváraním smerovania na strane klienta (anglicky client-side routing), vtedy prichádza do hry niektorý z robustných JavaScript frameworkov, ako je Vue.JS, React alebo Angular. Aplikácia s viacerými podstránkami, ktorými je možné prechádzať bez nutnosti poslať na server nový dotaz, sa nazýva jednostránková aplikácia (anglicky single-page application).

JavaScript frameworky sú nástroje, ktoré vývojárom uľahčujú vývoj interaktívnych stránok so single-page aplikácií. Frameworky pomáhajú programátorom vytvoriť plne funkčné webové aplikácie, ktoré dokážu manipulovať s komplikovanými dátami, zobrazovať ich, ovládať smerovanie na strane klienta bez nutnosti spoľahnúť sa na server. Prípadne aj vytvorenie plnej webovej stránky, ktorá potrebuje na kompletné načítanie všetkých podstránok len jeden dotaz na server. [25]

6.1.1 JavaScript

JavaScript bol vyvinutý v roku 1995 Brendanom Eichom, softvérovým vývojárom v spoločosti Netscape Communications Corporation. Jeho prvý vývoj bol veľmi rýchly a kritici mu vyčítali málo plánovania do budúcnosti. Vo veľa veciach bol JavaScript nadčasový a trvalo 15 rokov, kým si ho všimli aj mainstreamový vývojari a ocenili jeho sofistikovanosť. Pôvodný názov JavaScriptu bol Mocha, neskôr nakrátko LiveScript a nakoniec bol oficálne pomenovany JavaScript, tak ako ho poznáme aj teraz. Slovo Java v JavaScript nebolo zvolené náhodne, ale zároveň je aj veľmi zmätočné. JavaScript má viac spoločné s programovacím jazykom Self[1] než s Javou. Názov JavaScript bol však marketingovým ťahom a pokusom zviezť sa na vlne popularity programovacieho jazyku Java.

[1] Programovací jazyk vyvinutý v spoločnosti Xerox PARC v 80. rokoch.

V roku 1997 vznikol štandard ECMA-262, ktorého najznámejšou implementáciou bol práve JavaScript. Veľká aktualizácia štandardu bola ECMAScript verzia 5.1 (skratka ES5) publikovaná v júni 2011. Práve tá sa stala štandardom naprieč webovými prehliadačmi a dnes ju podporuje väčšina webových prehliadačov. V roku 2015 prišla ďalšia veľká aktualizácia v podobe ECMAScript 6 (skratka ES6), ktorá priniesla množstvo vylepšení a nových funkcionalít. Od roku 2016 sa komisia rozhodla, že bude vydávať novú verziu štandardu každý rok, aby sa predišlo komplikáciám s veľkou aktualizáciou, ako to bolo v prípade ES6. Posledná verzia je tohtoročná ES10 (2019). [14]

6.1.2 Výber JavaScript frameworku

Na trhu s JavaScript frameworkami je dnes niekoľko hráčov:

- React – React sám o sebe nie je kompletným frameworkom. Poskytuje základ, ale pre pokročilé funkcie musíme hľadať a použiť ďalšie knižnice, čo zvyšuje komplexitu.
- Angular – Framework vhodný na väčšie a komplexnejšie projekty, naprogramovanie jedného komponentu vyžaduje napísanie dlhšieho kódu, ako v prípade Reactu alebo Vue.js. Viď. obr. 6.1.
- Vue.js – Zdrojový kód aplikácie vo Vue.js je jednoduchý na čítanie, má jasnú logiku a kompletne separuje logiku aplikačnú od zobrazovacej. Vue.js má za sebou skvelú komunitu a množstvo dostupných rozšírení či knižníc. [13]

Pre vývoj frontendu som si vybral Vue.js, kvôli prívetivej dokumentácii, množstvu podporovateľov a predchádzajúcim skúsenostiam.

6.1.3 Gridsome

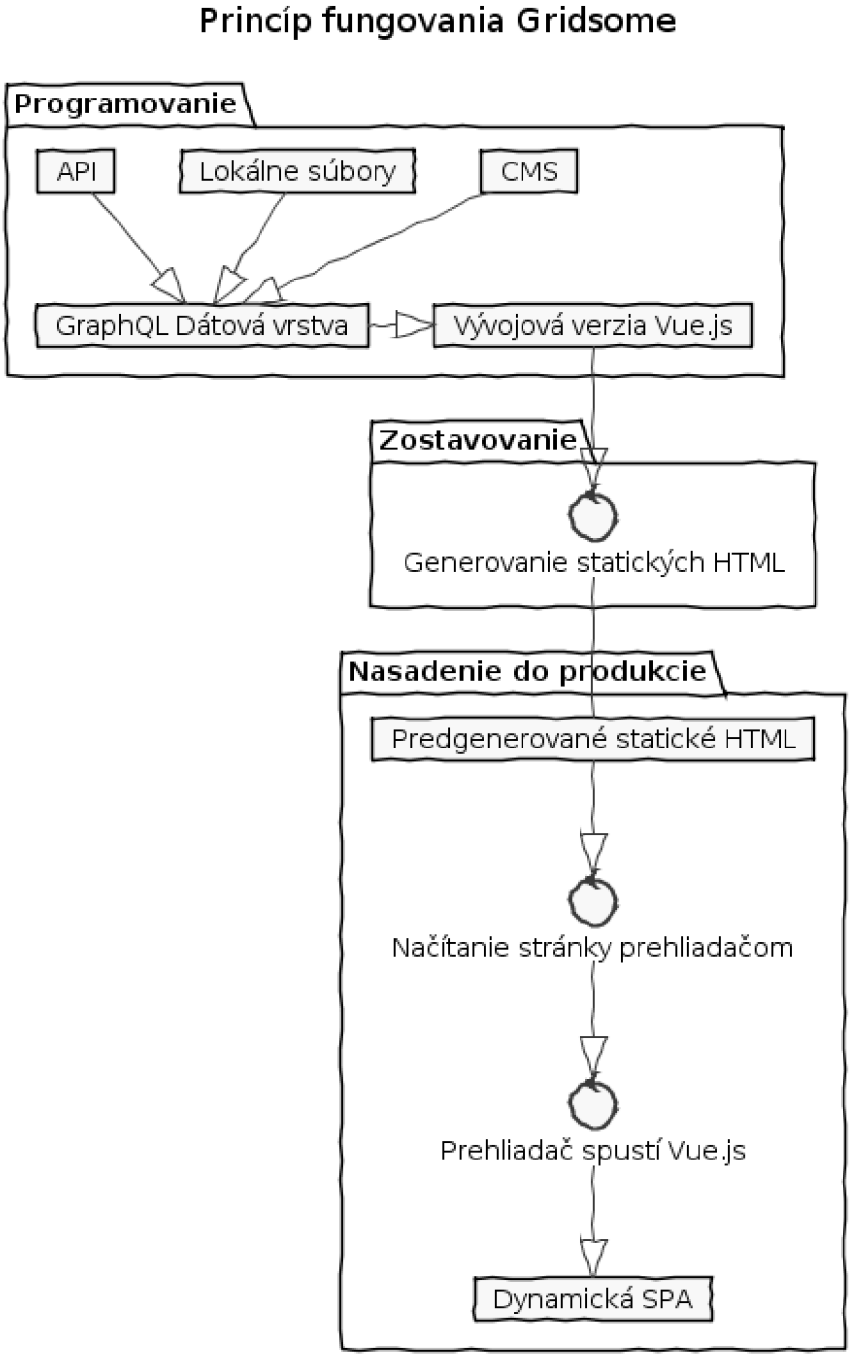
Gridsome je Vue.js knižnica, ktorá umožňuje vytvárať rýchle single-page aplikácie napojené takmer na akýkoľvek zdroj dát. Gridsome dokáže predgenerovať statické HTML súbory, ktoré sa po načítaní v prehliadači sfunkčnia a zapnú plne funkčnú Vue.js single-page aplikáciu. Načítavanie stránky je tak veľmi rýchle a nestráca sa pri tom žiadna funkčnosť aplikácie. Zdroje dát môžu byť v lokálnych súboroch, externej API alebo databáze. Gridsome má vlastnú vrstvu založenú na GraphQL, s pomocou ktorej je možné vytiahnuť len tie potrebné dáta pre konkrétny prípad. Využité dáta sú počas procesu generovania statickej stránky uložené vo forme JSON. Vygenerované statické HTML stránky sú SEO prívetivé a môžu byť nasadené na akomkoľvek webovom serveri (obr. 6.2). S Gridsome je zároveň jednoduché použiť CDN[2] pre rýchle doručenie obsahu kdekoľvek na svete. [4]

