VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

VÝVOJ SOFTVÉRU PRE SPRACOVANIE VÝSLEDKOV VIZUÁLNEHO POZOROVANIA METEOROV

SOFTWARE FOR PROCESSING DATA FROM VISUAL OBSERVATIONS OF METEORS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ADAM BARČA**
AUTHOR

VEDOUCÍ PRÁCE — **Mgr. JOZEF DRGA**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Barča Adam**
Program: Informační technologie
Název: **Vývoj software pro zpracování výsledků vizuálního pozorování meteorů**
Software for Processing Data from Visual Observations of Meteors
Kategorie: Informační systémy

Zadání:

1. Nastudujte teorii meteorů a jejich vizuální pozorování.
2. Analyzujte existující řešení pro zpracování výsledků IMO (International Meteor Organisation).
3. Navrhněte aplikaci pro Android na zpracování výsledků vizuálních pozorování. Zapisovatel bude mít možnost zadávat data v přehledné podobě. Výstupem bude protokol o vizuálním pozorování dle standardu IMO.
4. Navrženou aplikaci propojte s aktuální databází vizuálních pozorování IMO VMDB.
5. Aplikaci implementujte na mobilním telefonu/tabletu.
6. Proveďte zkušební pozorování a zpracování výsledků.

Literatura:

- Drga, Petrík, at al., Vizuálne pozorovania ako alternatívne cvičenia z fyziky, https://ufv.science.upjs.sk/_projekty/smolenice/pdf_15/10_drga_petrik.pdf
- www.imo.net
- Begeni Peter, Základy pozorovania nočnej oblohy, HaP Prešov, http://astro.begi.sk/publ/zaknoc/zaklpoz.htm

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

Vedoucí práce: **Drga Jozef, Mgr.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 9. listopadu 2021

Abstrakt

Cieľom práce bolo navrhnúť a vytvoriť aplikáciu pre spracovanie výsledkov vizuálneho pozorovania meteorov. Toho bolo dosiahnuté, výsledkom je aplikácia určená pre platformu Android, ktorá je zverejnená na službe Google Play. Aplikácia IMO je implementovaná za pomoci rámcov Spring Boot a Ionic. Ako úložisko pre dáta je použitá databáza PostgreSQL.

Abstract

Goal of the thesis was to design and implement an application for processing data, gathered during visual observation of meteors. Result is the IMO application, meant for the Android platform and distributed via Google Play service. IMO is built on Spring Boot and Ionic frameworks, PostgreSQL is used as data storage.

Klíčová slova

mobilná aplikácia, Spring Boot, PostgreSQL, Ionic, softvér, pozorovanie meteorov, Android

Keywords

mobile application, Spring Boot, PostgreSQL, Ionic, software, meteor observation, Android

Citace

BARČA, Adam. *Vývoj softvéru pre spracovanie výsledkov vizuálneho pozorovania meteorov*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Mgr. Jozef Drga

Vývoj softvéru pre spracovanie výsledkov vizuálneho pozorovania meteorov

Prohlášení

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Mgr. Jozefa Drgu. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Adam Barča
8. května 2022

Poděkování

Chcem sa poďakovať pánovi Mgr. Jozefovi Drgovi za odbornú pomoc pri návrhu a tvorbe tejto bakalárskej práce.

Obsah

1 Úvod **2**

2 Pozorovanie nočnej oblohy **3**
2.1 Amatérske pozorovanie meteorov 3
2.2 Základné pojmy 4
2.3 Proces pozorovania meteorov 6

3 Princíp tvorby mobilných aplikácii **10**
3.1 Mobilná aplikácia 10
3.2 Trojvrstvová architektúra 12

4 Použité technológie **14**
4.1 Back-end 14
4.2 Front-end 15

5 Analýza aplikácie **17**
5.1 Analýza IMO protokolu 17
5.2 Diagram prípadov použitia 21

6 Návrh aplikácie **24**
6.1 ER diagram 24
6.2 Návrh užívateľského rozhrania 28
6.3 Architektúra MVC 29

7 Implementácia aplikácie **35**
7.1 Back-end 35
7.2 Front-end 41

8 Testovanie **47**
8.1 Testovanie vývojárom 47
8.2 Testovanie užívateľmi 48

9 Záver **50**

Literatura **51**

A Obsah priloženého pamäťového média **53**

Kapitola 1

Úvod

Mobilné aplikácie zjednodušujú každodennú prácu a sú navrhnuté tak, aby boli, čo najefektívnejšie a najjednoduchšie. Vďaka tomu nám umožňujú pracovať s dátami v prehľadnej a ľahko pochopiteľnej podobe. Pomocou mobilných aplikácii si môžeme plánovať dni, zisťovať informácie o svojom bankovom účte alebo uľahčovať si nejakú pravidelnú rutinu. V súčasnej dobe dominujú dva operačné systémy na trhu, a to Android a iOS. Z toho dôvodu musíme vytvárať aplikácie na dve platformy, čo znamená, že buď budeme písať dve aplikácie v rôznych programovacích jazykoch alebo jednu aplikáciu, ktorá bude môcť bežať na obidvoch operačných systémoch.

Táto práca sa venuje vytvoreniu mobilnej aplikácie, ktorá uľahčuje zapisovanie meteorov a tieto dáta pôjdu poslať do databázy IMO VMDB. Pomocou tejto aplikácie bude mať používateľ v prehľadnej podobe usporiadané dáta, ktoré sú nutné pri každom pozorovaní meteorov a tiež bude mať možnosť vytvárať protokoly počas pozorovania, ktoré bude môcť poslať na stránku IMO (International Meteor Organization)[1], kde by mal mať založený účet. Používateľ tejto aplikácie by mal mať základné znalosti pozorovania meteorov, a ako sa zapisuje do IMO protokolu, ktorý zaznamenáva dáta o pozorovaní. Aplikácia bola vytvorená ako hybridná aplikácia a navrhnutá pre operačný systém Android s možnosťou rozšírenia na iOS alebo web. V kapitole 2 sú popísané základné pojmy ohľadom pozorovania meteorov a tiež priebeh takého pozorovania. Ďalej je v tejto práci popísané aké typy mobilných aplikácií poznáme a tiež, pomocou ktorých technológii ich môžeme tvoriť. Technológie, ktoré boli použité v tejto práci či už za účelom implementácie aplikácie alebo tvorenia dizajnu sú popísané v kapitole 4. V kapitole 5 je popísaná analýza IMO protokolu a jeho časti a tiež vytvorený diagram prípadov použitia. Návrh aplikácie či už databázy pomocou ER (Entity-relationship) diagramu alebo dizajnu pomocou rôznych náčrtov je popísaný v kapitole 6. V kapitole 7 je popísaný postup implementácie aplikácie pomocou rámca Spring a Ionic, kde sú vysvetlené základné pojmy spojené s týmito technológiami a tiež ukážky kódu, ktoré sú súčasťou tejto práce. Priebeh testovania aplikácie spolu s tabuľkami protokolov je popísané v kapitole 8. Na záver, v kapitole 9 vyhodnocujem celkovú prácu a tiež popisujem možné rozšírenia aplikácie. Výsledná aplikácia je zverejnená v obchode Google Play[2] pre jej jednoduchú inštaláciu.

[1] oficiálna stránka IMO: https://www.imo.net

[2] aplikácia IMO dostupná v obchode Google Play https://play.google.com/store/apps/details?id=sk.adambarca.imo

Kapitola 2

Pozorovanie nočnej oblohy

Pokiaľ nie je uvedené inak, tato kapitola bola prevzatá z návodu Základy pozorovania nočnej oblohy, ktorý napísal pán Peter Begini [23]. Pozorovanie nočnej oblohy je proces, pri ktorom sa pozorovateľ pozeraná na nočnú oblohu za určitým cieľom. Tento cieľ môže byť zber štatistík, hľadanie nových telies vo vesmíre alebo fotografovanie telies. Pozorovať telesá vo vesmíre môžu vedci, prístroje, ale aj obyčajní ľudia, ktorí by mali záujem o pozorovanie meteorov. Pozoruje sa podstate všetko vo vesmíre, čo nie je prázdnota a nemusia to byť ani pevné objekty, ale kľudne môžu byť plynného skupenstva, ako je hmlovina alebo plazmového. V tejto kapitole sa zameriame na pozorovaciu metódu meteorov, ktorú môžu robiť obyčajní ľudia, ktorí majú záujem a chuť pozorovať meteory. Budú tu vysvetlené základne pojmy, ako sú meteor, meteorický roj, rektascenzia alebo deklinácia a základný postup, ako prebieha pozorovanie meteorov.

2.1 Amatérske pozorovanie meteorov

Amatérske pozorovanie meteorov je aktivita, do ktorej sa môže zapojiť ktokoľvek kto by mal o to záujem. Väčšina ľudí to má ako koníček alebo ako oddychovú činnosť. Pri pozeraní meteorov ide hlavne o zber štatistík, ktorý slúži na vypočítanie **zenitovej hodinovej frekvencie** ktorá je lepšie popísaná v sekcií 2.2 a **populačného indexu**. Meteory sa pozorujú väčšinou v skupinkách, ale môžu pozorovať aj jednotlivci za pomoci techniky. Táto práca hovorí o amatérskom pozorovaní meteorov, kde ľudia nemusia mať vôbec žiadnu techniku na to, aby dokázali správne odpozorovať meteory. Záznamy, ktoré sa robia počas pozorovania sa zapisujú buď do protokolu alebo sa použije nejaké zariadenie na nahrávanie hlasu a potom následne sa tento záznam prevedie do protokolu, alebo do počítačového programu, ktorý by nám umožňoval daný záznam zmeniť do protokolu. Tento protokol, ktorý sa vyplní počas pozorovania je možné poslať buďto papierovou formou alebo elektrickou. IMO[1] (International Meteor Organization) je celosvetová organizácia združujúca amatérskych pozorovateľov meteorov. Táto organizácia nám umožňuje posielať protokoly elektrickou, formou ktoré sa ukladajú do ich databázy VMDB. Databáza je verejne prístupná a slúži na výpočet rôznych štatistík. Pretože človek nedokáže pozorovať meteory v takej kvalite, ako prístroje tak sa táto aktivita stratila do pozadia a teraz je to už len voľno časová aktivita. To ale neznamená že úplne vymrela stále ešte môžu aj amatérski pozorovatelia prispieť svojimi pozorovaniami, a to tým, že si budú zaznamenávať počet videných meteorov za noc. Tento údaj a ešte

[1] oficiálna stránka IMO: https://www.imo.net/

niektoré, ktoré môžu byť na vyše slúži k lepšej štatistike o rojoch a k vypočítaniu zenitovej frekvencie.

2.2 Základné pojmy

V tejto sekcií budú vysvetlené základne pojmy, ktoré sa týkajú pozorovania meteorov a ktoré je treba pochopiť a naučiť sa k tomu, aby sme dokázali správne vyplňovať protokol. Sú tu vysvetlené pojmy ako meteor, meteorický roj, ale aj fyzikálne vzorce, ktoré nám popisujú, ako sa vypočíta daná štatistika.

