VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SYSTÉM PRO VYUŽITÍ TECHNOLOGIE ROZŠÍŘENÉ REALITY V MUZEÍCH A GALERIÍCH

SYSTEM FOR AUGMENTED REALITY UTILISATION IN MUSEUMS AND GALLERIES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. FREDERIK MÜLLER**
AUTHOR

VEDOUCÍ PRÁCE **Ing. VÍTĚZSLAV BERAN, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2019/2020

Zadání diplomové práce

Student: **Müller Frederik, Bc.**
Program: Informační technologie Obor: Bezpečnost informačních technologií
Název: **Systém pro využití technologie rozšířené reality v muzeích a galeriích**
System for Augmented Reality Utilisation in Museums and Galleries
Kategorie: Uživatelská rozhraní

Zadání:

1. Prostudujte možnosti technologií pro rozšířenou realitu, zaměřte se na tvorbu, interakci a vizualizaci multimediálních popisků rozmístěných ve scéně na základě detekce 3D objektů.
2. Navrhněte systém, který umožní návštěvníkům výstav zobrazovat multimediální obsah s využitím rozšířené reality. Systém musí umožňovat vytvářet, spravovat, distribuovat a zobrazovat multimediální obsah, včetně správy 3D objektů.
3. Implementujte navržený systém s využitím relevantních dostupných technologií. Zaměřte se na efektivitu uživatelské interakce.
4. Vyhodnoťte vlastnosti výsledného systému na základě experimentů v reálném prostředí.
5. Prezentujte klíčové vlastnosti řešení formou plakátu a krátkého videa.

Literatura:

- Dieter Schmalstieg, Tobias Hollerer. *Augmented Reality: Principles and Practice.* Addison-Wesley, 2016. ISBN: 978-0321883575.
- Russ Unger, Carolyn Chandler. A Project Guide to UX Design: For user experience designers in the field or in the making. New Riders, 2012. ISBN: 0132931729.
- Dále dle pokynu vedoucího.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Beran Vítězslav, Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 20. května 2020
Datum schválení: 10. března 2020

Abstrakt

Cieľom diplomovej práce je poskytnúť návštevníkom rôznych typov objektov – typicky galérií, múzeií apod. doplnkové informácie o vystavovaných objektoch s dôrazom na zobrazenie v rozšírenej realite. Súčasťou je analýza už existujúcich riešení, návrh a implementácia celého systému, potrebného na nasadenie do prevádzky. Systém z pohľadu tvorcu obsahu, respektívne administrátora, predstavuje komplexné riešenie umožňujúce tvorbu interaktívnej prechádzky – skenovanie 3D objektov, snímanie 2D objektov a pridávanie obsahu k daným objektom. Na opačnej strane, nástroj určený pre návštevníkov, teda užívateľov, ktorí sú súčasťou interaktívnej prechádzky poskytuje zobrazenie doplnkových informácií o objektoch primárne nachádzajúcich sa v blízkosti užívateľa, ktoré sú automaticky detekované objektívom mobilného telefónu. Súčasťou riešenia je využitie rozšírenej reality, ktorá je implementovaná pomocou technológie *ARKit*, takže výsledná aplikácia je postavená na platforme *iOS*. Práca rieši problematiku detekcie 3D objektov a ich následné rozpoznanie, spolu so spôsobom, ako s týmito informáciami pracovať, ukladať a následne využiť pre potreby tejto práce. Vo výslednom riešení sa kladie dôraz na jednoduchosť ovládania (vodiace značky, nápovedy. . .), resp. celkový uživateľský zážitok.

Abstract

The aim of the master thesis is to provide visitors of various types of objects – typically galleries, museums, etc. with additional information about exhibited objects with emphasis on visual display in augmented reality. It includes an analysis of already existing solutions, design and implementation of the entire system needed for deployment. From the point of view of the content creator (administrator), the system represents a complex solution enabling the creation of an interactive walk – scanning 3D objects, scanning 2D objects and adding content to the given objects. On the other hand, the tool for visitors (users) who are part of an interactive walk, provides additional information about objects primarily located near the user, which are automatically detected by the camera on the mobile phone. Solution includes augmented reality, which is implemented using *ARKit* technology, so the final application is built on the *iOS* platform. The work addresses the issue of detection of 3D objects and their subsequent recognition, along with the way to work with this information, how to store it and then use it for purposes of this thesis. In the final solution, emphasis is placed on the simplicity of the usage (guide marks, hints. . .) and overall user experience.

Kľúčové slová

Aplikácia pre múzeá a galérie, rozšírená realita, 3D objekty, vizualizácia, rozpoznanie objektov, skenovanie objektov, ARKit, iOS, Swift, React, Express.js

Keywords

Museum & Gallery application, augmented reality, 3D objects, visualisation, object recognition, object scanning, ARKit, iOS, Swift, React, Express.js

Citácia

MÜLLER, Frederik. *Systém pro využití technologie rozšířené reality v muzeích a galeriích.* Brno, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vítězslav Beran, Ph.D.

Systém pro využití technologie rozšířené reality v muzeích a galeriích

Prehlásenie

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

.....
Frederik Müller
2. júna 2020

Poďakovanie

Ďakujem pánovi Ing. Vítězslavovi Beranovi, Ph.D. za vedenie diplomovej práce, odbornú pomoc a cenné rady pri vypracovaní práce. Ďalej patrí vďaka mojej rodine, kolegom a priateľom, ktorí mi dodávali potrebnú motiváciu a podporu.

Obsah

Kapitola 1

Úvod

Rozšírá realita sa stala v posledných rokoch diskutovanou témou. Významný podiel na šírení využitia rozšírenej reality veľkým masám užívateľov má spoločnosť *Apple*, ktorá v roku 2017 predstavila *framework ARKit* – sadu softvérových nástrojov a knižníc uľahčujúcich vývojárom mobilných aplikácií priniesť zážitok z rozšírenej reality užívateľom priamo do ich mobilných telefónov. Tento zážitok z rozšírenej reality je dostupný širokej verejnosti, nakoľko *Apple* má dnes prvenstvo v predajnosti mobilných telefónov [2]. Iba 6 mesiacov po sprístupnení rozšírenej reality do *Apple* zariadení bolo stiahnutých až 13 miliónov aplikácií, ktoré využívali rozšírenú realitu [14]. Z toho vyplýva, že užívatelia sa nových možností neboja, sú otvorení novým technológiám a priestor na trhu mobilných aplikácií je tak v tomto ešte nepreskúmanom a veľkými hráčmi neobsadenom segmente obrovský.

Cieľom tejto diplomovej práce je poskytnúť návštevníkom špecifických objektov ako sú napríklad galérie či múzea multimediálny obsah – doplnkové informácie o exponátoch, ktoré sú súčasťou týchto objektov. Návštevníkovi takýto systém umožní po namierení telefónu na exponát automaticky s využitím mobilnej aplikácie tento objekt rozoznať a v okolí tohto objektu zobrazí v rozšírenej realite videá, obrázky a iný dostupný obsah. Tento obsah je možné vytvárať a spravovať pomocou gest, rovnako ako pri návštevníkovi, v rozšírenej realite. V prvom rade je pre dosiahnutie výsledku potrebné na základe zanalyzových existujúcich riešení navrhnúť a implementovať všetky časti potrebné na nasadenie systému, ktorý poskytuje definované funkcionality. Na systém možno nahliadať z pohľadu návštevníka a z pohľadu administrátora. Z pohľadu návštevníka sa jedná o mobilnú aplikáciu, ktorá slúži ako interaktívna prechádzka, typicky priestormi galérie, múzea či iných priestorov, v ktorých sa naskytá príležitosť objavovania užívateľmi nepoznaných objektov, respektívne ich vlastností či príbehov. Z pohľadu administrátora sa jedná o webovú a mobilnú aplikáciu, ktorá slúži na zadávanie základných informácií o mieste, skenovanie a snímanie objektov a následné pridávanie informácií k objektom. Obe aplikácie pre svoju funkčnosť potrebujú online získavať informácie o vystavovaných objektoch. K tomuto účelu je potrebné vytvoreniť server, ktorý poskytuje informácie uložené v databáze a prípadne verifikuje administrátora.

Neoddeliteľnou súčasťou navrhnutej aplikácie je automatická detekcia 3D objektov, skenovanie a ich uloženie do vhodnej štruktúry pre možnosť následného rozoznania týchto objektov v priestore. Práca sa okrem iného venuje konkrétnemu riešeniu spracovania mračna bodov a detekcií objektov. Rovnako dôležitou časťou je zoznámenie sa s návrhom a praktikami pri tvorbe užívateľského rozhrania pre dosiahnutie intuitívneho ovládania aplikácie. V skutočnosti sa teda jedná o niekoľko častí, ktoré vzájomne komunikujú a tvoria tak jeden komplexný systém. Práca sa venuje popisu architektúry systému, jeho vývoju, behu

a následnej údržbe. Takáto architektúra musí byť odolná voči chybám a musí umožňovať škálovateľnosť, poprípadne zmenu jednotlivých subsystémov.

Predkladaná diplomová práca pozostáva zo šiestich kapitol. Teoretický základ počítačového videnia, algoritmy používané pri práci s rozšírenou realitou a spôsoby využitia sú popísane v kapitole 2. Nasleduje kapitola 3, ktorá obsahuje nutný základ pre vývoj informačných systémov, konkrétne pozostávajúcich z webových a mobilných aplikácií na platformách *React*, respektívne *iOS* a popis kľúčových technológií, ktoré boli použité pri riešení práce. Kapitola 4 vymedzuje už existujúce podobné aplikácie na mobilnom trhu a obsahuje návrh aplikácií, špecifikuje požiadavky a definuje ciele týchto aplikácie. Ďalej obsahuje návrh celého systému a zameriava sa na spôsob uloženia dát, ktoré umožňujú využiť rozšírenú realitu pre dosiahnutie navrhnutých cieľov. V kapitole 5 je popísaný postup realizácie, implementácia využitých algoritmov, ktorých výsledkom je systém ako celok. Na konci tejto kapitoly sa nachádza zhodnotenie výsledného systému na základe vykonaného testovania a experimentov. Posledná kapitola 6 obsahuje záver a teda zhodnotenie práce s možným vylepšeniami do budúcnosti pre možnosť nasadenia aplikácie na trh do rúk bežných užívateľov.

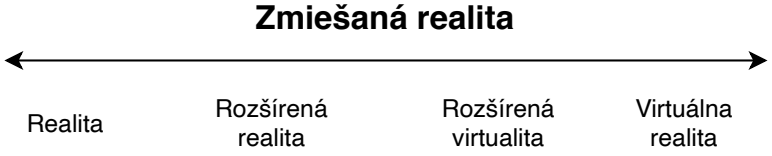
Kapitola 2

Rozšírená realita

Aby bolo možné navrhnúť a implementovať riešenie, ktoré využíva rozšírenú realitu, je potrebné sa predovšetkým oboznámiť s týmto termínom a existujúcimi aplikáciami, ktoré ju využívajú. Cieľom tejto kapitoly je poskytnúť odpovede na základné otázky, a to napr. aké ďalšie typy realít existujú, aký je medzi nimi rozdiel, aké sú súčasné limity a možnosti pri práci s rozšírenou realitou. Konkrétne je popísaný spôsob lokalizácie a mapovanie prostredia, zoznámenie sa so základnými pojmami a algoritmami ako sú významné body, systém SLAM a VIO. Nasleduje rozbor vizualizácie objektov v scéne s možnosťami *frameworkov SceneKit* a *Unity*. V predposlednej časti sa nachádza popis pravidiel pri tvorbe užívateľského rozhrania aplikácií, ktoré využívajú rozšírenú realitu. Posledná časť sa venuje *frameworku ARKit* pre potreby aplikácie na vizualizáciu 3D objektov a multimediálneho obsahu vo všeobecnosti.

2.1 Rozšírená realita

Rozšírená realita (*Augmented reality* – AR), virtúalna realita (*Virtual reality* – VR), zmiešaná realita (*Mixed reality* – MR). To sú pojmy, ktoré sa často medzi sebou nesprávne zamieňajú. Preto je vhodné si najskôr význam týchto termínov objasniť, respektívne poukázať na ich rozdiely. Virtuálna realita umiestňuje užívateľa do počítačom generovaného prostredia. Rozšírená realita poskytuje informácie, ktoré sú priamo vložené do existujúceho fyzického prostredia. Rozšírená realita tak zmazáva priepasť medzi virtuálnym a skutočným svetom. V rozšírenej realite sa digitálne informácie stávajú súčasťou reálneho sveta, prinajmenšom z pohľadu používateľa. Existujú však aj iné definície rozšírenej reality, kde napríklad rozšírená realita musí spĺňať tieto 3 vlastnosti – kombináciu reality a virtuality, zobrazenie v reálnom čase a registrovanie 3D okolia [21]. Pojem zmiešaná realita popisuje časť kontinua medzi virtuálnym a reálnym prostredím. Z obrázku 2.1 znázorňujúci vzťah medzi týmito pojmami vyplýva, že zmiešaná realita je všeobecný pojem zahŕňajúci všetky možné kombinácie reálneho sveta a virtuálnej reality. Rozšírena realita a virtuálna realita je teda podmnožinou zmiešanej reality.



Obr. 2.1: Kontinuum zmiešanej reality [13]

Kontinuum zmiešanej reality pozostáva z nasledujúcich častí:

- **Realita**. Reálne prostredie bez počítačom generovaných prvkov.
- **Rozšírená realita**. Reálne prostredie, v ktorom sa nachádzajú počítačom generované prvky.
- **Rozšírená virtualita**. Počítačom generované prostredie, v ktorom sa nachádzajú prvky z reálneho prostredia.
- **Virtuálna realita**. Počítačom generované prostredie bez prvkov z reálneho prostredia.

Pre prvé zmienky o začiatkoch rozšírenej reality netreba ísť ďaleko do minulosti. V 60. rokoch 20. storočia Ivan Sutherland vytvoril prvý displej, ktorý sa nasadzoval na hlavu (*head-mounted display* – HMD) a umožnoval zobraziť objekty do scény. Jednalo sa však o prístroj, ktorý bol ťažký a neprenosný. Ďalšie posuny nastali v 80. a 90. rokoch, kde vznikol práve pojem rozšírená realita a postupne sa vyvinul do formy, v akej ju dnes poznáme.

Rozšírená realita v mobilných funguje na nasledujúcom princípe. Reálne prostredie je snímané pomocou kamery, ktorá naživo prenáša snímaný obraz na obrazovku mobilného telefónu. Tento obraz je doplnený o digitálny obsah. Druhou možnosťou ako poskytnúť rozšírenú realitu je pomocou priehľadného displeja, na ktorý sa premieta počítačom generovaný doplnkový obsah obohacujúci reálne prostredie. Tento spôsob poskytuje užívateľovi prirodzenejší spôsob zobrazenia počítačom generovaného obsahu, keďže reálne prostredie je priamo bez nutnosti prostredníka obohatené o digitálne informácie, avšak z technických dôvodov tento spôsob zatiaľ nie je možné uplatniť s použitím mobilného telefónu.

Mnohí z nás sa stretli s rozšírenou realitou, hoci si to možno ani neuvedomujú. Typickým príkladom (obrázok 2.2) použitia AR je použitie v aute. Prvým príkladom je zadná parkovacia kamera, kde je digitálne znázornená vzdialenosť od okolitých objektov a prípadné vytočenie kolies pridaním vrstvy cez obraz. Tento obraz je v najnovších modeloch doplnený o takzvanú 360 stupňovú kameru, ktorá zobrazuje automobil spolu s okolím z vtáčej perspektívy. Ďalsím použitím v aute je *head-up display* (HUD), ktorý premieta na obrazovku dôležité informácie o jazde (typicky aktuálnu rýchlosť, nasnímané značky či navigáciu). Posledným príkladom je pokročilá navigácia, ktorá používa kameru snímajúcu dianie pred vizodlom a dokáže tak vložiť navigačné tabule priamo do obrazu. Okrem použitia v aute je taktiež časté použitie v televízii, kde sa pri rôznych spravidla športovných zápasoch analyzuje priebeh hry napríklad pomocou pera, ktoré kreslí do obrazu, poprípadne automaticky generovaných šťastík jednotlivých hráčov (ich pohyb, rýchlosť a ďalšie).

Obr. 2.2: Rozšírená realita v aute[1]

[1] https://mercedes-benz.com

Dnes, keď je rozšírená realita dostupná prostredníctvom telefónov a tabletov, ktoré sú denne používané miliardami ľudí na svete, vznikla možnosť využívať rozšírenú realitu pri riešeniach bežných problémov. Stala sa tak súčasťou množstva sektorov – od vzdelávania, cez zdravotníctvo až po priemysel.

Aplikácia *Google Translate*[2] (obrázok 2.3) slúži ako prekladač z jedného jazyka do druhého. Obsahuje jednu AR funkciu, ktorá je užitočná pri preklade textu. Stačí namieriť telefón na text a aplikácia preloží text na fotografii v reálnom čase. Ak je aplikácia pripojená k internetu, podporuje veľké množstvo jazykov, ale používatelia si môžu tiež vybrať z niekoľkých dostupných jazykových balíkov, ak chcú pokračovať v používaní funkcie okamžitého prekladu v režime bez použitia internet. Táto funkcionalita je obzvlášť užitočná v prípade návštevy krajiny s jazykom, ktorým nielenže užívateľ nehovorí, ale nie je ani schopný prepísať videné znaky (napr. pri návšteve Číny).

Obr. 2.3: Aplikácia *Google Translate*[2]

Obr. 2.4: Aplikácia *Measure*[3]

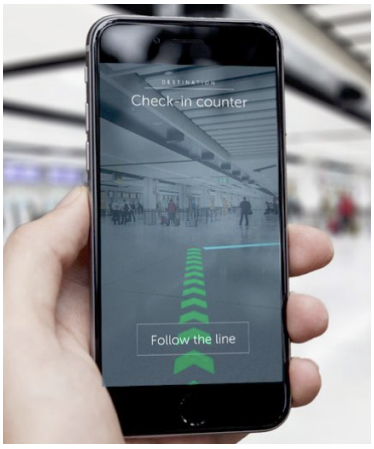
Aplikácia *Measure*[3] (obrázok 2.4) vyvinutá priamo spoločnsťou *Apple* a dostupná od *iOS 12* slúži na meranie objektov. Po namierení na objekt, respektívne bod od ktorého užívateľ chce spustiť meranie, stlačí tlačidlo "+", posunie telefónom na koniec objektu, ktorý meria a následne znovu stlačí tlačidlo pre ukončenie merania. Aplikácia vizualizuje dĺžku zmeraného objektu spojením začiatočného bodu a koncové bodu bielym pásom. Aplikácia dokáže rozoznať niektoré objekty ako je papier a automaticky poskytnúť informácie o rozmeroch rozoznaného objektu.

Aplikácia *Pokémon GO*[4] (obrázok 4) sa stala v posledných rokoch obrovským hitom. Táto hra si získala popularitu desiatok miliónov hráčov a zaradila sa tak na popredné miesta v rebríčkoch popularity [10]. Rozšírená realita spočíva v chytení pokémona. Na obrazovke mobilu sa v reálnom prostredí zobrazí pokémon spolu s krabičkou, do ktorej sa má chytiť pokémon. Užívateľ potiahnutím prsta hádže túto krabičku a chytá tak pokémona. Aplikácia okrem rozšírenej reality používa údaje z globálneho polohového systémy (*Global Positioning System* – GPS), vďaka čomu hráča zobrazuje na mape a umocňuje tak pocit reality.

[2] https://apps.apple.com/us/app/google-translate/id414706506
[3] https://apps.apple.com/us/app/measure/id1383426740
[4] https://apps.apple.com/us/app/pokémon-go/id1094591345

Aplikácia *Gatwick Airport*[5] (obrázok 2.6) získala ocenenia (mobilná inovácia roka na *National Technology awards* a mobilná aplikácia roka na *Real IT awards*) za kreatívne využitie rozšírenej reality. Viac ako dve tisír *beaconov* je umiestnených na letisku v dvoch termináloch, nakoľko v tomto prípade je dôležitá presnosť polohy a GPS takúto presnosť v rámci budovi neposkytuje. Cestujúci môžu použiť rozšírenú realitu na navigáciu v rámci letiska. Užívateľ iba namieri telefónom pred seba a na dlážke sa objavia šípky navigujúce k zvolenému cieľu. Letisko plánuje aj vďaka aplikácii poskytnúť užívateľom bezproblémové cestovanie.

Obr. 2.5: Aplikácia *Pokémon GO*[4]



Obr. 2.6: Aplikácia *Gatwick Airport Official*[5]

2.2 Lokalizácia a mapovanie prostredia

V rámci rozšírenej reality hovoríme o troch dôležitých aspektoch, ktoré su nevyhnutné pre správne fungovanie (obrázok 2.7):

- **Kalibrácia**. Proces porovnávania výsledkov dosiahnutých referenčným a kalibrovačným zariadením. Známa referenčná hodnota môže nahradzovať referenčné zariadenie.
- **Registrácia**. V tomto kontexte sa jedná o pojem, kde dochádza k zarovnaniu koordinačného systému reálneho a virtuálneho sveta.
- **Stopovanie**. Tejto pojem popisuje dynamické snímanie a kalkuláciu systému rozšírenej reality. Orientácia a pozícia displeja musí byť čo najpresnejšia, aby bolo možné hodnoverne zobraziť virtuálne objekty do reálneho sveta. [21]

Po spustení aplikácie s rozšírenou realitou, systém o prostredí veľa nevie, preto okamžite začne spracovávať dáta z rozličných senzorov. Kamera sa používa ako hlavný senzor, ale pre zlepšenie presnosti sa kombinujú dáta z ďalších senzorov, ako napríklad akcelerometer či gyroskop. Na základe týchto dát sa systém snaží vytvoriť mapu prostredia a umiestniť zariadenie do zmapovaného prostredia. V prípade, že by sme mali k dispozícii neobmedzený počet tzv. *beaconov* – čiže „majákov“ so známou polohou, ktoré využívajú technológiu *Bluetooth 4.0*, ktorá na základe sily signálu a triangulácii vzdialeností medzi jednotlivými

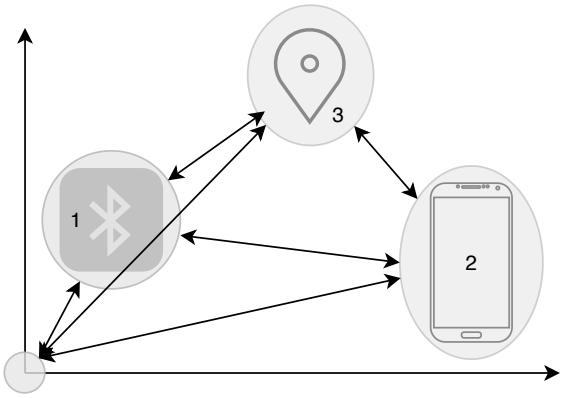
[5] https://apps.apple.com/gb/app/gatwick-airport-official/id1247492685

Obr. 2.7: Vzťah medzi kalibráciou, stopovaním a registráciou [21]

zdrojmi dokážu určiť polohu zariadenia. V iných prípadoch by mohlo stačiť GPS, ale pre potrebu rozšírenej reality potrebujeme použiť iný princíp, nakoľko GPS má v budovách slabú presnosť a *beacony* tiež nie sú všade prístupné.

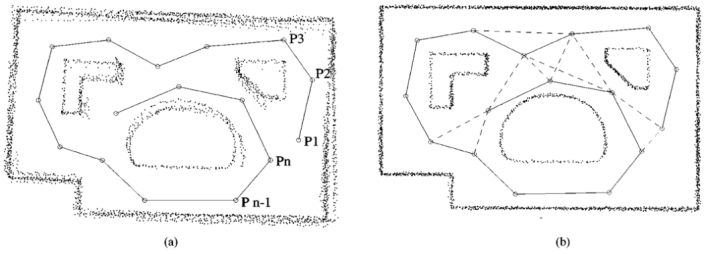
Autori [22] popisujú vytvorenie stochastickej mapy pre neurčité priestorové vzťahy. Pozrime sa na obrázok 2.8, ktorý ilustruje pozíciu *beaconu* (1) a pozíciu mobilu (2). Ak je úlohou presunúť mobil do bodu 3, vieme vypočítať o koľko a akým smerom je potrebné hýbať mobilom – avšak len za ideálnych podmienok, kedy je známa presná lokácia mobilu a presná lokácia *beaconu*. Avšak toto nie je možné v prípade, kde vôbec nepoznáme prostredie. Tento problém sa označuje ako simultánna lokalizácia a mapovanie (*Simultaneous localization and mapping* – SLAM), kde sa vytvára a aktualizuje mapa neznámeho prostredia a kde sa počíta s nepresnosťami. V prípade SLAMu majú všetky body navzájom vytvorené vzťahy. Výsledkom je rozdelenie pravdepodobností polohy každého bodu, kde niektoré body majú vyššiu presnosť a niektoré výrazne menšiu. *Extended Kalman Filter* (EKF), *Maximum a Posteriori estimation* (MAP) alebo *Bundle Adjustment* (BA) sú algoritmy používané pre výpočet pozícií na základe nepresných polôh. Napríklad *ARKit* používa *Kalman Filter*, ktorý používa dáta z kamery (*Visual system*) spolu s dátami z akcelerometru a gyroskopu (*Inertial system – Inertial Measurement Unit* alebo IMU). *Kalman Filter* rozhodne, ktorý systém podáva lepšie výsledky a tieto použije [12]. Pretože pozície bodov sú navzájom na sebe závislé (majú medzi sebou vytvorené vzťahy), každým príchodom nových dát zo senzorov sa prepočítava celá mapa. Tento prepočet mapy je pochopiteľne náročná operácia, ktorá vyžaduje množstvo matematických operácií.

Nakoľko žiadny senzor nie je dokonalý, pridáva do výpočtov nepresnosti. Tieto nepresnosti vznikajú pri obrazoch zachytených fotoaparátom i pri použití akcelerometra, gyroskopu apod. To je jedna z kľúčových vlastností algoritmu SLAM, ktorý vie s týmito nepres-



Obr. 2.8: Neurčité priestorové vzťahy medzi objektami

nosťami pracovať. Tento problém znázorňuje obrázok 2.9. Na ľavom obrázku (a) pri pohybe z bodu $P_1..P_n$ pozorujeme akumuláciu chýb v priebehu času. Na pravom obrázku (b) je zobrazené potrebné neustále prepočítavanie a zarovnanie tejto mapy. Jedným z riešení je zachovanie všetkých nasnímaných dát a vzťahov medzi týmito bodmi [9]. Hlavným problémom je riešenie problému uloženia týchto záznamov. Pamäťovo a výkonovo je nemožné ukladať a počítať so všetkými nasnímanými dátami. Algoritmus preto redukuje uloženú históriu dát a používa princíp, kde pri výpočtoch používa iba dáta, ktoré vykazujú najväčšiu presnosť a ostatné dáta maže.

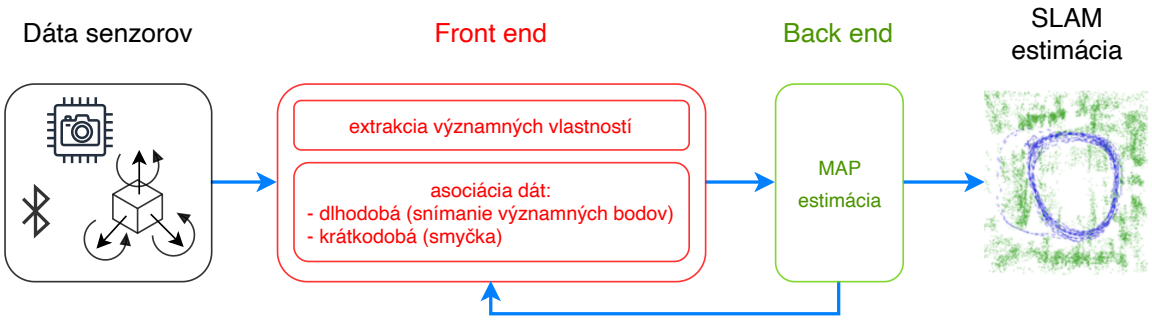


Obr. 2.9: Zarovnanie nasnímanej mapy z nepresných senzorov [9]

Ako znázorňuje 2.10, systém SLAM sa skladá z nasledujúcich štyroch častí [3]:

- `Dáta senzorov.` GPS, hĺbkové senzory, *beacon* apod. V mobilných zariadeniach sa však jedná hlavne o kameru, gyroskop a akcelerometer.
- `Front end.` Prvým krokom je extrakcia významných vlastností, z ktorých sú vytvorené body v 3D priestore. Tieto body sú sledovateľné počas pohybu, čiže medzi jednotlivými sekvenciami obrazov. Presnost týchto bodov sa s ich opakovaným výskytom zvyšuje.
- `Back end.` Táto časť sa stará o vytváranie vzťahov medzi sekvenciami obrazov, lokalizáciu kamery v priestore a celkovú tvorbu mapy priestoru. Pozorujeme 2 prístupy, kde jeden prístup algoritmov vytvára husté 3D mračná bodov prostredia, a druhý, ktorý vytvára riedku rekonštrukciu založenú na významných bodoch.

- `SLAM estimácia.` Časť obsahujúca nasnímané vlastnosti, ich lokáciu, vzťahy aj pozíciu kamery v snímanom priestore.

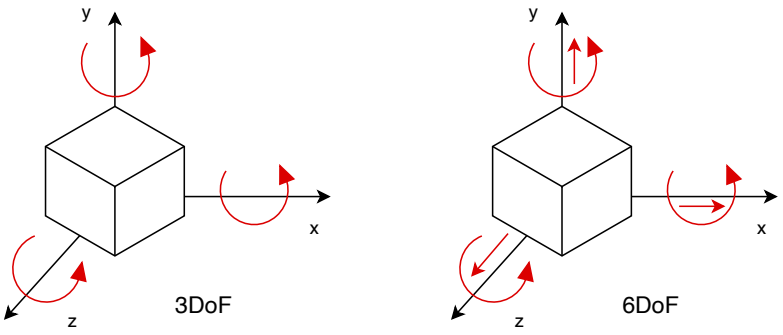


Obr. 2.10: Schéma SLAM [3]

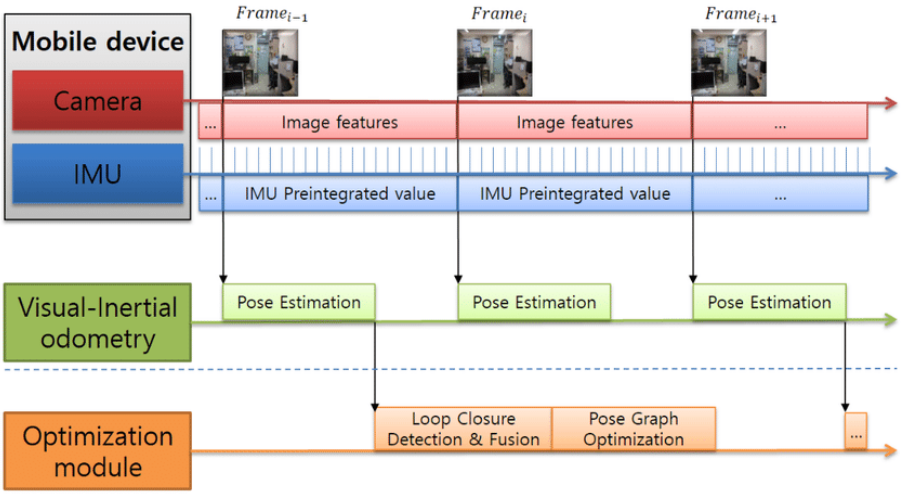
Ďalší často používaný spôsob mapovania je systém nazývany vizuálna zotrvačná odometria (*Visual Inertial Odometry* – VIO). Jedná sa o kontinuálne stopovanie 6DoF (šesť stupňov slobody, anglicky *six degrees of freedom*) kamery k začiatočnej polohe. Rozdiel medzi 6DoF a 3DoF je v počte možných pohybov. Zatial čo 3DoF poskytuje iba rotácie, 6DoF poskytuje aj posun okolo všetkých 3 osí. Tento rozdiel je znázornený na obrázku 2.11. Podstatnou vlastnosťou VIO je, že sa jedná o inkrementálne, resp. zotrvačné snímanie prostredia. Vo vzťahu k SLAM sa líši v spôsobe propagácie chyby a relokácie. VIO je tak možné označiť za jednu z kľúčových častí SLAMu. Proces VIO sa skladá z nasledujúcich krokov (obrázok 2.12):

1. Získanie vstupných obrazov z jednej alebo viacerých kamier.
2. Korekcia obrazu aplikovaním techník spracovania obrazu pre skvalitnenie dosahovaných výsledkov.
3. Použitie korelácie na zistenie zhody dvoch obrázkov bez dlhodobého sledovania.
4. Extracia významných bodov (*feature points*).
5. Konštrukcia pola optického toku prostredníctvom *Lucas-Kanade metódy*.
6. Kontrola vektorov pola optického toku z dôvodu vzniku potencionálnych chýb a odstránenie odľahlých hodnôt.
7. Odhad pohybu kamery z optického toku aplikovaním *Kalman filtru*.
8. Periodická populácia nasnímaných bodov pre održanie pokrytia celého obrazu.