6.2 Aplikačné rozhranie a systém pre správu obsahu

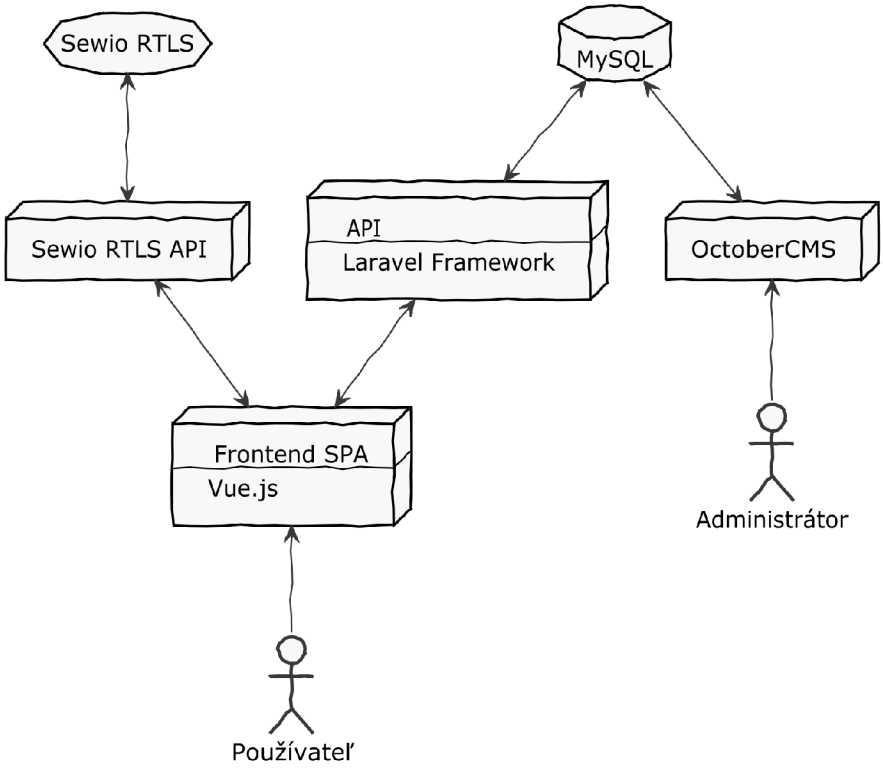
Aplikácia v platforme, ktorou sa v tejto práci zaoberám je zložená z niekoľkých na seba napojených komponentov, ktoré popisuje diagram 6.3. Ako systém pre správu obsahu som zvolil OctoberCMS, kvôli jeho flexibilite, možnosti rýchleho vývoja a kvôli tomu, že je založený na PHP frameworku Laravel, ktorý je veľmi dobre dokumentovaný a má okolo seba výbornú komunitu. Pri vývoji API zostávam pri programovacom jazyku PHP a ako framework volím Laravel. S Laravelom mám dobré skúsenosti z predchádzajúcich školských aj komerčných

[2] Z angličtiny Content Delivery Network; slovensky: sieť pre doručovanie obsahu.

Obr. 6.1: Porovnanie súborovej štruktúry zdrojových kódov rovnakej aplikácie napísanej v React, Angular a Vue.js (zľava). [13]



Obr. 6.2: Princíp fungovania knižnice Gridsome. [4]



Obr. 6.3: Diagram platformy a jej jednotlivých častí.

projektov a jeho výhodou je aj to, že podporuje najnovšie PHP 7.3. Ako som spomínal v kapitole 6.1.2, na frontend vyberám framewrok Vue.js. Frontend komunikuje so Sewio RTLS API, ktoré čerpá dáta v reálnom čase zo systému Sewio RTLS. Viac o Sewio RTLS nájdete v časti 3.3.

6.2.1 Systém pre správu zdrojových kódov

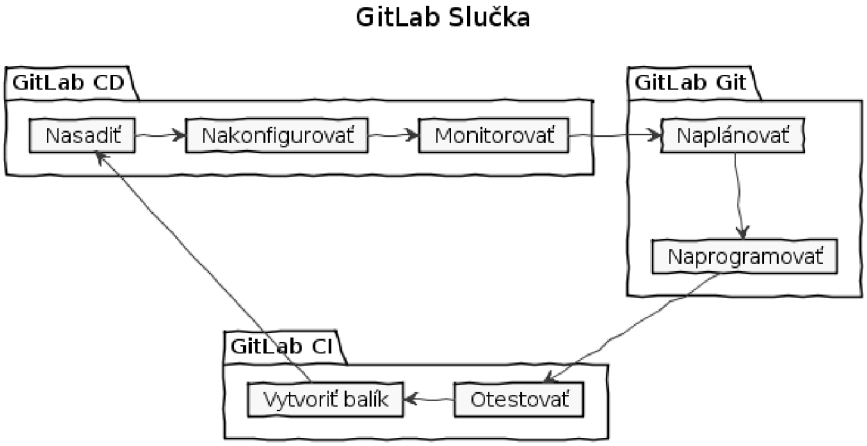
Zdrojové kódy spravujem v systéme pre správu zdrojových kódov Git. Git je v dnešnej dobe štandardom medzi jednotlivcami, malými tímami aj tímami vo veľkých korporáciách.