Meteor

Meteor je malé meteoroidné teliesko putujúce slnečnou sústavou, ktoré vošlo do našej zemskej atmosféry. Meteor je známy ako krátky svetelný záblesk na oblohe.
Meteory môžeme deliť podľa ich jasnosti a veľkosti do šiestich skupín:

- **mikrometeory**, ktoré niesu viditeľné
- **teleskopické meteory**, ktoré sú viditeľné len pomocou teleskopu
- **lietavice**
- **bolidy**
- **detonujúce bolidy** sú bolidy sprevádzajúce so zvukovým efektom
- **meteority** väčšinou dokážu dopadnúť na zemský povrch

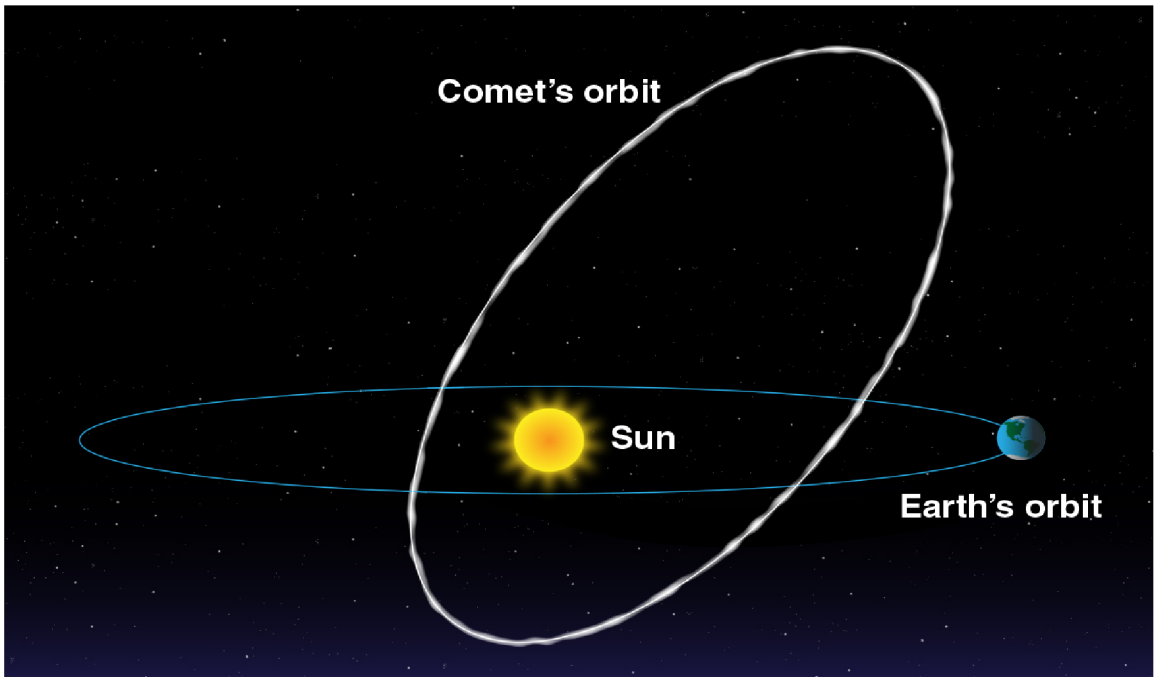
Ďalej môžeme deliť meteory aj podľa ich typu k rojovej príslušnosti:

- **sporadické meteory** sú meteory bez rojovej príslušnosti. Tieto meteory sú úplne náhodne a prichádzajú z všetkých smerov. Tieto meteory sú aktívne počas celého roka, a to frekvenciou 5 až 20 meteorov za hodinu.
- **rojové meteory** sú meteory, ktoré patria do nejakého konkrétneho meteorického roju. Ak si nakreslíme dráhu rojových meteorov k ich začiatku pretnú sa v malej oblasti na oblohe, ktorá sa nazýva **radiant**. Jednotlivé meteorické roje sú pomenované podľa toho, v ktorom súhvezdí sa nachádza ich radiant. Rojové meteory sú na rozdiel od sporadických aktívne iba v určitú dobu, ktorá sa každoročne opakuje v ten istý deň. Vznik rojových meteorov je ukázaný na obrázku 2.1. Meteorické roje vznikajú rozpadom a uvoľňovaním častíc z komét alebo planétiek spravidla v blízkosti Slnka, kde sa rozptyľujú malé častice popri celej jej obežnej dráhy. Obežná dráha sa musí krížiť s obežnou dráhou Zeme, a preto sú meteorické roje aktívne iba v určitú dobu v roku, keď Zem prechádza v blízkosti dráhy.

Zenitová hodinová frekvencia

Zenitová hodinová frekvencia (ZHR) je počet meteorov, ktorý je možné vidieť za hodinu na čistej a jasnej oblohe. Táto štatistika sa hlavne vyhodnocuje pri amatérskych pozorovaniach meteorov. Vzorec na výpočet ZHR, ktorý je prevzatý z [3] je vysvetlený v 2.1.

$$ZHR = \frac{N * F * r^{6.5-lm}}{T_{eff} * \sin(h)} \tag{2.1}$$



Obrázek 2.1: Priebeh meteorického roju prevzatý z [21]

- **N** je počet pozorovaných meteorov
- **F** je faktor obmedzenia pozorovacieho poľa vzorec na výpočet 2.2

$$F = \frac{1}{1 - K} \tag{2.2}$$

  kde K je priemerné zakrytie pozorovacieho poľa v desatinnom tvare

- **r** je populačný koeficient roja vzorec na výpočet 2.3

$$r = \frac{N(m+1)}{N(m)} \tag{2.3}$$

  kde N(m+1) je počet meteorov m+1 hviezdnej veľkosti a N(m) je počet meteorov m-tej hviezdnej veľkosti

- **lm** je medzná hviezdna veľkosť
- $\mathbf{T_{eff}}$ je efektívny čas pozorovania
- **h** je výška riadantu nad obzorom

Medzná hviezdna veľkosť

Medzná hviezdna veľkosť vyjadruje citlivosť vášho oka na dane pozorovacie podmienky. Čiže kvalita pozorovania sa odvíja od zraku pozorovateľa, tmavosti oblohy a zamračenosti. Každý pozorovateľ má inú medznú veľkosť, a preto by sa mala vypočítať individuálne. Jednotka medznej veľkosti je **magnitúda**. Platí čím nižšia magnitúda tým je hviezda jasnejšia

napr. Slnko má -26 mag, Mesiac v splne -12.5 mag a bolidy viac ako -4 mag. IMO (International Meteor Organization) k vypočítaniu mezdnej veľkosti doporučuje tzv. trojholníkovú metódu. **Trojuholníková metóda** spočíva v tom, že máme isté oblasti definované na oblohe podľa IMO, ktoré sú buď trojuholníkového tvaru alebo štvoruholníkového tvaru príklad oblasti α Vir - ζ Vir - γ Vir je vidieť na obrázku 2.2. V týchto oblastiach spočítame koľko vidíme hviezd vnútri daného útvaru ako aj na okrajoch. Pomocou týchto dvoch údajov, ako je názov oblasti a počet videných hviezd môžeme pomocou tabuliek, ktoré sú na stránke IMO[2] vyjadriť medznú veľkosť.

Rektascenzia a deklinácia

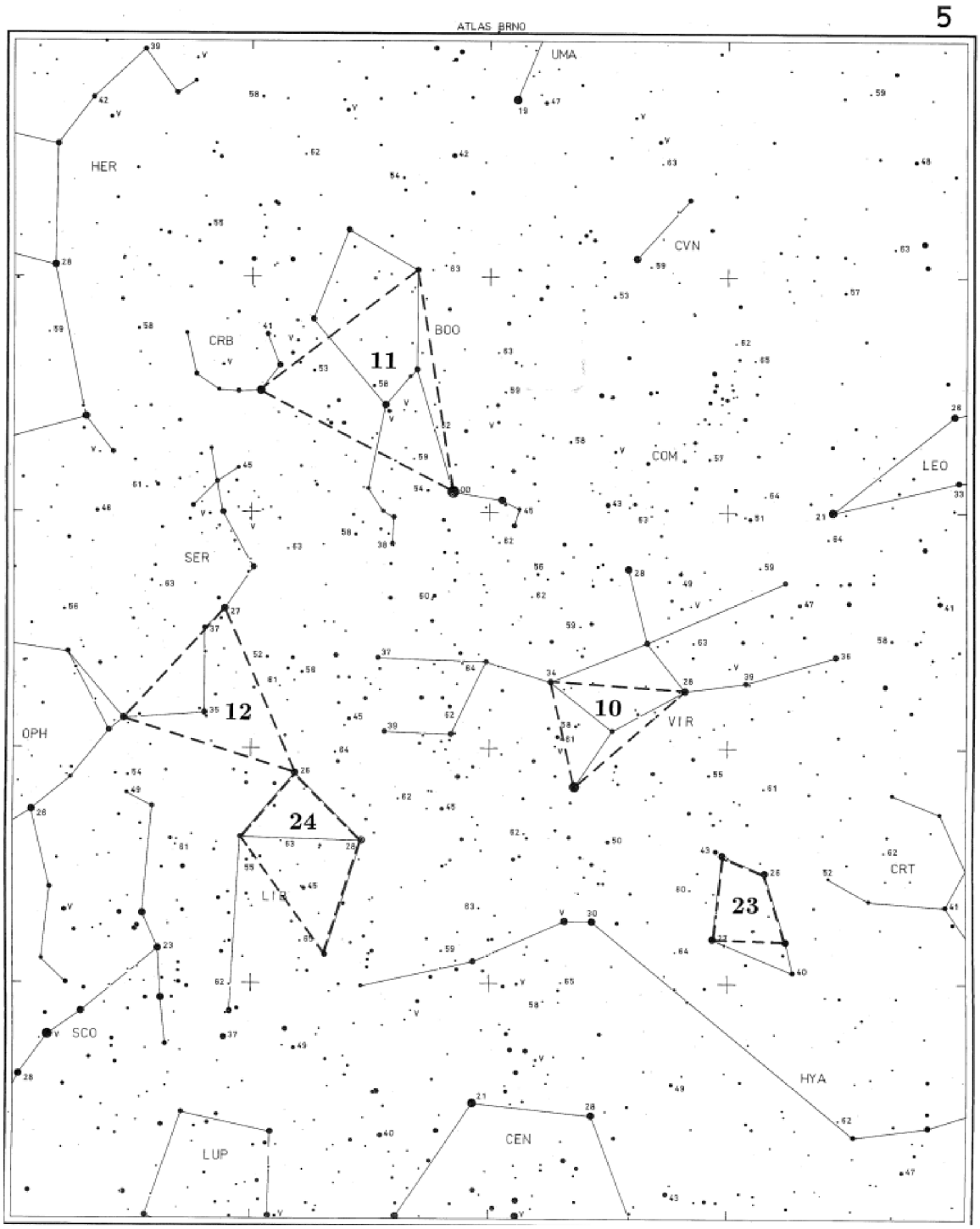
Rektascenzia a deklinácia určujú presné umiestnenie objektu na oblohe. Podobne, ako je to so zemepisnou šírkou a dĺžkou je to aj s rektascenziou a deklináciou pričom rektascenzia zodpovedá zemepisnej dĺžke a deklinácia zemepisnej šírke na nebeskej sfére. Tieto súradnice a ich znázornenie môžete vidieť na obrázku 2.3. Rektastenzia sa meria od nulového bodu, ktorý sa nachádza na mieste, kde sa Slnko nachádza v čase jarnej rovnodennosti. Oblohu možno považovať za hodiny, pretože sa Zem otáča, preto sa rektascenzia vyjadruje iba v jednom smere na východ. Môže mať hodnoty od 0 až po 24 hodín pričom platí, že 1 hodina sa rovná 15°, pretože $360/24 = 15$. Deklinácia môže mať hodnoty od -90° do 90° pričom platí že všetko na sever od rovníka má kladné znamienko až po 90°, čo je deklinácia severného nebeského pólu a všetko južne od rovníka má záporné znamienko až po -90°, čo je deklinácia južného nebeského pólu.