Extrakcia a uloženie významných bodov je dôležitý proces pri snímaní prostredia. Vo všeobecni sa za významný bod považuje bod, ktorý je výrazný – napríklad rohy. Samotná informácia by nestačila na odlíšenie jedného body od druhého a nebolo by možné spoľahlivo opätovne nájst tento bod a preto je analyzované celé okolie. Najdôležitejšou vlastnosťou je spolahlivosť. Algoritmus musí nájsť tento bod za rôznych podmienok – zmena svetla, rotácie, zaostrenia, priblíženia atď. Zároveň musí byť tento algoritmus dostatočne rýchly, aby bolo možné pracovať s obrazom v reálnom čase. Pre mobilné zariadenia sa počet vhodných algoritmus výrazne znižuje práve kvôli spomenutej rýchlosti spracovania.



Obr. 2.11: Porovnanie 6DoF a 3DoF

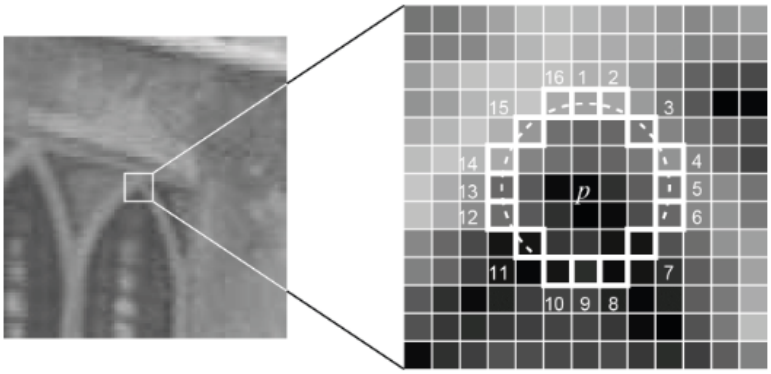


Obr. 2.12: Proces VIO [17]. *Poznámka: IMU je inerciálna meracia jednotka, ktorá slúži na detekciu lineárneho zrýchlenia. Väčšinov sa skladá z niekoľkých akcelerometerov, gyroskopov a magnetometerov.*

Detekcia významného bodu je viackrokový proces, ktorý sa líši od použitého algoritmu. Detekovaný bod je uložený prostredníctvo kľúčového bodu (*Keypoint*). Väčšina z týchto algoritmov nie je verejných, ale napríklad algoritmus BRISK (*Binary Robust Invariant Scalable Keypoints*)[8] je volne prístupný. Postavený je na algoritme FAST (*Features from Accelerated Segment Test*) [20]. FAST analyzuje kruhové okolie každého pixelu p a funguje na princípe zistenia, či je jas okolitého pixelu nižší/vyšší ako p. Ak určitý počet susedných pixelov spĺňa toto kritérium, algoritmus našiel roh. To je naznačené bodkovanou čiarou na obrázku 2.13. Pixely 7 až 10 sú tmavšie ako p a ostatné sú svetlejšie.

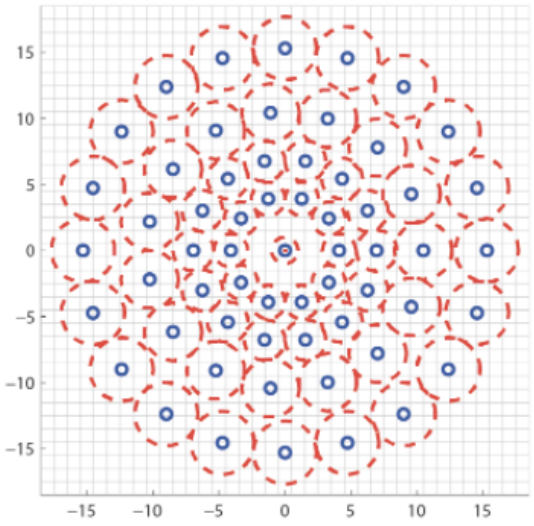
Algoritmus BRISK vyžaduje najmenej 9 po sebe nasledujúcich pixelov v 16-pixelovom kruhu dostatočne jasnejších alebo tmavších ako stredný pixel. BRISK okrem toho používa aj obrazy zmenšenej veľkosti, aby sa dosiahla lepšia spoľahlivosť – dokonca dosahuje presnosti na úrovni sub-pixelov.

Deskriptor bodu BRISK je binárny reťazec veľkosti 512 bitov. V podstate ide o zreťazenie výsledkov porovnania jasu medzi rôznymi vzorkami obklopujúcimi *keypoint* umiestnený v strede.



Obr. 2.13: Algoritmus FAST [20]

Ako je znázornené na obrázku 2.14, modré bodky tvoria sústredné kruhy pre vzorkovacie pozície. Červené kruhy označujú jednotlivé vzorkované oblasti na základe *Gaussovského vyhladenia*, aby sa predišlo aliasingovým efektom.



Obr. 2.14: Algoritmus BRISK [8]

2.3 Vizualizácia objektov v scéne

Obor počítačovej grafiky sa okrem iného zaoberá vizualizáciou objektov. Existuje velké množstvo technológií, ktoré poskytujú vykresľovať objektov. Tieto technológie možno deliť na nízkoúrovňové a vysokoúrovňové.

Nízkoúrovňové, ako napríklad *OpenGL* alebo *Metal* vyžadujú komplexné znalosti o počítačovej grafike a sú pomalšie na vývoj, nakoľko je potrebné vytvoriť celý systém, ktorý obsluhuje aplikáciu. Výhodou je plná kontrola nad vizualizáciou. Tieto nástroje umožňujú priamu komunikáciu s grafickým procesorom.

Vo väčšine prípadov sa pri vývoji používajú technológie aplikačné, ktoré majú vytvorenú nízkoúrovňovú vrstvu, ktorá je prístupná pomocou rozhrania, ktoré umožňuje prácu s 3D grafikou na vyššej úrovni. Niektoré vysokoúrovňové nástroje poskytujú aj svoje vývojárske nástroje (*Integrated Development Environment* – IDE), ktoré poskytuje funcionality ako animáciu objektov, komunikáciu po sieti či systém s fyzikálnymi vlastnosťami (gravitácia, odraz svetla).

ARCore[6] vyrobený spoločnosťou *Google* je *framework*, ktorý nepodporuje iba *Android* zariadenia, ale niektoré jeho funkcie sú dostupné aj pre platformu *iOS*. *ARCore* využíva tri kľúčové vlastnosti pre integráciu virtuálneho obsahu do reálneho sveta prostredníctvom kamery. Prvou je sledovanie pohybu, ktoré umožnuje telefónu porozumieť a sledovať pohyb vo vzťahu k reálnemu svetu. Druhou je enviromentálne porozumenie, ktoré umožnuje zistiť veľkosť a umiestnenie rôznych typov povrchu ako napríklad vertikálne a horizontálne plochy, stoly apod. Poslednou vlastnosťou je odhad svetla, ktorý umožnuje telefónu odhadnúť aktuálne svetelné podmienky prostredia.

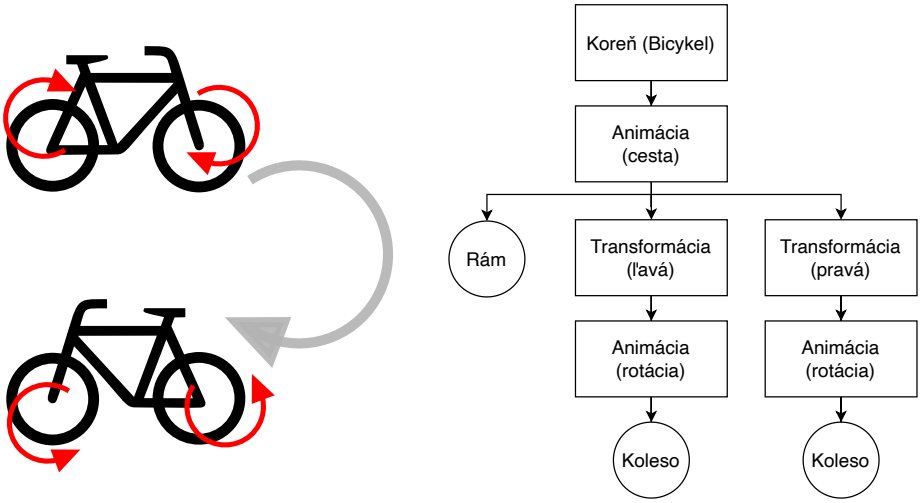
Vuforia[7] je *framework*, ktorý podporuje väčšinu dnes používaných platforiem – *Unity*, *iOS*, *Android* a *Universal Windows Platform*. Podporuje rozoznávanie objektov, textu, prostredia a vďaka obrovskému množstvu funkcionalít sa stal populárny pri práci s rozšírenou realitou. V aktuálnej verzii 9.1 podporuje nielen rozoznávanie objektov, ale objekty je možné sledovať a pripojiť k nim ďalší obsah. Oproti svojim natívnym konkurentom poskytuje napríklad vstavanú funkciu, kde je možné na objekty cylindrického tvaru prilepiť obrázok. V iných natívnych *frameworkoch* je takéto chovanie taktiež možné vytvoriť, ale vyžaduje výrazne väčšie znalosti a celé vlastné riešenie je potrebné implementovať.

Wikitude[8] je *framework* veľmi podobný *Vuforii*. Podporuje rozoznávanie a snímanie objektov spolu s pridávaním virtuálneho obsahu. V rámci veľkých projektov je možné použít *cloudové* riešenie, kde sú uložené rozoznávané objekty na internete. Nechýba podpora *Hololens* a dokonca sú podporované aj multiplatformné riešenia vývoja mobilných aplikácií *Xamarin*, *Flutter* a *Cordova*. *Framework* je možné bezplatne vyskúšať, ale vo výslednom obraze sú vodoznaky *Wikitude*.

SceneKit[9] patrí medzi populárny *framework* od spoločnosti *Apple* a je zameraný na pokročilú správu pomocou dátovej štruktúry grafu scény. Príklad takéhoto grafu scény je ilustrovaný na obrázku 2.15. Táto štruktúra je bežne používaná v editoroch vektorovej grafiky, ktorej úlohou je priestorová reprezentácia scény a objektov v nej, pomocou stromovej štruktúry. Vďaka tomu, že sa jedná o stromovú štruktúru, každý uzol má svojho predchodcu, poprípadne následníkov. Táto vlastnosť sa využíva pri nastavovaní vlastností objektov. Aplikáciou vlastností na uzol vyvolá rovnakú vlastnosť na všetkých jeho potomkov. Príkladom použitia je pohyb modelu bicykla, ktorý je zložený z viacerých častí – kolesá, rám, riadenie, sedlo a ďalšie. Aplikácia translačnej operácie na rám bicykla spôsobí aplikovanie rovnakej operácie aj na ďalšie časti, a teda nie je potrebné aplikovať túto operáciu zvlášť na každú časť jednotlivo.

SceneKit je proprietárna knižnica, ktorá je dostupná len pre operačné systémy *macOS*, *iOS* a *tvOS*. Výhoda oproti ostatným knižniciam spočíva aj v tom, že *SceneKit* knižnica je obsiahnutá v operačnom systéme a tak nie je potrebné ju prikladať k aplikácii, čím sa u ostatných knižníc častokrát výrazne zvyšuje veľkosť výslednej aplikácie. Nízkoúrovňovo používa knižnicu *Metal*, avšak pre staršie verzie operačných systémov používa knižnicu

[6] https://developers.google.com/ar
[7] http://vuforia.com
[8] https://wikitude.com
[9] https://developer.apple.com/documentation/scenekit



Obr. 2.15: Vizualizácia grafu scény [18]

OpenGL, určenú na vykreslovanie objektov. Ďalšou výhodou je komunikácia s *Core Animation* – knižnicou, ktorá umožňuje animáciu objektov. V neposlednom rade obsahuje systém pre simuláciu fyzikálnych zákonov.

Vývojové prostredie *Xcode* podporuje návrh scény pomocou vstavaného grafického nástroja *Interface Builder*, ktorý ponúka veľkú množinu preddefinovaných tried a základných geometrických primitív. Na základe týchto primitív sa vytvárajú zložitejšie modely. *SceneKit* podporuje import aj export navrhnutej scény vo formáte formáte COLLADA, ktorý poskytuje prenostiteľnosť medzi jednotlivými systémami. Keďže *ARKit* a *SceneKit* sú integrované v operačných systémoch *iOS*, je práve táto kombinácia častým a vhodným spôsobom tvorby rozšírenej reality.

Unity[10] je jedným z najznamejších konkurentov *ARKitu* pre platformu *iOS*. Často označované ako multiplatformové grafické jadro počítačovej hry (*engine*), ktoré si našlo využitie pri vývoji 2D/3D hier a simulátorov. Za multiplatformový sa označuje vďaka podpore až 27 platforiem.

Ponúka vývojárske prostredie, ktoré slúži na vytvorenie scény, logiky a správu aplikácie. Logika je písaná v jazyku *C#* alebo v jazyku *Javascript*. Za interpretáciu a kompiláciu je zodpovedný *framework .NET*.

Hlavnou výhodou je využitie nízkoúrovňových natívnych grafických knihovní, čo prináša teoretický vysoký výkon. Z toho vyplýva, že pre *iOS* je možnosť využitia knižnice *Metal* alebo *OpenGL*. Ďalšou výhodou je možnosť jednoduchej transformácie celého projektu pre vytvorenie virtuálnej reality zobrazenej pomocou špeciálnych okuliarov. IDE poskytuje možnosť optimalizácie renderovania pre zariadenia určené pre virtuálnu realitu a ponúka kontroléry kamery a podporu snímania pohybu používateľa. Konkrétne sa jedná o podporu produktov *HTC Vive* alebo *Oculus Rift*.

Unity je možné použiť v kombinácii s *ARKitom*. Stačí importovať balíčky, ktoré sprístupnia túto knižnicu. Knižnica je ponúkaná v štyroch verziách, ktoré sa líšia typom licencie. V prípade, že autor presiahne určitý zisk z aplikácie, musí si zakúpiť vyšší typ licencie. Súčasťou zakúpenia vyššej licencie je možnosť kontaktovat užívateľskú podporu, poprípadne je možné upraviť úvodnú licenčnú obrazovku, respektívne ju odstrániť.

[10] http://unity3d.com

2.4 Užívateľská interakcia

Dobré užívateľské rozhranie (*User Interface* – UI) je z pohľadu úspešnosti aplikácie na trhu jednou z kľúčových vlastností aplikácie. Preto je potrebné venovať nemalú pozornosť práve návrhu uživateľského rozhrania, nakoľko je súčasťou tvorby uživateľského zážitku (*User Expierence* – UX). K návrhu UI sa používa *wireframe*. „*Wireframe* je prototyp s nízkou presnosťou webovej stránky alebo obrazovky aplikácie. *Wireframe* sa používa na identifikáciu prvkov, ktoré budú zobrazené na stránke alebo na obrazovke, ako napríklad navigácia, obsah sekcií, snímok a/alebo potreba médií, prvky formulára a výzvy na akciu [23]."

Koncept rozšírenej reality je pomerne nový v sfére mobilných aplikácií. Užívatelia tak zatiaľ nie sú zvyknutí na spôsob ovládania takýchto aplikácií. Z uvedeného vyplýva, že každá nová aplikácia môže pristupovať k tvorbe užívateľského rozhrania inak. Avšak, po určitej dobe možno očakávať isté ustálenie spôsobu ovládania. Spoločnosť *Apple* s príchodom *ARKitu* pripravila pre dizajnérov aktualizáciu odporúčaní pri tvorbe uživateľského rozhrania, ktoré dolnila aj o časť venovanú rozšírenej realite[11]. V nasledujúcej časti sú popísané niektoré z týchto odporúčaní.

Aplikácia by mala používať čo najväčšiu možnú plochu a teda podľa možností celý displej zariadenia. Je vhodné poskytnúť čo najväčšiu plochu pre zobrazenie obrazu snímaného kamerou, a zároveň objektov umiestnených v tomto obraze. Treba sa vyvarovať zakrývaniu obrazu zbytočnými ovládacími prvkami, ktoré pôsobia rušivým dojmom a zhoršujú zážitok z rozšírenej reality.

Pri umiestňovaní objektov je vhodné brať ohľad na realistické umiestňovanie objektov do scény. Objekty by mali používať vysokokvalitné textúry, správnu mierku, či reflektovať svetelné podmienky ako odraz svetla a tieň. Svetelné podmienky sa v čase a s pohybom menia, takže je potrebné dbať aj na tento problém, a aktualizovať vzhľad objektov.

Správne fungovanie *ARKitu* je možné iba za vhodných svetelných podmienok. Napríklad v tme či za zníženej viditeľnosti nie je knižnica schopná poskytovať potrebnú presnosť. Túto skutočnosť je potrebné poskytnúť užívateľovi v prípade, že aplikáciu spustí za zlých svetelných podmienok, prípadne obmedziť určité funkcie aplikácie, ktoré vyžadujú *ARKit*. Pri používaní aplikácie je tiež potrebné vyhnúť sa pokynom, ktoré by viedli k zníženému komfortu užívateľa pri používaní aplikácie, ako napríklad neprirodzené držanie telefónu alebo držanie telefónu v jednej polohe po príliš dlhý čas. Mať na pamäti bezpečnosť je jednou z kľúčových bodov, a preto je dobré vyhnúť sa pokynom, ktoré vyžadujú náhle či veľké pohyby.

Odporúča sa použitie zvukových a hmatových indikácií ako spätnej väzby po vykonaní daného pokynu. Toto pravidlo sa uplatňuje bez rozdielu na výsledok – čiže pri úspešnom, ale i neúspešnom úkone. S tým súvisia nápovedy, ktoré je vhodné zobrazovať prakticky kdekoľvek, kde je to možné. 3D vizualizácia nápovedy je efektívnejšia a intuitívnejšia než zobrazenie textu s pokynom. V prípade, keď je použitie textu nevyhnutne potrebné, je použitie správnej terminológie kľúčové najmä z pohľadu bežného používateľa, ktorý často nie je expertom na počítačovú grafiku. Namiesto oznámenia chyby je vhodné navigovať užívateľa pokynmi, ktoré chybu odstránia. Napríklad, miesto použitia hlášky, že bol detekovaný príliš rýchly pohyb, je vhodnejšie zvoliť hlášku oznamujúcu, aby sa užívateľ snažil hýbať s telefónom pomalšie.

Po spustení aplikácie, *ARKit* musí prejsť cez inicializačný postup, ktorý trvá zvyčajne niekoľko sekúnd. Počas tejto doby by mal byť užívateľ informovaný, že prebieha inicializácia, a je vhodné, aby sa s telefónom pohyboval po miestnosti a snímal okolie kamerou.

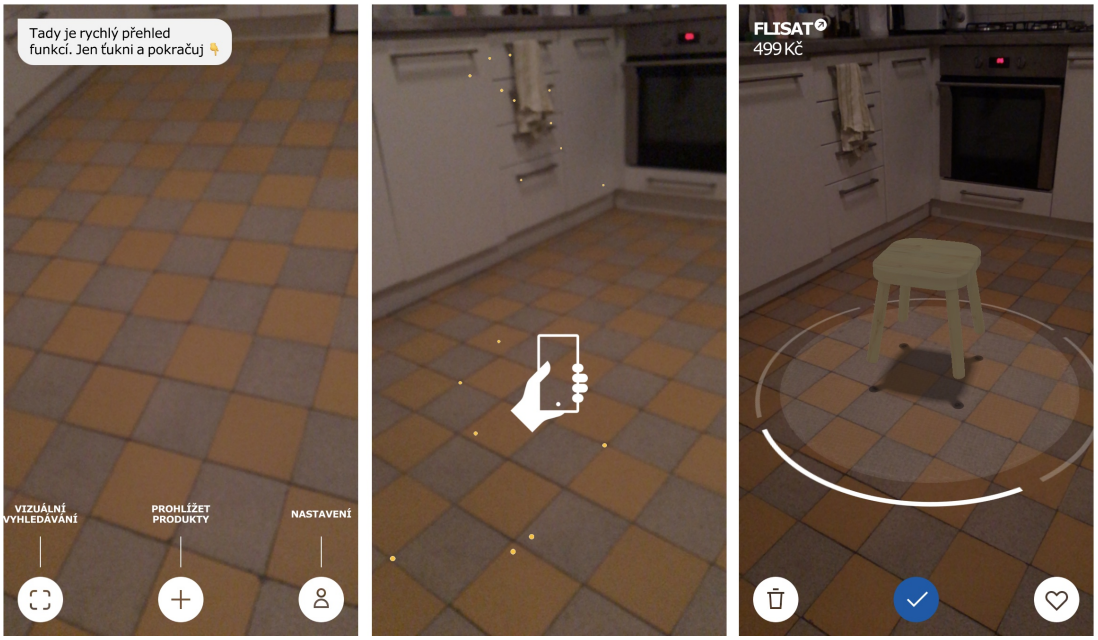
[11] https://developer.apple.com/design/human-interface-guidelines/

Jedným zo základných pravidiel návrh užívateľského rozhrania je použitie uživateľovi známych ikon. Preto je dobrým zvykom vyhnúť sa novým ikonám, ktoré užívateľ nepozná. Pre detekciu plochy sa používa skosený (priestorový 3D efekt) štvorec, ktorý v strede každej strany obsahuje nespojenú časť, indikácia umiestnenia objektu je reprezentovaná skoseným štvorcom bez týchto nespojených častí.

V prípade, že aplikácia využíva interakciu s objektami, uprednostňuje sa priama manipulácia s objektom oproti ovládacím prvkom, ktoré sú menej intuitívne a umiestnené mimo sledovaný objekt. Preto je použitie známych gest ako napríklad potiahnutie prstom pre posunutie objektu či rotáciu objektu v prípade rotácie objektu vhodnejšou alternatívou.

S príchodom *iOS 11* a *ARKitu* bola predstavená aplikácia *IKEA Place*, ktorá ako je z názvu poznateľné, pochádza z dielne úspešnej švédskej firmy *IKEA*. V čase písania tejto práce mala aplikácie hodnotenie 4,7 hviezdy z možných 5, ktoré znamenajú najlepší výsledok. Aplikácia slúži na umiestnenie predávaných produktov do priestoru. Aplikácia tak dáva možnosť potenciálnemu zákazníkovi vyskúšať si, ako by vyzeral daný produkt v mieste, kde by ho po zakúpení umiestnil. Každý model má reálne rozmery a textúru, respektívne farbu.

Na obrázku 2.16 je zobrazená hlavná časť aplikácie využívajúca rozšírenú realitu. Aplikácia spĺňa prakticky každý jeden bod z vyššie uvedených odporúčaní. Obraz kamery je zobrazený cez celú obrazku, sú použité animované vodiace prvky a užívateľ je sekvenciou jednoduchých krokov zoznámený s aplikáciou. Zobrazenie 3D objektov v scéne je hodnoverné – reflektuje svetelné podmienky, textúra materiálu je vo vysokej kvalite a rozmery objektu sú realistické. Aplikácia sa ovláda intuitívne priamou interakciou s virtuálnymi objektami.



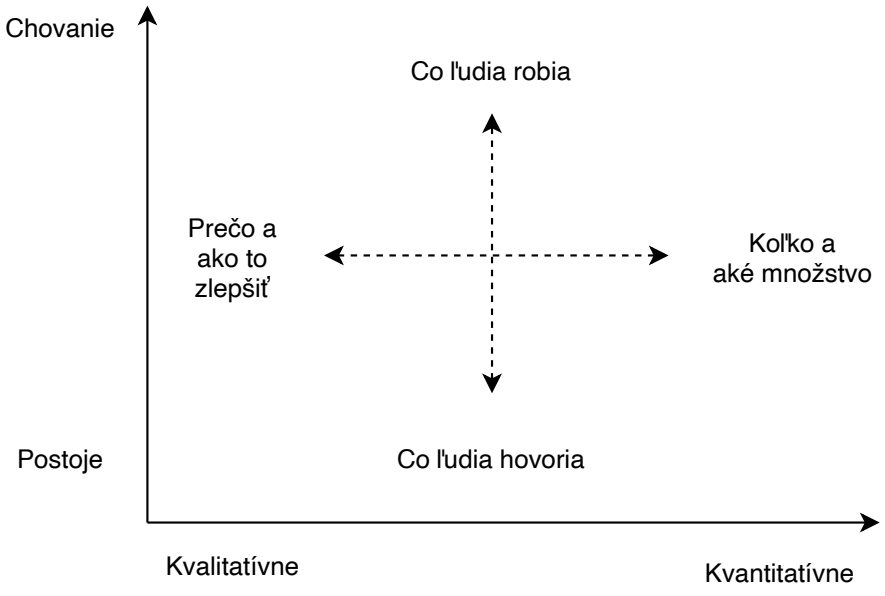
Obr. 2.16: Rozšírená realita v aplikácii IKEA Place[12]

Dôležitou oblasťou užívateľského rozhrania je overenie funkčnosti, ktoré sa spravidla robí na užívateľoch. Tento proces sa nazýva UX výskum, poprípadne užívateľské testovanie. Sleduje sa pritom chovanie užívateľa a získavajú sa odpoveďe na otázky typu čo užívateľ robí, je užívateľ spokojný a zvládne splniť úlohu užívateľ v dostatočnom čase? Hodnotí sa

[12] https://apps.apple.com/us/app/ikea-place/id1279244498

tak celkový dojem užívateľa. Dôležitým aspektom testovania na užívateľoch je správna identifikácia budúcich teoretických užívateľov. V skratke, ak sa jedná o animovanú hru, skupina testovaných užívateľov nebude vytváraná z dôchodcov. Pri tvorbe testov je hlavná úloha vytvoriť scenáre a určiť ako bude testovanie prebiehať. Často sa UX výskum robí vo fáze pred uvedením produktu na trh a pred samostatnou implementáciou, aby sa predišlo plytvaniu časových a finančných zdrojov na produkty a funcionality, ktoré nie sú správne navrhnuté. UX výskum slúži nie len na overenie myšlienky, ale aj na získanie nových nápadov, ktoré môžu prichádzať počas rozhovoru a sledovaní práce užívateľov pri samotnom testovaní a evaluácii výsledkov. Testovanie nemusí prebiehať výlučne priamo v kontakte s užívateľmi, ale dnes sa často používajú metódy A/B testovania (skupina užívateľov je rozdelená na 2 skupiny, kde jednej skupine sa zobrazí jedna verzia aplikácie a druhej skupine drúha verzia), online dotazníkov, nahrávania pohybu užívateľa aplikáciou (napríklad *smartlook*[13]) a mnoho ďalších.

Podľa Christiana Rohrera [19], experta na UX, vzniká tak delenie metód na kvalitatívne a kvantitatívne. Kvalitatívne štúdie skôr generujú údaje o správaní alebo postojoch na základe ich priameho pozorovania, zatiaľ čo v kvantitatívnych štúdiách sa údaje o príslušnom správaní alebo postojoch zhromažďujú nepriamo prostredníctvom merania alebo nástroja, napríklad prieskumu alebo analytického nástroja. Kvôli povahe ich rozdielov sú kvalitatívne metódy oveľa vhodnejšie na zodpovedanie otázok o tom, prečo a ako problém vyriešiť, zatiaľ čo kvantitatívne metódy dokážu oveľa lepšie zodpovedať, koľko a aké množstvo typov otázok. Takéto čísla pomáhajú pri určovaní priorít zdrojov, napríklad pri zameraní sa na problémy s najväčším dopadom. Ďalšie delenie je podľa postoju a chovania. Toto delenie sa dá zhrnúť kontrastom toho, „čo ľudia hovoria“ oproti tomu, „čo ľudia robia“ (veľmi často sa úplne líšia). Účelom prieskumu postojov je zvyčajne porozumieť alebo merať presvedčenie ľudí, a preto sa často tento typ prieskumu používa aj v oblasti marketingu. Tabuľka 2.17 ilustruje, ako tieto spôsoby ovplyvňujú typy otázok, ktoré je možné položiť a typy odpovedí.



Obr. 2.17: Typy otázok a odpovedí podľa spôsobu testovania [19]

[13] https://smartlook.com

2.5 ARKit

V roku 2017 spoločnosť *Apple* vydala knižnicu *ARKit*, ktorá je určená k tvorbe aplikácií, ktoré používajú rozšírenú realitu. Podmienkou kompatibility je *iPhone* s operačný systémom *iOS 11* a vyšším, čo pri údajoch zo začiatku roku 2020 predstavuje viac než 94 percentnú podporu zo všetkých iPhonov[14]. Spoločnosť *Apple* sa zameriavala predovšetkým na propagáciu funkcionality *ARKit*, ako jednej z hlavných funkcii operačného systému *iOS 11*. Dokonca bola do obchodu aplikácií *AppStore* pridaná aj sekcia pre aplikácie, ktoré využívajú rozšírenú realitu. Už po niekoľkých dňoch sa objavovali prvé aplikácie, ktoré využívali túto knižnicu a firmy sa predbiehali v tom, ktorá prvá poskytne funkciu s rozšírenou realitou.

Framework je postavený na technike zvanej VIO (*Visual Inertial Odometry*), ktorá je popísaná v kapitole 2.2. Vďakda tejto tychnológii je možné priniesť rozšírenú realitu do mobilného zariadenia, ktoré nepotrebuje žiadne špeciálne senzory. Vďaka tomu je *ARKit* podporovaný už *iPhonom 6s*, ktorý bol predstavený v roku 2015. V tejto dobe však existovala alternatíva hlavného konkurenta *Google*, a tou bol projekt *Tango*. Táto platforma mala nesmiernu nevýhodu oproti technológii *ARKitu*, a to predovšetkých v tom, že vyžadovala špeciálne senzory. Do tejto doby boli vyrobené iba 2 zariadenia, ktoré túto technológiu podporujú [24]. *Google* neskôr priznal, že sa vybral nesprávnou cestou. Ukončil projekt *Tango* a vytvoril projekt *ARCore* [6].

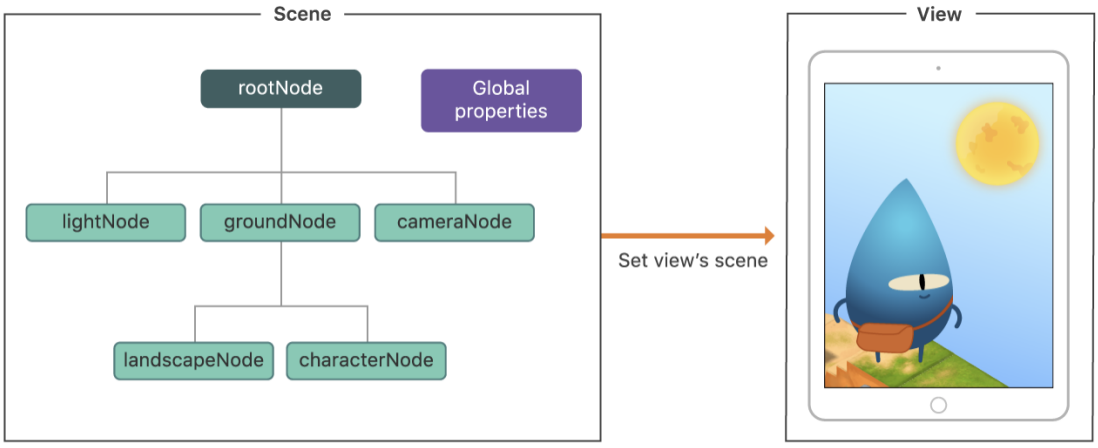
Knižnica je aktuálne vo verzii označenej *ARKit 3*. V prvej verzii podporovala detekciu horizontálnych plôch, ktoré mohli byť použité ako objekty pre ukotvenie virtuálnych predmetov. Ďalej podporovala meranie vzdialeností v scéne, estimáciu svetelných podmienok pre osvetľovanie virtuálnych predmetov, estimáciu tieňa a prácu s významnými bodmi v podobe mračna bodov (*point cloudu*). Aktualizácia na verziu 1.5 umožňovala detekovať okrem horizontálnych plôch aj plochy vertikálne. Ďalej pribudla podpora detekcia 2D objektov. Práve táto vlasnosť bude v tejto práci využitá vo veľkej miere. Verzia 2 priniesla funkcionalitu zdieľania scény s ostatnými užívateľmi, uloženie scény naviazanú na konkrétnu lokáciu, detekciu 3D objektov spolu a sledovanie 2D objektov. Spolu s uverejnením novej verzie *Apple* vydal novú verziu *iOS 12*, v ktorej je k dispozicii aplikácie *Measure*, v ktorej užívateľ môže jednoducho a pomerne presne merať dĺžku objektov. Okrem toho *Apple* v spolupráci s firmou Pixar vytvoril formát *usdz*, vďaka ktorému môže užívateľ jednoducho vložiť do scény 3D objekt[15]. Aktuálna verzia poskytuje realitstické vloženie AR obsahu pred/za ľudí v reálnom svete, ktoré spolu so zeleným plátnom ponúka pohlcujúci zážitok takmer v každom prostredí. Medzi ďalšie funkcie patrí napríklad súčasné snímanie prednej a zadnej kamery, detekciu pohybov snímaného človeka, súčasné snímanie viacerých tvárí, podpora simultánnej tvorby mapy reálneho sveta viacerými užívateľmi či zlepšenie detekcie 3D objektov a mnoho ďalšieho[16].