Git, na rozdiel od iných podobných systémov, spracúva dáta inak. Chápe ich ako sadu snímkov (anglicky snapshots) vlastného malého súborového systému. Vždy, keď sa v systéme vykoná zmena, Git vytvorí snímku, ako všetky súbory v danom okamihu vyzerajú a uloží si referenciu na túto snímku. Git je rýchly, pretože väčšina bežných úkonov prebieha lokálne, bez nutnosti prístupu do siete. Decentralizovanosť Gitu znamená, že každý účastník v tíme má stiahnutú celú históriu projektu a nespolieha sa na jeden centrálny zdroj. [15]

6.2.2 GitLab

GitLab je plne integrovaná platforma pre vývoj softvéru, ktorá je transparentná, rýchla a efektívna (obr. 6.4). Je prvou platformou, ktorá zabezpečuje vyvíjaným aplikáciám celý DevOps[3] životný cyklus. GitLab v sebe integruje nástroje pre správu projektu, agilný ma-

[3]Z anglického Development and Operations; slovensky vývoj a operácie.



Obr. 6.4: GitLab je plne integrovaná platforma pre vývoj a nasadzovanie softvéru. [9]

nažment, správu Git repozitárov. Zastrešuje tiež kompletný proces zostavenia aplikácie a jej nasadenie do produkcie prostredníctvom GitLab Ci/CD. [9]

Kapitola 7

Implementácia platformy

Pri programovaní *backendu* a *API* som využíval IDE[1] PhpStorm od spoločnosti JetBrains, ktoré je hodnotené ako jedno z najlepších IDE pre programovanie v PHP. *Backend* aj *API* sú založené ako samostatné projekty, ktoré sú aj samostatne verzované v nástroji Git.[2] Počas vývoja som hojne využíval krokovanie a inšpekciu spúšťaného kódu pomocou nástroja XDebug, ktorý má v PhpStorm natívnu podporu.[3] Samotný framework Laravel, jeho závislosti a ďalšie knižnice, som nainštaloval a spravoval pomocou balíčkovacieho systému Composer.[4] Aby PhpStorm lepšie porozumel zdrojovému kódu, v ktorom som využíval framework Laravel, nainštaloval som balíček Laravel IDE Helper, ktorý automaticky vygeneroval pomocné súbory, a tie následne PhpStorm priebežne analyzoval.[5]

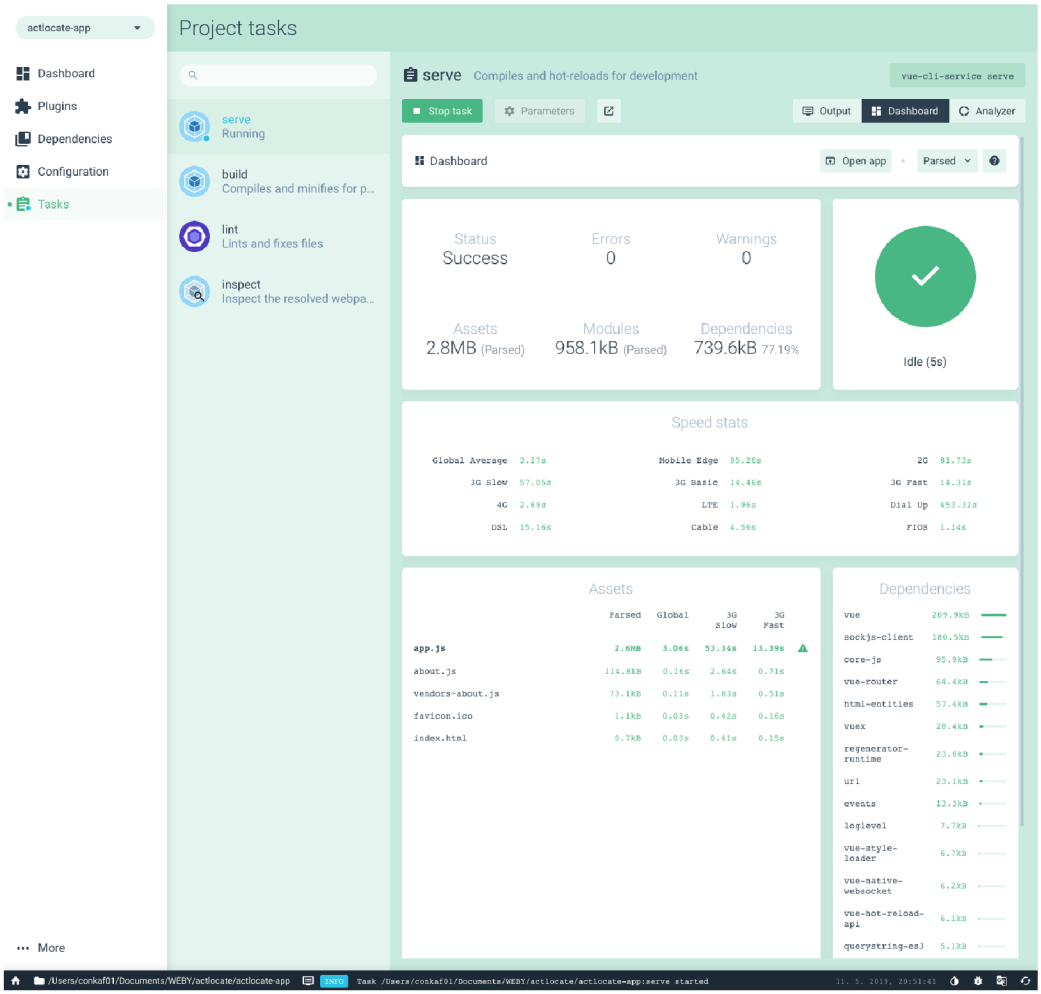
Počas implementácie *API* som využil aj balík Laravel Craftsman, ktorý umožňuje predgenerovanie zdrojového kódu podľa zadaných parametrov, a tým šetrí drahocenný čas.[6] Zároveň som využil odporúčania a znalosti z článku „Build a REST API with Laravel API resources“, kde sú uvedené cenné rady, ako správne a udržateľne postaviť API vo frameworku Laravel.[7]

Pre implementáciu *frontendu* som využil IDE WebStorm od spoločnosti JetBrains. WebStorm som zvolil, pretože má podobné prostredie ako PhpStorm, s ktorým mám veľa skúseností. Podobne ako pri programovaní v PHP, kde som využil balíčkovací systém Composer, som pri programovaní frontendu využil balíčkovací systém NPM.[8] Na správu frontend Vue.js SPA som využil nástroj Vue CLI (obr. 7.1), ktorý umožňuje inštaláciu dodatočných knižníc a doplnkov z NPM, spúšťanie build procesu[9] a uľahčuje sledovanie a opravu prípadných chýb počas tohto procesu.[10]