2.3 Proces pozorovania meteorov

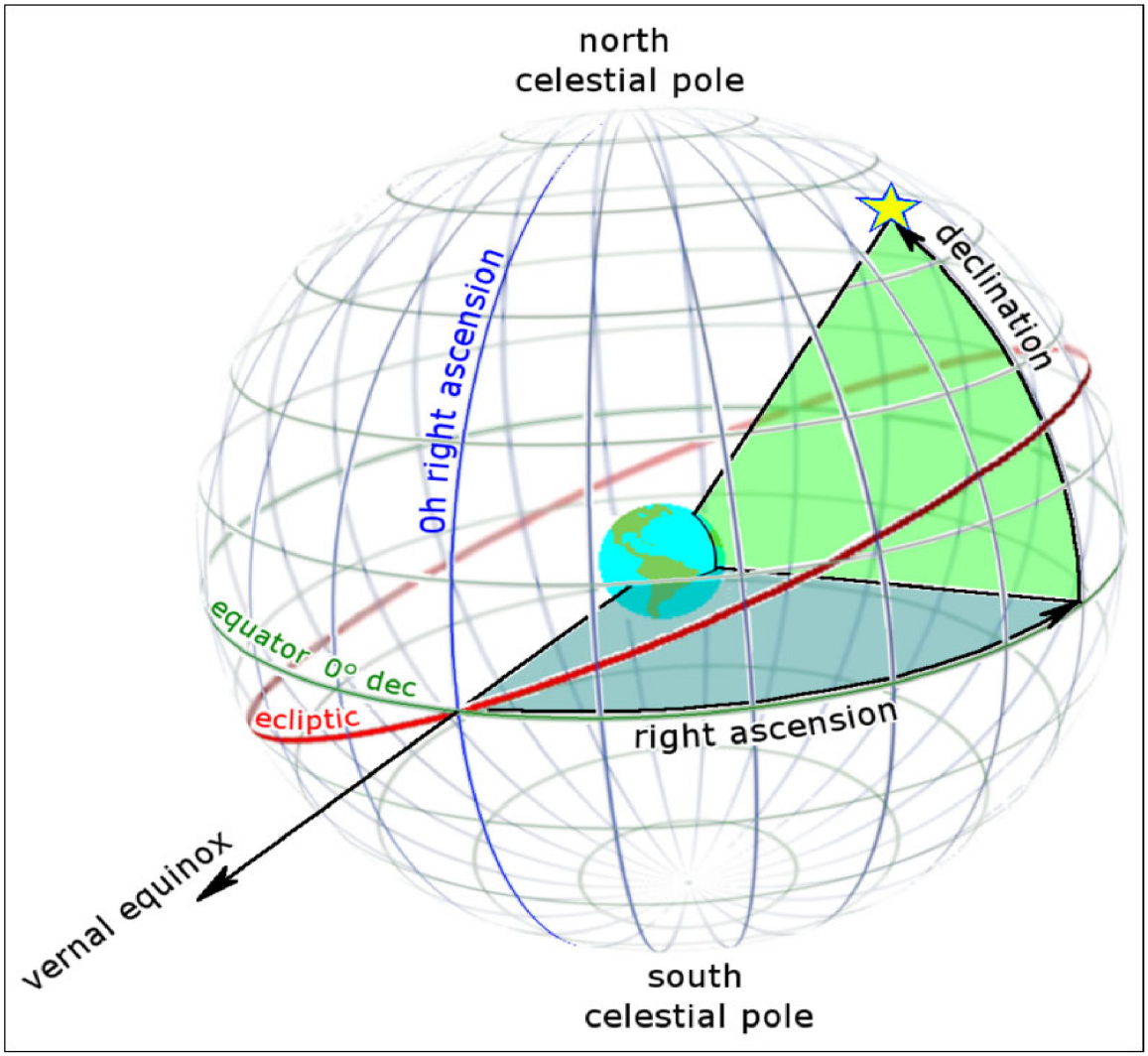
Pozorovanie prebieha väčšinou v menších skupinkách do desať ľudí pričom by aspoň jeden mal byť zapisovateľ. Pozorovanie by sa malo odohrávať na odľahlých miestach, kde kvalitu pozorovania nebude narušovať svetelný smog. Pretože pozorovanie prebieha večer a je to oddychová aktivita, pri ktorej sa človek veľmi nehýbe doporučuje sa dobre sa obliecť alebo zobrať so sebou spací vak. Ako prvé zapisovateľ zaznamená, kde sa celé pozorovanie odohráva. Potom sa skupinka dohodne, ktoré meteorické roje sa idú pozorovať väčšinou to bývajú iba roje, ktoré sú v danú dobu aktívne a sporadické meteory, ktoré sa zaznamenávajú vždy pri každom pozorovaní. Tieto údaje sa počas celého pozorovania nemenia. Ak sú predošlé údaje vyplnené tak si pozorovatelia rozdelia oblohu či už podľa svetových strán alebo iných preferencii. Počas toho, ako si každý pozorovateľ vyberie svoju časť oblohy nahlasujú rektascenziu a deklináciu zapisovateľovi. Zapisovateľ vytvára toľko protokolov koľko je aj pozorovateľov pre každého jeden. Pozorovatelia by sa nemali navzájom ovplyvňovať a mali by hlásiť údaje, ktoré sú špecifické pre nich. S týmto súvisí medzná hviezdna veľkosť, ktorá je pre každého individuálna a zaznamenáva sa pri každom intervale. Oblačnosť sa musí zadávať vždy pri jej zmene v percentách. Pokiaľ máme vyplnené tieto údaje u každého pozorovateľa môže začať pozorovanie, kde si zapisovateľ musí zaznamenať začiatok pozorovania. Pozorovatelia pozerajú na oblohu v intervaloch, ktoré trvajú približne 30 minút až 1,5 hodiny pričom by si pozorovatelia mali dávať prestávky, aby si oddýchli a hlavne, aby si oddýchol ich zrak. Počas pozorovania pozorovatelia hlásia, keď uvidia meteor na toto sa používa výraz „stop“. Pozorovateľ hlási svoje meno, aby zapisovateľ vedel komu má daný meteor pripísať, jeho rojovú príslušnosť, ktorá musí byť jedna z tých, na ktorých sa dohodli

[2]Tabuľky na vyjadrenie medznej hviezdnej veľkosti: https://www.imo.net/observations/methods/visual-observation/major/observation/

na začiatku, ktoré budú pozorovať a jeho jasnosť, čo sa udáva v magnitúdoch na presnosť 0.5 magnitúdy. Keď už pozorovateľ chce prestávku zahlási to zapisovateľovi, ktorý si musí zaznačiť dĺžku prestávky kvôli presnému efektívnemu určeniu času pozorovania. Počas tejto prestávky nemôže byť pozorovateľovi zaznamenaný žiadny meteor a mal by túto prestávku využiť na odpočinok. Po prestávke si vyberie opäť miesto na oblohe, kde sa bude pozerať a nahlási zapisovateľovi nové miesto, kde sa pozorovateľ pozerá. Čiže musí povedať novú rektascenziu, deklináciu, oblačnosť pokiaľ ju zaznamenávajú na začiatku intervalu a nie na konci a vypočítať novú medznú veľkosť. Tento proces sa opakuje do vtedy do kedy majú pozorovatelia náladu môžu skončiť už za 30 minút a ďalej len odpočívať alebo môžu pozorovať celú noc. Na konci pozorovania zapisovateľ skontroluje protokol či má všetky povinné polia vyplnené a zapíše čas konca pozorovania. Z tohto pozorovania by mali vzniknúť protokoly, ktoré sa môžu poslať na stránku IMO (International Meteor Organization) kde sa z nich vyhodnocuje zenitová hodinová frekvencfia a ďalšie štatistiky.



Obrázek 2.2: Oblasť α Vir - ζ Vir - γ Vir prevzatá z [9]



Obrázek 2.3: Rovníková sústava súradníc prevzatá z [14]

Kapitola 3

Princíp tvorby mobilných aplikácii

V tejto kapitole bude vysvetlené aké typy mobilných aplikácii poznáme aké sú medzi nimi rozdiely a aké majú výhody a nevýhody oproti ostatným. Tiež tu bude vysvetlená trojvrstevná architektúra, ktorá sa najviac využíva pre aplikácie klient-server.

3.1 Mobilná aplikácia

Mobilná aplikácia je typ aplikácie navrhnutý pre mobilné zariadenia, ktorým môže byť smartfón alebo tablet. Aplikácie na mobil bývajú väčšinou malé aplikácie zamerané na obmedzujúcu funkčnosť, ako napríklad kalkulačka alebo poznámkový blok. Mobilné aplikácie môžeme deliť podľa technológie vývoja mobilných aplikácii, a to na natívne a hybridné [20].

Natívna mobilná aplikácia

Natívna aplikácia je označenie pre aplikáciu, ktorá je vytvorená pre konkrétnu platformu ako je Android, iOS, Windows a podobne. Na žiadnej inej platforme nie je možné natívne aplikácie spustiť [17]. Oficiálne vývojové prostredie pre Android je Android Studio, kde je možné vytvárať aplikácie pomocou jazyku Kotlin, Java alebo C++. Všetky jazyky iných ako JVM (Java virtual machine) ako napríklad Go, JavaScript, C, C++ alebo assembly potrebujú pomoc jazykového kódu JVM [28]. Xcode je oficiálne IDE (integrated development environment) pre platformu iOS od spoločnosti Apple. Xcode nám poskytuje nástroje pre vytváranie softvéru, ktoré bežia na zariadeniach s operačným systémom iOS, iPadOS, tvOS, macOS a watchOS. V Xcode môžeme vyvíjať pomocou jazyku Swift, C, C++ a Objective-C. Xcode má nevýhodu, že musí bežať na systéme macOS pričom existujú nástroje na tvorbu aplikácii pre Apple aj vo Windowse alebo v Linuxe, ale tie nemusia byť také spoľahlivé [12].

Výhody natívnych aplikácii

- Môže fungovať aj bez pripojenia k internetu
- Vie komunikovať s hardvérom zariadenia, ako je napríklad fotoaparát alebo batéria
- Vie lepšie využiť vlastnosti zariadenia a tým pádom sú rýchlejšie
- Môže použiť ďalšie vlastnosti systému, ako sú pohybové gestá
- Využitie animácii, komponentov a prvkov pre danú platformu

Nevýhody natívnych aplikácii

- Musíte mať aplikáciu nainštalovanú na danom zariadení
- Aktualizácia aplikácie
- Drahší vývoj pokiaľ chceme aplikáciu na viacej platforiem

Webová aplikácia

Webová aplikácia je aplikácia, ktorá používa webový prehliadač ako prostriedok na komunikáciu s užívateľom. Je závislá na internete, ktorý slúži ako lacný komunikačný kanál medzi klientom a serverom. Webová stránka sa vyvíja v dvoch typoch jazykoch. Kód na strane servera sa zaoberá ukladaním a získavaním informácii a programuje sa napríklad jazykmi ako je Python, PHP alebo Java. Kód na strane klienta má na starosti prezentáciu informácii užívateľovi a vyvíja sa pomocou jazykov HTML, CSS a JavaScript, ktoré podporujú webové prehliadače [25].
Webová aplikácia potrebuje tri veci:

- webový server na spracovanie požiadaviek od klienta
- aplikačný server na vykonanie požiadaviek od klienta
- databázu na ukladanie informácii

Výhody webových aplikácii

- Nemusí sa inštalovať
- Aktualizácia aplikácie, pretože aktualizácia sa aplikuje centrálne
- Pomocou webového prehliadača môžete pritupovať k webovým aplikáciam odkiaľkoľvek
- Znižujú pirátstvo vo webových aplikáciach založených na predplatnom

Nevýhody webových aplikácii

- Je závislá na internete od jeho kvality sa odvíja rýchlosť toku dát a aj samostatná práca s aplikáciou
- Bezpečnostné riziká úniku dát v prípade nespoľahlivej poskytnutej služby
- Nemožnosť vývoja náročných aplikácii

Hybridná mobilná aplikácia

Hybridné aplikácie sú zmesou natívnych a webových riešení. Jadro aplikácie je napísané webovými technológiami ako HTML, CSS a JavaScript 3.1, ktoré sú následne zapuzdrené v natívnej aplikácii. Namiesto toho, ale aby sa aplikácia spustila v klasickom prehliadači, je spustená z natívnej aplikácie, ktorá má svoj vlastný vstavaný prehliadač. Operačný systém iOS používa na zobrazovanie **WKWebView** zatiaľ, čo Android používa na tú istú funkciu **WebView**. Kód napísaný pomocou HTML, CSS a JavaScript sa vloží do natívneho obalu

pomocou riešenia, ako je Apache Cordova alebo Ionic´s Capacitor. Cordonova a aj Capacitor majú systém modulov, ktoré umožňujú prístup k hardvérovým prvkom zariadenia, ako je fotoaparát alebo TouchID. Na tvorbu hybridných aplikácii sa používajú technológie Ionic, NativeScript, Xamarin, React Native a ďalšie [6].