Vytvorenie aplikácie s využitím knižnice *ARKit* a *SceneKit*, teda aplikácie, ktorá pracuje s rozšírenou realitou, vyžaduje zoznámenie sa so základnou terminológiu, ktorá je uvedená nižšie.

- **Kotva** (`ARAnchor`). Kotva slúži na umiestnenie virtuálnych objektov do prostredia. Skladá sa z pozície a rotácie. Využíva sa na tranformácie – zmena pozície a rotácie, ktorá je priradená k 3D objektu. Umiestnenie kotvy je možné na základe vyhľadania významných bodov v okolí umiestnenej kotvy a vytvoreniu vzťahov medzi nimi.

[14] https://developer.apple.com/support/app-store/
[15] https://www.apple.com/newsroom/2018/06/apple-unveils-arkit-2/
[16] https://developer.apple.com/augmented-reality/arkit/

- **Významné body** (`vector_float3`). Sú to body, ktoré sa svojimi vlastnostňami dajú odlíšiť od ostatných bodov v obraze. Väčšinov sa jedná o rohy či texturované prvky. Najdôležitejšou vlastnosťou je spolahlivosť. Algoritmus musí byť schopný nájsť daný významný bod z rôznych uhlov či za rôznych svetelných podmienok. Rovnako tak je dôležité okolie, respektívne vzdialenosť k ostatným významným bodom. Čím bližšie tohto bodu sa nachádzajú iné významné body, tým jednoduchšie sa dá lokalizovať daný bod. Z toho vyplýva, že aj tento bod je vhodné označiť ako významný.
- **Body mračien** (`ARPointCloud`). Jedná sa o zhluk významných bodov.
- **Scéna** (`SCNScene`). Kontajner pre hierarchiu uzlov a globálne vlastnosti, ktoré spolu tvoria 3D scénu. Ak chceme zobrazovať 3D obsah pomocou *frameworku SceneKit*, najprv je potrebné vytvoriť scénu, ktorá obsahuje hierarchiu uzlov a atribúty, ktoré spolu spolu predstavujú vizuálne prvky. Príklad je možné ilustrovať obrázkom 2.18.
- **Obraz** (`SCNView`). Podobne ako `UIView`, ktorý je mimochodom jeho nadtriedou, `SCNView` slúži pre zobrazovanie obsahu, ale na rozdieľ od už predstaveného `UIView`, ktorý slúži na zobrazovanie 2D obsahu, `SCNView` slúži na zobrazenie 3D obsahu.
- **Uzoľ** (`SCNNode`). Prvok grafu scény predstavujúci polohu a transformáciu v 3D súradnicovom priestore, ku ktorému je možné pripojiť geometriu, osvetlenie, kamery alebo iný zobraziteľný obsah. Sám o sebe nemá žiadny viditeľný obsah – predstavuje iba transformáciu (pozícia, mierka a orientácia) vzhľadom k jeho nadradenému uzlu. Hierarchia uzlov je označovaná ako graf scény.



Obr. 2.18: Príklad scény[17]

[17] https://developer.apple.com/documentation/scenekit/scnscene

Kapitola 3

Vývoj informačných systémov

Navrhovaný systém možno rozdeliť do niekoľko hlavných častí – mobilná aplikáci, webová aplikácia a server. Každá časť má svoje špecifiká, využívajú sa rôzne jazyky a postupy. V nasledujúcich kapitolách sa čitateľ zoznámi so základnými princípmi a stavebnými prvkami, z ktorých sa skladajú jednotlivé subsystémy. Na začiatku je nutné zoznámit sa s prostredím *iOS*. Táto kapitola obsahuje popis architektúry operačného systému *iOS* a životný cyklus aplikácie. Ďalej nasleduje zoznámenie sa s vývojom webových aplikácií, konkrétne na platforme *React* a s vývojom logiky na serveri prostredníctvom softvérového systému *Node.js* a jeho *frameworkov*. Všetky 3 časti spolu určitým spôsobom komunikujú a preto je potrebná znalosť trendov v návrhu takýchto rozhraní, čomu sa venuje podkapitola 3.4.

3.1 Vývoj mobilných aplikácií

V tejto časti sa čitateľ oboznámi s programovacím jazykom *Swift*, jeho zatriedením a základnými črtami. Následuje popis základných stavebných prvkov pri tvorbe aplikácie a jej životného cyklu.

Programovací jazyk *Swift*

Programovací jazyk *Swift*, vytvorený spoločnosťou *Apple* v roku 2014, nahradzuje programovací jazyk *Objective-C*, dovtedy jediný jazyk na tvorbu mobilných aplikácií na platformu *iOS*. Oproti svojmu predchodcovi má na prvý pohľad poznateľne jednoduchšiu syntax [15]. Tento jazyk bol v roku 2014 publikovaný ako proprietárny jazyk a neskôr v roku 2015 sa z neho stal otvorený software (*open source*). Jazyk je možné používať na všetkých platformách *Apple* (*iOS*, *watchOS*, *tvOS*, *macOS*). Tento prakticky nový jazyk rýchlo naberá na popularite a dokonca sa už stal 10. najpopulárnejším programovacím jazykom [16]. Jazyk je podporovaný aj platformou *Linux* a kedže je podpora vyvojárov vysoká, vývojárske tímy pracujú aj na podpore iných platforiem, medzi ktoré patrí napríklad aj *Windows*[1]. Aktuálna verzia jazyka je 5.2.3 a každou aktualizáciou prináša množstvo nových funkcií[2]. Posledná verzia okrem iného priniesla napr. vstavanú možnosť iterácie cez výčtový dátový typ, pribudli direktívy pre zobrazenie varovaní a chýb počas kompilácie, príkaz na generovanie pseudonáhodných čísel, príkaz na pseudonáhodné usporiadanie čísel v kolekcii či jednoduchšie a bezpečnejšie vytváranie hashu objektov a mnoho ďalšieho.

Swift sa radí medzi jazyky, ktoré využívajú viaceré paradigmá a to konkrétne nasledovné:

[1] https://swiftforwindows.github.io
[2] https://docs.swift.org/swift-book/RevisionHistory/RevisionHistory.html

- **Protokolovo orientovaná paradigma**. Podporuje používanie protokolov (špecifikácia požiadaviek na funkčnosť) so štruktúrami, ktoré ich implementujú, za použitia hierarchií dedičnosti tried za účelom opätovného použitia kódu a kategorizácie typov.
- **Objektovo orientovaná paradigma**. Výkonný kód je pridružený k dátam (metódy sú zapúzdrené k dátam).
- **Functionálna paradigma**. Výpočet chápe ako vyhodnotenie matematických funkcií.
- **Imperatívna paradigma**. Popisuje výpočet pomocou postupnosti príkazov a určuje presný postup, ako danú úlohu riešiť.
- **Blokovo štrukturovaná paradigma**. Skladanie implementovaného algoritmu z riadiacich štruktúr s jedným vstupným a jedným výstupným bodom namiesto neomedzeného používania skokov.

Základné črty jazyka sú:

- **Bezpečnosť.** Priamo na stránke `swift.com` je *Swift* definovaný ako univerzálny programovací jazyk vytvorený pomocou moderného prístupu k **bezpečnosti**, výkonu a návrhom softvérových vzorov. Tento jazyk je navrhnutý tak, aby bol bezpečnejší než jazyky typu *C*, a aby eliminoval celé skupiny nebezpečných kódov. Napríklad, premenné sú vždy inicializované pred prvým použitím, polia a čísla sú kontrolované na pretečenie a pamäť je spravovaná automaticky. Ďalšiou bezpečnou vlastnosťou je, že objekty nikdy nemôžu byť typu `nil` a snaha o použitie objektu, ktorého výsledkom je objekt typu `nil` skončí chybou v dobe kompilácie. Aj vďaka tejto vlastnosti je písanie kódu bezpečnejšie, čistejšie a zabraňujú častému dôvodu pádu aplikácie. Avšak, sú miesta, kde je použitie typu `nil` vhodné a pre tento spôsob použitia slúží vlastnosť známa ako *voliteľný typ*[3]. Tento typ môže obsahovať aj hodnotu typu `nil`, ale syntax núti programátora použiť znak otázniku pre indikáciu kompilátoru, že programátor rozumie chovaniu a bude s touto hodnotou zachádzané bezpečne.
- **Rýchlosť.** *Swift* je určený ako náhrada za jazyky typu *C* (*C*, *C++* a *Objective-C*). *Swift* musí byť pre väčšinu úloh porovnateľný s výkonnými jazykmi. Výkon musí byť tiež predvídateľný a konzistentný. Existuje mnoho nových jazykov s množstvom nových funkcií – rýchlosť je však ale zriedkavá. Komplexné triedenie objektov je v jazyku *Objective-C* 2,8 krát rýchlejšie než jazyku *Python* a v jazyku *Swift* až 3,9 krát rýchlejšie[4]. Tejto rýchlosti vďačí architektom, ktorý použili prekladač LVVM, ktorý zdrojové kódy pretransformuje do optimalizovaného natívneho kódu, ktorý je schopný použiť veľmi efektívne nový hardvér.
- **Komplatibilita s jazykom *Objective-C*.** *Swift* je plne kompatibilný s jazykom *Objective-C*. To znamená, že jedna aplikácia môže byť písaná súčasne v jazyku *Objective-C* a *Swift*. *Swift* môže teda volať rozhrania písané v *Objective-C* a naopak, s výnimkou, kde *Objective-C* nevie reprezentovať všetky objekty vytvorené v *Swifte* (napr. výpočtový typ s hodnotami typu reťazec).

[3] `https://developer.apple.com/documentation/swift/optional`
[4] WWDC 2014 `https://www.youtube.com/watch?v=w87f0AG8fjk`

Stavebné prvky aplikácie

Typická štruktúra aplikácie je znázornená na obrázku 3.1. Kontroléry obrazoviek sú mozgom celej aplikácie. Kontrolér pristupuje k dátam, respektívne k modelu, ktorý obsahuje aplikačné dáta a tieto dáta poskytuje obrazovkám, ktoré sa často skladajú z komponentov *frameworku UIKit* s možnosťou vytvorenia vlastných, neštandardných komponentov. *Frameworky UIKit* a *Foundation* tak poskytujú základné stavebné prvky. *UIKit* poskytuje prvky na vytvorenie vzhľadu aplikácie, kde základným prvkom je objekt `UIView`, ktorý je zodpovedný za zobrazenie obsahu na obrazovke. *Foundation* poskytuje dátové typy (čísla, reťazce, polia...), pomocou ktorých sa definuje model (dáta aplikácie). Úlohou `UIApplication` je príjem udalostí a manažment životného cyklu popísaného v sekcii 3.1.

`UIViewController`[5] je jeden zo základných stavebných prvkov aplikácie. Je to trieda, respektívne objekt, ktorý riadi hierarchiu obrazoviek. Definuje zdielané správanie, ktoré je spoločné pre všetky kontroléry obrazoviek. Väčšinou sa nevytvára inštancia priamo, ale vytvára sa trieda, ktorá dedí z tejto základnej triedy. Následne sa do vytvorenej triedy pridávajú metódy a vlastnosti, ktoré riadia hierarchiu obrazoviek.

Medzi hlavné zodpovednosti patrí aktualizácia obsahu obrazoviek, ktorá je väčšinou vyvolaná zmenou dát. Rovnako sa tak stará o zmeny vyvolané užívateľom, zmenu veľkosti obrazovky, rozloženie celkového rozhrania či komunikáciu s ostatnými kontrolérmi. Typicky, jeden kontrolér sa napr. stará o zobrazenie jednej tabuľky, kde druhý kontrolér sa stará o detail zobrazenia jednej položky po vybratí prvku z tabuľky. Býva zvykom, že objekty typu `UIView` patriace iba jednému kontroléru sú viditelné v jeden moment na obrazovke.

`UIViewController` má svoj životný cyklus a v dôsledku zmeny stavu sa volajú nasledujúce metódy, ktoré sa používajú na nasledovné akcie. Väčšina z nasledujúcich metód existuje v podobe `Will` – čize do tohto stavu sa ešteln dostane v podobe `Did`, ktorá je volaná bezprostredne po stave, ktorý nastal. Pre jednoduchosť uvádzam iba vybrané metódy.

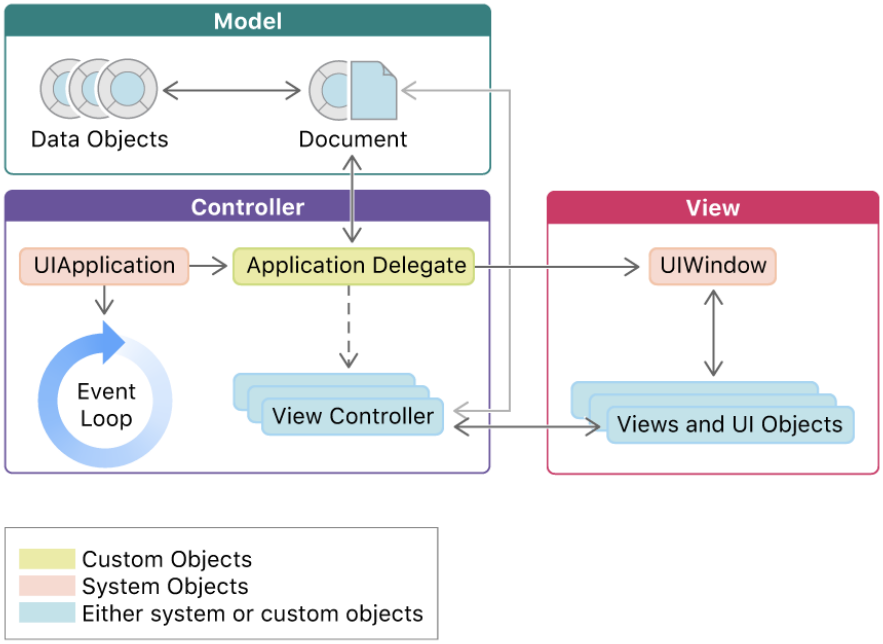
- `viewWillAppear`. Metóda sa volá pred tým, než je jeho obrazovka pridaná do hierarchie obrazoviek. Obrazovka pozná svoje rozmery, ale napríklad orientácia ešte nie je známa.
- `viewDidLoad`. Metóda sa volá ihneď po tom, čo je obrazovka vytvorená a načítaná do pamäťe. Je to vhodné miesto na inicializáciu objektov.
- `viewWillDisappear/viewDidDisappear`. Tieto metódy, ako je z názvu predpokladateľné, sa volajú pred/po tom čo obrazovka zmizne. Používajú sa častokrát na rôzne grafické animácie.

Životný cyklus aplikácie

Aplikácie sú spojením systémových *frameworkov* a vlastného kódu. Systémové *frameworky* poskytujú základnú infraštruktúru potrebnú k behu vlastných aplikácií a teda kódu, ktorý upravuje dizajn a chovanie aplikácie. *Framework iOS* je postavený na architektúre *Model-View-Controller* (MVC). Tento návrhový vzor oddeľuje aplikačné dáta a biznis logiku od vizuálnej reprezentácie týchto dát.

[5] https://developer.apple.com/documentation/uikit/uiviewcontroller

[6] https://developer.apple.com/library/archive/referencelibrary/GettingStarted/RoadMapiOS-Legacy/chapters/KnowtheCoreObjectsofYourApp/KnowCoreAppObjects/KnowCoreAppObjects.html



Obr. 3.1: Typická štruktúra aplikácie[6]

Vstupným bodom každej aplikácie založenej na jazyku *C* je funkcia `main` a rovnako to platí aj pri *iOS* aplikáciách. *Xcode* však túto funkciu vytvorí sám, takže ju už nie je potrebné písať. Jedinou úlohou funkcie `main` je prenechanie riadenia *UIKit frameworku*. Tak sa dostáva k slovu funkcia `UIApplicationMain`, ktorá vytvorí základné objekty aplikácie a prenechá riadenie aplikácie programátorovi – inicializačný kód. Základom každej aplikácie je objekt `UIApplication`, ktorého úlohou je uľahčiť interakciu medzi systémom a inými objektmi v aplikácii.

Aplikácie sú založené na princípe príchodu udalostí, ktoré vyvolal užívateľ. Vybrané udalosti:

- **Dotyk.** Udalosť prijíma objekt obrazovky v ktorej vznikla udalosť.
- **Vzdialené ovládanie, udalosť pohybu trasenia.** Udalosť prijíma prvý objekt, ktorý je schopný takýto udalosť spracovať.
- **Akcelerometer, magnetometer, gyroskop.** Udalosť prijíma objekt, ktorý je na to určený.
- **Lokácia.** Udalosť prijíma objekt, ktorý je na to určený.
- **Prekreslenie.** Udalosť prijíma obrazovka, ktorá potrebuje aktualizovať.

Dotykové udalosti v objektoch na ovládanie – napr. tlačidlá sú spracovávané inak ako dotyky popísané vyššie. Tieto používajú systém správ, ktoré sú doručené určeným objektom. Tento návrhový vzor je známy pod názvom cieľ-akcia (*target-action*) a umožnuje vyvolať vlastný kód po kliknutí na ovládací prvok.

V ľubovolnom momente sa aplikácia nachádza v jednom z nasledujúcich stavov:

- **Aktívna.** Aplikácia je v popredí a prijíma udalosti.

- **Neaktívna.** Aplikácia je v popredí a neprijma žiadne udalosti.
- **Na pozadí.** Aplikácia vykonáva kód a je na pozadí.
- **Suspendovaná.** Aplikácia nevykonáva žiaden kód, zostáva v pamäti a je na pozadí.
- **Vypnutá.** Aplikácia nebola spustená alebo bola ukončená systémom.

Pri prechode medzi týmito stavmi sú volané delegátne metódy, v ktorých je možné vykonávať vlastný kód a reagovať tak na zmenu stavu aplikácie. Pre znázornenie, prvá šanca na vykonanie vlastného kódu je v tele funkcie `application:willFinishLaunchingWithOptions:` a naopak posledná šanca je vo funkcii `applicationWillTerminate:`, ktorá je vyvolaná tesne pred ukončením aplikácie, teda pred prechodom do stavu vypnutia[7].

3.2 Vývoj webových aplikácií

Na začiatku 90. rokov 20. storočia vznikali prvé webové stránky. Tie sa však nedajú porovnať so stránkami, ako ich poznáme dnes. Väčšinou obsahovali iba jednoduché statické informácie, ktoré si užívatelia zobrazovali prostredníctvom svojich prehliadač. Jednalo sa o čistý hypertextový značkovací jazyk (*HyperText Markup Language* – HTML), bez kaskádových štýlov (*Cascading Style Sheets* – CSS) a bez prakticky žiadnej logiky. To však už neplatí pre dnešné weby, ktoré dnes často slúžia k úplne iným účelom ako kedysi a dajú sa považovať za aplikácie.

Webové aplikácie sú v podstate počítačové programy typu klient-server, kde klient beží vo webovom prehliadači. Pre ilustráciu – medzi bežné webové aplikácie možno zaradiť rôzne e-shopy, elektronické bankovníctvo či sociálne siete. Výhodou takýchto aplikácií je, že užívateľ nemusí inštalovať špeciálny softvér pre každú službu zvlášť, ale vystačí si s bežným webovým prehliadačom. Druhou veľkou výhodou je fakt, že aktualizácie prebiehajú centralizovane na jednom mieste – na serveri, na ktorom beží daná aplikácie. Vďaka tomu sa užívateľ nemusí starať o žiadne aktualizácie a súčastne výhodou pre vývojarov je možnosť opravovať chyby prakticky okamžite vydaním novej verzie bez povšimnutia užívateľov.

Webové aplikácie niekedy mávajú spoločné požiadavky a funkcie. Preto vznikli knižnice, ktoré urýchlujú vývoj a zjednodušujú tvorbu veľkých aplikácií. Podľa ankety [1], najpopulárnejší jazyk medzi programátormi sa stal *Javascript* a preto nie je náhoda, že najpopulárnejšie webové *frameworky* používajú práve tento jazyk. Na čele rebríčka sa umiestnili *jQuery*, následne *React.js* a *Angular.js*. Medzi jednotlivými knižnicami sú značné rozdiely. Medzi hlavné funkcie *jQuery* možno zaradiť predovšetkým priamu manipulácii s objektovým modelom dokumentu (*Document Object Model* – DOM), AJAX (*Asynchronous JavaScript and XML*) volaní a obrovský počet rôznych javascriptových pluginov.

React[8] je zameraný na tvorbu užívateľského rozhrania a to deklaratívnych spôsobom. Vyznačuje sa vysokou efektívnosťou a rýchlosťou, nakoľko vykresluje jednotlivé komponenty iba v prípade, že sa zmenia dáta. Vďaka deklaratívnemu spôsobu programovania sa kód stáva predvídateľnejším a jednoducho sa v ňom hľadajú chyby. Používa tzv. stav aplikácie, na základe ktorého sa riadi zobrazovanie obrazoviek. *React* sám o sebe neobsahuje nástroje pre udržovanie stavu a preto často býva používaný spoločne s knižnicou *Redux*, ktorá je založena na nasledujúcich princípoch. Stav je iba pre čítanie a je uložený na jednom mieste

[7] https://developer.apple.com/library/archive/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/TheAppLifeCycle/TheAppLifeCycle.html

[8] https://reactjs.org

v objekte stromovej štruktúry. Zmeny stavu je možné vykonať iba vyvolaním akcie, vďaka čomu je zamedzené vzniku časovo závislých chýb, čo uľahčuje ladenie programu.

Užívateľské rozhranie sa skladá z komponent, kde každá komponentá má vlastný stav. Následné skladanie týchto komponent tvorí komplexné užívateľské rozhranie. Každá *React* komponenta implementuje metódu `render()`, ktorá manipuluje so vstupnými dátami a určuje, čo sa má zobraziť. Komponenty sú malé, znovu použiteľné kusy kódu. Vstupné dáta, ktoré sú odovzdané do komponenty sú prístupné pomocou `this.props` (určené iba na čítanie). Okrem vstupných dát obsahuje komponenta aj interné dáta stavu, ktoré sú prístupné pomocou `this.state`. Každá zmena dát stavu komponenty vyvoláva nové vykreslenie komponenty opätovným volaním metódy `render()`. Komponenty je možné rozdeliť na rôzne časti podľa funkčnosti a použiť ich v rámci iných komponent. Komponenty môžu vracať ďalšie komponenty. V prípade, že sa nejaká časť užívateľského rozhrania používa viackrát alebo je dostatočne komplexná, je dobrým pravidlom z tejto časti vytvoriť komponentu. Príkladom častej komponenty je tlačidlo a v prípade komplexnosti sa môže jednať o formulár.

React narušil štandard oddelovania UI šablón a logiky. Nejedná sa teda o šablonovací systém, kde úlohou je oddeliť kód od popisu štruktúry stránky. Šablóna obsahuje reťazec zložený z HTML, CSS a Javascriptu. Šablónovací systém v skutočnosti pomocou regulárnych výrazov naplní šablónu dátami a poskladá z nej výsledný reťazec, ktorý sa zobrazí v prehliadači – ide teda o spájanie reťazcov. Pri prenačítaní stránky sa celý obsah nanovo vytvorí – poskladá sa celý nový reťazec. Ak chceme editovať konkrétnu časť stránky, môžeme nad DOM elementom použiť `Element.innerHTML`. Na prvý pohľad sa zdá byť toto riešenie rýchle a funkčné. To však neplatí pre prípad, ak danú komponentu sú nastavené tzv. *listenery* (napr. `onMouseOver`), ktoré automaticky po zmene zmiznú a je ich potrebné znovu nastaviť. Čiastočné riešenie je použitie metódy `Node.replaceChild()`, ale toto vedie k neprehladnému imperatívnemu programovaniu, kde vznikajú dlhé série `Node` metód, ktoré niekoľkými sadami `Node` metód dostávame komponentu z jedného stavu do druhého. Práve tu je sila *Reactu*, kde iba definujeme jednotlivé stavy, resp. podobu komponenty. To je možné prostredníctvom virtuálnehom DOMu, ktorý sa vytvorí na základe už spomínaných metód `render()`. Následne sa tento virtúalny DOM pomocou metód `appendChild` či `replaceChild` premietne do DOMu skutočného a efektívne sa prekreslí v prehliadači.

Dostávame sa tak k tomu, prečo sa pri práci s *Reactom* často používa syntax rozšírenie jazyka *Javascript* (*Javascript XML* – JSX), ktoré zvyšuje čitateľnosť kódu. Avšak jeho použitie nie je to podmienkou. JSX sa teda používa na popis užívateľského rozhrania v rámci *Javascript* kódu. V skutočnosti sa jedná o prevod HTML na funkcie, kde názvy elementov nahradíme názvami funkcií a z atribútov sa stane jeden veľký atribút.

```
<form className="form">
    <input type="text" placeholder="Name" />
    <input type="text" placeholder="Surname" />
    <input type="submit" value="Send" />
</form>
```

sa zapíše ako

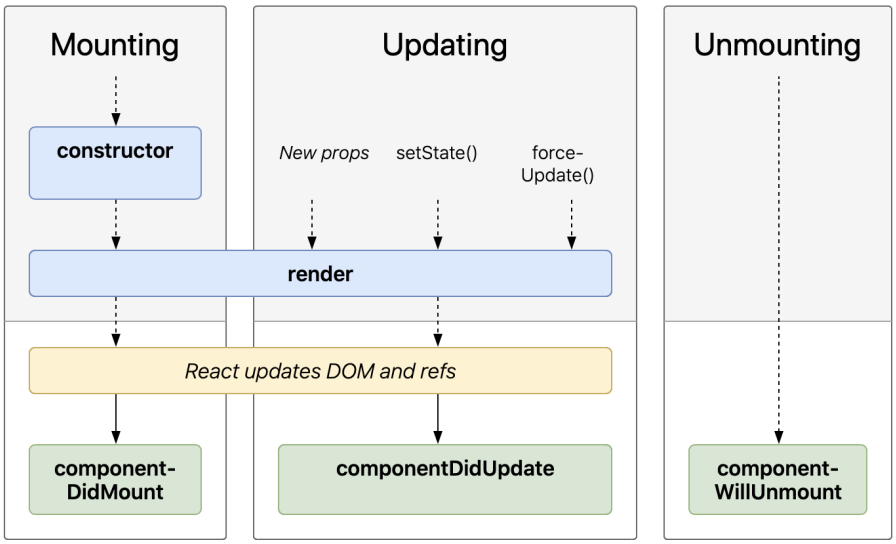
```
React.DOM.form({className: 'form'},
    React.DOM.input({type: 'text', placeholder: 'Name'}),
    React.DOM.input({type: 'text', placeholder: 'Surname'}),
    React.DOM.input({type: 'submit', value: 'Send'})
)
```

Každá komponenta má svoj životný cyklus. Pri každej zmene sa volá príslušná metóda, ktorú je možné nahradiť vlastnou implementáciou. Nie všetky metódy sa zvyknú používať – napríklad metóda `getSnapshotBeforeUpdate`, ktorá sa volá priamo pred tým, ako sa uloží posledný vykreslený výstup do DOMu, sa používa iba výnimočne. Obrázok 3.2 znázorňuje najčastejšie používané metódy.

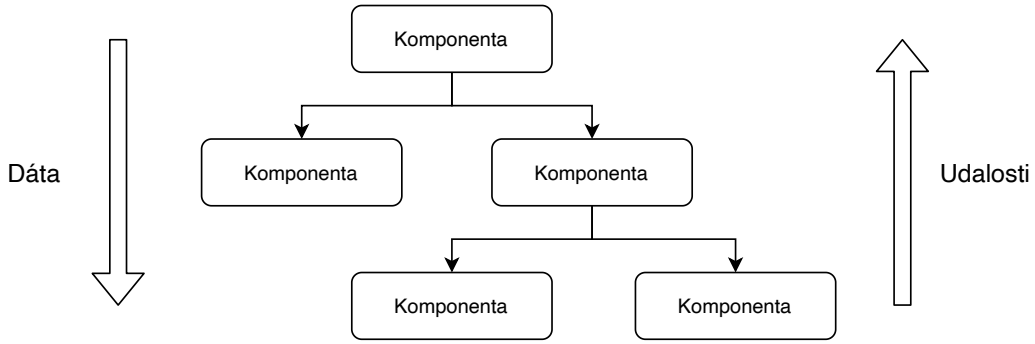
- `constructor()`. V prípade, že nie je potreba inicializovať stav a nie je potrebné naviazanie (*binding*) metód, nie je potrebné implementovať konštruktor. Konštruktor *React* komponenty je volaný pred tým, než je pripojená komponenta. Typicky sa používa k dvom účelom – inicializácia lokálneho stavu priradením objektu do `this.state` a *binding* metód na spracovanie udalostí.
- `render()`. Táto metóda je jediná metóda životného cyklu, ktorej implementácia je povinná v každej triede komponenty. Jedná sa o tzv. čistú metódu, čo znamená, že nemodifikuje stav komponenty a vracia zakaždým rovnaký výsledok. Na základe vstupných parametrov môže vracať následujúce typy:
  - ***React* elementy**. Typicky vytvorené prostredníctvom JSX, napríklad `<div />` alebo `<AwesomeComponent />`.
  - **Polia a fragmenty**. Umožnujú vracať viacej elementov.
  - **Portály**. Umožňujú vykreslovať potomkov do rôznych podstromov DOMu.
  - **Reťazce a čísla**. Sú vykreslované ako textové uzly v DOMe.
  - ***Booleovské hodnoty* a `null`**. Nevykreslujú nič. Používajú sa pre účely testovania.
- `componentDidMount()`. Je vyvolaná okamžite po pripojení komponenty, respektívne po tom, čo je komponenta vložená do DOM stromu. Ak je potrebné načítať dáta zo vzdialeného serveru, toto je správne miesto, kde zahájiť sieťový požiadavok. V tejto metóde je už možné nastaviť stav, avšak spôsobí to extra vykreslenie, ktoré nastane ešte pred tým, než prehliadač aktualizuje obrazovku, takže užívateľ ho neuvidí.
- `componentDidUpdate()`. Zavolá sa ihneď po tom, čo nastane aktualizácia. V prípade, že sa jedná o prvotné vykreslenie, metoda sa nevolá. Používa sa na operácie nad DOMom po tom, čo je komponenta aktualizovaná. Taktiež je to dobré miesto, kde je možné vyvolať sieťové operácie v závislosti na predchádzajúcich a aktuálnych dátach, ktoré sa predávajú vo vstupných parametroch.
- `componentWillUnmount()`. V prípade, že je komponenta odstránená z DOMu, okamžite sa vyvolá táto metóda. Slúži na vyčistenie komponenty – napríklad zrušenie časovačov, sieťových operácií a podobne.

Tok dát v *Reacte* je jedným smerom. *React* obsahuje nemennú množinu hodnôt, ktoré sa presúvajú do komponenty ako vlastnosti v HTML tagoch. Keďže komponenta nemôže priamo modifikovať žiadne vlastnosti, používa sa koncept, kde sa pomocou *callback* funkcie, ktorá je vložená ako parameter a prístupná v `props` danej komponenty, modifikuje potrebná hodnota. Tento proces sa v angličtine nazýva „*properties flow down; actions flow up*“, čo by sa dalo preložiť ako „vlastnosti prúdia smerom dole; akcie prúdia smerom hore“. Tento proces je ilustrovaný na obrázoku 3.3.

[9] https://github.com/wojtekmaj/react-lifecycle-methods-diagram



Obr. 3.2: Životný cyklus *React* komponenty[9]



Obr. 3.3: Tok dát v *Reacte*

Angular.js, ktorý sa umiestnil v rebríčku tretí, je na rozdiel od prvých dvoch umiestnených *framework*. Preto poskytuje komplexné riešenie pre vytváranie aplikácií s užívateľským prostredím. *Framework* má svoje pravidlá, ktoré definujú ako vytvoriť štruktúru aplikácie a akým spôsobom sa riešia jednotlivé problémy. Naopak, *React* a *jQuery.js*, obe knižnice, neobsahujú žiadne pravidlá ani obmedzenia, avšak táto flexibilita prináša so sebou zodpovednosť v podobe vytvorenia správnej architektúry aplikácie. Výhodu tiež je, že sa tieto knižnice môžu kombinovať s rôznymi inými knižnicami. Ďalší rozdiel medzi *Reactom* a *Angularom* je ten, že zatiaľ čo *Angular* používa architektúru MVC, *React* má iba *View*.

3.3 Vývoj serverovej časti aplikácií

Čo je to serverová časť aplikácie nazývaná tiež *backend*? Význam je možné odvodiť z anglického prekladu. Je to niečo, čo je vzadu a naopak, *frontend* je niečo, čo je vpredu, resp. čo je vidieť. Tieto pojmy zahŕňajú technológie podľa toho, či sa pomocou nich robia veci, ktoré sú viditelné, alebo sa pomocou nich robí vnútorné chovanie aplikácie. Druhé, možno jednoduchšie vysvetlenie je, že *frontend* určuje ako veci vyzerajú a *backend* určuje, ako

sa veci fungujú. HTML a CSS sú typické príklad jazykov, ktoré sa používajú pri tvorbe *frontendu*, na druhej strane je typický príklad medzi *backend* jazykmi *Python* alebo *C*.