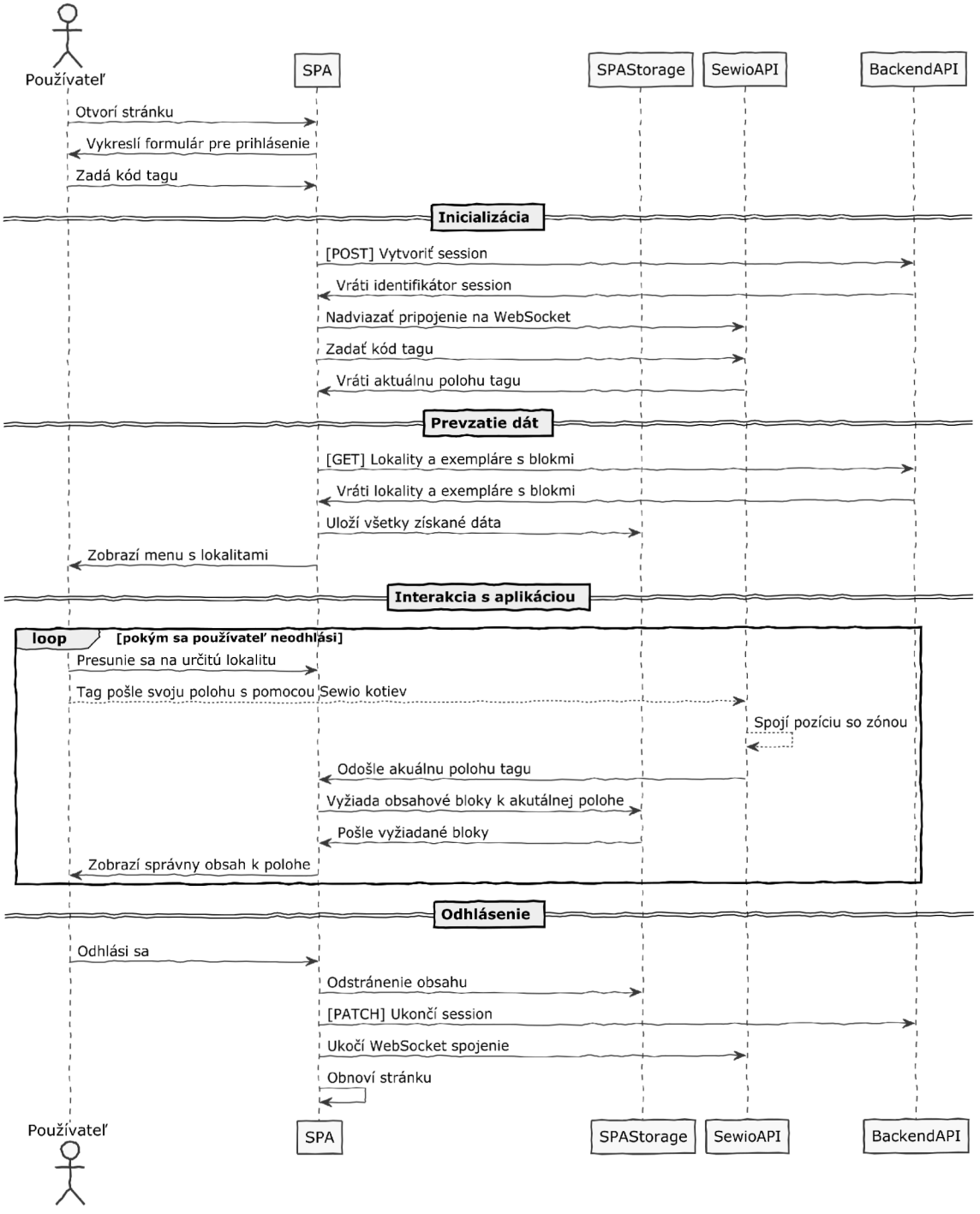
7.1 Diagram toku správ a dát v platforme

Sekvenčný diagram na obr. 7.2 znázorňuje výmenu správ a dát medzi jednotlivými aktérmi v procese práce s platformou a jej vrstvami.

[1] Z angl. Integrated development environment; slovensky integrované vývojové prostredie.
[2] Viac o nástroji Git nájdete v kapitole 6.2.1.
[3] Viac o nástroji XDebug nájdete na https://xdebug.org/.
[4] Viac o balíčkovacom systéme Composer nájdete na https://getcomposer.org/.
[5] Laravel IDE Helper – https://github.com/barryvdh/laravel-ide-helper
[6] Laravel Craftsman – https://github.com/mikeerickson/laravel-craftsman
[7] Celý článok je dostupný na https://blog.pusher.com/build-rest-api-laravel-api-resources/.
[8] Viac o balíčkovacom systéme NPM nájdete na https://www.npmjs.com/.
[9] Slovensky proces zostavovania aplikácie.
[10] Viac o nástroji Vue CLI nájdete na https://cli.vuejs.org/.



Obr. 7.1: Snímka obrazovky počas používania UI nástroja Vue CLI



Obr. 7.2: Diagram toku správ a dát v platforme.

Kapitola 8

Automatické a používateľské testovanie platformy

Testovanie je dôležitá časť vývoja akéhokoľvek softvéru. Počas implementácie je nutné overovať funkčnosť programu a zároveň kontrolovať, do akej miery odpovedá súčasný stav návrhu aplikácie. Každá časť platformy má vlastné špecifické spôsoby testovania.

8.1 Testovanie backend CMS

Vytvorené administračné rozhranie a implementovaný plugin do October CMS som testoval priebežne s využitím nástrojov dostupných v IDE PhpStorm (hlavne nástroj xDebug) a potom priamym testovaním cez administračné rozhranie October CMS. Testovanie spočívalo vo vytvorení rôznorodých položiek v jednotlivých častiach. Išlo teda hlavne o rôzne obsahové bloky, ich priradenie k teritóriám, vytváranie ankiet a kvízov.

Testovanie potom prebehlo aj v spolupráci s firmou Actlocate, kde som na základa spätnej väzby opravoval chyby, menil poradie jednotlivých prvkov administračného prostredia a upresňoval ich popisky.

8.2 Testovanie API

Testovanie API naprogramovaného vo frameworku Laravel spočívalo v dvoch častiach: unit tests[1] a HTTP testovanie API.

Framework Laravel už v základe obsahuje ukážkovú konfiguráciu PHP testovacej knižnice PHPUnit[2], čo značne uľahčuje vytváranie, spúšťanie a udržiavanie unit testov.

Laravel vo svojej dokumentácii odporúča a popisuje dva hlavné spôsoby testovanie pomocou PHPUnit:

- Čisto jednotkové testy, ktoré sa zaoberajú veľmi malou a izolovanou časťou zdrojového kódu. Najčastejšie sa jedná o jednu metódu.
- `Feature`[3] testy sa zaoberajú väčšou časťou zdrojového kódu, často pracujú s viacerými objektami, prípadne testujú celý koncový bod API. [8]

[1] Slovensky jednotkové testy.
[2] Viac o knižnici PHPUnit nájdete na https://phpunit.de/.
[3] Slovensky vlastnosť.