Výhody hybridných aplikácii

- Podpora viacerých platforiem
- Možnosť využívať hardvérových prvkov daného zariadenia
- Lacnejší vývoj pokiaľ chceme aplikáciu do viacej platforiem

Nevýhody hybridných aplikácii

- Musíte mať aplikáciu nainštalovanú na danom zariadení
- Je závislá na internete od jeho kvality sa odvíja rýchlosť toku dát a aj samostatná práca s aplikáciou
- Majú menší výkon ako natívne mobilné aplikácie

3.2 Trojvrstvová architektúra

Trojvrstevná architektúra, ktorá je zobrazená na obrázku 3.1 je dobre zavedená architektúra softvérových aplikácii, ktorá organizuje aplikáciu do troch vrstiev:

- prezentačná vrstva alebo používateľské rozhranie
- aplikačná vrstva
- dátová vrstva

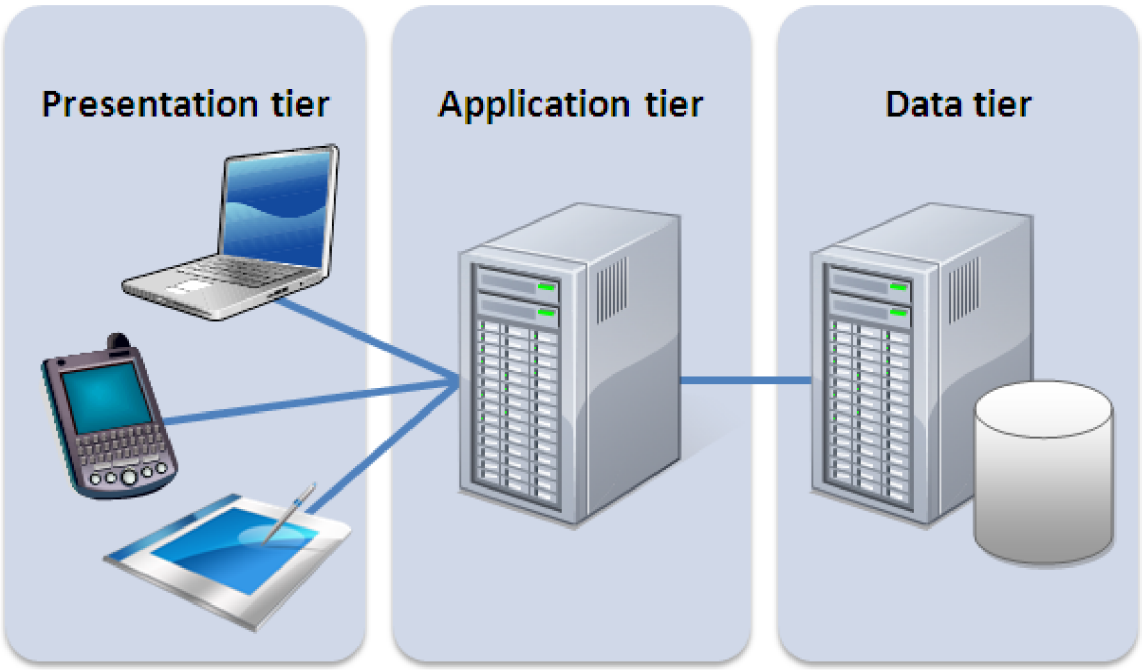
Hlavnou výhodou trojvrstvovej architektúry je, že každá vrstva beží na svojej vlastnej infraštruktúre vďaka čomu môže byť súčasne vyvíjané viacej vrstiev a môže byť škálovateľná, alebo aktualizovaná bez vplyvu na ostatné vrstvy [8].

Prezentačná vrstva

Prezentačná vrstva je používateľské rozhranie a komunikačná vrstva aplikácie, kde používateľ interaguje s aplikáciou. Jej hlavnou úlohou je prezentácia dát používateľovi a reagovať na príkazy používateľa.

Aplikačná vrstva

Aplikačná vrstva alebo logická vrstva je vrstva, kde sa informácie spracovávajú, ktoré sú zhromaždené z prezentačnej vrstvy. Aplikačná vrstva môže pridávať, odstraňovať a upravovať údaje v dátovej vrstve.



Obrázek 3.1: Schéma trojvrstvovej architektúry prevzatá z [16]

Dátová vrstva

Dátová vrstva alebo databázová vrstva je vrstva, kde sa ukladajú a spravujú informácie spracované aplikáciou. Môže to byť systém riadenia správy dát, ako je PostgreSQL alebo to môže byť databázový server NoSQL, ako je MongoDB. V trojvrstvovej aplikácii všetka komunikácia prechádza cez aplikačnú vrstvu. Prezentačná vrstva a dátová vrstva nemôžu spolu priamo komunikovať.

Výhody trojvrstvovej architektúre

- Rýchlejší vývoj
- Vylepšená škálovateľnosť
- Vylepšená spoľahlivosť
- Vylepšená bezpečnosť

Kapitola 4

Použité technológie

V tejto kapitole sú popísané jednotlivé technológie, ktoré boli použité na vytvorenie aplikácie. Každá z týchto technológii slúži na niečo iné či už na vytvorenie serverovej časti (back-end), klientskej časti (front-end), databázy alebo pri návrhu dizajnu aplikácie.

4.1 Back-end

Back-end (serverová časť) aplikácie je tá časť aplikácie, ktorá ma na starosti všetku logiku aplikácie. To znamená, že všetky výpočty, práca s databázou a poskytovania API (application programming interface). Serverová časť aplikácie bola vytvorená v jazyku Java s využitím rámca (framework) Spring.

Spring

Spring je najpopulárnejší aplikačným rámcom pre podnikovú Javu (enterprise Java). Pomocou springu dokážeme vytvárať vysoko výkonný, ľahko testovateľný a opakovane použiteľný kód. Spring rámec je open source a prvú verziu napísal Rod Johnson a bol vydaný pod licenciou Apache 2.0 v júni 2003. Hlavným prvkom Springu je jeho IoC (inversion of control) kontajner. Kontajner je zodpovedný za správu životných cyklov objektov, ako je vytváranie, volanie ich inicializačných metód a konfigurácie týchto objektov ich prepojením. Objekty je možné získavať pomocou vyhľadávania závislostí alebo pomocou dependency injection (predávania závislostí) [27]. Triedy aplikácie by mali byť, čo najviac nezávislé od iných tried vďaka čomu budú triedy ľahko znovu použiteľné a testovateľné.

Databáza

Databáza je organizovaná zbierka štruktúrovaných informácií alebo údajov, zvyčajne uložených elektronicky v počítačovom systéme. Databáza je zvyčajne riadená systémom bázy dát (SRBD). Spoločne sa dáta a SRBD spolu s aplikáciami, ktoré sú s nimi spojené označujú ako databázový systém. V relačných databázach sú dáta modelované v riadkoch a stĺpcov, ktoré sú v tabuľkách, čo umožňuje spracovanie a dopytovanie údajov efektívnejšie. Väčšina databáz používa na dopytovanie a písanie údajov štruktúrovaný jazyk SQL (Structured Query Language) [22]. V aplikácii sa použila databáza PostgreSQL, ktorá patrí medzi relačné databázi. PostgreSQL je známy svojou spoľahlivosťou, robustnosťou funkcií, ako je napríklad MVCC (Multi-Version Concurrency Control) a výkonom. PostgreSQL je open source, čo znamená, že jeho kód je verejne dostupný vďaka čomu môžeme databázu ľahko

rozširovať o nové funkcionality a tiež vďaka tomu má obrovskú komunitu vývojárov, ktoré vytvárajú tieto funkcionality. V PostgreSQL môžeme definovať svoje vlastné dátové typy, vytvárať vlastné funkcie a tiež písať kód v inom programovacom jazyku bez rekompilácie databázy [24].

4.2 Front-end

Front-end (klientská časť) je tá časť aplikácie, ktorú môže užívateľ vidieť a pracovať s ňou. Slúži na reprezentáciu objektov a procesov zo strany servera. Front-end by mal byť ľahko použiteľný a iteratívny, aby užívateľ dokázal, čo najrýchlejšie a najľahšie pracovať s aplikáciou [26]. Front-end aplikácie je odlišný na základe platformy, na ktorej beží. Napríklad inak vyzerá desktopová aplikácia oproti tej mobilnej pričom aj záleží na operačnom systéme, na ktorej aplikácia beží, pretože iOS má trochu odlišný prístup k zobrazovaniu niektorých prvkov ako Android.

HTML

HTML (HyperText Markup Language) je značkovací jazyk, ktorý sa používa na vytváranie základnej obsahovej kostry väčšiny webových stránok. Kedysi HTML slúžil tiež aj k formátovaniu vzhľadu, ale toto už neplatí, pretože k tomuto teraz slúži jazyk CSS (Cascading Style Sheets). Určité formátovanie, ale stále môže prebiehať v HTML, ale nie k vizuálnemu ale k významovému znázorneniu, ako je napríklad kurzíva alebo hyperlinky.

CSS

CSS (Cascading Style Sheets) je jazyk, ktorý sa používa na popis zobrazenia prezentácie dokumentu napísaneho v značkovacom jazyku, ako je HTML (HyperText Markup Language) alebo XML (Extensible Markup Language). Princíp jazyka je oddelenie obsahu dokumentu od jeho vizuálnej reprezentácie, ako je farba písma, alebo rozloženie. Toto umožňuje priehľadnejšiu stránku obsahu, ktorej znižuje zložitosť a opakovanie štrukturálneho obsahu [4].

Angular

Angular je open source rámec, ktorý slúži na vytváranie jednostránkových aplikácii napísaných v jazyku TypeScript, čo je vlastne striktná nadmnožina JavaSriptu. Angular je vyvinutý a podporovaný spoločnosťou Google, ktorá má veľkú dôveru v jeho stabilitu. Pôvodne mal byť druhou verziou rámca AngularJS, ale kvôli nedostatku spätnej kompatibility bol vydaný ako samostatný rámec. JavaSript je najčastejšie používaný programovací jazyk webových aplikácii na strane klienta. Je vložený do dokumentov HTML, aby umožnil interakciu s online stránkami rôznymi spôsobmi [18].

Ionic

Ionic je rámec na vývoj hybridných mobilných aplikácii. Hybridná aplikácia je vytvorená pomocou technológii, ako je HTML, CSS a JavaScript, ktoré sú kompatibilné aj s webovými aplikáciami. Viac o hybridných aplikáciách je popísané v sekcií 3.1. Ionic nám ponúka knihovňu komponentov pre užívateľské rozhranie. Medzi základne komponenty, čo nám Ionic ponúka patrí zoznam, tlačidlo, karta atď. Tieto komponenty sú pripravené vo dvoch verziách

a to pre iOS a Android, ktoré prispôsobia svoj vzhľad na základe operačného systému, kde aplikácia beží. Rámce, ktoré môžeme využiť pre vývoj Ionic aplikácii sú Angular, React, Vue, ale môžeme tvoriť aplikácie aj bez použitia rámca, čiže v čistom jazyku JavaScript. [24].