Framework Express[10] sa zaradil medzi TOP 5 najpoužívanejšími webovými *frameworkami* a zároveň v rebríčku medzi najobľúbenejšími webovými *frameworkami* obsadil tretie miesto [1]. Predstavíme si ho v nasledujúcich riadkoch. Je rychlý, minimalistický a založený na *Node.js*. Jedna z kľúčových vlastností je nedogmatickosť, čo v preklade znamená, že programátorovi nenúti postupy ani komponenty, ale necháva na ňom, aké postupy a komponenty použije. Oproti *Node.js* je kód *Expressu* značne kratší, pretože *Node.js* je nízkoúrovňový *framework* a ak by sa použil pri tvorbe web serveru priamo napr. *http* modul, bolo by potrebné vysporiadať sa s nastavovaním hlavičiach, spracovaním požiadavkov, smerovaním pomocou regulárnych výrazov atď. *Express* poskytuje práve túto funkcionalitu a zároveň zachováva nedogmatickosť a vysokú rýchlosť behu programu.

Medzi hlavné problémy, ktoré rieši *Express* patrí *routing*. Pojem *routing* v tejto súvislosti znamená určenie, ako aplikácia reagujuje na požiadavok klienta, respektívne, ktorá konkrétna funkcia je za daný *endpoint* zodpovedná a má reagovať na požiadavok klienta. Jedná sa teda o namapovanie konkrétneho jednotného identifikátora zdroja (*Uniform Resource Identifier* – URI) a HTTP (*Hypertext Transfer Protocol*) metódy (GET, POST atď.) na príslušnú akciu *controlleru*. *Routa* pre GET metódu sa vytvára pomocou príkazu `get` nad objektom `express`:

```
app.get(route_path, ...callbacks);
```

Obdobne to funguje pre ostatné metódy (POST, PUT, DELETE, PATCH), kde `get` by sa zamenil za príslušnú metódu. Pre každú metódu existuje funkcia, ktorá ako argument príjma na prvom mieste cestu, akú daná *routa* spracováva a následne ľubovolný počet *callback* funkcií. *Express* ponúka aj `app.all()`, ktorá spracováva všetky HTTP metódy. Poslednou špecialnou metódou je `app.use()`, ktorá špecifikuje *middleware* ako *callback* funkciu. *Middleware* funkcie sú funkcie, ktoré majú prístup k objektu požiadavku, odpovedi požiadavku a k ďalšej *middleware* funkcii v aplikačnom požiadavok-odpoveď cykli. Používajú sa na vykonanie autentizácie alebo na prípadné modifikácie požiadavku. Zápis vyzerá následovne:

```
app.get('/', function(req, res, next) {
    next();
})
```

- `get`. HTTP metóda, na ktorú sa *middleware* aplikuje.
- `/`. Cesta, na ktorú sa *middleware* aplikuje.
- `function`. *Middleware* funkcia.
- `req`. Argument HTTP požiadavok extrahovaný do objektu, ktorý obsahuje metódu, URL (*Uniform Resource Locator*) parametre, telo požiadavku, atď.
- `res`. Argument HTTP odpoveď extrahovaný do objektu, ktorý je modifikovaný a následne odoslaný metódou `send`.
- `next`. *Callback* funkcia, pomocou ktorej prechádza do ďalšej *callback* funkcie. V prípade, že funkcia nie je vyvolaná, prechod medzi funkciami je ukončený.

[10] https://expressjs.com

Controller, ktorý odpovedá na validný požiadavok je väčšinov posledná *callback* funkcia v danom reťazci

```
app.get(route_path, ...middlewares, controller.action);
```

Express uľahčuje extrakciu URL parametrov. V prípade, že sa v ceste nachádza parameter, ktorý je potrebné extrahovať, použije sa nasledovný zápis:

```
router.put('/category/:categoryId', function(req, res, next) { ...
```

Ako je z príkladu vidieť, na tento účel sa používa symbol dvojbodky. Následne sú všetky extrahované parametre so zhodným názvom v objekte `request.params`, ktorý je parametrom *callback* funckie. V tomto konkrétnom príklade by bolo `id` kategórie uložené v `req.params.categoryId`. *Express* dokonca podporuje v rámci cesty zadávať regulárne výrazy.

3.4 Architektúra informačných systémov

Martin Flowler sa vo svojej knihe [5] venuje architektúre aplikácií. V nej sa zmieňuje, že pojem architektúra nemá presnú definíciu a veľa ľudí sa tento termín snaží definovať, avšak každý inak. Existujú medzi definíciami isté spoločné prvky. Napríklad vnímanie architektúry ako vysokoúrovňové rozdelenie systému na časti a rozhodnutia, ktoré sú náročné zmeniť v priebehu času. Prevláda názor, že sa nejedná iba o jednu architektúru systému, ale o viacej architektúr v systéme a tento pohľad sa v priebehu životného cyklu systému môže zmeniť. Platí, že ak sa jedná o rozhodnutie, ktoré je v budúcnosti jednoduché zmeniť, nejedná sa o architektonické rozhodnutie.

Výsledkom každej architektúry je systém, ktorý je:

- **Nezávislý od *frameworkov***. Používať ich iba vo forme podporných nástrojov a nie tak, aby sa v budúcnosti mohli stať obmedzením funkčnosti.
- **Testovateľné**. Biznis logiku je možné testovať bez používateľského rozhrania, databázy, webového servera alebo akéhokoľvek iného externého prvku.
- **Nezávislé od používateľského rozhrania**. Používateľské rozhranie sa môže ľahko meniť bez zmeny zvyšku systému. Webové používateľské rozhranie by sa mohlo napríklad nahradiť konzolovým používateľským rozhraním bez zmeny biznis logiky.
- **Nezávislé od databázy**. Možnost vymeniť napr. *SQL Server* za *MongoDB*. Biznis logika nie je viazaná na databázu.
- **Nezávislý od akejkoľvek externej služby**. Biznis logika nevie o vonkajšom svete.

Rozdelenie na vrstvy je jedna z najbežnejších techník, ktoré používajú softvérový architekti pre rozdelenie komplexného systému. *Príklad: sieťové modely sa skladajú z viacerých vrstiev. V prípade prakticky najpoužívanejšieho TCP/IP modelu sú to aplikačná vrstva, transportná vrstva, internetová vrstva a vrstva sieťového rozhrania. Sieťové protokoly sa dajú zaradiť do týchto vrstiev. V praxi sú potom dáta prenášané po sieti rozdelené na menšie dátové jednotky, ku ktorým sa pridávajú hlavičky jednotlivých protokolov. Rôzne sieťové zariadenia spracovávajú protokoly na rôznych vrstvách, od prenosových médií na najnižšej vrstve, cez prepínače, rozbočovače, smerovače až po samotné spracovanie dát softvérom*

bežiacom na procesore počítača. Vrsty v rámci systému je možné predstaviť si ako vrstvy koláča, kde jednotlivé vrstvy predstavujú hlavné subsystémy. Každá vrstva sa opiera o vrstvy uložené pod ňou, využíva ich služby. Spodné vrstvy nevedia o vyšších vrstvách, ale zároveň najvrchnejšia vrstva nevie o najnižšej vrstve. Používa iba vrstvu, s ktorou susedí. Medzi hlavné výhody patria:

- Vrstva je súvislý funkčný celok, ktorý sa dá pochopiť bez znalosti ostatných vrstiev.
- Vrstvy môžeme nahradiť alternatívnymi implementáciami rovnakých základných služieb.
- Minimalizujú sa závislosti medzi vrstvami.
- Vrstva sa po vytvorení môže znovu použiť na mnoho služieb vyššej úrovne.
- Dobré miesto pre štandardizáciu, keďže definijú, ako spolu komunikujú ostatné vrstvy.

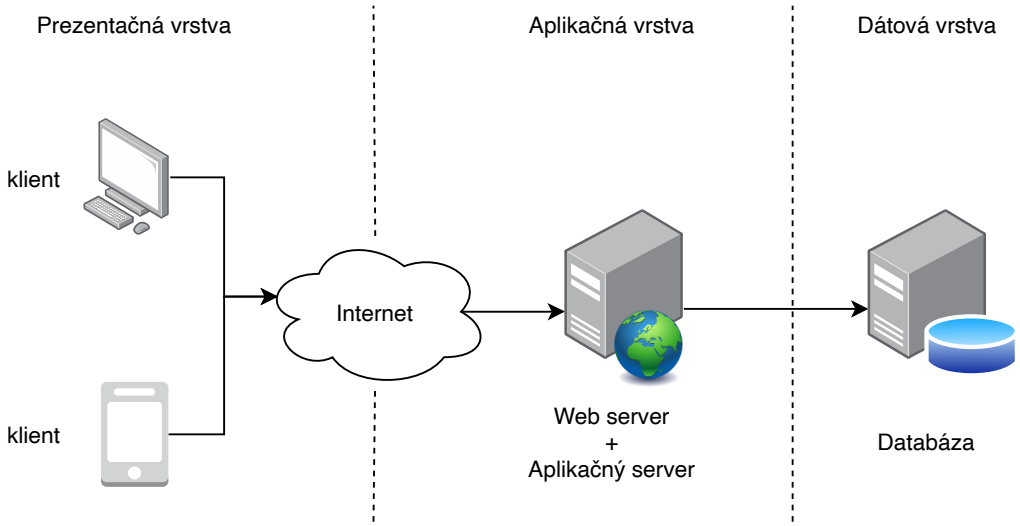
Naopak, za hlavné nevýhody možno považovať:

- Vrstvy nedokážu niektoré veci zapúzdriť správne. Výsledkom sú kaskádové zmeny. Klasickým príkladom je pridanie prvku, ktorý musí byť zobrazený v užívateľskom rozhraní, preto musí byť v databáze, a teda musí byť pridaný aj medzi každú vrstvu.
- Nižší výkon. Každá vrstva zpravidla pracuje s vlastnou reprezentáciou dát, takže je zakaždým potrebná transformácia dát.

N-úrovňová architektúra je klient-server architektúra, v ktorej sú funkcie prezentácie, spracovania aplikácií a správy údajov fyzicky oddelené. N-úrovňová architektúra poskytuje model, pomocou ktorého môžu vývojári vytvárať flexibilné a opakovane použiteľné aplikácie. Rozdelením aplikácie do úrovní vývojári získajú možnosť úpravy alebo pridania konkrétnej vrstvy namiesto prepracovania celej aplikácie. Aj keď sa pojmy vrstva (anglicky *layer*) a úroveň (anglicky *tier*) často zamieňajú a používajú v rovnakom význame, v niektorých literatúrach existujú rozdiely. Vrstva je mechanizmus logického štruktúrovania pre prvky, ktoré tvoria softvérové riešenie, zatiaľ čo úroveň je mechanizmus fyzického štruktúrovania pre systémovú infraštruktúru. Najrozšírenejším využitím viacstupňovej architektúry je trojstupňová architektúra. Trojvrstvá architektúra sa zvyčajne skladá z prezentačnej vrstvy, aplikačnej vrstvy (tiež označovaná ako biznis alebo logická vrstva) a dátovej vrstvy.

Vrstvy architektury (obrázok 3.4):

- **Prezentačná vrstva**. Tá časť, ktorá je viditeľná pre užívateľov. Zaisťuje vstup požiadaviek a prezentáciu výsledkov. Je závislá na platforme (napr. webová aplikácia, aplikácia pre *Windows*, *Android* aplikácia atď.). Môže byť teda rôzna pre rôzne zariadenia či platformy.
- **Aplikačná vrstva**. Prostredná vrstva modelu (*middleware*), zabezpečuje výpočty a operácie vykonávané medzi vstupno-výstupnými požiadavkami a dátami. Tiež nazývaná ako aplikačný server.
- **Dátová vrstva**. Najnižšia vrstva modelu, zaisťuje prácu s dátami, teda s databázový systém a základné dátovo-funkčné operácie zaisťujúce ukladanie, výber, agregáciu, predspracovanie, integritu a audit dát.

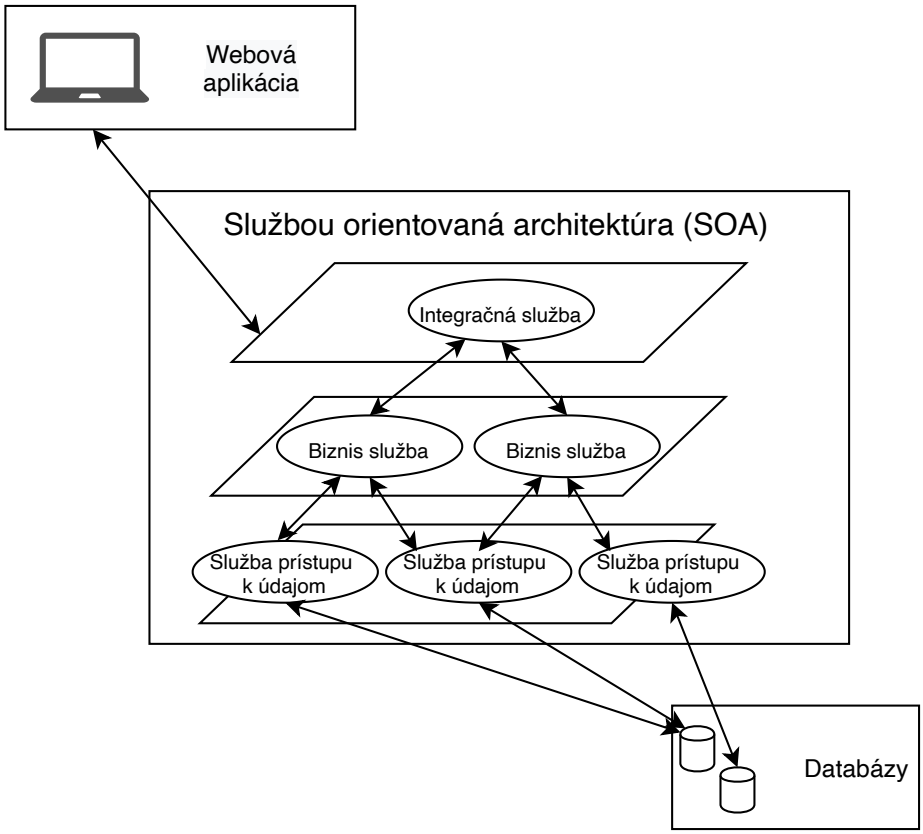


Obr. 3.4: Trojvrstvá architektúra

Konkrétna podoba architektúry sa značne líši od použitia. Trendy v architektúrach pre webové a mobilné aplikácie sa neustále vyvíjajú. Jedným z trendov je použitie službou orientovanej architektúry (*Service Oriented Architecture* – SOA), ktorá je ilustrovaná na 3.5. Aplikácia sa skladá z niekoľkých jednotiek až desiatok služieb podľa zložitosti aplikácie. Každá služba logicky predstavuje biznis aktivitu so špecifickým výsledkom, je samostatne funkčná a prístupná pomocou svojho rozhrania. Vznikajú tak pomerne vysoko nezávislé komponenty, ktoré sú jednoducho udržovateľné. To umožnuje skladať aplikáciu a ďalšie služby z už existujúcich slúžieb prostredníctvom svojho sprístupneného API. Práve tento prístup by mal znížit finančné náklady na vývoj a zvýšiť flexibilitu. SOA presadzuje 6 hlavných hodnôt:

- **Biznis hodnote** sa prikladá väčší význam ako technickej stratégii.
- **Strategickým cieľom** sa prikladá väčší význam ako prínosu pre konkrétny projekt.
- **Vnútornej interoperabilite** sa prikladá väčší význam ako integrácii na mieru.
- **Zdieľaným službám** sa prikladá väčší význam ako vykonávaniu na špecifické účely.
- **Flexibilite** sa prikladá väčší význam ako optimalizácii.
- **Evolučnému zdokonaleniu** sa prikladá väčší význam ako snahe o počiatočnú dokonalosť.

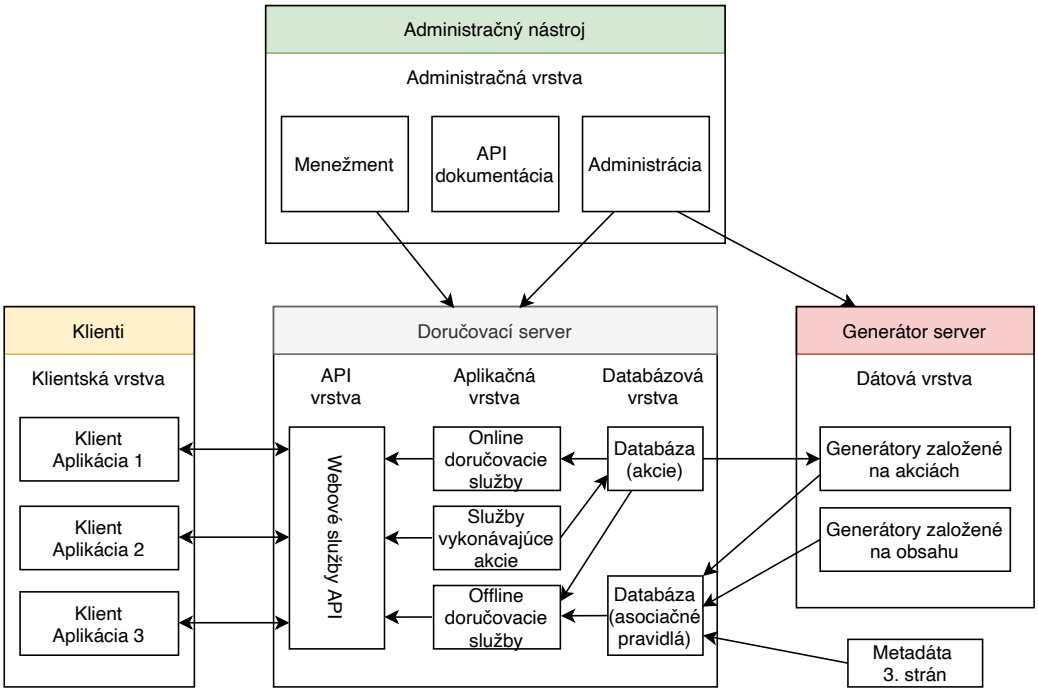
Diagram architektúry webových a mobilných aplikácií je zobrazený na 3.6. Diagram popisuje niekoľko časti, ktoré navzájom medzi sebou komunikujú. Klient (pomocou mobilnej aplikácie, či webového prehliadača) odosiela požiadavok na server. API vrstva spracuje tento požiadavok a ďalej deleguje jej spracovávanie do aplikačnej vrstvy. Aplikačná vrstva spracuje požiadavok, prípadne požiada aplikačnú vrstvu o dáta. Následne sa vracia výsledok klientovi cez aplikačné rozhranie. Dôležitou vlastnosťou je, že klient obsahuje minimum biznis logiky, nakoľko klientov môže byť viac. Preto sa biznis logika do maximálnej možnej miery implementuje práve na servery a eliminuje sa tak duplicitný kód. Zo schémy vidíme, že



Obr. 3.5: Službou orientovaná architektúra (SOA)

okrem klienta a serveru je prítomná časť, ktorá obsahuje vrstvu, ktorej úlohou je dolovanie dát z databázy a poslednou časťou je administračný nastroj, ktorý obsahuje administráciu serveru, či API dokumentáciu.

Ďalším trendom je architektúra jednostránkových aplikácií. Užívateľské rozhranie sa prezentuje pomocou *Javascript* aplikácie. To zostáva nemenné aj po užívateľovej interakcii a pomocou AJAX volaní alebo *Websockets* sa asynchrónne alebo synchrónne vykonávajú požiadavky bez potreby prekreslenia celej stránky. Vždy sa teda zmení iba nevyhnutná časť stránky. Užívateľ tak získa prirodzený zážitok bez preblikávajúcej obrazovky.



Obr. 3.6: Diagram architektúry webovej a mobilnej aplikácie

Kapitola 4

Analýza a návrh riešenia pre galérie a múzeá s využitím rozšírenej reality

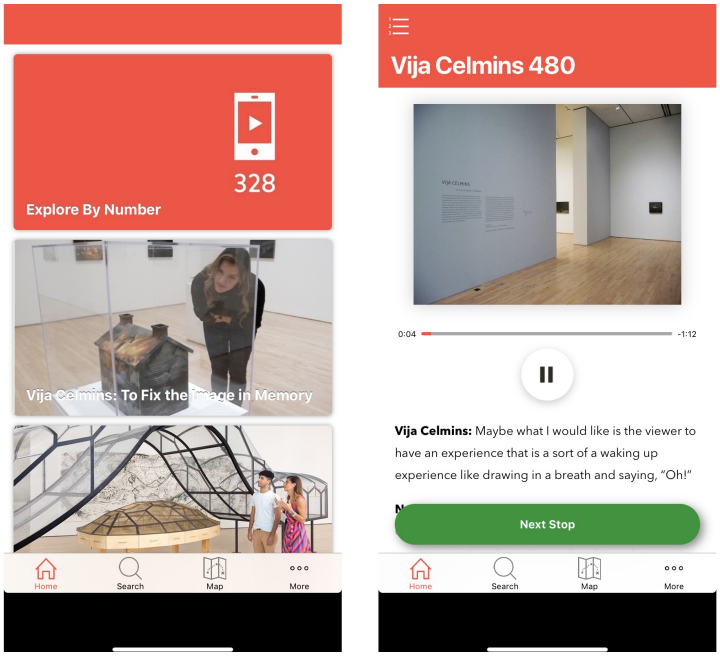
Cieľom práce je vytvoriť systém a aplikácie, ktoré správcovi galérie umožnia spravovať multimediálny obsah k jednotlivým exponátom a návštevníkovi tento obsah pomocou rozšírenej reality zobrazovať. Vytvorenie aplikácie s vhodným užívateľským rozhraním a funcionalitou vyžaduje znalosť ostatných aplikácií, ktoré sa používajú na rovnaký učel, poprípadne majú podobnú funkcionalitu. Tieto aplikácie som analyzoval v prvej časti tejto kapitoly a na základe získaných poznatkov z podobných, už existujúcich aplikácií, v ďalšej časti práce špecifikujem požiadavky pre vytvárané aplikácie a navrhujem celý informačný systém z pohľadu architektúry systému. Na základe požiadavkov navrhujem systém, ktorý sa využije pri práci v rozšírenej realite tak, aby splňoval definované požiadavky. Nasleduje návrh dátových štruktúr a jednotlivé aplikácie sú popísané v nasledujúcich častiach.

4.1 Špecifikácia požiadavkov

Množstvo múzeí a galérií poskytuje audio prehliadky, avšak väčšina z nich používa špeciálne prístroje slúžiace iba na tento jeden účel. Takéto riešenie je finančne nákladné a vyžaduje veľkú počiatočnú investíciu pre zakúpenie potrebného hardvéru a tvorby softvéru podľa presných požiadaviek. Nezanedbateľnou finančnou čiastkou je správa takýchto zariadení, ktoré sa časom opotrebujú, prípadne pokazia. Preto sa nasledujúca časť práce venuje iba prehliadkam, ktoré sú vo forme mobilnej aplikácie dostupné pre užívateľov, ktorí si ich môžu nainštalovať a spustiť vo svojom mobilnom zariadení. Tieto aplikácie analyzujem a na základe ich vlastností navrhujem zlepšenia, prípadne nové, chýbajúce funkcie.

Múzeum moderného umenia v San Francisku (*San Francisco Museum of Modern Art* – SFMOMA) patrí medzi múzeá s najväčšou tržbou v danej lokalite [7]. V roku 2016 vytvorili aplikáciu *SFMOMA Audio* (obrázok 4.1), ktorej cielom je audio sprevádzanie naprieč celým múzeom. Hneď po otvorení aplikácie sa zobrazí možnosť objavovania diela po zadaní čísla, ktoré je umiestnené priamo pri vystavovanom exponáte. Táto funkcia umožnuje návštevníkom veľmi rýchlo vyhľadať exponát bez nutnosti vyhľadávania podľa celého názvu, prípadne vyhľadaním v dlhom zozname. Ďalšou možnosťou je voľba jednej z ponúkaných prehliadok podľa obdobia či témy daného poschodia, respektívne niekoľkých miestností. Po zvolení prehliadky sa zobrazí možnosť spustenia začatia audio prehliadky. Text je do-

plnený o audionahrávky, vďaka čomu nemusí návštevník neustále pozerať do mobilu, ale môže spoznávať exponáty a zároveň sa dozvedať nové informácie. Navrhovaná aplikácia by mala rovnako kombinovať rôzne spôsoby sprístupnenia informácií (audio, video, text...), nakoľko užívateľ si tak môže vybrať preferovanú formu. Následne je zobrazená obrazovka s náhľadom exponátu vo formáte fotky, pod ktorou sa spúšťa zvuková nahrávka, a pod touto nahrávkou je zároveň umiestnený aj jej prepis. Užívatelia, ktorý tak preferujú text, majú možnosť náhrady za audionahrávku. Užívateľovi je umožnené preskočiť daný exponát a manuálne môže vybrať iný exponát, ktorý ho zaujíma. Po skončení nahrávky o jednom exponáte je užívateľ nútený kliknuť na tlačidlo „Pokračovať“ pre posun na ďalšiu nahrávku. Poslednou možnosťou je takzvaná prechádzka umením, kde môže užívateľ po spustení nahrávky vložiť mobil do vrecka a nechať sa navigovať priestormi múzea. Aplikácia pozná na základe *beaconov* presnú polohu užívateľa a napríklad cestu vo výťahu spríjemňuje spustením príjemnej muziky. Toto riešenie je však z finančného hladiska značne náročné a je možné ho nahradiť napríklad použitím AR, ktoré po rozoznaní exponátu vie identifikovať miestnosť, v ktorej sa návštevník nachádza. Aplikácia okrem vyššie spomenutých vlastností obsahuje mapu múzea a základné informácie o cene lístkov či ako sa stať členom. Aplikácia je hodnotená 2,7 hviezdy z možných 5. Užívatelia sa sťažujú na málo prehliadok, nefunkčnú synchronizáciu polohy či neprispôsobenie aplikácie na modely *iPhone X* a vyššie. Týmto problémom je potrebné vyvarovať sa a podporovať čo najväčšie množstvo modelov s dôrazom na najnovšie zariadenia.

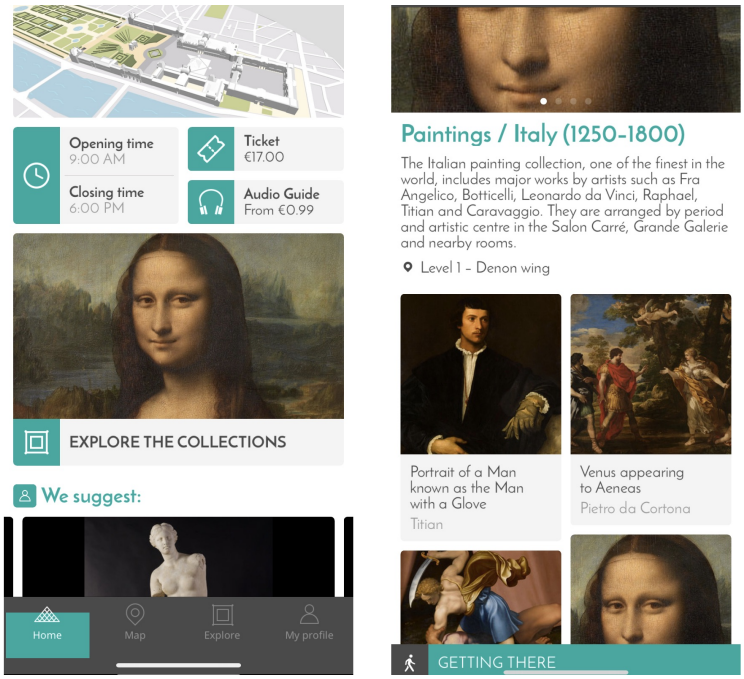


Obr. 4.1: Aplikácia *SFMOMA Audio*[1]

Aplikácia *Louvre* (obrázok 4.2) je oficiálnou aplikáciou tohto rovnomenného francúzskeho múzea. Aplikácia pôsobí príjemným, moderným dojmom. Preto je potrebné, aby navrhovaná aplikácia taktiež poskytovala rovnaký príjemný pocit z používania a využívala najnovšie druhy ovládacích prvkov ako napríklad modálne prezentovanie v rámci iOS 13. Na hlavnej stránke sú prehľadne zobrazené základné informácie – otváracie hodiny, cena

[1] https://apps.apple.com/us/app/sfmoma-audio/id1376087175

lístkov a mapa. Užívateľ vidí na mape fotky diel a ich názvy. Mapa tak zjavne patrí do štandardného obsahu aplikácií pre múzea, čo umožnuje užívateľovi nájsť všetky exponáty, ktoré ho zaujímajú. Okrem toho aplikácia poskytuje prehľad stálych kolekcií, v ktorých užívateľ nájde zoznam exponátov, kde každý exponát obsahuje fotku, názov a rok vzniku. Toto sú minimálne informácie, ktoré by mala navrhovaná aplikácia poskytovať a je ich vhodné doplniť o videá a ďalší multimediálny obsah.



Obr. 4.2: Aplikácia *Musée du Louvre official app*[2]

Na porovnanie oficiálnej aplikácie *Louvre* existuje aplikácia *Louvre Museum Visitor Guide* (obrázok 4.3) od spoločnosti *eTips LTD*, ktorá sa špecializuje na tvorbu aplikácií určených pre prehliadky celých miest či budov. Keďže nejde o oficiálnu aplikáciu, okamžite po otvorení aplikácie je užívateľovi ponúknuté odomknutie platených informácií za 3,49 eura. Bez zakúpenia aplikácia zobrazuje iba všeobecné informácie (do 30 viet) o múzeu. Navrhnutá neplatená aplikácia môže využiť tento priestor po prvom zapnutí aplikácie na zoznámenie užívateľa s aplikáciou a predstavenie funkcií, poprípadne ovládania. Po zakúpení sa užívateľovi odomknú informácie o otváracích hodinách, histórii múzea, praktických informáciách, reštauráciách, obchodoch a spôsobu akým sa môže dostať do múzea. Tieto všeobecné informácie boli dostupné vo všetkým skúmaných aplikáciách a preto nesmú chýbať ani v navrhovanej aplikácií. Aplikácia obsahuje mapu múzea a informácie o jednotlivých oddeleniach múzea. Všetky tieto informácie si môže užívateľ prehrať do sluchátok, hoci hlas pôsobí strojovo. Na hlavnej stránke je dostupný zoznam ostatných aplikácií vyvinutých touto firmou, a po kliknutí je presmerovaný do *AppStoru*. Aplikácia je hodnotená 4,2 hviezdami a medzi hlavné výčitky patrí malé množstvo poskytovaných informácií. Navrhnutá aplikácia by mala poskytovať dostatočné množstvo informácií, aby užívatelia našli všetky potrebné informácie na jednom mieste bez potreby vydávania brožúr apod.

[2] https://www.louvre.fr/en/mediaapps/musee-du-louvre-official-app
[3] https://apps.apple.com/us/app/louvre-museum-visitor-guide/id586619009

Obr. 4.3: Aplikácia *Louvre Museum Visitor Guide*[3]

Na základe analýzy vlastností a kľúčových prvkov existujúcich aplikácií slúžiacich na rovnaký účel, sú vybrané kľúčové funkcie, ktoré takýto typ aplikácie musí poskytovať, prípadne vylepšenia, ktoré zjednodušia prácu užívateľom a poskytnú lepší zážitok z navštívenej výstavy.

Rozšírená realita

V múzeách a galériách dochádza k situáciám, kde viacero ľudí zaujíma jeden konkrétny exponát a vzniká tak situácia, kedy sa okolo jedného exponátu postaví väčší počet návštevníkov. Doplnkové informácie však bývajú častokrát umiestnené vedľa exponátu a umožňujú iba malému počtu návštevníkov dozvedieť sa o vystavovanom exponáte dodatočné informácie. Riešením sú práve mobilné aplikácie, ktoré poskytujú riešenie tohto problému. Každý návštevník má možnosť nájsť tieto informácie vo svojom mobilnom telefóne. Avšak žiadna z existujúcich aplikácií určených pre múzeá a galérie neposkytuje možnosť objavovať exponáty pomocou rozšírenej reality. Rozšírená realita bude hlavnou funkciou vytváranej aplikácie vďaka čomu užívateľom zefektívni prístup k informáciám, čím sa tak aj odlíši od zbytku aplikácií, ktoré sú momentálne dostupné.

Aplikácia pre návštevníka bude schopná kamerou rozoznať exponát, a po namierení telefónu na tento exponát zobrazí detail exponátu spolu s multimediálnym obsahom, ktorý je k tomuto exponátu priradený. Práve táto funkcionalita okrem iného dokáže prepojiť digitálny a skutočný svet, vďaka čomu užívateľom poskytne nový spôsob získavania informácií o exponátoch. Aplikácia bude podporovať zobrazenie obrázkov, videí, audio nahrávok, 3D modelov a textov priamo v okolí tohto rozoznaného exponátu. Rovnako tak bude možné označením obsahu zobraziť obsah na celú obrazovku. Užívateľ tak bude zbavení nutnosti pracného vyhľadávania medzi množstvom dostupných exponátov v dlhom zozname. Stačí mu tak iba namieriť kamerou na objekt a zobrazí sa mu všetok dostupný obsah k danému

exponátu. V prípade, že užívateľ preferuje spôsob vyhľadávania exponátov zo zoznamu podľa kategórií, bude mu umožnený použiť aj tento klasický spôsob.