8.3 Testovanie frontendu – Vue.js SPA

Dokumentácia Vue.js bližšie popisuje jeden z možných spôsobov testovania – podobne ako v prípade testovania API sa jedná o unit testy. V prípade unit testov vo Vue.js sa testujú jednotlivé komponenty. Jednoducho testovateľné komponenty sa vyznačujú hlavne tým, že ich stav je závislý len na ich `properties`[4], príklad takejto komponenty a jej unit testu uvádzam nižšie. [11]

```
<template>
  <p>{{ msg }}</p>
</template>

<script>
  export default {
    props: ['msg']
  }
</script>
```

Výpis 8.1: Príklad jednoduchej a ľahko testovateľnej Vue.js komponenty. [11]

```
...
describe('Message', () => {
  it('renders correctly with different props', () => {
    expect(getRenderedText(MyComponent, {
      msg: 'Hello'
    })).toBe('Hello')

    expect(getRenderedText(MyComponent, {
      msg: 'Bye'
    })).toBe('Bye')
  })
})
```

Výpis 8.2: Unit test Vue.js komponenty. [11]

8.4 Testovanie kompletnej platformy

Pred testovaním kompletnej platformy je potrebné splniť nasledovné úlohy:

- rozostaviť *Sewio RTLS Kit* pozostavajúci z kotiev, sieťovej kabeláže a sieťového switchu,
- spustiť *Sewio RTLS server* v rovnakej sieti ako sú Sewio RTLS kotvy,
- nakonfigurovať *frontend SPA* na použitie správnej adresy *Sewio RTLS servera*,
- nastaviť miestnosti a zóny *v Sewio RTLS Manager*,
- naplniť informácie o exponátoch a teritóriách *do backend CMS platformy*,

[4] Slovensky vstupných parametroch.

- mať tablet alebo smartfón s pripojením na internet a napojením do rovnakej siete, v akej je aj *Sewio RTLS Server*.

Testovanie kompletnej platformy bolo vykonané v spolupráci so spoločnosťou Actlocate s.r.o. Pripomienky a spätnú väzbu som následne zapracoval do aplikácie.

Kapitola 9

Kontajnerizácia a nasadenie webovej aplikácie

Virtualizácia založená na kontajneroch je v súčastnosti jednou z najperspektívnejších technológii v cloude. Poháňa inovácie a zmeny v tom, ako sa aplikácie vyvíjajú a prevádzkujú. Je ťažké nájsť väčšiu technologickú firmu, ktorá by do kontajnerových technológií neinvestovala čas a peniaze. Idea kontajnerov však nie je nová. V operačných systémoch založených na Linuxe je táto technológia dostupná už od začiatku tohto milénia. Avšak chcelo to práve Docker, aby sa táto technológia dostatočne spopularizovala a dostala do mainstreamu.

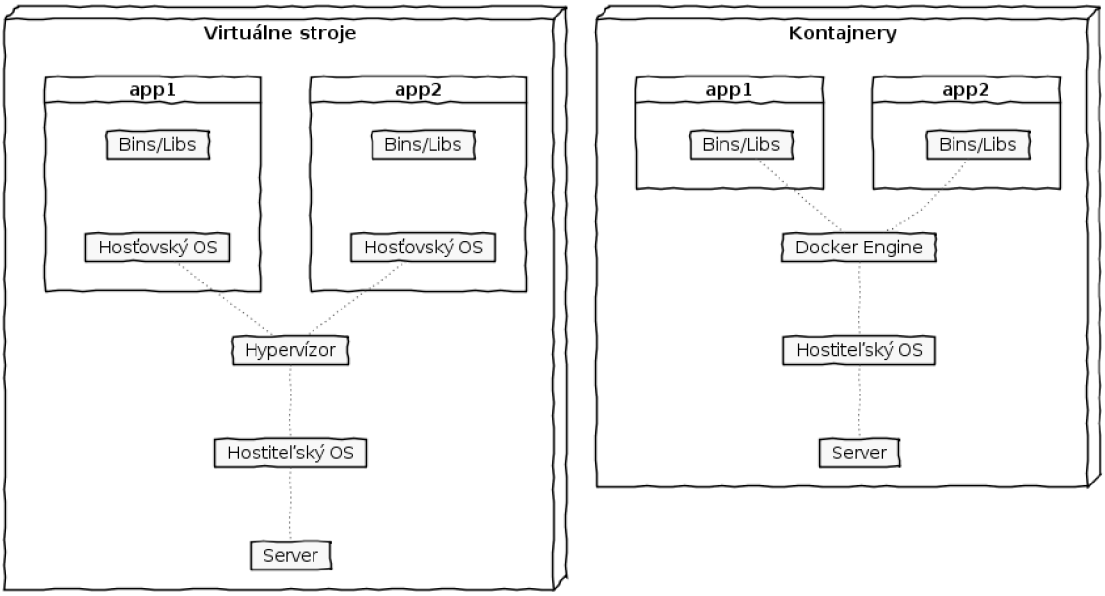
Kontajnery – virtualizácia na úrovni operačného systému – sú odľahčený prístup k virtualizácii, ktoré poskytujú úplné minumum, toho čo aplikácia potrebuje na svoj beh a fungovanie. V istom zmysle ich môžeme chápať ako super-minimalistické virtuálne stroje, ktoré nebežia na hypervízore. Nástroje, ktoré sú zahrnuté v kontajneri:

- aplikáciu,
- závislosti aplikáce,
- knižnice,
- binárne súbory,
- konfiguračné súbory.

Kontajnerizovanie aplikácie umožňuje spoľahlivý beh v rôznych prostrediach tým, že sa abstrahuje operačný systém a fyzická infraštrukúra (obr. 9.1). Kontajnerizované aplikácie zdieľajú jadro hostiteľského operačného systému s ostatnánými kontajnermi, pričom tieto zdieľané časti operačného systému sú len na čítanie. Vo vnútri kontajneru býva zvyčajne len jedna spustiteľná mikro-služba. Veľkosť kontajnerov býva väčšinou v desiatkách megabajtov a jeho spustenie trvá pár sekúnd. Keď záťaž na aplikáciu stúpa, môžu byť spustené ďalšie kontajnery, a keď záťaž klesá, dodatočné kontajnery sa vypínajú. [5]

9.1 Docker Compose

Docker Compose je nástroj na definovanie a spúšťanie multi-kontajnerových Docker aplikácií. Docker Compose používa súbory typu YAML na konfiguráciu aplikačných služieb. Následným použitím jednoduchých príkazov je možné všetky služby naštartovať z konfigurácie. Docker Compose funguje vo všetkých prostrediach – produkčných, testovacích aj vývojových. Používanie Docker Compose je trojkrokový proces:



Obr. 9.1: Rozdiel medzi kontajnermi a klasickými virtuálnymi strojmi. [5]

1. Zadefinovať prostredie aplikácie s použitím `Dockerfile` tak, že ho je možné znovu používať kdekoľvek.
2. Zadefinovať služby, ktoré aplikácia poskytuje v `docker-compose.yaml` súbore, aby mohli byť spúšťané vo vlastnom izolovanom prostredí.
3. Spustiť príkaz `docker-compose up`, čo naštartuje celú aplikáciu. [6]

V rámci konfigurácie aplikácie sú vytvorené dva súbory `docker-compose.yaml`. Ten hlavný sa používa pri vývoji a druhý `docker-compose.build.yaml` sa používa v prostredí GitLab CI/CD. V samotných definičných `Dockerfile` súboroch je využitá jedna z najnovších vlastností, tzv. *Multistage builds*[1], čo je možné využiť práve pri rozličnej konfigurácii kontajnerov pre beh vo vývojovom prostredí a produkčnom prostredí.