Figma

Figma je cloud-based kolaboračný dizajnový nástroj, kde sa naviac zmeny prejavujú v reálnom čase. Na raz vo Figme môže pracovať až 100 ľudí na jednom projekte. Figma nám umožňuje vytváranie komponentov, ktoré môžeme viac krát použiť, prezentáciu projektu pomocou interaktívneho prototypu a ukazovanie CSS vlastnosti jednotlivých prvkov. Aj keď Figma beží v cloude je možnosť ju používať aj v off-line režime pričom sa samozrejme nedá pracovať v reálnom čase, čo nám neumožňuje vidieť prácu ostatných členov v reálnom čase a tiež ani našu prácu nevidia ostatní členovia tímu v reálnom čase [13].

Kapitola 5

Analýza aplikácie

Hlavným cieľom analýzy je vytvoriť model, ktorý zachytáva požiadavky zákazníka. Model špecifikuje aké možnosti môže používateľ so systémom robiť a tiež akého typu používateľov poznáme. Model by mal byť, čo možno najjednoduchší a hlavne najpresnejší. Hlavnou úlohou aplikácie je uľahčiť prácu zapisovateľovi pri meteorových pozorovaniach. Zapisovateľ by mal mať možnosť vytvárať pomocou aplikácie protokoly, ktoré sú podľa štandardu IMO (International Meteor Organisation) a následne mať možnosť tieto protokoly posielať do databáze vizuálneho pozorovania IMO VMDB. Pretože väčšinou pozorovanie prebieha v skupinách a na rovnakých miestach budú mať používatelia možnosť vytvárať aj lokalizáciu a pozorovateľov. Požiadavky na aplikáciu boli vytvorené na základe konzultácii s vedúcim práce Mgr. Jozef Drga, ktorý už 22 rokov pozoruje meteory a tiež bude budúcim používateľom aplikácie.

5.1 Analýza IMO protokolu

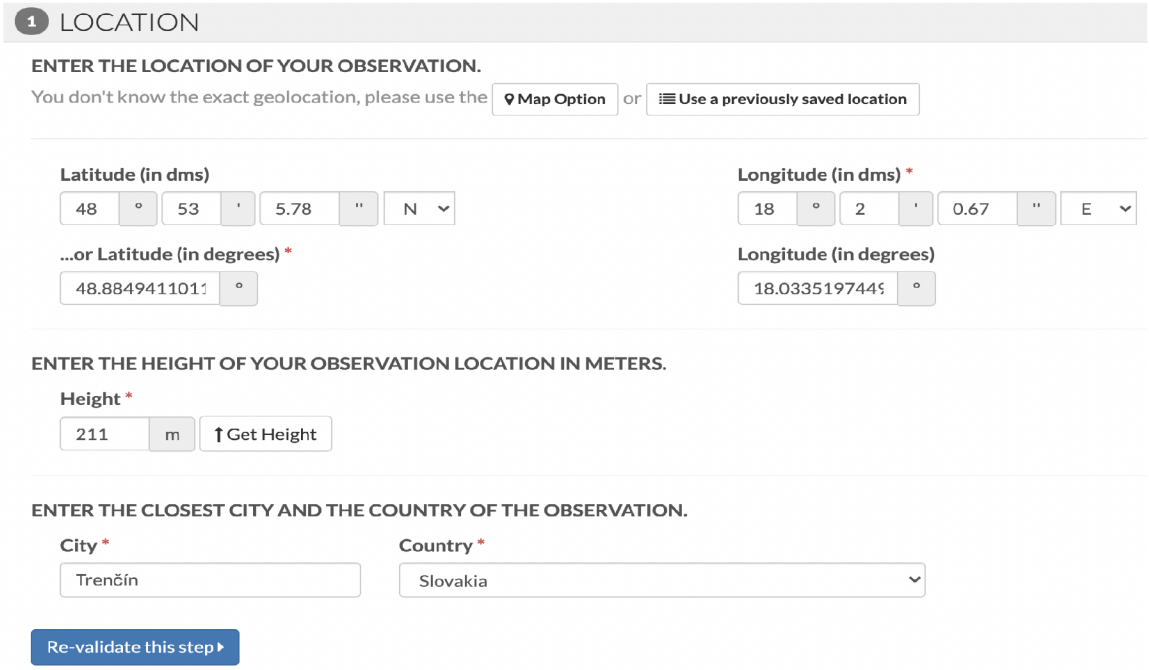
V tejto časti sa podrobne pozrieme na IMO protokol, ktorý je možné vyplniť na adrese `https://www.imo.net/members/imo_observation/add_observation` alebo pomocou csv (comma-separated values) súborov na adrese `https://www.imo.net/members/imo_observation/upload_observation`. Stránka `https://www.imo.net/` delí protokoly na šesť časti pokiaľ sa jedná o prvú možnosť posielania alebo na štyri časti z toho sú dve časti spojené v prípade keď to posielame pomocou možnosti csv súborov.

Lokalizácia

Lokalizácia je miesto, kde sa dané pozorovanie uskutočnilo, čiže je to miesto vytvorenia protokolu. IMO ponúka možnosť vyhľadať danú lokalitu pomocou adresy alebo môže používateľ použiť nejakú z predošlých lokalít, kde už uskutočnil pozorovanie predtým. Formulár je zobrazený na obrázku 5.1.
Polia, ktoré treba vyplniť v lokalizácii:

- **mesto**
- **štát**
- **nadmorskú výšku** zadanú v metroch

- **zemepisnú dĺžku** zadanú buď v stupňoch, minútach a sekundách alebo v desatinných stupňoch
- **zemepisnú šírku** tiež buď zadanú v stupňoch, minútach a sekundách alebo v desatinných stupňoch



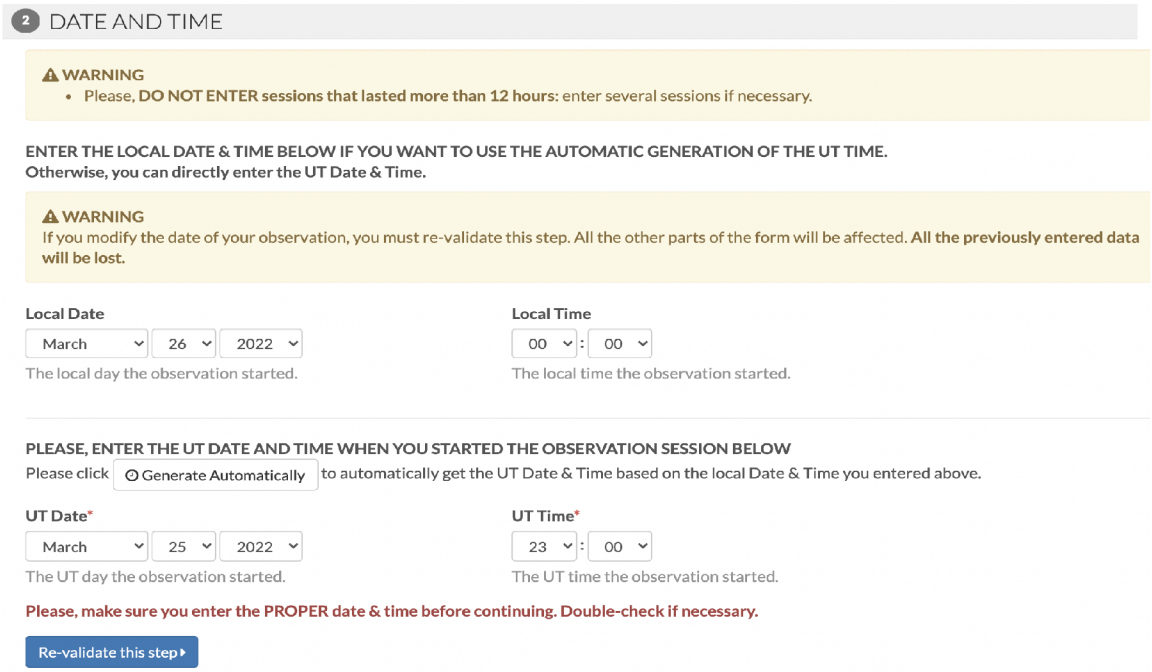
Obrázek 5.1: IMO formulár pre lokalizáciu prevzatý z [10]

Čas a dátum

Tu musí používateľ vyplniť čas a dátum začatia pozorovania vo formáte UT (Universal Time). Celý protokol stránka vyžaduje, aby sa časy vypĺňali v UT. Stránka tiež ponúka možnosť vyplniť miestny čas a potom automaticky vygenerovať UT čas pomocou miesta pozorovania, ktoré sa vypĺňalo vyššie. Tiež stránka neakceptuje formát 24:00, ale musíme použiť formát 00:00. Stránka tiež vyžaduje, aby sa nezadávali protokoly, ktoré trvali viac, ako dvanásť hodín a namiesto toho tento protokol rozdelili na kratšie protokoly. Tento formulár môžete vidieť na obrázku 5.2.

Meteorické roje

V tejto časti musí používateľ zadať, ktoré meteorické roje počas protokolu sledoval, a to aj v prípade pokiaľ nevidel žiadny meteor, ktorý by patril do daného roja. Roje sú vytriedené na základe dátumu začatia, počas ktorého je roj aktívny, dátumu konca jeho aktivity a vyplnením času a dátumu začatia pozorovania vyššie. Používateľ má síce možnosť pridať do protokolu aj roje, ktoré neboli v danú dobu aktívne. Používateľ má tiež možnosť vytvoriť svoj vlastný roj, kde musí ale zadať rektascenziu, deklináciu, rýchlosť a komentár. Automaticky sa pridá aj možnosť pozorovať sporadické meteory, čiže túto možnosť nie je možné vylúčiť. Tento formulár je zobrazený na obrázku 5.3.

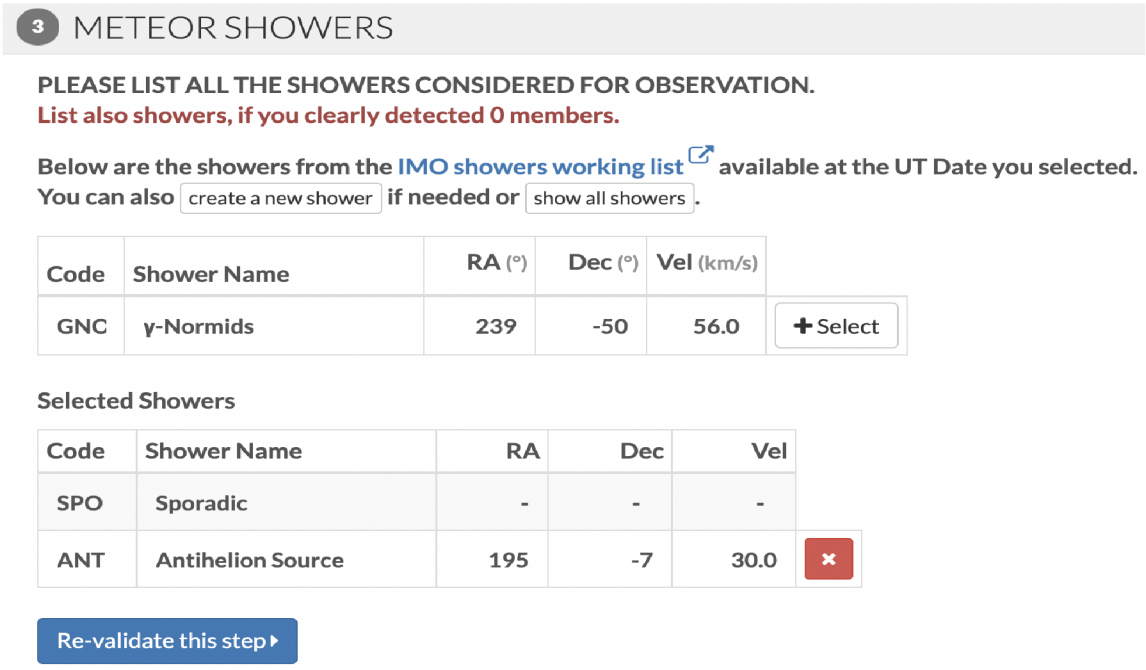


Obrázek 5.2: IMO formulár pre čas a dátum prevzatý z [10]