Keďže aplikácia musí na základe nejakých údajov rozoznávať exponáty, tvorcovi obsahu bude umožnené tieto objekty nasnímať a spravovať. K tomu bude vytvorená druhá aplikácia, ktorá bude slúžiť k tomuto účelu. Dostupné budú 2 režimy – automatický a manuálny. V automatickom režime stačí namieriť telefón na exponát a automaticky sa bez ďalších potrebných úkonov vytvorí sken exponátu, na základe ktorého sa neskôr rozozná nasnímaný objekt. V druhom, manuálnom režime, administrátor vloží exponát do manuálne vytvorené virtuálneho obalu a následne oskenuje tento exponát. V prípade, že sa jedná o plochý objekt (napríklad obraz), stačí poskytnúť fotku takéhoto objektu. Administrátor následne určí obsah, ktorý k danému exponátu prislúcha. Následne je možné prostredníctvom rozšírenej reality konfigurovať polohu, orientáciu a veľkosť pridaného multimediálneho obsahu v 3D priestore v závislosti na polohe naskenovaného objektu. Ďalším spôsobom, ako konfigurovať zobrazenie multimediálneho obsahu bude možnosť priamo meniť hodnoty rotácie, polohy a veľkosti bez nutnosti zobrazenia v rozšírenej realite a teda tieto údaje budú prístupné vo forme formulára. Práca s rozšírenou realitou bude spĺňať pravidlá uvedené v kapitole 2.4.

Základné informácie

Aplikácia bude obsahovať základné informácie o budove (galéria, múzeum...) vrátane lokality, mapy budovy, otváracích hodín či ceny vstupného. Každý exponát môže obsahovať obrázok, zvukový záznam a textový popis. Každému exponátu je priradená kategória, do ktorej patrí a je možné zobraziť všetky exponáty v danej kategórii. Kategória tiež obsahuje obrázok, zvukovú nahrávku a popis kategórie. Exponát bude možné vyhľadať pomocou kódu, ktorý užívateľ spravidla nájde umiestnený v blízkosti exponátu. Aplikácia obsahuje aj klasické fulltextové vyhľadávanie podľa názvu exponátu.

Správca obsahu

Systém bude obsahovať okrem aplikácie pre užívateľa aj dedikované aplikácie (webová a mobilná) pre tvorcu a správcu obsahu. Teoreticky je možné zlúčiť tieto dve aplikácie do jednej keďže pracujú s rovnakými dátami, ale každá aplikácia slúži na úplne iný účel. Hlavným rozdielom je, že rozšírená realita – t.j. tvorba skenov a konfigurácia multimediálneho obsahu vo virtuálnej realite bude prístupná iba z mobilnej aplikácie pre administrátora. Rozdiely sú vyžadované aj v spôsobe zobrazovania dát. Administrátor potrebuje vidieť informácie prehľadne, bez zbytočných rušivých elementov, zatiaľ čo návštevník požaduje peknú aplikáciu, kde je niekedy dobrý dojem z užívateľského rozhrania dôležitejší ako samotný obsah. Pomocou webovej aplikácie bude možné meniť všetok obsah rovnako ako v mobilnej aplikácií. Aplikácia pre správcu bude zabezpečená, takže bude požadovať autentizáciu užívateľským menom a heslom.

4.2 Návrh informačného systému

Pred implementáciou samotného riešenia, ktoré poskytuje užívateľom možnosť zobrazovať dodatočné informácie o exponentoch v múzeách či galériách a zároveň administrátorom poskytuje správu týchto exponátov, je potrebné navrhnúť, z akých častí sa bude takýto systém skladať a ako budú navzájom tieto časti komunikovať. Preto je problém potrebné dekomponovať na jednotlivé časti a k problému pristupovať tradičnou metódou *z dola hore*,

kde sa najprv vytvoria základné stavebné kamene, z nich sa vytvoria ďalšie väčšie a väčšie celky až nakoniec vznikne celá architektúra systému, ktorá je ilustrovaná na obrázku 4.4.

Mobilná aplikácia pre návštevníka plní tri hlavné funkcie, resp. sa skladá z troch častí, ktoré plnia dané funkcie. Prvá časť je automatická detekcia exponátu po namierení kamerou na daný exponát. Ďalšou časťou je zobrazenie multimediálneho obsahu v AR a poslednou je vizualizácia informacií o exponátoch, kategóriách atď. Štandardne sa zvykne táto časť nazývať *frontendom*. Mobilná aplikácie sa skladá z niekoľkých základných vrstiev. Hlavná, najväčšia časť je UI vrstva, ktorá zobrazuje dáta vo forme zrozumitelnej pre používateľa. Táto vrstva je napojená na vrstvu, ktorá sa nazýva logickou vrstvou. Logická vrstva získava dáta prostredníctvom sieťovej vrstvy, ktorá dotazuje dáta zo serveru. Poprípadne získava dáta od dátovej vrstvy, ktorá komunikuje s databázou. Logická vrstva následne získané dáta spracuje a ďalej posúva do UI vrstvy, ktorá dáta zobrazí. Každú vrstvu je možné rozdeliť na ďalšie a ďalšie vrstvy, avšak vyššie definovaný výčet vrstiev je minimálny.

Dáta, ktoré sa zobrazujú návštevníkom v mobilnej aplikácii je potrebné spravovať. K tomuto účelu slúži mobilná aplikácia pre administrátorov. Rovnako sa táto časť nazýva ako *frontend*, hoci aplikácia obsahuje istú mieru logiky napr. pri skenovaní exponátov a konfigurácií multimediálneho obsahu v AR. Prakticky sa jedná o rovnaké rozloženie vrstiev ako v aplikácii pre návštevníkov. UI vrstva, ktorá zobrazuje dáta vo forme vhodnej pre administrátora, ktorá je napojená na logickú vrstvu. Logická vrstva spracováva do UI vrstvy dáta, ktoré získava zo sieťovej vrstvy.

Treťou časťou, ktorá spadá pod *frontend* je webová aplikácia, ktorá umožnuje oprávneným administrátorom spravovať obsah. Vrstvy vo webovej aplikácii spravidla závisia od použitého *frameworku*, avšak aj v tomto prípade možeme definovať základné 3 vrstvy: vrstva služieb, zobrazovacia vrstva a logická vrstva. V niektorých *frameworkoch* je možné logickú vrstvu spojiť so zobrazovacou vrstvou.

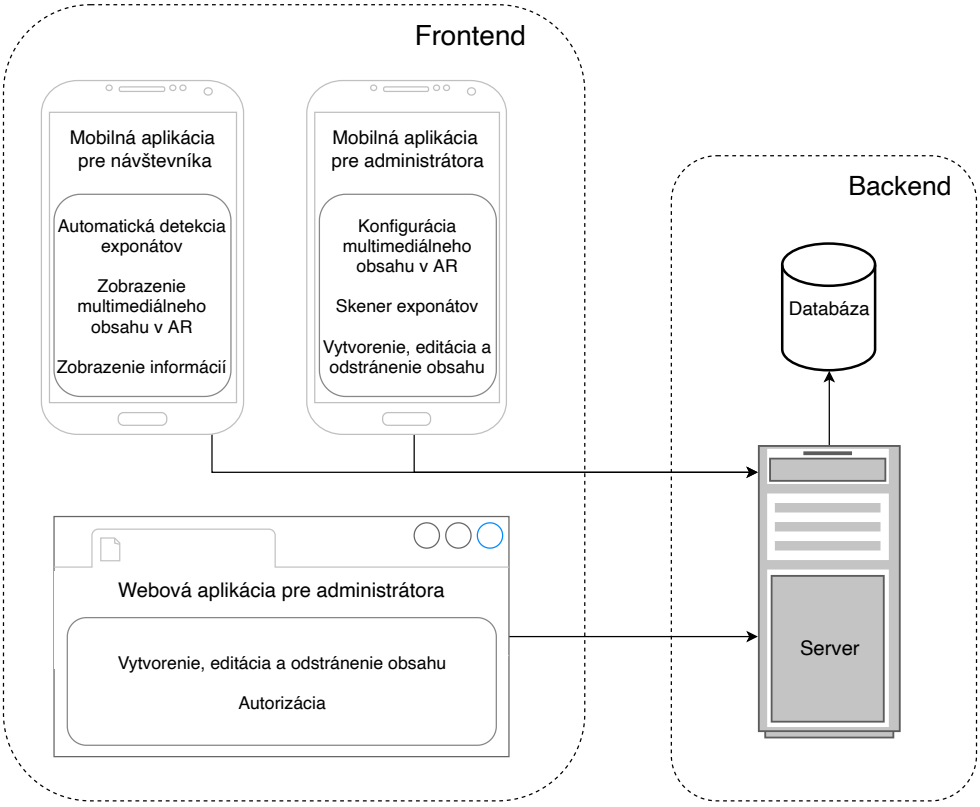
Všetky *frontend* časti komunikujú so serverom prostedníctvom svojej sieťovej vrstvy, resp. všeobecne pomocou vrstvy služieb. Úlohou serveru je spracovávať HTTP požiadavky. Prostredníctvom REST (*Representational State Transfer*) rozhrania (*Application Programming Interface* – API) je poskytnutý prístroj k zdrojom. Server obsahuje definíciu dátových modelov, vrstvu na komunikáciu s databázou a logickú vrstvu, ktorá spracováva požiadavky. Logická vrstva registruje endpointy, spracováva parametre HTTP požiadavku, komunikuje s databázovou vrstvou, transformuje dáta a posiela HTTP odpoveď.

Poslednou časťou je databáza, ktorá slúži na persistenciu dát. Databáza sa nachádza aj v mobilnej aplikácii pre užívateľa. Databáza ukladá údaje o exponátoch, kategóriách exponátov, poschodiach obsahujúcich mapu vo forme obrázku, dostupných jazykov, otváracích hodinách, novinkách, lístkoch, užívateľoch a obecných informáciách o galérii či múzeu.

4.3 Systém rozšírenej reality

Táto práca je zameraná na vytvorenie systému, ktorý poskytuje návštevníkom výstav zobrazovať multimediálny obsah s využitím rozšírenej reality. Tento mediálny obsah je potreba vytvárať, spravovať a distribuovať. Návrh takéhoto systému je zobrazený v schéme 4.6. Tento systém sa skladá z viacerých častí, ktoré je možné od seba oddeliť a vedia tak fungovať nezávisle od seba.

Prvá časť nazvaná *Skener exponátov* slúži k vytvoreniu referenčného objektu, ktorý sa následne používa k detekcii exponátov. Finálny produkt tohto modulu je súbor obsahujúci informácie o skenovanom objekte. Obsahuje tak významné body, ktoré sú detekované na skenovanom objekte, poprípadne v jeho blízkom okolí. Práve vďaka uloženiu týchto bo-

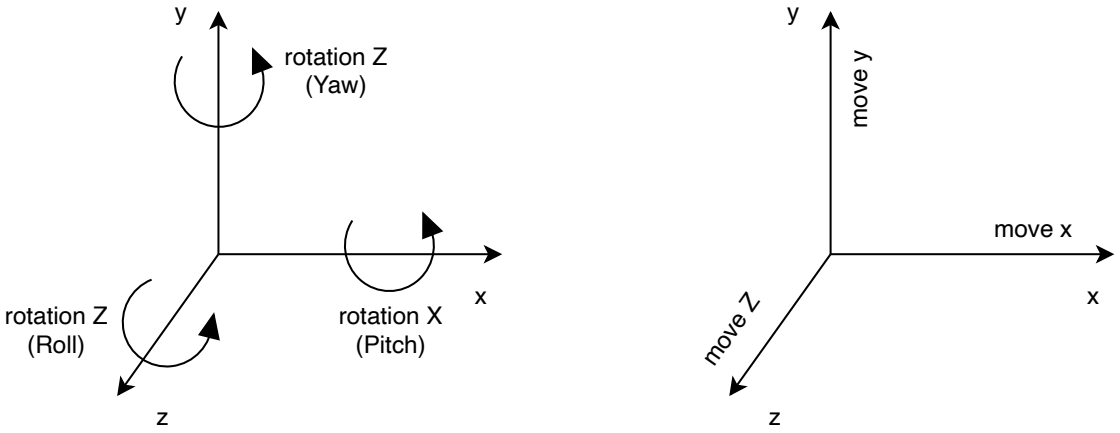


Obr. 4.4: Návrh systému

dov je možné tieto body následne v scéne znovu rozoznať a identifikovať tak objekt, okolo ktorého sa zobrazuje multimediálny obsah. Spolu s významnými bodmi je uložený *pivot* – transformácia medzi súradnicovým priestorom objektu, ktorou sa ovplyvňuje jeho poloha, rotácia a mierka. Platí, že čím viac významných bodov obsahuje referenčný objekt, tým presnejšie, rýchlejšie a stabilnejšie je možné následné rozoznanie objektu. Skener exponátov funguje v dvoch režimoch. Prvý dostupný režim je automatický, kde systém bez asistencie užívateľa detekuje ohraničenie objektu, ktorý sa následne skenuje a iba v rámci tejto vytvorenej obálky objektu sa zaznamenávajú významné body, ktoré sú uložené a ostatné body sú ignorované. Druhý, manuálny režim, vyžaduje od užívateľa pomocou gest definovať ohraničenie skenovaného objektu a až následne sa spustí snímanie bodov v rámci ohraničenia, v ktorom sa nachádza objekt.

Druhá časť, *Konfigurátor multimediálneho obsahu* poskytuje užívateľovi vložiť multimediálny obsah, ktorý je neskôr možné konfigurovať. Tento multimediálny obsah slúži k zobrazeniu doplnkových informácií do okolia detekovaného objektu v rozšírenej realite. Nie je možné vložiť ľubovolný multimediálny obsah, ale iba taký, ktorý je podporovaný, nakoľko každý typ obsahu potrebuje jasnú definíciu toho, akou formou sa má zobraziť. Podporované sú nasledovné typy obsahu: audio nahrávka, video, obrázok, text a 3D model. Keďže jeden exponát môže obsahovať viac obsahu, je potrebné ukladať tento obsah do zoznamu. Každý obsah musí držať informáciu o svojej polohe – 6DoF, tým pádom je možné daný obsah zobraziť podľa želania administrátora na ľubovolné miesto do scény. Táto poloha je definovaná posunutím a rotáciou v závislosti od pozície naskenovaného exponátu. Rotácia a posunutie sú znázornené v obrázku 4.5. Okrem pozície obsah drží informáciu o svojich

rozmeroch, respektívne mierke zobrazenia. Vlastnosti obsahu je možné konfigurovať prostredníctvom formulára, ale hlavne prostredníctvom rozšírenej reality. V prípade využitia rozšírenej reality je v prvom rade potrebné detekovať exponát pomocou referenčného objektu, ktorý je vytvorený *skenerom exponátov*. Následne po úspešnej detekcii a zobrazení multimediálneho obsahu na preddefinovaných pozíciách je nutné označiť obsah, ktorý chce administrátor konfigurovať. Po vyznačení tohto obsahu je možné vybrať druh transformácie. Po vybraní transformácie sú sledované gestá užívateľa a v závislosti na vykonaných gestách sa v reálnom čase mení príslušná vlastnosť obsahu.

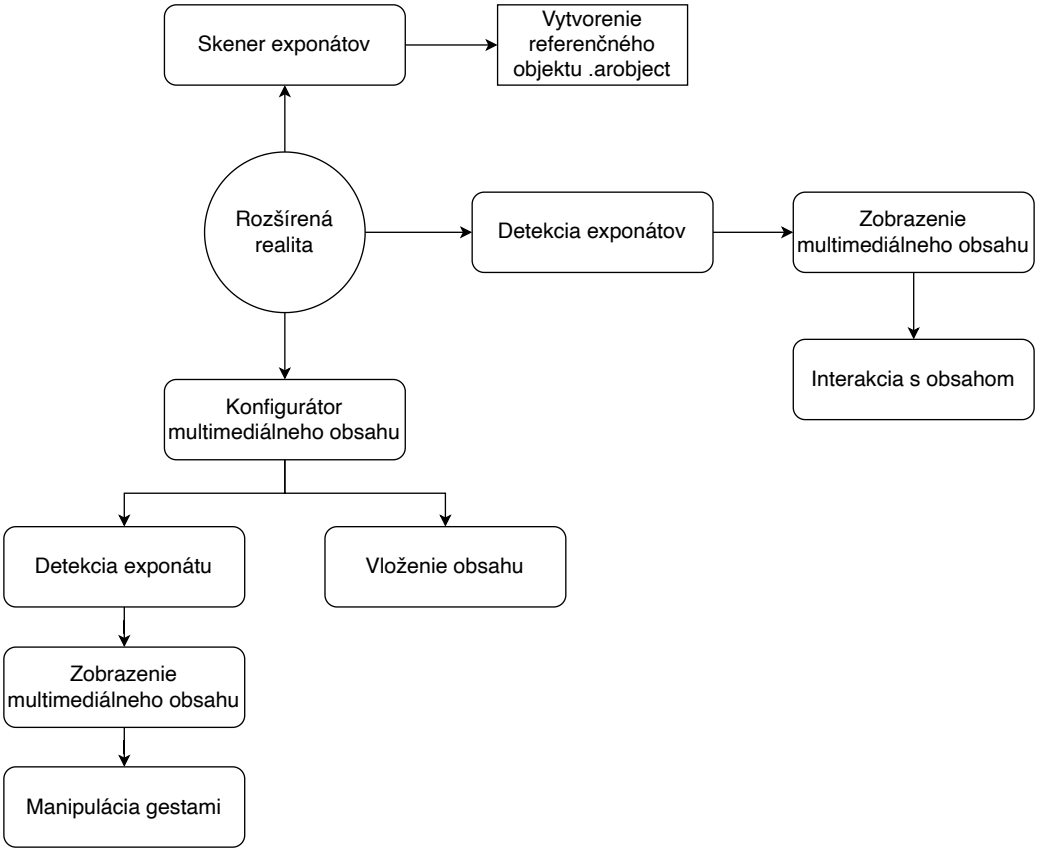


Obr. 4.5: Vľavo rotácia a vpravo posunutie

Poslednou časťou, ktorá je v rozšírenej realite dostupná pre návštevníkov výstav je detekcia exponátov. Kľúčový vstupný prvok, ktorý potrebuje modul *detekcie exponátov* je zoznam všetkých exponátov vrátane ich referenčných objektov, bez ktorých nie je možné tieto exponáty v scéne detekovať. Prvým krokom je spustenie vyhľadávania exponátov v scéne na základe vytvorených referenčných objektov. V prípade, že je nejaký referenčný objekt v scéne nájdený, následuje vyhľadávanie v databáze, ktoré určí, ktorému exponátu je priradený nájdený referenčný objekt. Keďže multimediálny obsah pre všetky exponáty môže dosahovať stovky MB, na zariadení sú uložené iba odkazy na daný obsah. Zatiaľ čo sa sťahuje multimediálny obsah, v rozšírenej realite sa objavuje indikátor znázorňujúci načítavanie pre každý z týchto sťahovaných obsahov a to presne na mieste, kde bude po úspešnom stiahnutí tento obsah zobrazený. Poslednou úlohou tejto komponenty je registrácia gest, kde napríklad v prípade, že užívateľ klikne na ikonu „Prehrať“, začne sa prehrávať daný súbor, poprípadne po dvojkliku sa zobrazí obsah na celú obrazovku. Pre takúto funkcionalitu je potrebné ukladať všetky zobrazené objekty v scéne a momente, kěd užívateľ interaguje so scénou, kontroluje sa, či nebolo kliknuté na nejaký objekt zo scény, ktorý by následne vyvolal akciu. Po detekcii ďalšieho referenčného objektu je všetok obsah prislúchajúci k predchádzajúcemu objektu odstránený zo scény a opakuje sa vyššie popísaný proces pre novodetekovaný exponát.

4.4 Dátová štruktúra

Na základe požiadavkov definovaných na začiatku tejto kapitoly boli definované nasledujúce dátové modely zobrazené na obrázku 4.7.

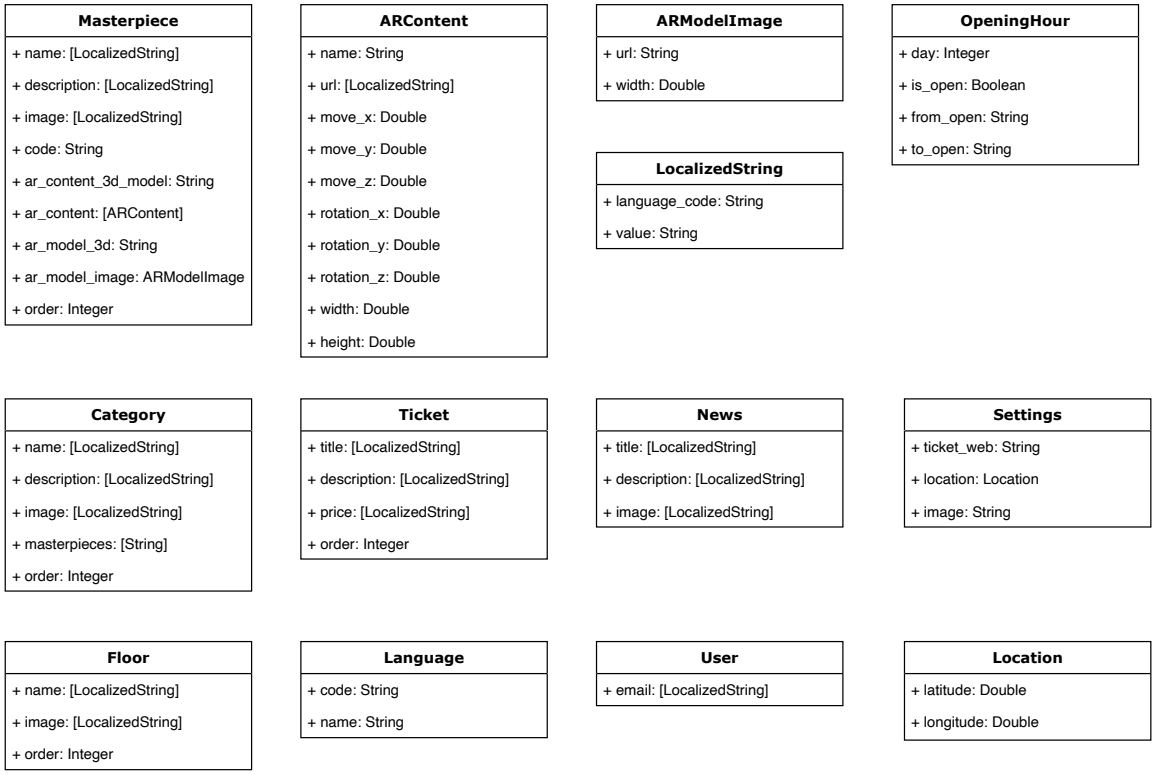


Obr. 4.6: Schéma práce s rozšírenou realitou

Štruktúra `Masterpiece` obsahuje všetky potrebné dáta k zobrazeniu obsahu v rozšírenej realite. Detekcia objektu je na základe 2 parametrov, `ar_model_image` a `ar_model_3d`, v závislosti od typu snímaného objektu. Prvý spomenutý `ar_model_image` je vhodný pre obrazy a iné 2D objekty. Druhý spomenutý `ar_model_3d` obsahuje URL adresu objektu, ktorý je vytvorený prostredníctvom 3D skeneru, ktorý poskytuje mobilná aplikácia pre administrátora. Tento objekt je vo formáte `ARReferenceObject` s koncovkou *.arobject*. Štruktúra `ARContent` obsahuje názov a lokalizované objekty (odkazy na objekty), ktoré sa majú po detekcii zobraziť. Ďalej každý objekt (text, obrázok, zvuková nahrávka) obsahuje výšku a šírku v metroch, rotácie okolo každej osi v stupňoch a posun v každej osi v metroch. Tieto parametre presne definujú umiestnenie objektu v rozšírenej realite a poskytujú absolútnu kontrolu nad zobrazením. Aplikácia podporuje lokalizáciu, ktorú poskytuje štruktúra `LocalizedString`. Táto štruktúra obsahuje kód jazyku a asociovanú hodnotu.

4.5 Mobilná aplikácia pre návštevníka

Diagram prípadu použitia pomáha pri tvorbe užívateľského rozhrania. Definuje sa ním, komu je aplikácia určená a aké funkcie užívateľ od aplikácie požaduje, respektívne za akým účelom aplikáciu spúšťa. Následujúcim krokom je vytvorenie *wireframu* kľúčových obrazoviek mobilnej aplikácie pre návštevníkov, ktoré sú zobrazené na obrázku 4.9. Ako posledný krok návrhu aplikácie je popis jednotlivých obrazoviek s dôrazom na odporúčania *Apple* pri tvorbe aplikácií poskytujúcich rozšírenú realitu, ktoré boli predstavené v kapitole 2.4.



Obr. 4.7: Dátové modely

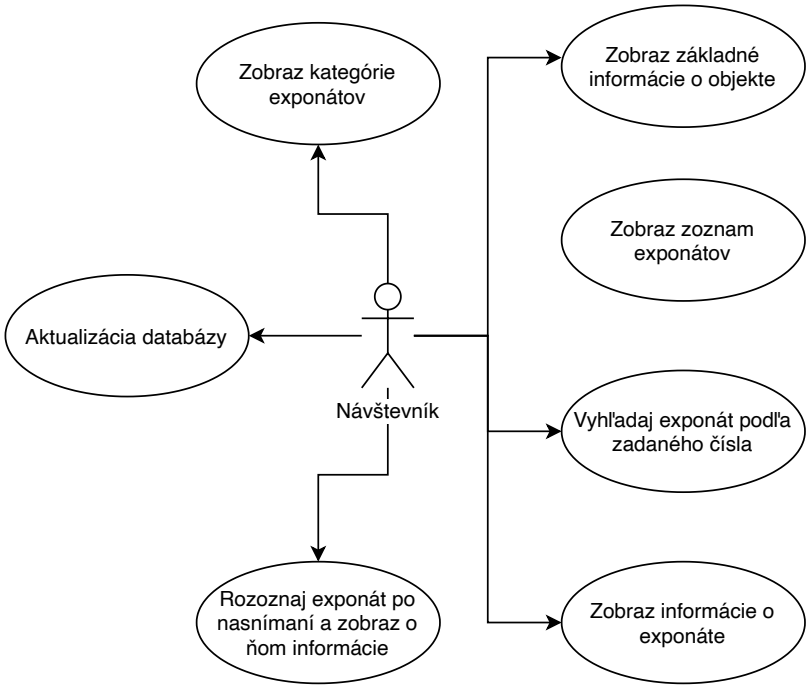
Diagram prípadu použitia

Diagram zobrazený na obrázku 4.8 definuje hlavné fukcie aplikácie očakávané návštevníkom. Bežný užívateľ, teda návštevník, spúšťa aplikáciu za účelom získania dodatočných informácii o exponátoch, ktoré ho v rámci prehliadky galérie či múzea zaujímajú. Hlavnou funkciou je rozoznanie exponátu pomocou kamery telefónu, a následné zobrazenie informácií o rozoznanom exponáte v rozšírenej realite, ktorý popisuje prípad užitia špecifikovaný v prílohe A, označený ako *Prípad užitia č. 1.* Okrem iného, návštevník chce získať základné informácie o otváracích hodinách, cene vstupenky apod. V neposlednom rade chce zobraziť kategoricky usporiadaný zoznam exponátov.

Úvodná obrazovka

Úvodná obrazovka sa líši v závislosti od toho, či je aplikácia spustená po prvýkrát na danom mobilnom telefóne, alebo už bola aplikácia na danom zariadení už aspoň raz spustená.

V prípade prvého spustenia aplikácie sa zobrazí uvítacia obrazovka s prehľadom funkcií, aby užívateľ vedel, aké aplikácia poskytuje funkcie, poprípadne, ako jednotlivé funkcie ovládať. Užívateľ má možnosť tento prehľad preskočiť stlačením tlačidla „Preskočiť“. Medzi prezentované funkcie určite patrí použitie rozšírenej reality, dostupnosť nových exponátov prostredníctvom aktualizácií priamo v aplikácii, základné informácie o objekte a detailné informácie o jednotlivých exponátoch. Jednotlivé prezentované funkcie sa nachádzajú na rozličných obrazovkách, medzi ktorými sa naviguje pomocou štandardného gesta – potiahnutím prstu doľava/doprava, na ktoré je užívateľ z operačného systému *iOS* zvyknutý.



Obr. 4.8: Diagram prípadu použitia

Keďže aplikácia k svojej funkčnosti potrebuje databázu exponátov, ktoré sa vystavujú a ktoré sa majú v rozšírenej realite automaticky rozoznávať, je potrebné ju pred prvým použitím aplikácie stiahnuť, poprípadne neskôr aktualizovať. V tom prípade aplikácia užívateľovi ponúknuté stiahnuť aktuálnu databázu. Ak užívateľ má stiahnutú databázu, ale nemá aktuálnu verziu, môže pokračovať v používaní aplikácie. V prípade, že nemá stiahnutú žiadnu databázu, tento krok nie je možné preskočiť, nakoľko aplikácia by sa bez databázy nedala zmysluplne používať. Po úspešnom stiahnutí databázy sa zobrazí úvodná obrazovka so základnými informáciami o objekte, čo značí, že je vybraná prvá položka z menu. Ak už aplikácia bola spustená a obsahuje aktuálnu databázu, automaticky sa pochopitelne zobrazuje klasický obsah aplikácie.

Menu

Užívateľ držiaci mobil v jednej ruke ovláda celú obrazovku iba prostredníctvo palca. Dnešné mobily sú vysoké a užívateľ má problém dosiahnuť na celú obrazovku pri držaní jednou telefónu jednou rukou. Zaručene je však schopný dosiahnuť na spodok obrazovky, preto je vhodné umiestniť menu do spodnej časti aplikácie. Menu pozostáva z niekoľkých častí (tlačidiel). V prípade aplikácie pre návštevníkov sa jedná o nasledujúce časti (usporiadané zľava): domáca obrazovka (základné informácie), rozoznanie exponátov (rozšírená realita), zoznam kategórií a exponátov a nakoniec mapa objektu.

Základné informácie

Základné informácie sa zobrazujú po spustení aplikácie alebo po vybratí odpovedajúcej položky z menu. Zobrazujú sa informácie o otváracích hodinách, cene vstupného a adresa objektu. Po kliknutí na cenu vstupného je zobrazený detail s kompletným cenníkom. V prí-

pade, že je dostupná webová stránka na kúpu vstupeniek online, v pravom hornom rohu je zobrazené tlačidlo „Kúpiť“, ktoré presmeruje na webovú stránku s kúpou vstupeniek. Užívateľ si tak môže jednoducho kúpiť lístky a vyhnúť sa prípadnej rade na pokladni. Po kliknutí na adresu objektu je zobrazená mapa s možnosťou navigácie. Užívateľ je informovaný otváracích hodínách pre daný deň, poprípadne, ak je zatvorené, zobrazuje sa informácia o najbližšom dni, kedy je otvorené. Nemusí sa tak preklikávať na ďalšiu obrazovku a okamžite bez hľadania vidí aktuálnu informáciu o otvorenosti. Okrem týchto základných informácií je dostupná sekcia zobrazujúca text s aktualitami. Táto sekcia sa zobrazí iba v prípade, že je tento druh obsahu poskytnutý tvorcom obsahu. Pri každom spustení aplikácie sa sťahujú aktuálne informácie potrebné k zobrazeniu úvodnej obrazovky bez nutnosti aktualizovať celú databázu.

Mapa objektu

Užívateľ potrebuje prehľad o rozmiestnení jednotlivých exponátov v budove, aby si mohol naplánovať prehliadku a omylom tak nevynechal niektoré exponáty, ktoré by ho mohli zaujímať. K tomu slúži mapa objektu, ktorá sa zobrazuje po vybratí odpovedajúcej položky z menu. Mapa je statická (jedná sa o obrázok), užívateľ môže gestom priblíženia podľa potreby mapu priblížiť a následne oddialiť. V prípade, že má objekt viac poschodí, zobrazuje sa prepínač poschodí, a po vybratí daného poschodia sa zobrazí príslušné poschodie.

Detekcia exponátu a zobrazenie multimediálneho obsahu

Rozoznanie exponátu pomocou zadnej kamery telefónu sa aktivuje po vybratí odpovedajúcej položky z menu. Užívateľovi sú zobrazené pokyny pre úspešný proces inicializácie *ARKitu*. Textové pokyny sú zobrazené v hornej alebo spodnej často na priesvitnom pozadí. V prípade, že existuje možnosť znázornenia pokynu grafický prvkom, tento prvok sa zobrazí v strede obrazovky. V prípade, že to dáva zmysel, prvok sa zobrazí animovaný. Po namierení na exponát a úspešnom rozoznaní objektu, telefón vydá haptickú odozvu aby upozornil užívateľa na zmenu stavu a zo spodnej časti obrazovky sa zobrazí otvoriteľný detail exponátu. Detail je možné skryť potiahnutím smerom dolu, prípadne zobraziť na celú obrazovku, po potiahnutí detailu k vrchnému okraju displeja. Po detekcii je automaticky vložený dostupný multimediálny obsah do rozšírenej reality. S obsahom je možná interakcia a to dvojklikom na zobrazený multimediálny obsah. Po označení sa tento obsah zobrazí užívateľovi cez celú obrazovku, čo značne zlepšuje čitateľnosť.