9.2 Kubernetes

Kubernetes (ďalej len K8s) je open-source platforma pre manažment kontajnerizovaných aplikácii v produkčnom prostredí. S K8s je jednoduché aplikáciu automaticky škálovať, zredukovať chybovosť a zvýšiť bezpečnosť. Odpadá nutnosť skriptovania pre kontrolovanie stavu, reštartovania či zmeny počtu replikovaných Docker kontajnerov. Miesto toho sa v K8s nakonfiguruje požadovaný počet kontajnerov a zvyšok je automatický. K8s dokáže dokonca aj automaticky škálovať aplikácie na základe využitia systémových prostriedkov. K8s je hlavne o abstrahovaní komplexity z komplikovaných produkčných infraštruktúr. [21]

Toto je šesť vrstiev konceptu K8s začínajúcich od abstrakcie najvyššieho stupňa:

1. Deployment – Nasadenie,

[1] Slovensky viackrokové zostavovanie.

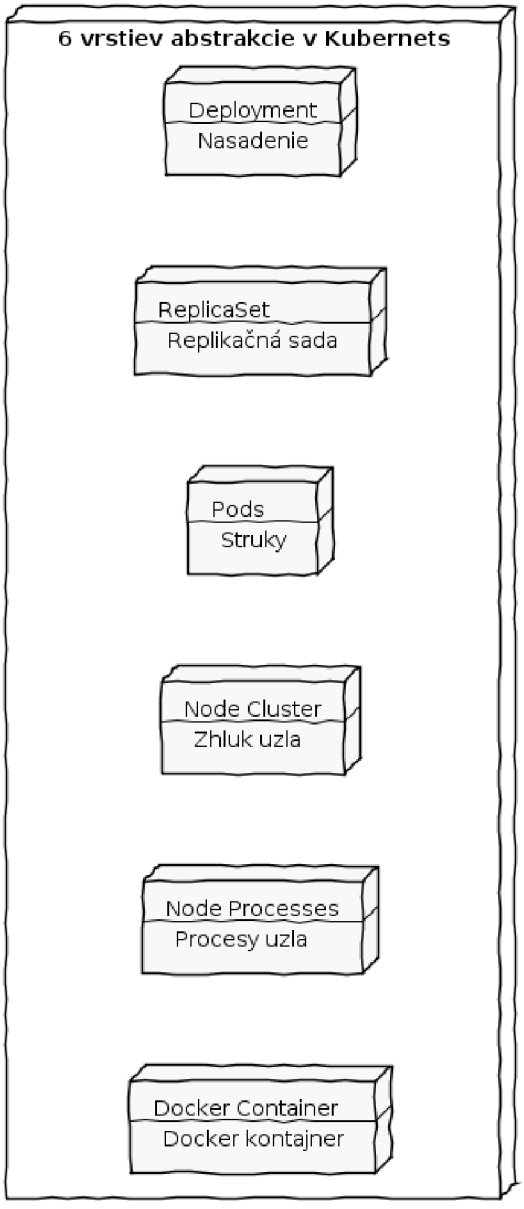
2. Replica Set – Replikačná sada,
3. Pods - Struky,
4. Node Cluster – Zhluk uzla,
5. Node Processes – Procesy uzla,
6. Docker Container – Docker kontajner.

Nasadenie (deployment) vytvára a manežuje *replikačnú sadu (replica set)*, ktorá vytvára a ovláda *struky (pods)*, ktoré sa spúšťajú na *uzloch (nodes)*, ktoré majú *kotajner runtime*, na ktorom sa spúšťa *aplikačný kód*, ktorý je zakomponovaný v *Docker kontajneroch*. [21] Viď. diagram na obr. 9.2.

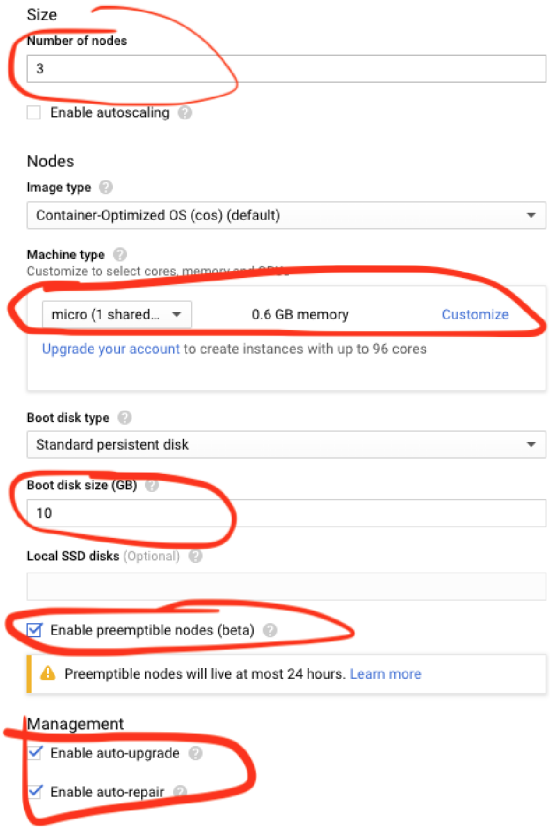
9.3 Nasadenie do Google Cloud

Výslednú aplikáciu som nasadil s použitím cloud platformy spoločnosti Google a ich produktu Google Kubernetes Engine. Využil som ponuku pre nové účty v Google Cloud, na ktoré poskytuje Google zadarmo kredit v hodnote približne 6700 Kč na jeden rok.

Pri nastavovaní Kubernetes cluster v Google Kubernetes Engine (ďalej len GKE) som sa snažil, o čo najefektívnejšie cenové nastavenie, aby som zbytočne všetok pridelený kredit neminul. Špecifiká tohto nastavenia je možné vidieť na obr. 9.3. Nastavil som počet Kubernetes nodes na 3, čo je zároveň vyžadované minumum v GKE. Každý node vychádza z typu stroja *micro*, ktorý má vyhradený 1 virtuálny CPU a 600 MB RAM. Spolu tak mám v clusteri dostupné 3 virtuálne CPU a 1,8 GB RAM, čo je pre beh mojej aplikácie na testovanie dostačujúce. V GKE clusteri som povolil a použil Google LoadBalancer vo forme Ingress Controller. Tento Ingress Controller otvára konektivitu do Kubernetes cluster a inteligentne vyrovnáva záťaž na jednotlivé Kubernetes Nodes.



Obr. 9.2: 6 vrstviev abstrakcie v K8s. [21]



Obr. 9.3: Nastavenie Kubernetes Cluster v Google Kubernetes Engine v službe Google Cloud s vyznačenými podstatnými časťami pre cenovo efektívnu konfiguráciu.