Distribúcia počtu

V distribúcii počtu sa zadávajú jednotlivé intervaly, v ktorých prebiehalo pozorovanie. Tiež by interval nemal obsahovať viac ako 20 meteorov. Tento formulár je možné vidieť na obrázku 5.4.
Polia, ktoré je treba vyplniť pri každom intervale:

- **dátum začatia intervalu** v UT
- **čas od kedy začal interval a kedy skončil** v UT
- **rektascenzia** zadaná v stupňoch
- **deklinácia** zadaná v stupňoch
- **čas trvania intervalu** buď zadaní v hodinách alebo v minútach tento formát si môže zvoliť užívateľ
- **korekcia prekážky poľa**, ktorá sa vypočíta pomocou 2.2
- **medznú veľkosť**
- u každého roja je potrebné zadať
  - **metódu pozorovania**, kde budeme používať metódu C (for counting), ktorá je predvolená pre pozorovanie
  - **počet videných meteorov**



Obrázek 5.3: IMO formulár pre meteorické roje prevzatý z [10]

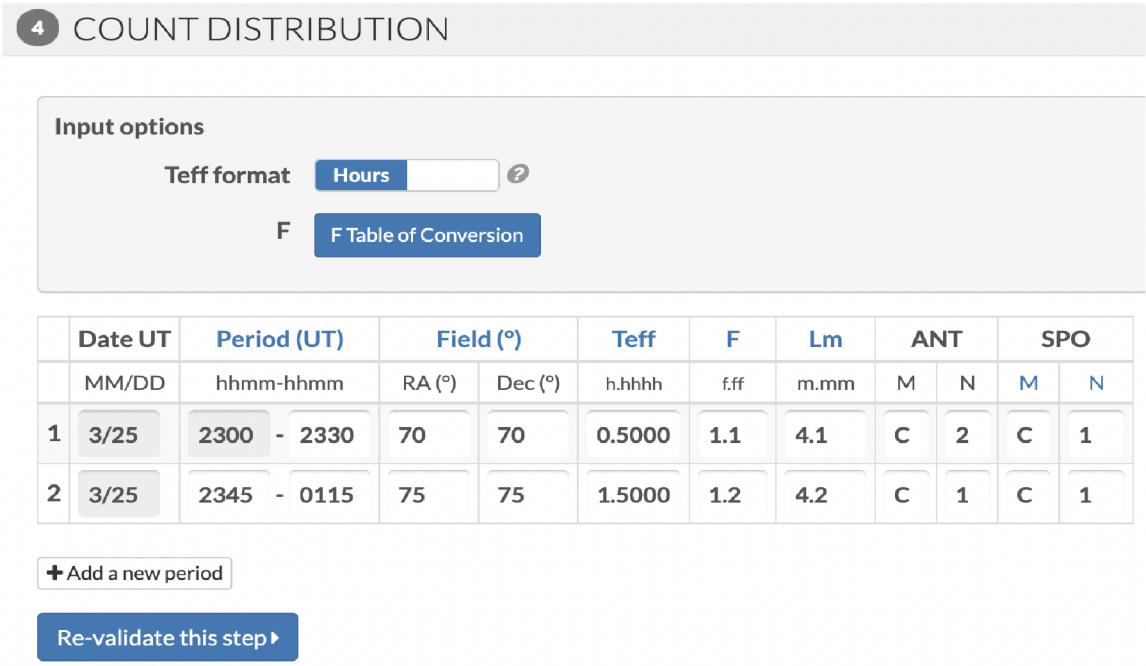
Distribúcia magnitúdy

V distribúcii magnitúdy sa zadáva počet meteorov v danom intervale s tou istou magnitúdou. Pokiaľ nejaký meteor má magnitúdu končiacu na 0.5 tento meteor sa rozdelí na dve polky meteoru pričom jedna polka bude zapísaná v dolnej hranici a tá druhá v hornej hranici takže dohromady budú dávať tú istú magnitúdu napr. ak je meteor s magnitúdou 4.5 tak sa zapíše, ako pol meteoru s magnitúdou 4 a pol meteoru s magnitúdou 5. Používateľ má tiež možnosť spojiť intervaly podľa rojov čím z jednoduchší tabuľku, ale stále platí obmedzenie maximálne dvadsať meteorov na interval. Tento formulár je zobrazený na obrázku 5.5.
Polia vo formulári:

- **meteorický roj**
- **čas od kedy začal interval a kedy skončil** v UT
- **13 stĺpcov pre magnitúdu** od -6 do 7
- **počet zadaných meteorov**
- **maximálny možný počet meteorov**

Komentár

V poslednom formulári sa zapisujú komentáre, ktoré pozorovateľ chce zanechať ohľadom svojho protokolu. Tento komentár by mal spresniť pozorovanie alebo zapísať isté komplikácie počas pozorovania. Používateľ tu má tiež možnosť tento protokol zverejniť pod iného



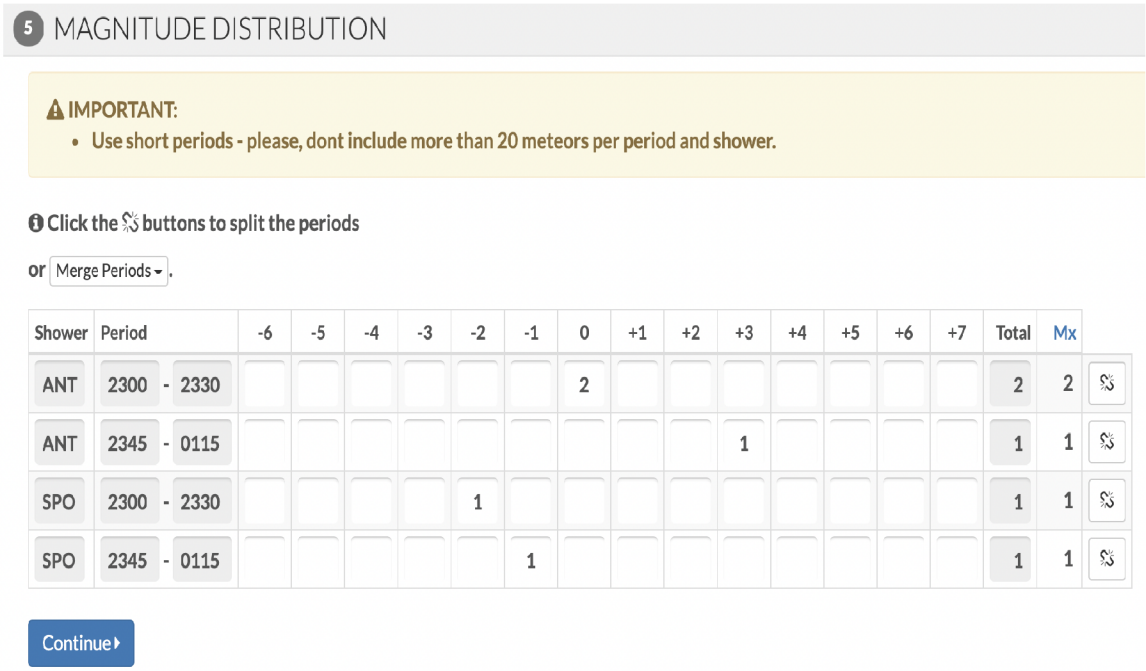
Obrázek 5.4: IMO formulár pre distribúciu počtu prevzatý z [10]

používateľa, ktorý je registrovaný na stránke IMO alebo používateľovi, ktorý nie je zaregistrovaný v takom prípade bude stále pozorovanie zverejnené pod aktuálnym účtom. Tento užívateľ sa dá vyhľadať pomocou jeho mena, ktoré má na svojom IMO účte. Tento formulár je zobrazený na obrázku 5.6.

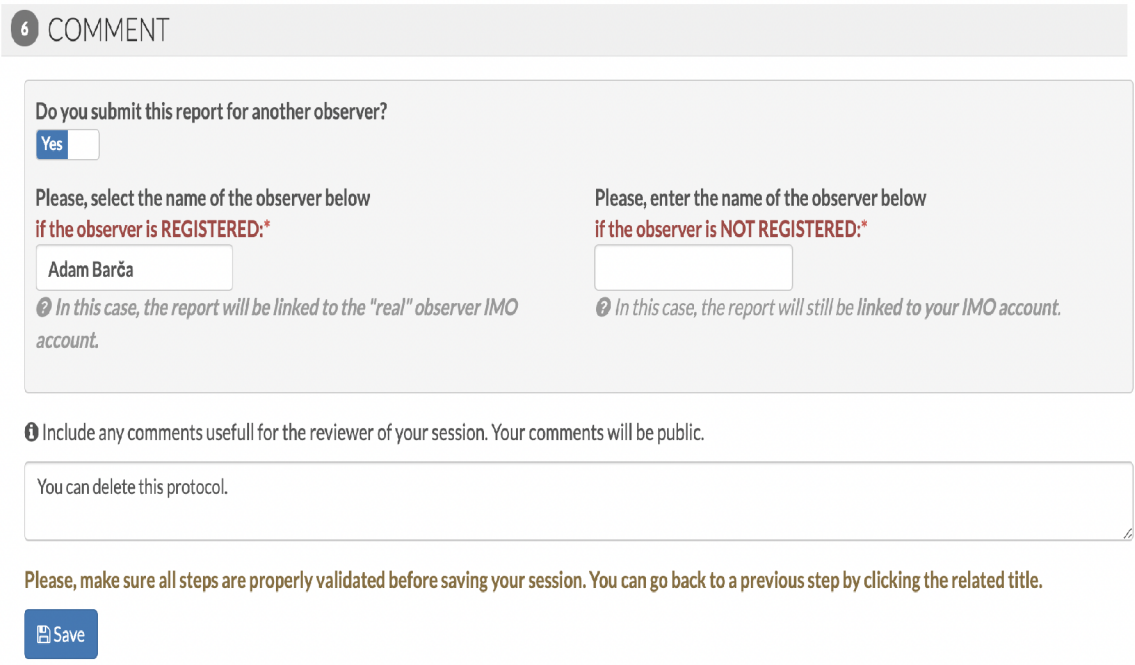
5.2 Diagram prípadov použitia

Diagram prípadov použitia (Use Case Diagram) opisuje, ako aktéri používajú systém a tiež identifikuje interakcie medzi systémom a jeho aktérmi. **Aktér** je roľa používateľa, ktorý interaguje so systémom. Tento aktér môže byť ľudský používateľ, čas alebo iný externý systém. Diagram prípadov použitia patrí medzi jazyky UML (Unified Modeling Language) [7]. Diagram prípadov použitia na túto prácu[1] je na obrázku 5.7. Diagram zachytáva tie najkľúčovejšie prípady užitia so systémom a koľko aktérov máme v systéme. Aktér administrátor má rovnaké prípady užitia ako aktér používateľ, ale naviac môže vytvárať týchto používateľov.