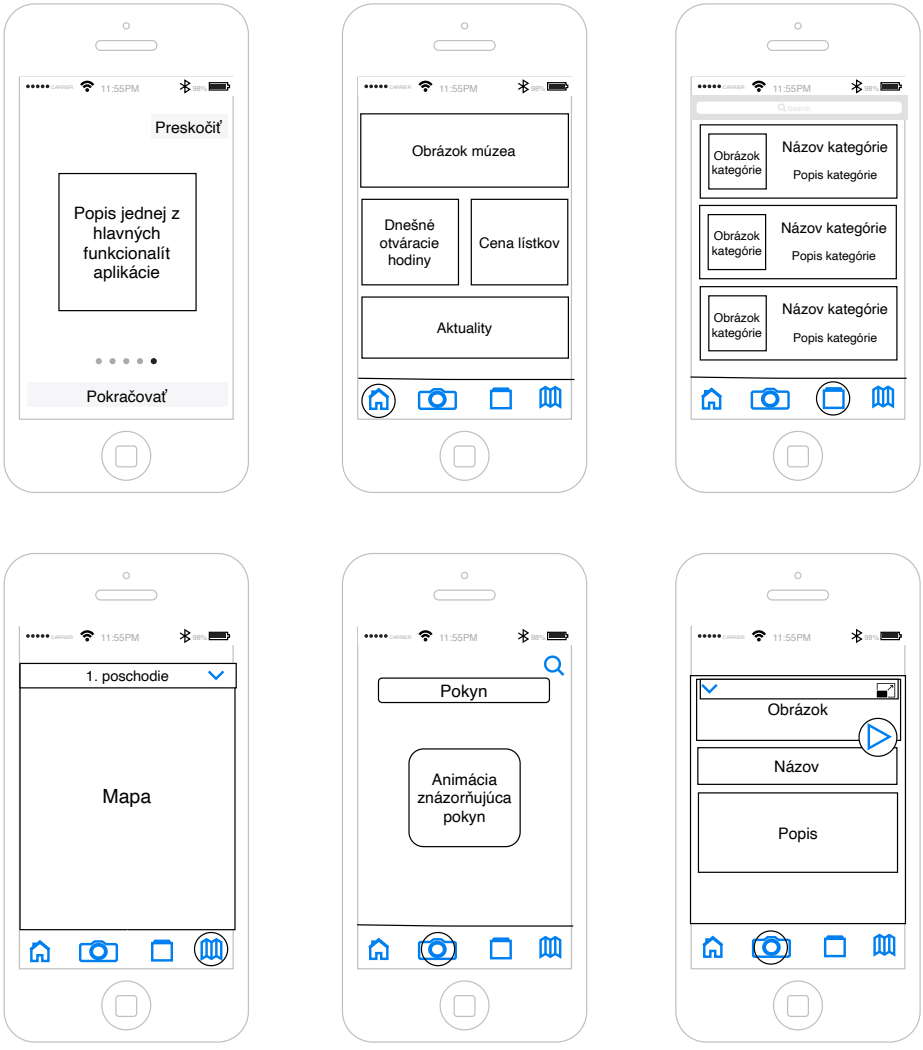
Zoznam kategórií s exponátmi

Nie každý užívateľ preferuje možnost využiť rozšírenú realitu, poprípadne to nemusia dovolovať podmienky (napr. svetelné) v rámci prehliadky. Zoznam kategórií s exponátmi sa zobrazuje po vybratí odpovedajúcej položky z menu. Každá kategória obsahuje názov, obrázok a krátky popis. Tento zoznam je vertikálne posúvateľný. V prípade, že objekt nemá exponáty rozdelené do kategórií, v prvej úrovni sú zobrazené všetky exponáty bez rozdelenia do kategórií. Po kliknutí na kategóriu sa zobrazí detail zvolenej kategórie. Detail obsahuje fotku kategórie, popis kategórie a v neposlednom rade zoznam exponátov, ktoré patria do danej kategórie. V hornej časti je možnosť vyhľadať exponát zadaním číselného kódu alebo názvu, ktorý je zpravidla zobrazený v blízkosti exponátu v realite. Zadanie kódu urýchľuje vyhľadanie oproti klasickej metóde, kedy je potrebný napísať celý názov exponátu. Po za-

daní každého znaku sa aktualizuje zoznam výsledkov. Po kliknutí na exponát sa zobrazuje detail exponátu, aby si užívateľ mohol zobraziť všetky dostupné informácie o exponáte.

Detail exponátu

Detail exponátu je jeden z najdôležitejších obrazoviek celej aplikácie a obsahuje podľa dostupnosti údajov k danému exponátu fotku, názov, popis (napr. prepis nahrávky alebo iný ľubovoľný text), audio nahrávku a video nahrávku. Zvukovú náhravku je možné pozastaviť, prípadne spustiť od začiatku. V prípade, že je dostupný 3D model objektu, užívateľ má možnosť umiestniť tento 3D model do rozšírenej reality.



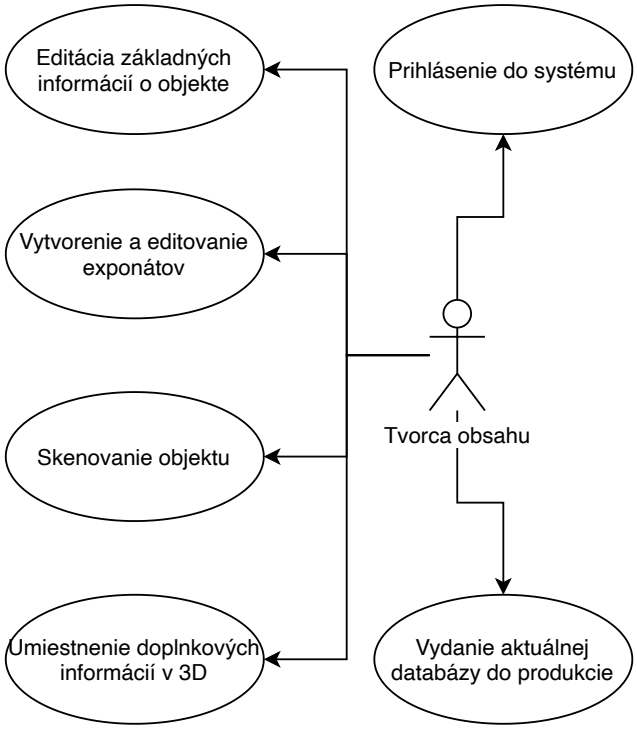
Obr. 4.9: *Wireframy* vybraných obrazoviek mobilnej aplikácie

4.6 Mobilná aplikácia pre administrátora

Rovnako ako pri návrhu mobilnej aplikácie pre návštevníka, aj v prípade aplikácie určenej administrátorovi je vhodné začať vytvorením diagramu prípadu použitia. Následne je potrebné navrhnúť jednotlivé obrazovky vyplývajúce z diagramu prípadu použitia.

Diagram prípadu použitia

Diagram zobrazený na obrázku 4.10 zobrazuje hlavné fukcie aplikácie. Tvorca obsahu chce vytvárať, editovať a mazať informácie o vystavovaných exponátoch. Prípad užitia vytvorenia exponátu je špecifikovaný v prílohe A, označený ako *Prípad užitia č. 2*. Prakticky sa jedná o veľmi podobnú aplikáciu ako pre návštevníka, ale zobrazenie pre administrátora poskytuje možnosť editovať položky. Okrem editácie položiek je možné pomocou rozšírenej reality vytvárať skeny objektov a konfigurovať k nim multimediálny obsah.



Obr. 4.10: Diagram prípadu použitia

Prihlásenie

Táto funkcionalita je dostupná iba v aplikácii určenej pre administrátorov. Pre správu objektu je potrebné prihlásenie, aby bolo možné ochrániť zmenu dát pred nepovolanými osobami. Prihlásenie je dostupné prostredníctvom prihlasovacieho mena, resp. emailu a hesla. Aplikácia disponuje funkciou automatického prihlásenia, takže v prípade, že administrátor sa už aspoň raz prihlásil, aplikácia sa ho pokúsi automaticky znovu prihlásiť.

Menu

Rovnako ako v mobilnej aplikácií pre návštevníkov, menu sa zobrazuje pre ľahkú dostupnost v spodnej časti aplikácie. Pozostáva z niekoľkých častí. V prípade aplikácie pre administrátora sa jedná o nasledujúce časti (usporiadané zľava): domáca obrazovka (základné informácie), mapa objektu, zoznam kategórií a exponátov a nakoniec nastavania.

Základné informácie

Dôraz je kladený na jednoduché a prehľadné zobrazenie editovateľných informácií. Základné informácie sa zobrazujú po spustení aplikácie alebo po vybratí odpovedajúcej položky z menu. Obrazovka umožňuje zmeniť hlavný obrázok, informácie o vstupenkách vrátane webovej adresy pre kúpu vstupeniek, informácie o otváracích hodinách, novinkách a lokalite objektu.

Mapa objektu

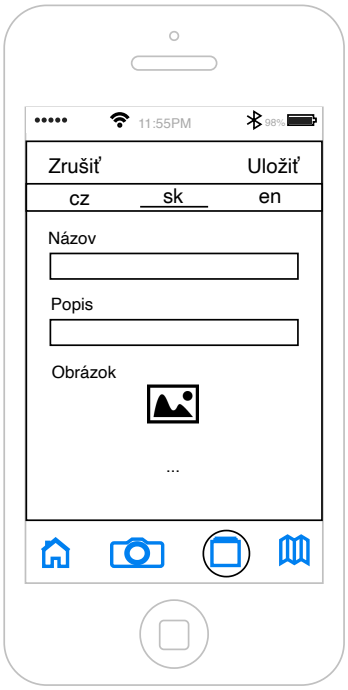
Mapa objektu sa zobrazuje po vybratí odpovedajúcej položky z menu. Aplikácie pre správcu umožnuje pridávanie, editovanie a mazanie mapy pre každé poschodie. Poschodie je možné zmazať štandardným gestom – potiahnutím prstu zprava do ľava. Pre každý dostupný jazyk je možné zvoliť iný obrázok mapy, čím je užívateľom následne ponúknutá možnosť zobrazovať mapu v preferovanom jazyku. Okrem názvu poschodia je možné zvoliť aj poradie poschodí, v akom sa majú užívateľovi zobrazovať. Toto poradie nie je závislé na jazyku.

Konfigurácia multimediálneho obsahu

Táto funkcionalita je podobná rozoznaniu exponátu pomocou kamery, avšak dostupná je iba v aplikácii určenej pre administrátorov. Pre konfiguráciu multimediálneho obsahu je v prvom rade potrebné detekovať exponát. Po rozoznaní exponátu sa zobrazí dostupný multimediálny obsah. Konfigurácia prebieha jednotlivo obsah po obsahu. Najprv je potrebné označit editovaný obsah. Dvojlikom na jeden z obsahov sa vybraný obsah svetelne odlíši od ostatných, resp. ostatné prvky zblednú. Po stlačení konfiguračného tlačidla v pravom hornom rohu je zobrazený výber vlastností, ktoré je možné meniť pomocou gest. Po vybraní jednej z možností sa po aplikovaní daného gesta okamžite aplikujú zmeny na vybranom obsahu. Takto je možné konfigurovať rotáciu, veľkosť a umiestnenie obsahu v závislosti od polohy exponátu.

Správa exponátov a kategórií

Spravovanie exponátov a kategórií sa zobrazuje po vybratí odpovedajúcej položky z menu. Exponát a kategóriu je možné pridať, editovať alebo odstrániť (potiahnutím prstu zprava doľava). Aby bolo možné zadávať lokalizované hodnoty parametrov, je možné meniť jazyk podľa dostupných jazykov, ktoré je možné nastaviť. Detail exponátu (obrázok 4.11) obsahuje obsahuje lokalizovaný názov, popis a obrázok. Poradie a kód je rovnaký naprieč všetkými jazykmi. Rovnako tak 3D model a 2D obrázok slúžiaci k detekcii exponátu, na základe ktorého sa identifikuje daný exponát. K 2D obrázku je vhodné nastaviť šírku reálneho obrazu. Posledným jazykovo nezávislím je polom je 3D model exponátu, ktorý je možné umiestniť do rozšírenej reality. Administrátor môže neobmedzený počet obsahu, ktorý sa zobrazí pri detekcii objektu návštevníkom galérie či múzea. Tomuto obsahu je možné upraviť pozíciu, na ktorej sa má v závislosti na detekovanom exponáte zobraziť. Konkrétne sa jedná o posunutie v smere osy x, y, z a rotáciu okolo osy x, y, z. Kategórie, obdobne ako exponáty, obsahujú názov, popis, obrázok a poradie nezávislé od jazyku. V detaile kategórie je možné priradiť, ktoré exponáty patria do danej kategórie. Použitie kategórií nie je povinné.



Obr. 4.11: *Wireframe* detailu exponátu

Nastavenia

Nastavenia sa zobrazujú po vybratí odpovedajúcej položky z menu. Administrátorovi je v nastaveniach umožnené povoliť prístup ku kamere, ktorý je potrebný pre skenovanie objektov pri spravovaní exponátu. Administrátor v nastaveniach spravuje jazyky, ktoré dané múzeum či galéria podporuje. Kľúčovou funkciou je zverejnenie aktuálnej verzie databázy ako produkčnej databázy, ktorá sa zobrazuje všetkým užívateľom aplikácie pre návštevníkov. Pre aplikovanie tejto funkcie je potrebné potvrdenie akcie. V neposlednej rade sa v nastaveniach nachádza možnosť odhlásenia.

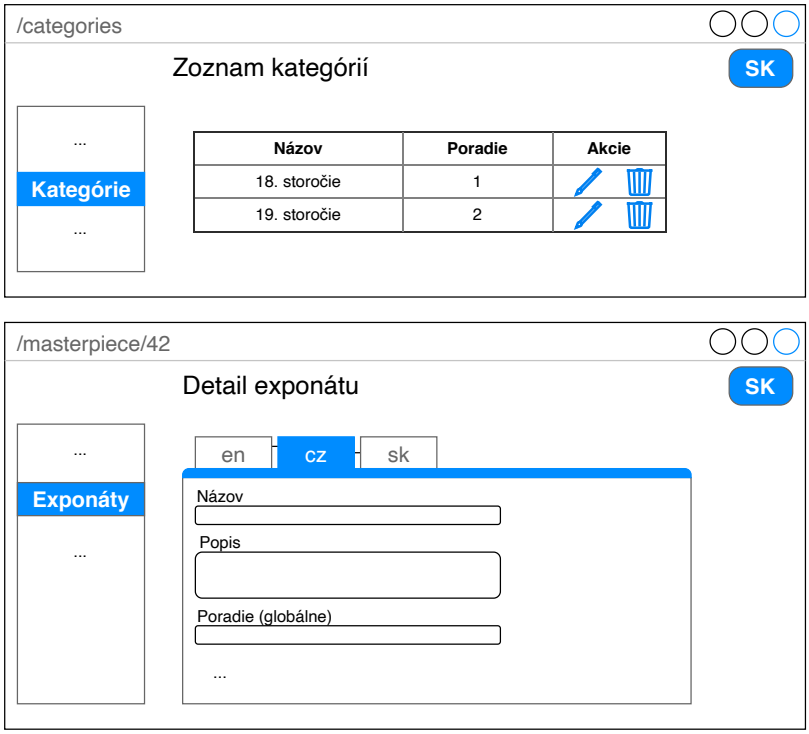
Identifikácia a skenovanie objektu

Administrátorovi musí byť umožnené vytvárať skeny objektov, podľa ktorých sa neskôr rozoznávajú objekty a zobrazuje sa multimediálny obsah. V rámci detailu exponátu po kliknutí na „3D model“ sa zobrazuje možnosť vložiť 3D model zo súborového systému alebo nasnímať 3D model. Po kliknutí na „Nasnímať 3D model“ sa zobrazí nová obrazovka bez zbytočných elementov (napr. schované menu v spodnej časti obrazovky) z dôvodu, aby bolo možné na čo najväčšej ploche možné skenovať exponát. Táto obrazovka sa dá ukončiť iba stlačením tlačidla v ľavom hornom rohu, aby sa predišlo strate údajov pri nesprávnom detekovaní gesta späť. Užívateľ bude textovo a graficky navigovaný. Umiestnenie a použitie elementov na pokyny sa riadi rovnakými pravidlami, ako pri identifikácii exponátu v prípade návštevníka. Predtým, ako začne skenovanie objektu, aplikácia musí identifikovať rozmery tohto objektu. Predvolený je automatický režim, ktorý je možné prepnúť na manuálny, prípadne proces detekcie začať znovu. V prípade manuálneho režimu užívateľ pohybmi prstov zväčšuje/zmenšuje virtuálny objekt tak, aby sa skenovaný objekt nachádzal v strede tohto virtuálneho objektu. Potvrdzujúcim tlačidlom sa začína skenovanie objektu, kde je

potrebné užívateľa grafickou animáciou upozorniť, že sa má okolo predmetu pohybovať tak, aby nasnímal objekt zo všetkých strán, ktoré sú prístupné návštevníkom. Po nasnímaní exponátu zo všetkých strán, prípadne po manuálnom ukončení, je následne tento sken uložený. V prípade automatického režimu stačí namieriť na skenovaný objekt, obkrúžiť okolo tohto objektu a ukončiť skenovanie, ktoré rovnako ako pri manuálnom režime uloží sken a odošle na server.

4.7 Webová aplikácia pre administrátora

Predchádzajúce časti sa zameriavali na užívateľské rozhranie a funkcie mobilných aplikácií. Nasledujúca časť sa venuje špecifikám webovej aplikácie, ktorá je prístupná výhradne administrátorom. Preto je potrebné užívateľa autorizovať. Po vložení prihlasovacích údajov (email a heslo) je užívateľovi zobrazená predvolená hlavná stránka na vkladanie a úpravu exponátov. V pravom hornom rohu je možnosť zmeny jazyka. Na ľavej strane je zobrazené hlavné menu, ktoré obsahuje nasledujúce položky: exponáty, podporované jazyky, otváracie hodiny, novinky, lístky, mapa, kategórie, zverejnenie databázy, nastavenia a odhlásenie. Webová aplikácia poskytuje prakticky rovnaké funkcie ako mobilná aplikácie pre administrátora s výnimkou tvorby 3D modelu pre detekciu exponátu, ktorú webová aplikácie neobsahuje. Výhodou webovej aplikácie je spravidla beh na väčšom displeji, takže práca je jednoduchšia, prehľadnejšia a vo výsledku rýchlejšia. Viacjazyčnosť na jednotlivých obrazovkách, napríklad pri detaile exponátu, je poskytnutá prostredníctvom prepínania medzi štítkami v hornej lište.



Obr. 4.12: *Wireframy* vybraných obrazoviek webovej aplikácie

Dostupnosť dát zo serveru

Pre dostupnosť nových exponátov nie je potrebná aktualizácia aplikácie. Všetky potrebné dáta sú uložené vzdialene na serveri, a tak jediným potrebným krokom je potvrdenie stiahnutia nových dát pri štarte aplikácie, prípadne aktualizácia databázy budovy prostredníctvom domovskej obrazovky. Pri prvom spustení aplikácie je potrebné stiahnuť databázu, inak je užívateľovi zamedzené v užívaní aplikácie.

Ako už bolo spomenuté, dáta zo serveru sú dostupné pre aplikáciu pomocou REST API. Server na základe požiadavkov zaslaných z aplikácie, požiadavok spracuje, vyhľadá dáta v databázi a vygeneruje odpoveď vo formáte JSON (*JavaScript Object Notation*), ktorá je následné odoslaná späť aplikácii. Nižšie sú uvedené všetky dostupné koncové body (*endpointy*), ktoré poskytuje server spolu s uvedenými parametrami a spôsobom použitia.

Autentifikácia

- `POST {{apiUrl}}/api/users.` Používa sa pri registrácii administrátora. Do tela sa vkladá objekt `UserRequestEntity`, ktorý obsahuje email a heslo. Odpoveď obsahuje objekt `UserResponseEntity`, ktorý obsahuje email a token.
- `POST {{apiUrl}}/api/users/login.` Používa sa pri vstupe užívateľa do systému. Do tela sa vkladá objekt `UserRequestEntity`. Odpoveď obsahuje objekt `UserResponseEntity`.
- `GET {{apiUrl}}/api/user.` Používa sa pri overení užívateľa, typicky pri automatickom prihlásení. Hlavička obsahuje autentifikačný token. Odpoveď obsahuje objekt `UserResponseEntity`.

Novinky

Server poskytuje prístupový bod `{{apiUrl}}/api/news`.

- `GET.` Používa sa pri získaní noviniek. Odpoveď obsahuje pole objektov `NewsResponseEntity`, ktorý obsahuje identifikátor, lokalizované nadpisy, lokalizované popisky, lokalizované adresy obrázkov a dátum vytvorenia.
- `PUT.` Používa sa pri vytvorení novinky. Do tela sa vkladá objekt `NewsRequestEntity`, ktorý obsahuje lokalizované nadpisy, lokalizované popisky a lokalizované adresy obrázkov. Hlavička obsahuje autentifikačný token. Odpoveď obsahuje objekt `NewsResponseEntity`.
- `UPDATE.` Používa sa pri aktualizácii novinky. Parameter je identifikátor novinky, ktorá sa aktualizuje. Do tela sa vkladá objekt `NewsRequestEntity`. Hlavička obsahuje autentifikačný token. Odpoveď obsahuje objekt `NewsResponseEntity`.
- `DELETE.` Používa sa pri odstránení novinky. Parameter je identifikátor novinky, ktorá sa odstráni. Hlavička obsahuje autentifikačný token.

Otváracie hodiny

Server poskytuje prístupový bod `{{apiUrl}}/api/openingHours`.

- `GET.` Používa sa pri získaní otváracích hodín. Odpoveď obsahuje objekt `OpeningHoursResponseEntity`, ktorý obsahuje pole objektu `OpeningHours`. `OpeningHours` obsahuje index dňa, čas „otvorené od“, čas „otvorené do“ a indikátor otvorenia.

- `PUT`. Používa sa pri vytvorení novinky. Do tela sa vkladá objekt `NewsRequestEntity`, ktorý obsahuje lokalizované nadpisy, lokalizované popisky a lokalizované adresy obrázkov. Hlavička obsahuje autentifikačný token. Odpoveď obsahuje objekt `NewsResponseEntity`.

Vstupné
Server poskytuje prístupový bod `{{apiUrl}}/api/tickets`.

- `GET`. Používa sa pri získaní vstupného. Odpoveď obsahuje pole objektov `TicketResponseEntity`, ktorý obsahuje identifikátor, lokalizované názvy, lokalizované popisky, lokalizovanú cenu a poradie.

- `PUT`. Používa sa pri vytvorení vstupného. Do tela sa vkladá objekt `TicketRequestEntity`, ktorý obsahuje lokalizované názvy, lokalizované popisky, lokalizované ceny a poradie. Hlavička obsahuje autentifikačný token. Odpoveď obsahuje objekt `TicketResponseEntity`.

- `UPDATE`. Používa sa pri aktualizácii vstupného. Parameter je identifikátor vstupného, ktoré sa aktualizuje. Do tela sa vkladá objekt `TicketRequestEntity`. Hlavička obsahuje autentifikačný token. Odpoveď obsahuje objekt `TicketResponseEntity`.

- `DELETE`. Používa sa pri odstránení vstupného. Parameter je identifikátor vstupného, ktorá sa odstráni. Hlavička obsahuje autentifikačný token.

Podporované jazyky
Server poskytuje prístupový bod `{{apiUrl}}/api/languages`.

- `GET`. Používa sa pri získaní podporovaných jazykov. Odpoveď obsahuje pole objektov `LanguageResponseEntity`, ktorý obsahuje identifikátor, kód a lokalizovaný názov.

- `PUT`. Používa sa pri vytvorení jazyka. Do tela sa vkladá objekt `LanguageRequestEntity`, ktorý obsahuje kód a lokalizovaný názov. Hlavička obsahuje autentifikačný token. Odpoveď obsahuje objekt `LanguageResponseEntity`.

- `UPDATE`. Používa sa pri aktualizácii jazyka. Parameter je identifikátor jazyka, ktorý sa aktualizuje. Do tela sa vkladá objekt `LanguageRequestEntity`. Hlavička obsahuje autentifikačný token. Odpoveď obsahuje objekt `LanguageResponseEntity`.

- `DELETE`. Používa sa pri odstránení jazyka. Parameter je identifikátor jazyka, ktorý sa odstráni. Hlavička obsahuje autentifikačný token.

Mapa budovy
Server poskytuje prístupový bod `{{apiUrl}}/api/floors`.

- `GET`. Používa sa pri získaní mapy budovy. Odpoveď obsahuje pole objektov `FloorResponseEntity`, ktorý obsahuje identifikátor, lokalizovanú adresy obrázkov, lokalizované názvy a poradie.

- `PUT`. Používa sa pri vytvorení novej mapy. Do tela sa vkladá objekt `FloorRequestEntity`, ktorý obsahuje lokalizované adresy obrázkov, lokalizované názvy a poradie. Hlavička obsahuje autentifikačný token. Odpoveď obsahuje objekt `FloorResponseEntity`.

- `UPDATE`. Používa sa pri aktualizácii mapy. Parameter je identifikátor mapy, ktorá sa aktualizuje. Do tela sa vkladá objekt `FloorRequestEntity`. Hlavička obsahuje autentifikačný token. Odpoveď obsahuje objekt `FloorResponseEntity`.

- `DELETE`. Používa sa pri odstránení mapy. Parameter je identifikátor mapy, ktorá sa odstráni. Hlavička obsahuje autentifikačný token.

Vystavované objekty
Server poskytuje prístupový bod `{{apiUrl}}/api/masterpieces`.

- `GET`. Používa sa pri získaní vystavovaných objektov. Odpoveď obsahuje pole objektov `MasterpieceResponseEntity`, ktorý obsahuje identifikátor, lokalizovanú adresy obrázkov, lokalizované názvy, lokalizované popisky, adresa 3D modelu pre detekciu objektu, adresa obrázku pre detekciu objektu s jeho šírkou, 3D obsah, pole objektov `ARContent`, kód a poradie. `ARContent` obsahuje názov, výšku (v metroch), šírku (v metroch), posun (v metroch) v smere osi X, Y, Z, rotáciu (v stupnoch) okolo osi X, Y, Z, zväčšenie a lokalizované adresy obsahov.

- `PUT`. Používa sa pri vytvorení vystavovaného objektu. Do tela sa vkladá objekt `MasterpieceRequestEntity`, ktorý obsahuje identifikátor, lokalizovanú adresy obrázkov, lokalizované názvy, lokalizované popisky, adresa 3D modelu pre detekciu objektu, adresa obrázku pre detekciu objektu s jeho šírkou, 3D obsah, pole objektov `ARContent`, kód a poradie. Hlavička obsahuje autentifikačný token. Odpoveď obsahuje objekt `MasterpieceResponseEntity`.

- `UPDATE`. Používa sa pri aktualizácii vystavovaného objektu. Parameter je identifikátor objektu, ktorý sa aktualizuje. Do tela sa vkladá objekt `MasterpieceRequestEntity`. Hlavička obsahuje autentifikačný token. Odpoveď obsahuje objekt `MasterpieceResponseEntity`.

- `DELETE`. Používa sa pri odstránení objektu. Parameter je identifikátor objektu, ktorý sa odstráni. Hlavička obsahuje autentifikačný token.

Kategórie
Server poskytuje prístupový bod `{{apiUrl}}/api/categories`.

- `GET`. Používa sa pri získaní kategórií. Odpoveď obsahuje pole objektov `CategoryResponseEntity`, ktorý obsahuje identifikátor, lokalizovanú adresy obrázkov, lokalizované názvy, lokalizované popisky, poradie a pole objektov `Masterpiece`, ktorý obsahuje identifikátor a lokalizované názvy.

- `PUT`. Používa sa pri vytvorení kategórie. Do tela sa vkladá objekt `CategoryRequestEntity`, ktorý obsahuje identifikátor, lokalizovanú adresy obrázkov, lokalizované názvy, lokalizované popisky, poradie a pole identifikátorov vystavovaných objektov. Hlavička obsahuje autentifikačný token. Odpoveď obsahuje objekt `CategoryResponseEntity`.

- `UPDATE`. Používa sa pri aktualizácii kategórie. Parameter je identifikátor kategórie, ktorá sa aktualizuje. Do tela sa vkladá objekt `CategoryRequestEntity`. Hlavička obsahuje autentifikačný token. Odpoveď obsahuje objekt `CategoryResponseEntity`.

- `DELETE`. Používa sa pri odstránení kategórie. Parameter je identifikátor objektu, ktorý sa odstráni. Hlavička obsahuje autentifikačný token.

Nastavenia

Server poskytuje prístupový bod `{{apiUrl}}/api/settings`.

- `GET`. Používa sa pri získaní nastavení. Odpoveď obsahuje objekt `SettingsResponseEntity`, ktorý obsahuje identifikátor, webovú stránku pre kúpu lístkov, lokáciu objektu (zemepisná šírka a zemepisná výška) a adresu hlavného obrázka.
- `PUT`. Používa sa pri zmene nastavení. Do tela sa vkladá objekt `SettingsRequestEntity`, ktorý obsahuje webovú stránku pre kúpu lístkov, lokáciu objektu (zemepisná šírka a zemepisná výška) a adresu hlavného obrázka. Hlavička obsahuje autentifikačný token. Odpoveď obsahuje objekt `SettingsResponseEntity`.

Zverejnenie zmien databázy

Server poskytuje prístupový bod `GET {{apiUrl}}/api/release`, ktorý sa používa pre aplikovanie všetkých zmien pre užívateľov. Hlavička obsahuje autentifikačný token.

Nahranie súborov

Server poskytuje prístupový bod `POST {{apiUrl}}/uploads`. Používa sa pre nahranie súborov na server. `Content-Type` obsahuje `multipart/form-data`. Podporovaný `MIME type` je `application/octet-stream`. Hlavička obsahuje autentifikačný token. Odpoveď obsahuje adresu nahraného súboru.

Kapitola 5

Realizácia navrhnutého systému

Prvá časť tejto kapitoly sa venuje jadru problému a teda implementácii rozšírenej reality, kde sa popisujú algoritmy a kroky, ktoré prinášajú požadované funkcionality definované v predchádzajúcej kapitole. Následujú sekcie rozdelené podľa platformy. Mobilná aplikácie, webová aplikácia a serverová časť. Každá z týchto častí sa venuje nastavenej architektúre, ktorá je najdôležitejším prvkom pri vytváraní aplikácie a použitými knižnicami, ktoré uľahčujú a urýchľujú vývoj.

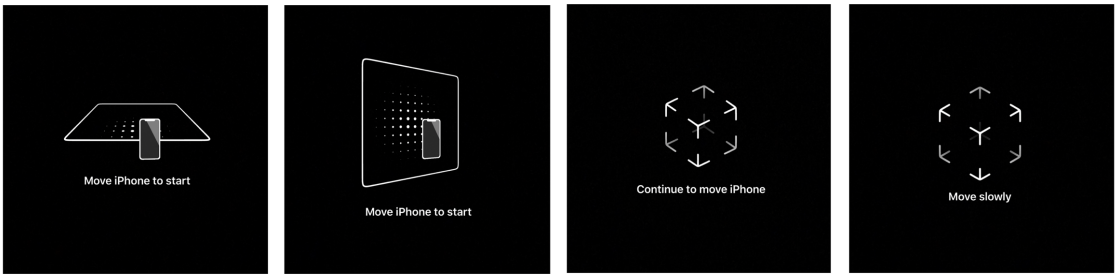
5.1 Integrácia rozšírenej reality

Výsledné riešenie bolo implementované na platforme *iOS*, kde konkrétne pri práci s rozšírenou realitov sa sa využívajú *frameworky ARKit* a *SceneKit*. Jedná sa o *frameworky*, ktoré boli predstavené v druhej kapitole venujúcej sa teoretickému úvodu. *ARKit* podporuje podľa potreby využitia niekoľko typov konfigurácii:

- `ARWorldTrackingConfiguration.` Sleduje polohu a orientáciu zariadenia vo vzťahu k akýmkoľvek povrchom, ľuďom alebo známym obrázkom a objektom, ktoré *ARKit* môže nájsť a sledovať pomocou zadnej kamery zariadenia.
- `ARBodyTrackingConfiguration.` Umožňuje sledovať osoby, roviny a obrázky pomocou zadného fotoaparátu zariadenia.
- `AROrientationTrackingConfiguration.` Sleduje iba orientáciu zariadenia pomocou zadného fotoaparátu.
- `ARImageTrackingConfiguration.` Sleduje iba známe obrázky pomocou zadného fotoaparátu zariadenia.
- `ARFaceTrackingConfiguration.` Sleduje iba tváre prostredníctvom prednej kamery zariadenia, vrátane pohybových a mimických výrazov.
- `ARObjectScanningConfiguration.` Používa zadnú kameru na zhromažďovanie vysoko kvalitných údajov o konkrétnych objektoch, ktoré má aplikácia rozoznať neskôr, za behu.
- `ARPositionalTrackingConfiguration.` Sleduje polohu zariadenia v 3D priestore.