Kapitola 10

Záver

V práci je popísané, ako som postupoval pri analýze, navrhovaní, implementácii, kontajnerizácii a nasadzovaní webovej platformy zobrazujúcej interaktívny obsah na základe aktuálnej polohy používateľa. Hlavným úspechom je vytvorenie funkčnej platformy, ktorú tvorí viac vrstiev: webová single-page aplikácia, HTTP API rozhranie pre získavanie a ukladanie dát, plugin do systému správy obsahu October, ktorým je možné dáta v aplikácii administrátorsky spravovať. Dôležitým výsledkom je, že single-page aplikácia nie je priamo závislá na konkrétnej implementácii HTTP API či podkladového systému pre správu obsahu. Do aplikácie je možné jednoducho integrovať iný konektor, ktorý možno napojiť na iné API a zdroj dát.

Aplikáciu je možné ďalej rozširovať o rôzne typy obsahu, ako sú napríklad: vstavané príspevky zo sociálnych sietí, hodnotenie vystaveného exempláru. Prípadne aj o zložitejšie funkcionality ako napríklad: prihlásenie používateľa pomocou identity na sociálnej sieti, následná integrácia do marketingový nástrojov prevádzkovateľa výstavy cez ktoré môže návštevníka ďalej kontaktovať; zdieľanie výsledkov kvízu na sociálnych sieťach; vytvorenie četovacieho robota, ktorý by poskytoval dodatočné informácie k vystavovaným exemplárom. Aplikáciu je tiež možné rozšíriť o multijazyčnosť, čiže správu a zobrazovanie rôznych jazykových verzií obsahu a rozhrania aplikácie.

Aplikáciu je tiež možné reálne použiť v praxi, stačí len pripraviť reálny obsah korešpondujúci s vystavenými exemplármi. O použitie na konkrétnej komerčnej výstave sa budem usilovať spolu so spoločnosťou Actlocate s.r.o. v blízkej budúcnosti.

Literatúra

[1] *About web applications*. Adobe Inc., [Online; navštívené 13.04.2019].
URL https://helpx.adobe.com/dreamweaver/using/web-applications.html

[2] *API Blueprint Tutorial*. Oracle Inc., [Online; navštívené 16.04.2019].
URL https://help.apiary.io/api_101/api_blueprint_tutorial/

[3] *CMS comparison 2018/2019: The most popular open source systems*. 1&1 IONOS In.c, [Online; navštívené 13.04.2019].
URL https://www.ionos.com/digitalguide/hosting/cms/cms-comparison-a-review-of-the-best-platforms/

[4] *How it works*. Grindsome, [Online; navštívené 16.04.2019].
URL https://gridsome.org/docs/how-it-works

[5] *Introduction to Containers: Concept, Pros and Cons, Orchestration, Docker, and Other Alternatives*. flow.ci, [Online; navštívené 16.04.2019].
URL https://medium.com/flow-ci/introduction-to-containers-concept-pros-and-cons-orchestration-docker-and-other-alternatives-9a2f1b61132c

[6] *Overview of Docker Compose*. Docker Inc., [Online; navštívené 16.04.2019].
URL https://docs.docker.com/compose/overview/

[7] *Real-time location system RTLS on UWB*. Sewio Networks, s.r.o., [Online; navštívené 16.04.2019].
URL https://www.sewio.net/real-time-location-system-rtls-on-uwb/

[8] *Testing: Getting Started*. Laravel, [Online; navštívené 02.05.2019].
URL https://laravel.com/docs/5.8/testing

[9] *The entire DevOps lifecycle in one application*. GitLab Inc., [Online; navštívené 16.04.2019].
URL https://about.gitlab.com/stages-devops-lifecycle/

[10] *Understanding Apiary*. Oracle Inc., [Online; navštívené 16.04.2019].
URL https://help.apiary.io/api_101/understanding-apiary/

[11] *Unit Testing*. Vue.js, [Online; navštívené 02.05.2019].
URL https://vuejs.org/v2/guide/unit-testing.html

[12] Ashwini, A.: *Should You Use NoSQL Or SQL Db Or Both?* [Online; navštívené 13.04.2019].
URL https://medium.com/swlh/should-you-use-nosql-or-sql-db-or-both-349cb26c9add

[13] Borick, S.: *I created the same app in React and Vue (Part 2: Angular).* [Online; navštívené 16.04.2019].
URL https://medium.com/javascript-in-plain-english/i-created-the-exact-same-app-in-react-and-vue-part-2-angular-39b1aa289878

[14] Brown, E.: *Learning JavaScript, 3rd edition.* O'Reilly Media, Inc., 2016, ISBN 978-1-491-91491-5.

[15] Chacon, S.: *Pro Git.* CZ.NIC, z. s. p. o., 2009, ISBN 978-80-904248-1-4.

[16] Chen, Y.; Kobayashi, H.: *Signal strength based indoor geolocation. Proceedings of the IEEE International Conference on Communication*, ročník 1, č. 2, 2002: str. 436–439.

[17] Curran, K.; Furey, E.; Lunney, T.; aj.: *An Evaluation of Indoor Location Determination Technologies. Journal of Location Based Services*, ročník 5, č. 2, 2011: s. 61–78, doi:10.1080/17489725.2011.562927.

[18] Curran, K.; Furey, E.; Lunney, T.; aj.: *Streetfight moves indoors – to map the last three feet. Geospatial World*, ročník 5, č. 1, 2014, ISSN 2277–3134.

[19] Evans, E.: *Domain-Driven Design: Tackling Complexity in the Heart of Software.* Pearson Education, 2003, ISBN 9780132181273.
URL https://books.google.cz/books?id=hHBf4YxMnWMC

[20] Gutmans, A.; Bakken, S. S.; Rethans, D.: *Mistrovství v PHP 5.* Brno: Computer Press, vyd. 2. vydání, 2007, ISBN 978-80-251-1519-0.

[21] Hale, J.: *Key Kubernetes Concepts.* [Online; navštívené 16.04.2019].
URL
https://towardsdatascience.com/key-kubernetes-concepts-62939f4bc08e

[22] Ilinčev, O.: *Dropdown – vaše poslední volba.* [Online; navštívené 16.04.2019].
URL https://www.ilincev.com/dropdown

[23] Ilinčev, O.: *Formuláře na mobilu.* [Online; navštívené 16.04.2019].
URL https://www.ilincev.com/formulare-na-mobilu

[24] Jin, B.; Sahni, S.; Shevat, A.: *Designing Web APIs – Building APIs That Developers Love.* O'Reilly Media, Inc., 2018, ISBN 978-1-492-02692-1.

[25] Macrae, C.: *Vue.js: Up and Running.* O'Reilly Media, Inc., 2018, ISBN 978-1-491-99724-6.

[26] McGee, J.: *Music City Center app guides visitors.* [Online; navštívené 16.04.2019].
URL https://eu.tennessean.com/story/money/tech/2014/11/12/music-city-center-app-guides-visitors/18945489/

[27] Michel, R.: *Information Management: Wearables come in for a refit.* [Online; navštívené 16.04.2019].
URL https://www.mmh.com/article/information_management_wearables_come_in_for_a_refit

[28] Ograca, H.: *DDD, Hexagonal, Onion, Clean, CQRS, ... How I put it all together.* [Online; navštívené 13.04.2019].
URL https://herbertograca.com/2017/11/16/explicit-architecture-01-ddd-hexagonal-onion-clean-cqrs-how-i-put-it-all-together/

[29] Zendulka, J.; Rudolfová, I.: *Databazove systemy IDS Studijni opora.* FIT VUT v Brně, 2006; rev. 5.2.2018.