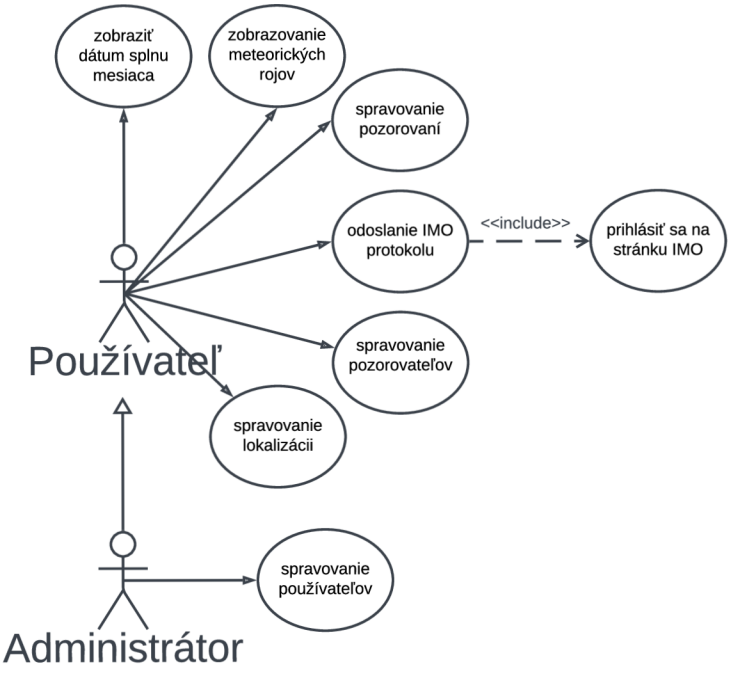
[1] diagram prípadov použitia pre túto prácu: https://lucid.app/documents/view/e7428ca2-1272-4418-90eb-dafee2c7b61e



Obrázek 5.5: IMO formulár pre distribúciu magnitúdy prevzatý z [10]



Obrázek 5.6: IMO formulár pre komentár prevzatý z [10]



Obrázek 5.7: Diagram prípadov použitia

Kapitola 6

Návrh aplikácie

V tejto kapitole sú vytvorené návrhy aplikácie, ktoré vychádzajú z analyzovaných požiadavkách z kapitoly 5. Pri navrhovaní projektu bol použitý ER diagram (Entity Relation Diagram), ktorý slúži na reprezentáciu databázy a návrhy užívateľského rozhrania pomocou mockup modelov. Tiež tu sú popísané štandardy, ktoré sa používajú na dizajn mobilných aplikácii na platforme Android ako aj jednotlivý postup vytvárania užívateľského rozhrania.

6.1 ER diagram

Pri návrhu databázy bol použitý ER diagram (Entity-relation diagram). ER diagram je konceptuálny model, ktorý sa využíva na znázornenie jednotlivých objektoch (entít) a vzťahy medzi týmito objektami. **Entita** je identifikovateľná svojim názvom a obsahuje ľubovoľný počet atribútov. **Atribúty** sú vlastnosti danej entity. Môžu byť jednoduché alebo zložené napr. adresa, kde adresa sa skladá z viacej jednoduchých atribútov, ako je číslo domu a ulica.
Medzi entitami môžu byť **vzťahy**, ktoré sú typu:

- **1 .. 1** vzťah vyjadruje vzťah jednej entity na ďalšiu entitu, ktorá môže byť tiež iba jedna. Napr. osoba a účet pričom predpokladáme, že účet môže vlastniť len jedna osoba
- **1 .. N, N .. 1** sú vzťahy, ktoré vyjadrujú vzťah jednej entity tiež nazývanú ako master k druhej entite, ktorá môže byť lubového počtu táto entita sa tiež nazýva ako detail. Detail entita môže však mať iba jednu master entitu. Napr. osoba a adresa, kde každá osoba má jednu trvalú adresu bydliska, ale na tejto adrese môže bývať viac osôb
- **M .. N** vzťah vyjadruje, že k jednej entite môže byť ľubovoľný počet entít druhého typu a tie zas môžu mať ľubovoľný počet prvého typu. Napr. film a herci, kde každý herec môže hrať vo viacerých filmoch a tiež v každom filme môže hrať viacej hercov.

ER diagram[1], ktorý bol vytvorený a použitý na túto prácu je možné vidieť na obrázku 6.1. V ďalších sekciách budú vysvetlené entity a ich vzťahy medzi nimi, ktoré boli použité na túto prácu.

[1]ER diagram pre túto prácu: https://lucid.app/documents/view/4faf0967-c282-4756-9035-d775c25e7559

Pozorovanie

Pozorovanie je hlavnou entitou v aplikácii. Slúži na to, aby zhromaždila všetky dôležité údaje do jedného celku.
Jej atribúty sú:

- **ID** je primárny kľúč
- **čas a dátum začatia** je čas a dátum kedy začalo pozorovanie tento údaj nie je v UT (Univerzal Time)
- **čas a dátum ukončenia** je čas a dátum kedy bolo ukončené pozorovanie ani tento údaj sa neukladá v UT. Tento atribút sa neposiela do IMO protokolu, ale slúži nám na to, či už bolo pozorovanie ukončené alebo či sa ešte v ňom pokračuje.

Lokalizácia

Lokalizácia je entita, ktorá vyjadruje reálne miesta na mape, kde sa môže uskutočniť pozorovanie. Táto entita má vzťah 1 .. N ku pozorovaniu pričom môže, ale byť vytvorená lokalizácia, v ktorej sa neuskutočnilo žiadne pozorovanie.
Táto entita má atribúty:

- **ID** je primárny kľúč
- **mesto** lokalizácie
- **štát** lokalizácie. Tento údaj sa odporúča vypĺňať anglicky, pretože IMO na svojich stránka používa preddefinované názvy štátov, z ktorých si musí užívateľ vybrať. Pokiaľ tomuto údaju nebude rozumieť IMO tak namiesto štátu tam dá „unknown“.
- **miesto** je nepovinný údaj, ktorý má informačnú hodnotu iba pre používateľa. Slúži na to, aby si svoju lokalizáciu lepšie spresnil, pretože môže mať aj viac lokalizácii v tom istom meste, čiže aj v tom istom štáte zároveň.
- **nadmorská výška** lokalizácie
- **zemepisná šírka** lokalizácie
- **zemepisná dĺžka** lokalizácie

Meteorický roj

Meteorický roj je entita, ktorá vyjadruje všetky meteorické roje, ktoré používa IMO. Táto entita má pevne preddefinované dáta meteorických rojoch, ktoré sú prevzaté zo stránky IMO[2]. Táto entita sa používa na vyjadrenie, ktoré meteorické roje sa pozorovali počas pozorovania a taktiež na to, aby vyjadrila do akého roja patrí daný meteor.
Jej atribúty sú:

- **ID** je primárny kľúč
- **kód** je unikátny troj znakový kód, ktorý vyjadruje daný roj.
- **meno** je názov meteorického roja

[2]IMO zoznam meteorických rojoch: https://www.imo.net/members/imo_showers/working_shower_list

- **dátum začatia aktivity**, ako názov atribútu napovedá je to dátum kedy začína byť roj aktívny. Tento údaj sa ukladá ako mesiac a deň.
- **dátum ukončenia aktivity** presne, ako u začatia aktivity sa aj tento údaj ukladá ako mesiac a deň.
- **rýchlosť** roja
- **rektascenzia** roja
- **deklinácia** roja

Protokol

Protokol je výsledný produkt ktorý, sa bude posielať na stránku IMO. Vďaka vzťahu k pozorovaniu má tento protokol prístup k lokalizácii a k meteorickým rojom, ktoré sa pozorovali. Tieto údaje by sa dali povedať, že slúžia ako hlavička pre každý protokol, ktorý sa uskutočnil v tom danom pozorovaní. Protokol môže vlastniť iba jeden pozorovateľ.
Atribúty tejto entity sú:

- **ID** je primárny kľúč
- **komentár** slúži na zaznamenávanie neočakávaných udalosti alebo k spresneniu pozorovania. Tento údaj je nepovinný.

Pozorovateľ

Pozorovateľ je osoba, ktorá môže pozorovať meteory. Aj tu platí rovnako, ako pri lokalizácii, že pozorovateľ nemusí vlastniť žiadne protokoly na to, aby existoval.
Atribúty tejto entity sú:

- **ID** je primárny kľúč
- **meno** pozorovateľa
- **priezvisko** pozorovateľa

Interval

Interval predstavuje časový úsek, počas ktorého pozorovateľ aktívne pozoroval meteory. Pretože prestávky sa do databázy neukladajú táto entita by mala reprezentovať nielen interval pozorovania, ale tiež aj efektívny čas pozorovania. Platí, že protokol sa musí skladať aspoň z jedného intervalu alebo viacerých.
Atribúty pre túto entitu sú:

- **ID** je primárny kľúč
- **čas začatia** intervalu
- **čas ukončenia** intervalu. Ako už bolo spomenutý rozdiel medzi časom začatia a časom ukončenia je efektívny čas pozorovania pre daný interval.
- **rektascenzia** intervalu

- **deklinácia** intervalu
- **oblačnosť** tento údaj je voliteľný pričom sa v takom prípade počíta, že nebola žiadna oblačnosť, čiže 0%.

IMO oblasť

Ako už bolo spomenuté v kapitole 2.2 tak IMO doporučuje na vyjadrenie medznej veľkosti trojuholníkovú metódu. Tieto dáta sú preddefinované a pochádzajú z tabuliek, ktoré uvádza IMO.
Atribúty tejto entity sú:

- **ID** je primárny kľúč
- **meno** je názov danej oblasti. Tento názov sa môže skladať buď z jednej trojice znakov alebo z dvoch trojíc znakov oddelenie pomlčkou.
- **počet hviezd** vyjadruje koľko hviezd vidí pozorovateľ v danej oblasti. Toto číslo môže byť od 0 do 110 pričom pokiaľ videl viac hviezd než oblasť špecifikuje berie sa to ako keby videl maximálny počet hviezd v danej oblasti.
- **magnitúda** vyjadruje citlivosť oka pozorovateľa na oblohu. Tento údaj nás najviac zaujíma na tejto entite a tiež sa. ako jediný zadáva do protokolu.

Meteor

Meteor je entita, ktorú daný pozorovateľ hlási zapisovateľovi. Platí, že každý interval môže mať maximálne 20 meteorov. Ako je vidieť pri každom meteore sa zaznamenáva hlavne jeho magnitúda a rojová príslušnosť.
Atribúty tejto entity sú:

- **ID** je primárny kľúč
- **čas vytvorenia** tento údaj je informatívny pre používateľa a nezadáva sa do IMO protokolu. Slúži na to, aby používateľ zreteľne videl, ktorý meteor je najnovší a ktorý zase nie.
- **magnitúda** je jasnosť meteoru táto hodnota môže byť od -6 do 7 pričom by sa mala udávať na presnosť 0.5 magnitúdy.

Túto entitu je možné rozšíriť aj o ďalšie atribúty ktoré, nás môžu pri pozorovaní zaujímať, ako je napríklad stopa alebo kvalita. Pretože ale IMO protokol tieto údaje nevyžaduje tak ich ani nezaznamenávame.

Používateľ a oprávnenie

Tieto entity slúžia na autentifikáciu a autorizáciu používateľa. Každý používateľ môže vlastniť viacej oprávnení. Oprávnenia, ktoré boli vytvorené pre danú aplikáciu sú používateľ a admin. Admin by sa mal nachádzať iba jeden v celej aplikácii.

6.2 Návrh užívateľského rozhrania

Užívateľské rozhranie (user interface) je veľmi dôležitou súčasťou aplikácii. Pomáha optimalizovať užívateľskú skúsenosť (user experience) a tiež reprezentuje našu aplikáciu z hľadiska výzoru. Cieľom dobrého návrhu je uľahčiť prácu používateľa na toľko, aby za čo najmenší počet krokov dosiahol maximálny požadovaný výsledok. Hlavným rozdielom medzi užívateľským rozhraním a užívateľskou skúsenosťou je, že užívateľské rozhranie je viac zamerané na dizajne stránke, ako užívateľská skúsenosť, ktorá je zameraná na potreby používateľa.