`ARWorldTrackingConfiguration` relácia používa techniku nazývanú vizuálna zotrvačná odometria (*visual-inertial odometry* – VIO). Tento proces kombinuje údaje snímača pohybu

s počítačovou víziou analýzy snímok z kamery na sledovanie polohy a orientácie zariadenia v skutočnom priestore, známeho tiež ako póza (*pose*), ktorá je vyjadrená v transformačnej vlastnosti `ARCamera`. Pre dosiahnutie najlepších výsledkov, táto konfigurácia potrebuje konzistentné údaje zo senzorov a kamery s rozoznateľnými vlastnosťami a vizuálne zložitými objektami (t.j. hladká biela stena nie vhodné prostredie na inicializáciu). Po začatí relácie `sceneView.session.run(configuration))`, *ARKitu* trvá nejaký čas, aby zhromaždil dostatok údajov na presné modelovanie polohy zariadenia. Počas relácie sa môžu zmeniť podmienky, ktoré ovplyvňujú kvalitu sledovania okolitého prostredia. Pre získanie týchto informácií sa používajú vlastnosti `ARCamera`. Konkrétne súbor *ARCameraTrackingState.swift* obsahuje všetky možné stavy kamery a v závislosti na stave poskytuje užívateľovi informácie o aktuálnom stave relácie a zároveň obsahuje odporúčanie čo robiť napr. pri rýchlych pohyboch telefónom (obmedziť rýchly pohyb telefónom). Ako pomôcka pri navádzaní sa používa `ARCoachingOverlayView`, ktorý poskytuje *iOS* od verzie 13. Jednotlivé stavy sú ilustrované na 5.1. Tieto navádzacie inštrukcie sú zobrazené v strede obrazovky.



Obr. 5.1: `ARCoachingOverlayView`[1]

V rámci *ARKitu* a *SceneKitu* je niekoľko dôležitých delegátnych metód, ktoré sa využívajú pri práce v rozšírenej realite.

```
func renderer(
    _ renderer: SCNSceneRenderer,
    didAdd node: SCNNode,
    for anchor: ARAnchor
)
```

Táto funkcia sa volá, keď *ARKit* nájde uzol v scéne, ktorý sa dá použiť ako kotva a pridať do scény. Tieto objekty sa môžu použiť na umiestnenie virtuálneho obsahu do reálneho sveta. V rámci detekcie objektu sa táto funkcia využíva na detekciu snímaného objektu (`ARObjectAnchor`) alebo obrázku (`ARImageAnchor`). V prípade skenovania je v prípade manuálneho režimu využitá pre umiestnenie ohraničovacieho rámu k povrchu (`ARPlaneAnchor`), resp. pre spodné ohraničenie v prípade automatického režimu.

```
func renderer(
    _ renderer: SCNSceneRenderer,
    didRemove node: SCNNode,
    for anchor: ARAnchor
)
```

[1] https://developer.apple.com/documentation/arkit/arcoachingoverlayview

Pri presune na iné miesto alebo pri veľkej zmene prostredie sa volá táto delegátna metóda a vyjadruje, že kotva už nie je k dispozícii. Z toho dôvodu pri automatickom skenovaní sa užívateľ dostáva na začiatok celého procesu.

```
func session(
    _ session: SCNARSession,
    didUpdate frame: ARFrame
)
```

Pre potreby práce so všetkými údajmi, ktoré *ARKit* poskytuje, sa používa táto delegátna metóda. Táto funkcia poskytuje všetky informácie, čo *ARKit* dokáže zistiť z prostredia. Prístupný je zachytený obrázok z kamery, kotvy a v neposlednom rade mračno bodov.

Mračná bodov sú objekty reprezentované bodmi v 3D priestore. Tieto body sa zvyčajne nazývajú významné vlastnosti (*features*). *ARKit* dokáže zistiť tieto vlastnosti a nájsť povrchy. Práve tieto údaje sa používajú pri automatickom režime skenovania objektom nazvaný `frameProcessor`, ktorý spracovává mračno bodov a vytvára tak obálku okolo skenovaného objektu.

Vytvorenie modelu

Niektoré nasledujúce časti boli inšpirované a prevzaté z manuálu pre tvorbu modelov z oficiálnych stránok *Apple*[2]. Na to, aby bolo možné zobrazovať multimediálny obsah pri detekovanom objekte, je potrebné najprv vytvoriť model, ktorý sa má detekovať. Tento model sa nazýva `ARReferenceObject`. Pre vytvorenie modelu sa používa konfigurácia `ARObject-ScanningConfiguration`, ktorá poskytuje najväčší výkon a detail pri snímaní okolia. Po spustení relácie je potrebné nasmerovať kameru na snímaný objekt a poskytnúť pohľad z rôznych uhlov, čo umožní *ARKitu* vytvoriť si vnútornú mapu objektu a jeho okolia. Pri snímaní sa odporúča objekt umiestniť pred matné, stredne sivé pozadie, osvetliť objekt 240 až 400 luxami zo všetkých strán a zabezpečiť teplotu svetla približne 6500 Kelvinov (D65). Pri dodržaní všetkých podmienok sa dosiahne najlepší výsledok skenovania a toho dôsledkom je neskôr rýchle rozoznanie objektu.

Postup tvorby modelu je nasledovný:

1. Spustenie konfigurácie `ARObjectScanningConfiguration`.
2. Inicializácia *ARKitu* a snímanie rovín.
3. Vytvorenie ohraničujúceho rámu.
4. Snímanie významných bodov vnútri rámu.
5. Export objektu.

Apple vo vzorovej aplikácii poskytuje riešenie, kde je nutné ručne nastaviť ohraničujúci rám manuálne pomocou gest. Následne po vytvorení rámu okolo snímaného objektu sa sníma každá strana objektu pre dosiahnutí čo najlepšieho výsledku pri následnej detekcii. Pred exportom modelu je možné upraviť pomocou gest kotvu, teda počiatok objektu. V poslednom bode sa vytvorí `ARReferenceObjekt` volaním metódy

[2] https://developer.apple.com/documentation/arkit/scanning_and_detecting_3d_objects

```
func createReferenceObject(
    transform: simd_float4x4,
    center: simd_float3,
    extent: simd_float3,
    completionHandler: @escaping (ARReferenceObject?, Error?) -> Void
)
```

kde

- `transform` je transformačná matica definujúca pôvod a orientáciu lokálneho súradnicového systému pre oblasť, ktorá sa má extrahovať
- `center` je bod vzhľadom na pôvod špecifikovaný transformáciou, ktorý definuje stred ohraničujúceho rámu pre oblasť, ktorá sa má extrahovať
- `extend` je šírka, výška a hĺbka oblasti, ktorá sa má extrahovať, centrovaná na stredový bod a orientovaná na lokálny súradnicový systém určený transformáciou
- `completionHandler` obsahuje výsledný objekt alebo error

Predvolené automatické riešenie zbavuje užívateľa nutnosti manuálne pozicovať ohraničujúci rám a okamžite začne snímať skenovaný objekt.

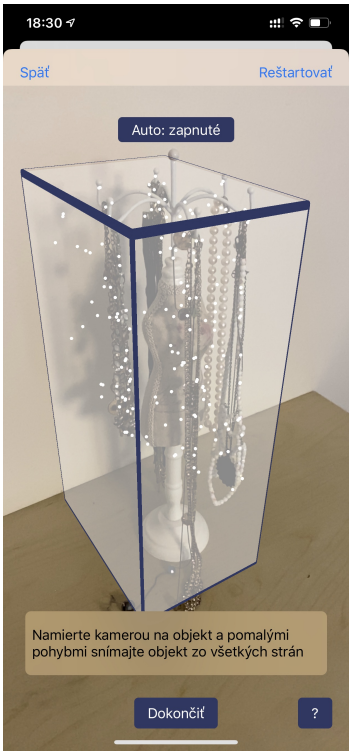
Skenovanie modelu

Na to, aby sa mohol detekovať objekt reálneho sveta, je nutné poskytnúť *ARKitu* informácie o detekovaných objektoch. Napríklad v prípade obrazu si *ARKit* vystačí s fotkou obrazu, orientáciou a šírkou. Avšak v prípade objektu, ktorý má hĺbku a je možné ho snímať z rôznych uhlov, nestačí iba 2D obrázok, ale *ARKit* potrebuje aj informácie o objekte zo všetkých pozorovateľných uhlov. K tomuto účelu slúži `ARReferenceObject`, ktorý je možné pomocou vyššie spomenutej funkcie `createReferenceObject(transform:center:extent:completionHandler:)`. Hlavným problémom je teda zistiť, kde sa skenovaný objekt nechádza, resp. kde je jeho stred a rozmery. V prípade manuálneho režimu je táto situácia jednoduchšia, kde užívateľ ručne vytvorí ohraničenie objektu a následne sníma objekt zo všetkých pohľadových strán pričom sa vytvorí objekt *.arobject*, ktorý je následne využitý pre detekciu. O celý tento proces sa stará trieda `Scan` a kontrolér `ScannerViewController`.

V rámci automatického režimu je postup značne iný. Inicializácia a detekcia plochy, na ktorej je umiestnený objekt zostáva v oboch režimoch rovnaká. Následne sa označí prvý bod reálneho objektu, ktorý sa vezme ako stred obrazovky, teda ten, na ktorý užívateľ miery kamerou. Tento bod sa používa pri inicializácii objektu `ARFrameProcessor`, ktorý spracováva informácie o významných bodoch (*feature points*) tvoriace mračno bodov poskytnuté *ARKitom* na základe ktorých postupne vytvára ohraničenie snímaného objektu. Ďalšie body sa získavajú v závislosti od vzdialeností k bodom, ktoré už sú označené ako body prislúchajúce k skenovanému objektu. Keďže poskytované body boli častokrát veľmi nepresné, nebolo možné spoľahlivo a dostatočne rýchlo vytvoriť rám objektu iba na základe aktuálne poskytnutých bodov vo video rámci. Preto vznikla potreba tieto údaje ukladať. Zvolená bola štruktúra `Octree`[3], ktorá riešila problém obrovského počtu detekovaných bodov a výmaz prípadných duplicít. Následne sa všetky body transformujú do jednej roviny a

[3] https://github.com/JaapWijnen/swift-algorithm-club/tree/master/Octree

vytvorí sa konvexná obalka pomocou algoritmu *ConvexHull*[4], aby bolo možné zistiť ohraničenie skenovaného objektu. Pre zjednodušenie je objekt považovaný za hranol k čomu slúžil algoritmus *rotating calipers*[4], ktorého výsledok je ilustrovaný na obrázku 5.3. Následne je nájdený bod s najväčšou výškou, ktorá tak určuje výšku výsledného ohraničenia objektu v ktorom sú umiestnené skenované body. Posledným krokom sú tak známe všetky údaje potrebné pre vytvorenie referenčného modelu. O vytvorenom ohraničení objektu je užívateľ vizuálne informovaný a tento proces je znázorený na obrázku 5.2.



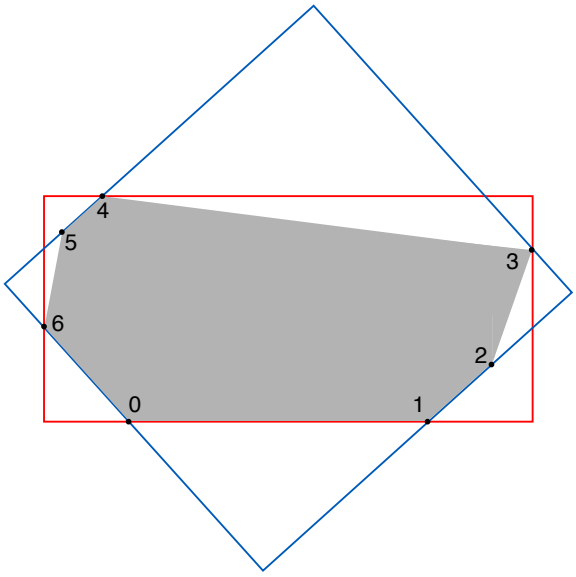
Obr. 5.2: Skenovanie objektu v automatickom režime

Zobrazenie a pozicovanie multimediálneho obsahu

V priebehu rozoznávania objektov pre zobrazenie doplnkového multimediálneho obsahu sa používa konfigurácia `ARWorldTrackingConfiguration`. Tejto konfigurácii sa nastavujú referenčné objekty, ktoré sa majú detekovať. Konkrétne sa jedná o objekty typu `ARReferenceImage` a `ARReferenceObject` podľa druhu súboru. `ARReferenceObject` požaduje súbor s koncovkou *.arboject* (detaily v ďalšej časti) a `ARReferenceImage` požaduje obrázok, jeho orientáciu a fyzickú šírku. Tieto referenčné objekty sa musia nachádzať na zariadení telefónu pred samotným spustením konfigurácie. O sťahovanie týchto súborov sa stará `Download-ViewController`, ktorý v prípade zmeny databázy pri štarte aplikácie stiahne ZIP archív obsahujúci tieto súbory zo serveru, rozbalí ho a uloží do priečinku *AR_app_downloaded_files* v dokumentoch telefónu. Po úspešnej detekcii sa táto informácia spropaguje do objektu `RecognizeRun`, ktorého úlohou je zobraziť multimediálny obsah do scény.

Aplikácia podporuje niekoľko druhov multimediálneho obsahu. V závislosti na koncovke súboru sa rozhoduje medzi podporovanými formátmi: video (*.mp4*, *.mov*), audio (*.mp3*),

[4] https://github.com/JaapWijnen/swift-algorithm-club/tree/master/Convex%20Hull



Obr. 5.3: Výsledok metódy *rotating calipers* [4]

obrázok (*.png*, *.jpg*), 3D model (*.usdz*) a text (*.txt*). Po detekcii sa na miesto, kde sa má zobraziť multimediálny obsah (podľa hodnôt štruktúry `ARContent`, ktorá obsahuje prípadné posunutia, rotácie, rozmery a mierku) najprv zobrazí indikátor načítania. Tento indikátor nazvaný `LoadingScene` je implementovaný v *SpriteKite*. Implementovaný je pomocou `SKSpriteNode`, ktorý má nastavenú textúru ikony načítavania, kde na tento uzol je aplikovaná akcia rotácie tohto uzlu o 360 stupňov v priebehu 2 sekúnd v nekonečnej smyčke. Následne po zobrazení indikátora načítania sa spustí sťahovanie multimediálneho obsahu, ktorý sa má zobraziť.

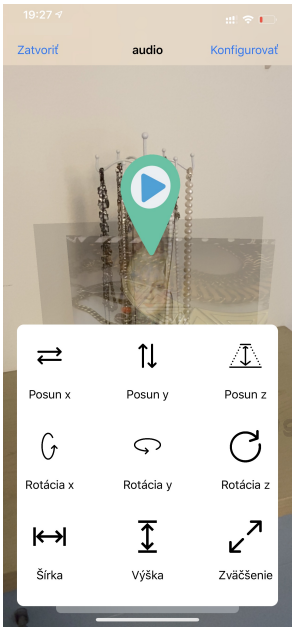
V prípade textu sa vytvorí inštancia `textNode` objektu `SCNNode`, ktorý má nastavenú rozmery podľa štruktúry `ARContent`. Tento objekt sa pridá do pola `imageNodes`, ktoré slúži na prípadné zmazanie celého multimediálneho obsahu zo scény po zrušení užívateľom alebo po detekcii nového objektu. Následne sa nastaví požadovaná pozícia voči detekovanému objektu, rotácia a mierka. Ďalej sa vytvorí objekt `SCNText`, ktorý sa inicializuje stiahnutým textom, ktorý sa nastavý ako geometria objektu textNode a pridá sa prostredníctom volania `addChildNode(textNode)` nad detekovaným uzlom.

Pri zobrazení audio nahrávky sa na rozdiel od ostatného typu obsahu nezobrazuje načítavací indikátor. V priestore sa zobrazí ikonka znázorňujúca prehrávanie na ktorú je možné kliknúť a po spustení je ju možné znovu zastaviť opätovným kliknutím na už zmenený obrázok znázorňujúci pauzu. Rovnako ako v prípade všetkých ostatných typov multimediálneho obsahu, aj tento je vložený do slovníka `contentNodes`, ktorý obsahuje všetky uzly a k ním príslušnú URL adresu obsahu a následne pridaný do scény volaním `addChildNode(imageNode)` nad detekovaným uzlom.

V prípade multimedíalneho obsahu typu obrázok sa vytvorí scéna obsahujúca indikátor načítania, ktorý sa po stiahnutí obrázka zo serveru nahradí uzlom `imageNode`, ktorému sa nastaví ako materiál tento stiahnutý obrázok.

Zobrazenie videa využíva `AVPlayer`, ktorý sa inicializuje odkazom na prehrávané video. Tento objekt je následne nastavený rovnakým princípom ako pri obrázku a teda ako materiál vytvorené uzlu `videoNode`.

Obr. 5.4: Zobrazenie multimediálneho obsahu



Obr. 5.5: Možnosti pozicovania obsahu

Posledným podporovaným obsahom je 3D model vo formáte *usdz* (*Universal Scene Description*). Jedná sa o formát súborov pre 3D modely, ktorý spoločnosť *Apple* predstavila v spolupráci so spoločnosťou *Pixar* pre *ARKit*. *USDZ* je kompaktný súbor obsahujúci všetky informácie o *mesh* dátach, binárnych údajoch a textúrach modelu, takže uľahčuje zobraziť alebo preniesť model bez straty údajov dôležitých pre správne rozbrazenie alebo straty textúry. Najprv je zobrazený indikátor načítania a po úspešnom stiahnutí objektu a vytvorení objektu typu `SCNReferenceNode` je tento objekt pridaný do scény pomocou rekurzie aj vrátane jeho všetkých uzlov.

Obrázok 5.4 znázorňuje zobrazenie všetkých typov multimediálneho obsahu.

Aplikácia pre návštevníkov a administrátorov sa líši vo funkcionalite – po dvojkliku na zobrazený multimediálny obsah. V prvom rade je potrebné zistiť, aký obsah, resp. uzol bol kliknutý, k čomu slúži `UITapGestureRecognizer` a aplikuje sa tzv. *hit test*, kedy sa vyberie uzol najbližšie nachádzajúci sa z pohľadu kamery. Aplikácia po dvojkliku na multimediálny obsah je znázornená na 5.5. Všetkým uzlom okrem vybraného sa nastaví polovičná priehľadnosť a zobrazí sa `ARSettingsView` – vyskakovacie okno, ktoré informuje o spôsobe manipulácie s objektom. K tomu sú využité viaceré typy `GestureRecognizerov`, konkrétne `UIPinchGestureRecognizer` (nastavenie mierky, výšky a šírky), `UIPanGestureRecognizer` (nastavenie posunu v smere osi X, Y a Z) a `UIRotationGestureRecognizer` (nastavenie rotácie okolo osi X, Y a Z). Následne sa menia hodnoty odpovedajúcich parametrov v štruktúre `ARContent`.

V prípade aplikácie pre administrátorov sa po dvojkliku zobrazuje multimediálny obsah cez celú obrazovku. Tento obsah sa zobrazuje v závislosti na vyššie popísanom slovníku `contentNodes`, ktorý mapuje vybraný uzol k URL adrese, resp. obsahu.

5.2 Mobilná aplikácia

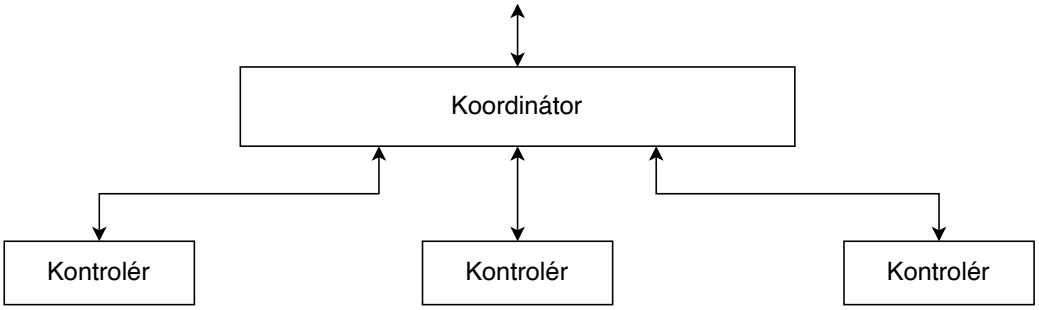
Nasledujúca sekcia obsahuje popis aplikovaného architektonického vzoru MVVM+C, ktorý umožňuje jednoduchú testovateľnost spolu s udržovateľnosťou a popis knižníc, ktoré boli použité pri implementácii mobilnej aplikácie s cielom vytvoriť kvalitné riešenie s využitím dostupných už existujúcich riešení.

Architektúra

Apple odporúča pri vývoji aplikácií použiť architektonický vzor *Model-View-Controller* (MVC), ktorého použitie so sebou nesie niekoľko problémov. Kontrolér obsahuje jadro logiky pre jednu obrazovku. Avšak, stará sa aj o navigáciu medzi týmito obrazovkami. V prípade, že je možné dostať sa z jednej obrazovky na väčší počet (3 a viac) iných obrazoviek, z kontroléra, ktorý sa má starať o chovanie jednej obrazovky sa stáva objekt, ktorý sa stará prevažne o navigáciu medzi obrazovkami. Z kódu sa tak stáva kód neprehľadný, ťažko čitateľný, ťažko refaktorovateľný a zle udržovateľný. Tento problém riešim využitím takzvaných koordinátorov. Koordinátor je objekt, ktorý má na starosti jeden alebo viacej kontrolérov. Je to teda objekt, ktorý obsluhuje životný cyklus logicky súvisiacej funkcionality, ktorá je obsiahnutá v niekoľkých kontroléroch, respektívne obrazovkách.

Obrázok 5.6 znázorňuje tok informácií medzi koordinátorom a kontrolérmi. Na komunikáciu medzi týmito objektami sa využívajú delegátne metódy. Jedná sa o prípady, kedy kontrolér požaduje o akciu spojenú s navigáciou alebo zmenu dát. Koordinátory udržujú históriu zobrazených obrazoviek pomocou objektov zvaných navigačné kontroléry, čo znamená, že kontroléry nepoznajú kontext v ktorom sú zobrazené. To zjednošuje znovupoužiteľnosť kontrolérov a možnosť použitia z rôznych miest v aplikácii prakticky bez dodatočného kódu priamo v kontroléry.

Úlohou koordinátoru nie je iba obsluha navigácie medzi kontrolérmi, ale napríklad aj synchronizačná logika či správa úložiska.



Obr. 5.6: Komunikácia koordinátor a kontrolér

V rámci aplikácie pre administrátora tak vzniklo až 17 koorinátorov:

- `ApplicationCoordinator`. Hlavný koordinátor spúšťajúci ostatné koordinátory.
- `TabBarCoordinator`. Navigácia medzi položkami v hlavnom menu.
- `HomeCoordinator`. Navigácia medzi položkami na hlavnej obrazovke.
- `MapCoordinator`. Správa (vytvorenie, aktualizácia, odstránenie) mapy.

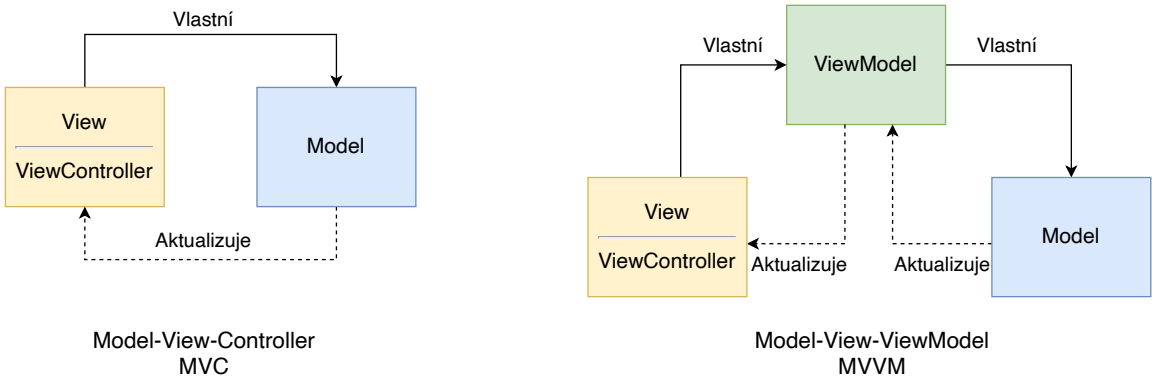
- `LibraryCoordinator.` Výber exponátov a kategórií.
- `SettingsCoordinator.` Práva aplikácie, vydanie novej verzie a odhlásenie.
- `CredentialsCoordinator.` Prihlásenie.
- `LanguageCoordinator.` Správa jazykov.
- `HomeImageCoordinator.` Správa hlavného obrázku.
- `HomeNewsCoordinator.` Správa noviniek.
- `HomeLocationCoordinator.` Správa lokality.
- `HomeTicketCoordinator.` Správa lístkov.
- `HomeCategoryCoordinator.` Správa kategórií.
- `MasterpieceCoordinator.` Správa exponátu.
- `ARContentConfigurationCoordinator.` Správa 3D obsahu exponátu.
- `OpeningHoursCoordinator.` Správa otváracích hodín.
- `DocumentPickerCoordinator.` Výber dokumentu.
- `ScannerCoordinator.` Skenovanie exponátu.

V rámci aplikácie pre návštevníka vzniklo 7 koorinátorov:

- `ApplicationCoordinator.` Hlavný koordinátor spúšťajúci ostatné koordinátory.
- `TabBarCoordinator.` Navigácia medzi položkami v hlavnom menu.
- `OnboardingCoordinator.` Sprievodca aplikáciou. Aplikovaný `ApplicationCoordinatorom` pri prvom spustení aplikácie.
- `HomeCoordinator.` Úvodná obrazovka obsahujúca všeobecné informácie (hlavný obrázok, lístky, otváracie hodiny, novinky).
- `MapCoordinator.` Mapa a výber poschodí.
- `RecognizeCoordinator.` Rozšírená realita. Identifikácia a informácie o exponáte.
- `LibraryCoordinator.` Knižnica exponátov a kategorizácia.
- `DownloadCoordinator.` Sťahovanie a aktualizácia databázy. Aplikovaný `ApplicationCoordinatorom` v prípade existencie novej verzie databázy.

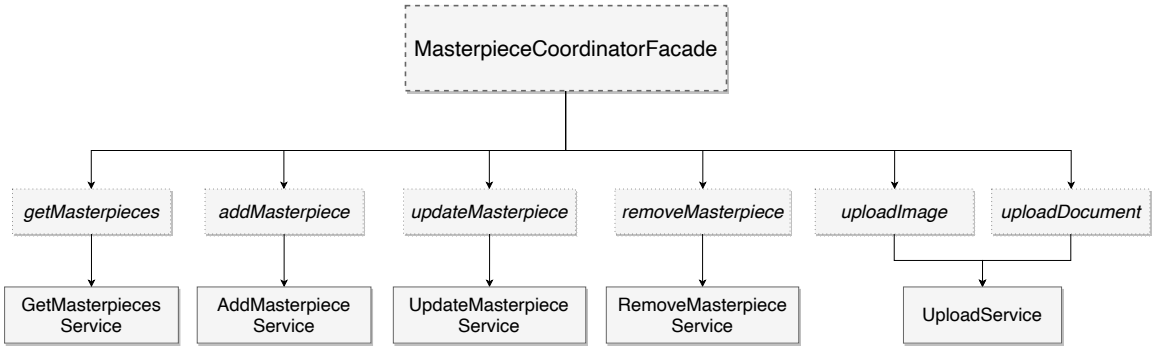
V rámci MVC – *ViewController* je úzko spojený s *View*. V dôsledku toho je často zložité vytvárať *unit testy* pre *View*, pretože celková obchodná logika, v tomto prípade, musí byť oddelená od kódu pre *View*. Aj keď je stále možné presunúť transformáciu dát a obchodnú logiku do *Modelu*, je možné zvoliť inú architektúru, ktorá tento problém rieši. Práve z tohto dôvodu bola použitá architektúra *Model-View-ViewModel* (MVVM), ktorá rieši všetky vyššie uvedené problémy.

Kľúčový rozdiel je, že v MVVM je nová triedu *ViewModel*, ktorá interaguje s *Modelom*. To znamená, že *ViewController* nezodpovedá za dáta a manipuláciu s nimi. Preto *Controller* v tejto architektúre robí iba to, čo má robiť, čiže pracovať s *View*. MVVM s využitím konceptu koordinátora sa označuje ako MVVM+C a práve táto architektúra je použitá pri implementácii mobilných aplikácií v rámci tejto diplomovej práce. Rozdiel medzi MVC a MVVM je ilustrovaný na 5.7.



Obr. 5.7: Porovnanie MVC a MVVM

Každý softvér by mal poskytovať možnosť testovať a práve z toho dôvodu bola použitá technika vkladania závislostí (*Dependency Injection* – DI). Táto tachnika v objektovo orientovanom programovaní slúži pre vkladanie závislostí medzi jednotlivými komponentami programu tak, aby jedna komponenta mohla použiť druhú, bez toho, aby na ňu mala v dobe zostavovania programu referenciu. Využitie je nasledovné. Ak chce objekt používať služby iného objektu bez použitia DI, potom zodpovedá za celý svoj životný cyklus, teda inicializáciu atď. Pri aplikácii DI je však objekt odbremenený od tejto správy, keďže ju zariaďuje poskytovateľ závislostí. Objekt teda potrebuje už len referenciu na poskytovateľa závislostí, ktorý mu je schopný dodať hneď niekoľko rôznych komponentov, spĺňajúcich určité rozhranie. Sila DI spočíva v tom, že každá z týchto komponentov môže objektu poskytovať rozdielne služby. Na DI sa v aplikácii používa pre inicializáciu koordinátorov objekt `CoordinatorFactoryImp` a pre inicializáciu modulov objekt `ModuleFactoryImp`. Vkladanie závislostí umožňuje nie len testovanie modulov, ale tiež umožňuje modifikáciu správania sa objektu bez zmeny kódu tohto objektu (princíp otvorenosti/zatvorenosti), ktorý popisuje Robert C. Martin [11] ako jeden z princípov SOLID. Výsledkom teda nie iba testovatelní kód, ale výsledkom je aj kód flexibilný. V projekte je často využitý okrem návrhového vzoru továreň aj návrhový vzor fasáda. Továreň slúži k vytvoreniu inštancie bez toho, aby sa pôvodná trieda neznečistila inicializačným kódom. Fasáda sa používa na vytvorenie jednotného rozhrania pre celú logickú skupinu tried, ktoré sa tak združia do subsystému. Protokoly k daným fasádam majú príponu *Facade.swift* a implementácia daného protokolu obsahuje príponu *FacadeImp.swift*. Fasáda je pomerne jednoduchý vzor, ktorý sa skladá z jednej triedy, ktorá fasádu tvorí. Tá je napojená na ďalšie triedy, s ktorými pracuje. Zvonku je však vidieť len fasáda (od toho názov), a tá zastupuje rozhranie pre celý subsystém. Celá zložitá štruktúra tried je v pozadí. Zníži sa tým počet tried, s ktorými aplikácie komunikuje. Subsystém sa vďaka tomu lepšie používa a testuje. Príklad fasády, ktorá sa používa pri správe exponátov je znázornený na 5.8. Jedná sa o `MasterpieceCoordinatorFacade`, ktorý poskytuje niekoľko služieb pre komunikáciu so serverom.



Obr. 5.8: Príklad fasády

Použité knižnice

CocoaPods[5] je manažér závislostí pre jazyky *Swift* a *Objective-C Cocoa* projekty, ktorý obsahuje vyše 72 tisíc knižníc a používa sa vo viac ako 3 miliónoch aplikáciách. *CocoaPods* pomáha elegantne škálovať projekty. Použitie je nasledovné. Do súboru *Podfile* sa vložia knižnice, ktoré sa majú nainštalovať. Následne po spustení príkazu `pod install` sa nainštalujú všetky knižnice vrátane ich závislostí. Zároveň sa vytvorí súbor *Podfile.lock*, ktorý obsahuje verzie nainštalovaných knižníc. V mobilných aplikáciach boli použité následovné knižnice:

- `Alamofire.` Volanie HTTP sieťových požiadavkov.
- `AlamofireImage.` Sťahovanie obrázkov.
- `SwiftyJSON.` Parsovanie JSON odpovedí zo serveru.
- `IQKeyboardManager.` Automatická správa (skrytie klávesnice a navigácia medzi objektami typu `UITextField`) klávesnice.
- `SwiftLint.` *Lintner* na vynútenie konvencií a *Swift* štýlu.
- `netfox.` Poskytuje rýchly prehľad o všetkých vykonaných sieťových požiadavkách aplikáciou. Okrem štandardných `URLRequestov` podporuje aj knižnice tretích strán ako napríklad použitej knižnici *Alamofire*.
- `R.swift.` Kód pre prácu s obrázkami, lokalizáciou atď. je silne typovaný, kontrolovaný v čase kompilácie a podporuje napovedanie. Zabraňuje tak pádu aplikácie v dobe jej behu napr. z dôvodu neexistencie obrázku, ktorá často vzniká kvôli preklepu alebo neúmyselnému zmazaniu.
- `TableKit.` Umožňuje vytvárať zložité zobrazenia tabuľky deklaratívnym spôsobom. Skrýva zložitosť metód `UITableViewDataSource` a `UITableViewDelegate`, takže kód je čistý, ľahko čitateľný a jednoduchý na údržbu.
- `SnapKit.` Práca s automatickým pozicovaním.
- `NVActivityIndicatorView.` Galéria načítavacích animácií.