[30] Zhou, Y.; Law, C. L.; Guan, Y. L.; aj.: *Indoor Elliptical Localization Based on Asynchronous UWB Range Measurement.* [Online; navštívené 16.04.2019].
URL https://ieeexplore.ieee.org/document/5466205

Príloha A

Obsah CD

- Priečinok `src` – zdrojové súbory platformy
- Priečinok `latex_src` – zdrojové súbory tejto technickej správy
- Priečinok `docs` – PDF s technickou správou a dokumentácia

Príloha B

Návod na inštaláciu – lokálny vývoj

B.1 Backend a REST API

Pred spustením projektu je potrebné mať v počítači nainštalovaný Docker CE. Projekt bol testovaný s Docker Engine v18.09.2.[1]

1. V termináli v priečinku `docker` spustite príkaz `docker-compose up -d` a počkajte na jeho dokončenie.
2. Po jeho skončení spustite príkaz `docker-compose exec workspace bash`, čím sa dostanete do terminálu spusteného docker kontajneru.
3. Najskôr v priečinku `backend` spustite príkaz `composer install`, potom odkopírujte predpripravený konfiguračný súbor `cp env.example .env` a spustite príkaz `php artisan october:up`.
4. Ďalej v priečinku `api` spustite príkaz `composer install` a potom odkopírujte predpripravený konfiguračný súbor `cp env.example .env`.
5. Backend je prístupný pod adresou `http://localhost:8080/backend`. Predvolené prístupové meno je `admin` a heslo `admin`.
6. API je prístupné pod adresou `http://localhost:8081`.

B.2 Frontend SPA

Pred spustením je potrebné mať v počítači nainštalovaný Node a NPM. Projekt bol testovaný s Node v11.3.0 a NPM v6.4.1.[2]

1. V termináli v priečinku `frontend` spustite príkaz `npm install`.
2. V súbore `.env` skontrolujte nastavenia adries URL pre API.

[1] Docker CE na stiahnutie pre rôzne OS – https://hub.docker.com/search/?type=edition&offering=community
[2] Node.js na stiahnutie pre rôzne OS – https://nodejs.org/en/

3. Potom príkazom `npm run serve` spustite vývojovú verziu frontend SPA. Vo výstupe príkazu nájdete adresu URL, ktorú môžete otvoriť v prehliadači a pristúpiť tak k projektu.

Príloha C

Detaily vybraných prípadov použitia

C.1 Prihlásiť sa do SPA tagom

ID: UC1
Účastníci: Používateľ

C.1.1 Vstupné podmienky

1. Používateľ má pri sebe tag.
2. Používateľ má otvorenú SPA v prehliadači svojho mobilu/tabletu.

C.1.2 Tok udalostí

1. Prípad použitia začína tým, že používateľ vyhľadá kód tagu, ktorý je uvedený na nálepke tagu.
2. Používateľ zadá kód tagu do formulára.
3. **KEĎ** používateľ chce zadať dodatočné údaje
   - 3.1. Používateľ môže zadať svoje celé meno.
   - 3.2. Používateľ môže zadať svoju emailovú adresu.
   - 3.3. Používateľ musí zaškrnúť checkbox, že súhlasí so spracovaním osobných údajov, pokiaľ zadal informácie v kroku 3.
4. Používateľ odošle formulár.

C.1.3 Následné podmienky

1. SPA príjme formulár, odošle informácie na API a započne novú session.
2. SPA presmeruje používateľa na ďalšiu obrazovku.

C.2 Automatické a manuálne zobrazenie obsahu

ID: UC2
Účastníci: Používateľ

C.2.1 Vstupné podmienky

1. Používateľ je prihlásený pomocou kódu tagu.

C.2.2 Tok udalostí

1. Prípad použitia začína tým, že SPA načíta do úložiska v prehliadači všetky potrebné dáta.
2. **KEĎ** sa používateľ dostane do blízkosti exponátu:
   2.1. SPA zobrazí používateľovi obsah prislúchajúci k danému exponátu.
   2.2. Používateľ prehliada obsah.
   2.3. **KEĎ** sa používateľ rozhodne vyplniť kvíz -> Prípad použitia: C.4.
   2.4. **KEĎ** sa používateľ rozhodne vyplniť anketu -> Prípad použitia: C.3.
3. **KEĎ** používateľ vyberie položku z menu:
   3.1. SPA zobrazí používateľovi vybraný obsah a ignoruje aktuálnu polohu používateľa, kým používateľ funkciu nedeaktivuje.
   3.2. Používateľ prehliada obsah.
   3.3. **KEĎ** sa používateľ rozhodne vyplniť kvíz -> Prípad použitia: C.4.
   3.4. **KEĎ** sa používateľ rozhodne vyplniť anketu -> Prípad použitia: C.3.
   3.5. Používateľ môže deaktivovať funkciu zafixovanie na vybranom obsahu, čím sa zobrazí obsah na základe aktuálnej polohy používateľa.

C.2.3 Následné podmienky

Nie sú uvedené.

C.3 Vyplniť anketu

ID: UC3
Účastníci: Používateľ

C.3.1 Vstupné podmienky

1. Používateľ je prihlásený pomocou kódu tagu.
2. Používateľ má zobrazený obsah, v ktorom je možnosť vyplniť anketu.

C.3.2 Tok udalostí

1. Používateľ odpovedá na anketové otázky.
2. **KEĎ** sa jedná o anketovú otázku s výberom jednej možnosti:
   2.1. Používateľ vyberie jednu z ponúkaných možností.
   2.2. SPA automaticky zaznamená voľbu používateľa.
   2.3. SPA ponúkne používateľovi ďalšiu anketovú otázku.
3. **KEĎ** sa jedná o anketovú otázku s výberom viacerých možnosti:
   3.1. Používateľ vyberie 1 až X z ponúkaných možností.
   3.2. SPA automaticky zaznamená voľbu používateľa.
   3.3. SPA ponúkne používateľovi ďalšiu anketovú otázku.

C.3.3 Následné podmienky

1. SPA odošle výber používateľa do API.

C.4 Vyplniť Kvíz

ID: UC4
Účastníci: Používateľ

C.4.1 Vstupné podmienky

1. Používateľ je prihlásený pomocou kódu tagu.
2. Používateľ má zobrazený obsah, v ktorom je možnosť vyplniť kvíz.

C.4.2 Tok udalostí

1. Používateľ odpovedá na kvízové otázky.
2. Používateľ vyberie jednu zo štyroch ponúkaných možností.
3. **AK** je odpoveď na otázku správna: SPA používateľovi zobrazí hlášku, že odpovedal správne.
4. **AK** je odpoveď na otázku nesprávna: SPA používateľovi zobrazí hlášku, že odpovedal nesprávne.
5. SPA automaticky zaznamená voľbu používateľa.
6. SPA ponúkne používateľovi ďalšiu kvízovú otázku.

C.4.3 Následné podmienky

1. SPA odošle odpovede používateľa do API.