[5] http://cocoapods.org

- `RealmSwift`. Náhrada za natívnu databázu *CoreData*. Objektová databáza, ktorá je prístupná pre viaceré platformy. Zjednodušuje definíciu schémy databázy a prácu s rôznymi vláknami a podporuje šifrovanie uložených dát.
- `SwipeCellKit`. Podpora mazania a editovania ťahom prstu po bunke.
- `SSZipArchive`. Archivácia súborov a rozbalenie archívu.

5.3 Webová aplikácia

Webová aplikácia bola implementovaná v jazyku *Javascript*. Konkrétne bola použitá *Javascript* knižnica pre tvorbu užívateľských rozhraní *React*. Nakoľko je *Javascript* dynamicky typovaný jazyk, v projekte bola použitá jeho typovaná varianta *Typescript*[6]. Typová kontrola umožnila urýchliť vývoj celej aplikácie a zachytiť tak množstvo chýb ešte pred samotným spustením programu. *Typescript* môže byť použitý v súvislosti s ľubovoľným webovým prehliadačom, ktorý podporuje *Javascript*. Na grafické prvky užívateľského rozhrania bola použitá knižnica *Material-UI*.

Architektúra

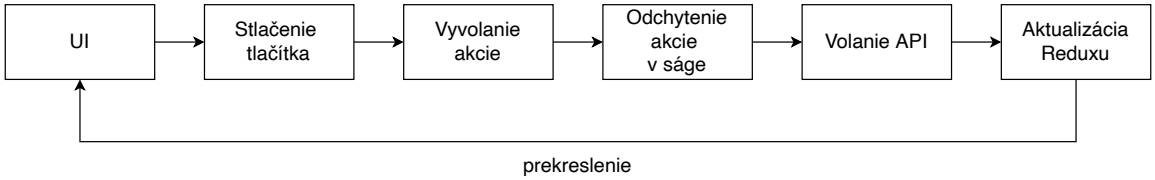
Pred samotnou implementáciou bol kladený dôraz na jednoduchú udržovateľnosť a rozšíriteľnosť projektu v budúcnosti. UI komponenty sú rozdelené podľa princípov atomického dizajnu, kde sa jednotlivé komponenty delia na atómy, molekuly a organizmy. Atómy tvoria najmenšie časti, ako napr. naštýlovaný nadpis, tlačítko apod. Z atómov sú zložené zložitejšie časti, zvané molekuly. Tie môžu na rozdiel od atómov pristupovať ku globálnemu stavu aplikácie. Pracujú však stále iba s určitou časťou stavu. Najkomplexnejšími prvkami sú organizmy. Tie majú prístup k stavu, môžu mať na seba napojenú napr. analytiku. Medzi hlavné výhody takéhoto prístupu sa považuje princíp, že jednotlivé komponenty sa spájajú do zložitejších celkov a vytvárajú tak komplexné prvky. Takto je minimalizovaná duplicita kódu a zmeny sa hneď prejavia na všetkých miestach kde sa daný atóm, molekula či organizmus používa. To prispieva k lepšej testovateľnosti a rozšíriteľnosti projektu.

Počas behu, je nutné si v aplikácii držať určité dáta. Ide o dáta týkajúce sa prihláseného užívateľa, stiahnutých diel atď. Na správu stavu bola použitá knižnica Redux[7]. Ide o stavový kontajner, ktorý ma na starosti udržať konzistenciu medzi aktuálnym stavom aplikácie a stavom databázy. Medzi jeho hlavné výhody patrí centralizovanosť dát, ktoré sú následne dostupné naprieč celou aplikáciou. S takto uloženými dátami sa jednoduchšie pracuje a aj prípadné chyby sa jednoduchšie odhaľujú.

Celá aplikácia komunikuje s REST API, aby obdržala aktuálne dáta. Pri tejto komunikácii sú využívané tzv. generátory[8] (*generator functions*). Ide o špeciálny druh funkcií, ktoré je možné za behu opustiť a znovu do nich vstúpiť, pričom ich kontext ostáva zachovaný naprieč jednotlivými vstupmi. Po zavolaní takejto funkcie je vrátený objekt, tzv. iterátor. Nad týmto objektom sa následne prevoláva metóda `next()`, ktorá vykoná časť tela funkcie po prvý príkaz `yield`. Tento príkaz špecifikuje hodnotu, ktorá má byť vrátená s ďalším volaním `next()`. Spolu s vyžitím `Promise`, sa stávajú mocným nástrojom asynchrónneho programovani.

Príklad pridania nového diela do databázy znázorňuje schéma 5.9.

[6] https://typescriptlang.org/
[7] https://redux.js.org/
[8] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/function*



Obr. 5.9: Tok akcií webovej aplikácie

Užívateľ pridá nové dielo cez formulár na stránke. Po stlačení tlačítka „Uložiť“ sa vyvolá akcia. Tá sa následne odchytí v ságach, kde sa zavolá generátor. Ten má za úlohu prevolať API endpoint a uložiť dáta z formulára ako nové dielo do databázy. Po úspešnom uložení do databázy sa stiahnu znova všetky diela a vyvolá sa akcia na aktualizáciu reduxového stavu. K prekresleniu UI komponenty dôjde na základe zmeny stavu, pričom stav aplikácie už obsahuje aj novo pridané dielo.

Viacjazyčný formulár

Asi najkomplikovanejšou časťou celej aplikácie bol viacjazyčný formulár na pridávanie nových diel. Pri implementácii bola využitá vlastnosť prekresľovania UI na základe zmeny stavu ako aj posielaní aktuálneho stavu smerom dole do podkomponent. Komponenta ktorá spájala logiku prekresľovania a bol v nej držaný aj aktuálny stav celého formuláru bola implementovaná pomocou triedy `AddMasterpieceFormUI`. Stav sa následne posielal smerom nadol a jednotlivé podkomponenty formuláru mali k dispozícii aktuálny stav. Na základe toho sa po prepnutí na iný jazyk zobrazili správne dáta v jednotlivých políčkach formuláru. Na zmenu stavu boli do komponent posielané metódy, ktoré modifikovali určitú časť stavu v nadradenej komponente. Napr. metóda `OnDataChange()` slúžila na zmenu názvu diela alebo jeho poradia či iných atribútov. Formulár však obsahuje aj vnorené dátové štruktúry ako napr. AR obsah. Na jeho zmenu bola implementovaná metóda `OnArContentDataChange()`, ktorá aktualizovala časť stavu, kde boli držané informácie o AR obsahu. Obdobne boli implementované aj ďalšie časti formuláru. Takýto formulár bol využitý všade tam, kde bola potreba ukladania rovnakého druhu položiek v rozličných jazykoch. Spoločným prvkom pre tieto komponenty bola nadradená komponenta, ktorá obsahovala logiku pre zmenu dát a držala v sebe aktuálny stav formuláru. Zmenou stavu došlo k prekresleniu nadradenej komponenty a nakoľko bol stav posielaný o úroveň nižšie, zmenili sa aj parametre podkomonent, ktoré boli taktiež prekreslené. Takto sme schopný vyvolať prekreslenie častí UI nachádzajúcich sa v DOM strome o n-úrovní nadol, od komponenty v ktorej nastala zmena.

Podpora jazykov

Aplikácia podporuje slovenský a anglický jazyk. Pri inicializácii na klientovi sa zavolá metóda `loadLocalizationSettings()`, ktorá deteguje aktuálny jazyk webového prehliadača. V prípade, že má užívateľ nastavený iný jazyk, použije sa slovenčina. Jazyk je možné meniť aj priamo v aplikácii. Tlačidlo na jeho prepnutie sa nachádza v hornom menu. Preklady sú uložené v konfiguračnom súbore *localizationConfig.ts* a je možné ich jednoducho doplniť o ďalšie jazyky.

5.4 Server

Server je implementovaný s využitím *frameworku Express*, ktorý je postavený na *Node.js*.

Ako databáza je použitá *MongoDB*. Jedná sa o multiplatformnú dokumentovú databázu. Patrí medzi *NoSQL* databázy a na rozdiel od tradičných relačných databáz využívajúcich tabuľky, používa dokumenty podobné formátu JSON (*MongoDB* formát nazýva BSON) a dynamickú databázovú schému, ktorá umožňuje jednoduchšie a rýchlejšie vytváranie a integráciu dát.

Na správu javascriptových balíčkov je použitý *Node Package Manager – npm*. Súbor *package.json* obsahuje zoznam všetkých knižníc, ktoré sú použité v projekte. Pomocou príkazu *'npm install'* sa nainštalujú všetky zmienené knižnice.

Server obsahuje nasledujúce závislosti:

- `expressjs`. Server na spracovanie a smerovanie HTTP požiadavkov.
- `express-jwt`. Prostredník slúžiaci na overenie *JSON Web Token* (JWT) pri autentifikácii. JWT je otvorená štandardná RFC 7519 metóda pre poskytnutie bezpečných nárokov komunikácie medzi dvoma stranami.
- `jsonwebtoken`. Generovanie JWT používaných pri autentifikácii.
- `mongoose`. Modelovanie a mapovanie údajov *MongoDB* do *Javascriptu*.
- `mongoose-unique-validator`. Spracovanie jedinečných chýb pri validácii v *Mongoose*. *Mongoose* spracováva validáciu iba na úrovni dokumentu, takže jedinečný index v kolekcii vyvolá výnimku na úrovni ovládača. Pomáha pri formátovaní chyby ako bežný *mongoose* `ValidationError`.
- `passport`. Práca s autentifikáciou užívateľa.
- `slug`. Kódovanie parametrov do URL podporovaného formátu.
- `multer`. Spracovanie `multipart/form-data`.
- `archiver`. Vytvorenie *.zip* archívu.
- `dateformat`. Formátovanie `Date` objektu ako reťazec.
- `supertest`. Poskytnúť abstrakciu na vysokej úrovni na testovanie HTTP a zároveň umožňuje prejsť na nižšiu úroveň rozhrania API poskytovanú superagentom.
- `underscore`. Poskytuje celú radu užitočných funkcií predovšetkým z funkcionálneho programovania bez nutnosti rozšírenia akýchkoľvek vstavaných objektov.

Štruktúra aplikácie je následovná:

- `app.js`. Vstupný bod aplikácie. Tento súbor definuje `express` server a pripája ho k *MongoDB* pomocou *mongoose*. Taktiež vyžaduje cesty a modely, ktoré aplikácia používa.
- `config/`. Tento priečinok obsahuje konfiguráciu pre *passport*, konfiguračné premenné a premenné prostredia.
- `routes/`. Tento priečinok obsahuje definície ciest pre API.

- `models/`. Tento priečinok obsahuje definície schém pre modely *Mongoose*.

Chyby sú riešené následovne: V *route/api/index.js* je definovaný *middleware* na spracovanie chýb pre prácu s `ValidationError` z *Mongoose*. Tento *middleware* odpovedá stavovým kódom 422 a naformátuje odpoveď.

Požiadavky na API sú overené pomocou hlavičky `Authorization` s platným JWT. V *route/auth.js* sú definované *express middlewary*, ktoré môžu byť použité na autentifikáciu požiadavkov. *Required middleware* konfiguruje *express-jwt middleware* pomocou tajomstva aplikácie a ak nie je možné žiadosť overiť, vráti stavový kód 401. K dátovému obsahu JWT je možné pristupovať pomocou `req.payload` priamo v *endpointe*. *Optional middleware* konfiguruje *express-jwt* rovnakým spôsobom ako *required*, ale ak nie je možne autentifikovať požiadavku, nevráti stavový kód 401.

Nahrávanie súborov sa rieši nasledujúcim spôsobom: Pre nahratie súboru je vyžadovaný autorizačný token, ktorý sa validuje pomocou *express-jwt*. Následne je požiadavok zaslaný na spracovanie *middlewarom uploadMiddleware.js*, ktorý používa vyššie spomenutý nástroj *multer*. Z požiadavku `multipart/form-data` sa vytiahne súbor s názvom `file`, ktorý sa spracuje a uloží sa do zložky *public/uploads*. Pre zabezpečenie jedinečnosti je k súboru priradený dátum vzniku. Maximálna veľkosť nahrávaného súboru je nastavená na 100 MB. Odpoveď obsahuje URL adresu nahraného obrázka, ktorý je verejne prístupný. V prípade, že nastane k chybe, server vráti stavový kód 404.

O aktualizáciu dát sa stará prístupový bod `GET /release`, ktorý vyžaduje autorizačný token. Následuje získanie všetkych exponátov z databázy. Pomocou nástroja *supertest* sa prevolajú všetky prístupové body s metódou GET. Tento proces je synchronný a po obdržaní poslednej odpovede sa pristúpy k poslednej sprístupnenej verzii databázy, ktorá sa porovná s aktuálnymi hodnotami. V prípade, že sa zistí odlišnosť v jazykoch, kategóriách, exponátoch alebo mape, vytvorí sa *json.txt*, ktorý obsahuje aktuálny dátum a aktuálnu kompletnú databázu vo forme JSONu a *homeJSON.txt*, ktorý obsahuje aktuálny dátum spolu s údajmi z databázy, ktoré sa používajú na úvodnej stránke mobilnej aplikácie pre užívatelov (novinky, otváracie hodiny, lístky a nastavenia). Ak nenastane odlišnosť, prepíšu sa aktuálne hodnoty (konkrétne novinky, otváracie hodiny, lístky a nastavenia), ale použije sa už existujúci dátum, ktorý mobilnej aplikácii signalizuje, že hlavná časť databázy sa nezmenila a stačí tak aplikovať zmeny iba na úvodnej stránke bez nutnosti stiahnutia celej databázy. Spolu s JSON súbormi sa vytvorí archív *Archive.zip* pomocou nástroja *archiver*, ktorý obsahuje okrem týchto JSON súborov aj všetky súbory na detekciu exponátu v rozšírenej realite (`ar_model_3d` a `ar_model_image`).

Server bol nasadený prostredníctvom služby *Heroku*[9], konkrétne server je prístupný na URL https://dry-journey-73039.herokuapp.com. *MongoDB* databázu poskytuje doplnok *mLab MongoDB*, ktorý ponúka *Heroku*. Implementovaná webová aplikácia a obe mobilné aplikácie používajú tento server.

5.5 Testovanie

V rámci diplomovej práce bolo vytvorených niekoľko aplikácií, ktoré spoločne tvoria systém, ktorý umožnuje návštevníkom múzeií a galérií zobrazovať multimediálny obsah s využitím rozšírenej reality a administrátorom vkladať a konfigurogovať tento multimediálny obsah. Kľúčovým aspektom celého systému je spoľahlivosť a funkčnosť prvkov v časti rozšírenej reality, ktorá bola jadrom práce. Z toho dôvodu boli otestované iba tieto časti.

[9] https://heroku.com

Spôsob testovania

Výsledný systém bol vyhodnotený na základe užívateľského testovania a experimentami v reálnom prostredí. Každú úlohu účastníci testu opakovali jedenkrát. Po prvom pokuse im bola poskytnutá detailná inštruktáž. Týmto postupom sa eliminovala možnosť, že riešenie je funkčné, avšak užívateľ nemá dostatok skúseností s aplikáciami podporujúcimi rozšírenú realitu, poprípadne nedostatočnou inštruktážou, resp. zlým návrhom UX. Boli vytvorené 4 úlohy, kde prvá úloha bola z aplikácie určenej pre návštevníka a dalšie tri boli z aplikácie určenej pre administrátora:

1. Namierením telefónu na objekt rozoznať tento objekt a zobraziť jeho multimediálny obsah.
2. Vytvoriť sken objektu v automatickom režime.
3. Vytvoriť sken objektu v manuálnom režime.
4. Pridať multimediálny obsah a konfigurovať jeho umiestnenie.

Priebeh testovania

Testovanie systému prebiehalo na vzorke desať ľudí v rokoch 12-70 rokov, ktorí mali za úlohu splniť vyššie definované úlohy. Výber ľudí úmyselne pokrýval čo najširšie spektrum možných používateľov. Pred samotným testovaním bolo potrebné vytvoriť testovacie sady na ktorých prebiehalo testovanie. Referenčné objekty boli vytvorené pomocou vytvorenej mobilnej aplikácie pre administrátora. Konkrétne sa jednalo o obraz, ktorý využíval ako referenčný objekt 2D obrázok a sochu, ktorá využívala ako referenčný objekt AR súbor pre detekciu 3D objektov. Oba tieto objekty mali nastavené všetky typy multimediálneho obsahu.

Výsledok testovania

Prvú úlohu testu splnili všetci účastníci, pričom behom testu sa nepýtali na žiadne otázky a komentovali, že kroky, ktoré vedú k rozoznaniu objektu a zobrazeniu multimediálneho obsahu sú intuitívne. Polovica z nich dokonca použila gesto dvojkliku pre zobrazenie vybraného multimediálneho obsahu cez celú obrazovku. Zvládnuť vytvoriť sken v automatickom režime na 1. pokus nezvládol nikto z účastníkov testu, ale v manuálnom režime sa to podarilo trom účastníkom a pridať multimediálny obsah spoločne s konfiguráciou polovici. Výrazná zmena však nastala po poskytnutí inštruktáže a všetky úkony úspešne zvládli všetci zúčastnení. Na základe výsledkov testovania odporúčam pre administrátorov obsahu obohatiť aplikáciu o animácie, ktoré budú užívateľa jasne informovať o požadovaných krokoch, aby nedochádzalo k situácií, kde užívateľ si nie je istý svojimi krokmi, prípadne doplniť aplikáciu o hlasové navádzanie, nakoľko skenovanie objektov, pridávanie a následná konfigurácia multimediálneho obsahu nie je na prvý krát dostatočne intuitívna a vyžaduje istý typ zoznámenia sa so systémom. Ak by nepomohli navrhnuté kroky, je možné zvážiť vytvorenie inštruktážnych videí, v krajnom prípade ponúknuť osobitné školenia. Z testovania vyplýva, že systém je funkčný a pre návštevníkov dostatočne intuitívny.

Experimenty na hraničných prípadoch

Testovanie na užívateľoch bolo uskutočnené na vhodných objektoch, ktoré boli kvalitne naskenované. Okrem toho boli uskutočnené experimenty, ktorými bolo zistené, že rozoznanie exponátu je silne závislé od kvality referenčného objektu vytvoreného pri skenovaní. V prípade kvalitne (výborné svetelné podmienky, snímanie zo všetkých zorných uhlov a vysoké množstvo nasnímaných bodov) naskenovaného objektu je prakticky okamžite po namierení telefónu na snímaný objekt tento objekt rozoznaný a zároveň je zobrazený multimediálny obsah prislúchajúci k tomuto objektu. V prípade, že je poskytnutý nekvalitne naskenovaný objekt, rozoznanie môže trvať až jednotky sekúnd a multimediálny obsah nie je stabilne zobrazený v okolí rozoznaného objektu. Za hlavnú príčinu možných problémov pri práci s rozšírenou realitou možno označiť nedostatok svetla v danom prostredí, kde až najnovšie rady telefónov eliminujú tento problém s využitím viacerých senzorov pre snímanie obrazu. Použitie technológie nie je vhodné pri veľmi podobných exponátoch v rámci vystavovaných exponátov, nakoľko môže prísť k zámene medzi podobnými exponátmi. Rovnako tak nie je vhodné vytvoriť referenčný 3D model pre exponáty do veľkosti $10x10x10\,cm$ (napr. mravec). V tomto prípade odporúčam vytvoriť fotografiu, kde je prípadne možné zahrnúť aj okolie exponátu pre uľahčenú detekciu. Posledným typom objektov, ktoré nie sú vhodné a teda je značne znížená schopnosť rozoznania sú objekty, ktoré nemajú žiadne body príznačné body (napr. biela stena) a objekty, ktoré odrážajú svetlo (krištáľové svietidlo či zrkadlo) alebo naopak, sú transparentné (plexisklo), pretože z pohľadu kamery tak menia svoj vzhľad. Pri týchto objektoch a objektoch obsahujúcich menší počet významných bodov často dochádzalo k problémom vytvorenia ohraničujúcej obálky objektu v rámci automatického skenovania. S výnimkou týchto uvedených prípadov, ktoré nie sú vhodné na takéto použitie, obecne možno považovať vytvorené riešenie za univerzálny systém, ktorý je v dostatočnej miere spoľahlivý a funkčný.

Kapitola 6

Záver

Cieľom tejto diplomovej práce bol návrh a implementácia systému, ktorý umožní návštevníkom výstav zobrazovať multimediálny obsah s využitím rozšírenej reality. Kľúčové pre takýto systém je umožniť administráciu multimediálneho obsahu, teda jeho tvorbu, správu a distribúciu.

Pojem rozšírená realita a princíp počítačového videnia je predstavený v úvode práce. Následne v prvej časti sa čitateľ dozvie prostredníctvom akých nástrojov je možné využiť rozšírenú realitu v mobilných telefónoch a ako vytvoriť vhodné užívateľské prostredie pre prácu s rozšírenou realitou. Poslednou témou prvej kapitoly *framework ARKit*, ktorý je neskôr použitý v rámci realizácie navrhnutého riešenia. Riešenie tohto problému vyžadovalo návrh komplexného riešenia, teda vrátane mobilnej užívateľskej aplikácie a administrátorských častí pre správcu systému. Teoretická časť teda pojednáva aj o vývoji informačných systémov, vrátane konkrétnych postupov a mechanizmov pri vývoji mobilných a webových aplikácií. Táto časť sa venuje najnovším trendom pri vývoji systémov a prináša prehľad rôznych prístupov, respektívne architektúr informačných systémov.

V ďalšej časti aj na základe analýzy existujúcich aplikácií pre návštevníkov galérií a múzeií navrhujem vlastné riešenie, avšak s použitím rozšírenej reality, ktoré návštevníkom umožňuje automatické vyhľadávanie exponátov a prehľadné zobrazenie multimediálneho obsahu. Následne po identifikovaní potrieb užívateľov je potrebná detailná špecifikácia každej obrazovky s dôrazom na UX. V prípade aplikácie pre užívateľa je potrebné dbať aj na samotný návrh grafického rozhrania. Návštevník galérie potrebuje jednoducho a rýchlo vyhľadať informácie o vystavovaných exponátoch. Zároveň tieto informácie zobraziť prehľadne a prirodzene, prostredníctvom rozšírenej reality. Administrátor potrebuje pomocou mobilnej aplikácie vytvárať skeny exponátov, ktoré neskôr slúžia na ich rozpoznanie. Okrem iného administrátor potrebuje pridávať a umiestňovať multimediálny obsah pomocou webového rozhrania, ktoré je na tieto úkony prehľadnejšie a teda vhodnejšie než použitie mobilného telefónu.

Pri implementácii boli využité najnovšie technológie, ktoré sú vhodné pre realizáciu navrhnutého systému a ktoré boli popísané v druhej kapitole tejto práce. Samotná implementácia kladie dôraz na kvalitný, jednoducho udržiavateľný a v neposlednom rade testovateľný kód, ktorý je umožňuje reagovať na zmeny zadania alebo prípadné pridávanie nových funkcionalít. Popis implementácie sa zameriava na prácu z rozšírenou realitou, ktorá je jadrom tejto práce.

Vznikli tak dve mobilné aplikácie, kde jedna slúži výhradne pre administrátora, ktorý spravuje danú galériu či múzeum, a druhá slúži výhradne pre návštevníka. Tieto aplikácie

boli doplnené o webovú aplikáciu, ktorá slúži administrátorovi pre prehľadnú správu. Spolu tak vytvárajú systém, ktorý poskytuje všetky požadované vlastnosti v rámci zadania.

Vytvorený systém je možné ďalej rozvíjať vo viacerých smeroch. Napríklad v rámci testovania bolo zistené, že nie vždy je v rámci automatického skenovania rozoznané umiestnenie exponátu. Tento algoritmus je teoreticky možné vylepšovať rôznymi spôsobmi, ktoré by za určitých podmienkok mohli dosahovať lepšie výsledky. Ďalším zlepšením by mohla byť optimalizácia pri aktualizácii databázy, kde by sa aktualizoval iba prvok v ktorom nastala zmena. Taktiež je možné pridávať nové funkcionality ako napríklad rozoznanie objektu nasnímaním QR kódu, ktorý by mohol obsahovať identifikátor objektu.

Výsledkom práce je systém, ktorý je bez ďalších úprav pripravený poskytnúť zážitok z rozšírenej reality návštevníkom galérií a múzeií.

Literatúra

[1] *Stack Overflow Developer Survey 2019*. [Online; navštívené 10.2.2020].
URL https://insights.stackoverflow.com/survey/2019

[2] *Global smartphone market Q4 and full year 2019*. 2020, [Online; navštívené 2.2.2020].
URL
https://canalys.com/newsroom/canalys-global-smartphone-market-q4-2019

[3] Cadena, C.; Carlone, L.; Carrillo, H.; aj.: Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age. *Trans. Rob.*, ročník 32, č. 6, December 2016: s. 1309–1332, ISSN 1552-3098, doi:10.1109/TRO.2016.2624754.
URL https://doi.org/10.1109/TRO.2016.2624754

[4] Eberly, D.: *Minimum-Area Rectangle Containing a Set of Points*. Máj 2015, [Online; navštívené 10.2.2020].
URL https://geometrictools.com/Documentation/MinimumAreaRectangle.pdf

[5] Fowler, M.; Rice, D.: *Patterns of Enterprise Application Architecture*. A Martin Fowler signature book, Addison-Wesley, 2003, ISBN 9780321127426.
URL https://books.google.cz/books?id=Jl5rkQnbfAIC

[6] Gershgorn, D.: *Google finally admits you don't need special hardware for augmented reality*. August 2017, [Online; navštívené 13.12.2018].
URL https://qz.com/1064898/google-finally-admits-all-you-need-for-augmented-reality-is-a-camera/

[7] Lee, J.; Revis, A.: *Largest Bay Area Museums*. August 2017, [Online; navštívené 14.12.2018].
URL https://www.bizjournals.com/sanfrancisco/subscriber-only/2017/08/25/sfmoma-california-academy-of-sciences.html

[8] Leutenegger, S.; Chli, M.; Siegwart, R. Y.: BRISK: Binary Robust invariant scalable keypoints. In *2011 International Conference on Computer Vision*, 2011, s. 2548–2555.

[9] Lu, F.; Milios, E.: Globally Consistent Range Scan Alignment for Environment Mapping. *Auton. Robots*, ročník 4, č. 4, Október 1997: s. 333–349, ISSN 0929-5593, doi:10.1023/A:1008854305733.
URL https://doi.org/10.1023/A:1008854305733

[10] MANSOOR IQBAL: *Pokémon GO Revenue and Usage Statistics (2020)*. 2020, [Online; navštívené 16.5.2020].
URL https://www.businessofapps.com/data/pokemon-go-statistics

[11] Martin, R. C.: The Open-Closed Principle. *C++ Report*, January 1996.
URL https://www2.cs.duke.edu/courses/fall07/cps108/papers/ocp.pdf

[12] Miesnieks, M.: *Why is ARKit better than the alternatives?* Júl 2017, [Online; navštívené 14.12.2018].
URL https://medium.com/6d-ai/why-is-arkit-better-than-the-alternatives-af8871889d6a

[13] Milgram, P.; Kishino, F.: A Taxonomy of Mixed Reality Visual Displays. *IEICE Trans. Information Systems*, ročník vol. E77-D, no. 12, 12 1994: s. 1321–1329.

[14] Nelson, R.: *ARKit-only Apps Surpass 13 Million Downloads in First Six Months, Nearly Half from Games*. Marec 2018, [Online; navštívené 28.12.2018].
URL https://sensortower.com/blog/arkit-six-months

[15] Ng, S.: *Beginning iOS 11 Programming with Swift*. AppCoda publisher, 2017, 28 s.
URL https://www.appcoda.com/swift/

[16] O'Grady, S.: *The RedMonk Programming Language Rankings: January 2018*. Marec 2018, [Online; navštívené 22.12.2018].
URL https://redmonk.com/sogrady/2018/03/07/language-rankings-1-18/

[17] Piao, J.-C.; Kim, S.-D.: Adaptive Monocular Visual–Inertial SLAM for Real-Time Augmented Reality Applications in Mobile Devices. *Sensors*, ročník 17, 11 2017: str. 2567, doi:10.3390/s17112567.

[18] Power, K.: *The Scene Graph*. Október 2013, [Online; navštívené 14.12.2018].
URL http://akmac.itcarlow.ie/~powerk/GeneralGraphicsNotes/scenegraph/scenegraph.htm

[19] Rohrer, C.: *When to Use Which User-Experience Research Methods*. 2015, [Online; navštívené 20.5.2020].
URL https://www.nngroup.com/articles/which-ux-research-methods/

[20] Rosten, E.; Drummond, T.: Machine Learning for High-Speed Corner Detection. In *Computer Vision – ECCV 2006*, editácia A. Leonardis; H. Bischof; A. Pinz, Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, ISBN 978-3-540-33833-8, s. 430–443.

[21] Schmalstieg, D.; Tobias, H.: *Augmented reality: principles and practice*. Addison-Wesley, 2016, ISBN 978-0-321-88357-5.

[22] Smith, R.; Self, M.; Cheeseman, P.: A Stochastic Map for Uncertain Spatial Relationships. In *Proceedings of the 4th International Symposium on Robotics Research*, Cambridge, MA, USA: MIT Press, 1988, ISBN 0-262-02272-9, s. 467–474.
URL http://dl.acm.org/citation.cfm?id=57425.57472

[23] Unger, R.; Chandler, C.: *A Project Guide to UX Design: For User Experience Designers in the Field or in the making*. Peachpit, 2009, ISBN 978-0-321-60737-9, 186 s.

[24] Wikipedia: *Tango (platform)*. [Online; navštívené 10.1.2019].
URL https://en.wikipedia.org/wiki/Tango_(platform)

Príloha A

Prípady užitia

Prípad užitia č. 1

Názov prípadu užitia	Vytvorenie exponátu v mobilnej aplikácií
Zúčastnený aktéri	Administrátor objektu
Tok udalostí	1. Administrátor v zozname exponátov vyberie možnosť vytvoriť nový exponát. 2. Administrátor pre každý jazyk vyplní základné údaje — názov, popis, poradie, obrázok. Poradie a kód sú údaje spoločné pre všetky jazyky. 3. V prípade, že exponát disponuje audio nahrávkou, administrátor pre každý jazyk zvolí audio súbor. 4. Administrátor zvolí 3D model, poprípadne obrázok s nastavenou šírkou v cm, na základe ktorého sa detekuje objektu. 5. V prípade, že exponát disponuje 3D model, administrátor zvolí 3D súbor. 6. Administrátor pre každý jazyk vloží doplnkové informácie o objekte, ktoré sa budú zobrazovať v rozšírenej realite vrátane posunutia v osy x, y, z a rotácie okolo osy x, y, z detekovaného exponátu. 7. Administrátor uloží exponát.
Vstupné podmienky	Administrátor sa prihlási svojimi prihlasovacími údajmi do aplikácie.
Výstupné podmienky	Administrátor vidí pridaný exponát v zozname všetkých exponátov.
Požiadavky kvality	Pred uložením dochádza k validácia povinných údajov. Po zadaní každého dátovo náročného objektu (fotka, video…) dochádza k uloženiu objektu na vzdialený server, aby sa znížila časová potreba pri ukladaní celého exponátu v poslednom kroku.

Prípad užitia č. 2

Názov prípadu užitia	Zobrazenie dodatočných informácií o vystavovanom exponáte návštevníkom múzea či galérie
Zúčastnený aktéri	Návštevník
Tok udalostí	1. Návštevník prepne aplikáciu do módu snímania exponátou zadnou kamerou mobilného telefónu. 2. Návštevník je vodiacimi značkami informovaný o priebehu a krokoch potrebných pre kalibráciu snímacieho zariadenia. 3. Návštevník namiery kameru na pozorovaný objekt. 4. Návštevníkovi sa zobrazia doplnkové informácie o objekte v rozšírenej realite — napríklad fotka, text, video či audio stopa. 5. Návštevník rozklikne detail exponátu. 6. Návštevníkovi je zobrazený textový popis exponátu, audio a video v závislosti na dostupnosti pre daný exponát.
Vstupné podmienky	Spustená aplikácia s aktuálnou databázou
Výstupné podmienky	
Požiadavky kvality	Detekcia musí prebehnút v uživateľsky prívetivom čase — do 3 sekúnd. Umiestnenie 3D objektov v scéne musí byť statické i v prípade, že sa návštevník s telefónom pohybuje v priestore.

Príloha B

Obsah priloženého pamäťového média

- /
  - `visitorMobileApp` ..... Zdrojové kódy mobilnej aplikácie pre návštevníka
  - `adminMobileApp` ..... Zdrojové kódy mobilnej aplikácie pre administrátora
  - `adminWebApp` ..... Zdrojové kódy webovej aplikácie pre návštevníka
  - `server` ..... Zdrojové kódy serveru
  - `text` ..... Zdrojové kódy textu
  - `documents`
    - `poster.pdf` ..... Plagát prezentujúci výsledok práce
    - `video.mp4` ..... Video prezentujúce výsledok práce
  - `dp.pdf` ..... Text práce

Príloha C

Plagát