Material Design

Material Design je stránka vyvinutá spoločnosťou Googlu v roku 2014[3]. Stránka dobre popisuje návrh dobrého dizajnu pre platformu Android ako aj rôzne inšpirácie pre dizajn aplikácie. Stránka sa zameriava na kompletný dizajn ako z estetického hľadiska tak aj logického. Vysvetľuje a ukazuje, že návrh dizajnu nie je len na papieri, ale že máme komponenty, ktoré medzi sebou komunikujú, čo môžeme implementovať pomocou animácii. Tiež popisuje výber správnej typografie a farebnej palety, ktorí majú obrovský vplyv na vzhľad a použiteľnosť aplikácie. Z tejto stránky som sa inšpiroval na návrhu dizajnu práce, ktorá bola navrhnutá v tmavom móde (dark mode) ako aj na výberu farebnej palety.

Fázy návrhu užívateľského rozhrania

Návrh užívateľského rozhrania aplikácie prebiehalo v niekoľkých fázach, počas ktorého sa vytváral celkový vzhľad aplikácie. Každá fáza definuje nové požiadavky na aplikáciu, ako sú jednotlivé prechody medzi obrazovkami, dáta, ktoré sa majú zobraziť alebo farba pozadia. Tento projekt bol najprv vytvorený pomocou skice, z ktorej potom boli navrhnuté drôtené rámy s vysokou presnosťou (high fidelity wireframes). Robievajú sa ešte aj drôtené rámy s nízkou presnosťou, ale pretože by bol rozdiel medzi rámy malý tak som tento postup vynechal.

Skica

Prvé návrhy boli nakreslené pomocou voľnej ruky, ceruzky a papiera. Táto metóda je dobrá v počiatočnej fáze návrhu, kde ešte len rozmýšľame aké dáta budeme zobrazovať a akou formou. Návrh tiež zobrazuje jednotlivé funkcionality aplikácie tak isto ako aj vzťahy medzi obrazovkami. Pri návrhu je dobrá spätná väzba zákazníka alebo používateľa, ktorý vidí veľmi jednoduchou formou budúci výzor aplikácie, a to mu pomáha rozmyslieť si údaje a funkcionality, ktoré má aplikácia mať. Vďaka tomu, že sa výzor a požiadavky na aplikáciu menia najviac počiatočnej fáze je veľmi rýchle a jednoduché toto zmeniť v skici, ako by sme to už museli meniť vo finálnom dizajne aplikácie. Skicu k aplikácii je možno vidieť na obrázku 6.2.

Drôtené rámy s vysokou presnosťou

Drôtené rámy s vysokou presnosťou (high fidelity wireframes) je dizajnový návrh, ktorý sa používa na konci fázy návrhu užívateľského rozhrania. Správny rám s vysokou presnosťou je zameraný hlavne na dizajn, typografiu a farby pričom by mal presne zobrazovať jednotlivé fonty ako aj medzery medzi prvkami. Oproti tomu rám s nízkou presnosťou je zameraný

[3]Material Design: https://material.io/design

na funkcie aplikácie a na rozmiestnenie komponentov. Rám by mal už predstavovať dokončený dizajn aplikácie s všetkými farbami, logami alebo grafikou. Drôtený rám s vysokou presnosťou bez interaktivity sa tiež nazýva ako *maketa*. Pokiaľ sa do makety pridajú ešte aj interakcie prechodov alebo animácii tak sa z nej stane *prototyp s vysokou presnosťou* [19]. Prototyp vďaka svojej interakcii vyzerá, ako už naprogramovaná aplikácia, ktorá dokáže jednoduchú logiku, ako sú animácie alebo reakcia na tlačidlá. Vďaka tomu sa aplikácie lepšie prezentuje zákazníkovi, ktorý má možnosť vidieť finálny dizajn aplikácie a "ohmatať si ju".
Výhody drôteného rámu sú:

- **ľahký prevod na prototyp**, kde len stačí pridať interakcie a z makety sa stane prototyp
- **zobrazuje finálny dizajn**, ktorý slúži na dobrú predstavu, ako bude aplikácia vyzerať
- **uľahčuje prácu vývojárom**, keď vývojári jasne vidia aké fonty sa použili alebo aká je veľká medzera medzi prvkami urýchľuje a uľahčuje to prácu vývojárovi, ktorý sa nemusí spoliehať na vlastnú predstavivosť
- **inšpiruje nadšenie** členov tímu, ktorý vidia finálnu verziu aplikácie ešte pred samým začatím implementácie má to na nich povzbudzujúci účinok

Rámy vysokej presnosti pre túto prácu boli vytvorené pomocou programu Figma[4], ktorá je popísaná v tejto sekcii 4.2, ako aj prototyp s vysokou presnosťou, kde nám Figma ponúka nástroje na ľahké zostrojenie tohto prototypu. Ukážka rámu s vysokou presnosťou je možné vidieť na obrázku 6.3 pričom sú tam zobrazené len tie obrazovky, ktoré sa nachádzajú aj vo skici vyššie 6.2

6.3 Architektúra MVC

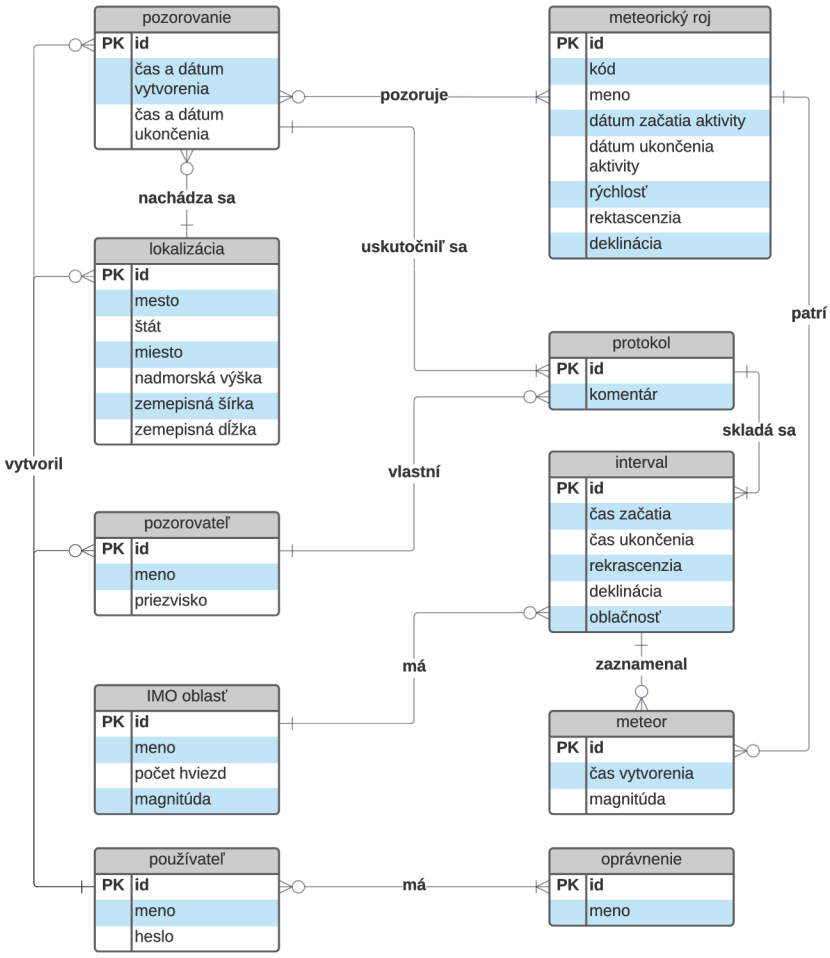
Architektúra MVC (model-view-controller), ktorá je zobrazená na obrázku 6.4 je obľúbená architektúra, ktorá oddeľuje riadiacu logiku aplikácie od dátového modelu a užívateľského rozhrania. Využíva sa hlavne pri návrhu webových aplikácii, aj keď pôvodne bola navrhnutá pre desktopové aplikácie. Architektúra rozdeľuje aplikáciu do troch nezávislých časti.
Tri časti MVC architektúry:

- **model** predstavuje dáta a aplikačnú logiku aplikácie. Uchováva v sebe dáta a má na starosti všetky výpočty. Môže tiež validovať dáta z formuláru napr. či má emailová adresa správny formát. Nevie o existencii ďalších komponent a nemal by ani obsahovať prvky, ktoré sú závislé na ostatných komponentoch.
- **view** (pohľad) zobrazuje dáta spracované modelom a užívateľské rozhranie. Určuje, ako majú byť prezentované dáta a ako má vyzerať výsledný vzhľad aplikácie viac v sekcii 6.2
- **controller** (radič) je riadiacou jednotkou architektúry, ktorý príma požiadavky a dáta z pohľadu a následne ich priradí aplikačnej logike a vracia výsledok z aplikačnej logiky späť do pohľadu.

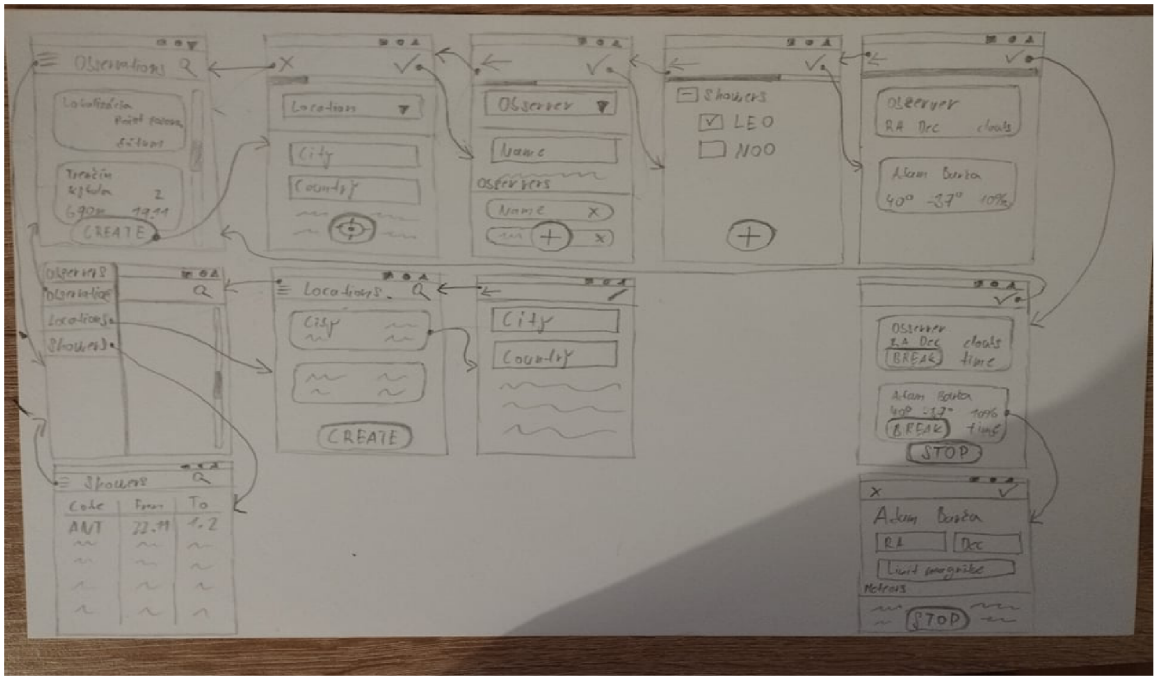
[4] rám vysokej presnosti pre túto prácu: https://www.figma.com/file/YutHIyzK7ZYfEvnenF1mV6/BachelorThesis?node-id=0%3A1

Architektúra MVC je výhodná hlavne pre projekty, v ktorých nie je známa ich kompletná funkcionalita. Vďaka tejto architektúre je náš projekt ľahšie rozšíriteľný a ľahšie udržiavateľný. Architektúra rieši takzvaný "špagetový kód", kde sa mieša dizajn, logika aplikácie alebo operácie s databázou [15].
Výhody architektúry MVC:

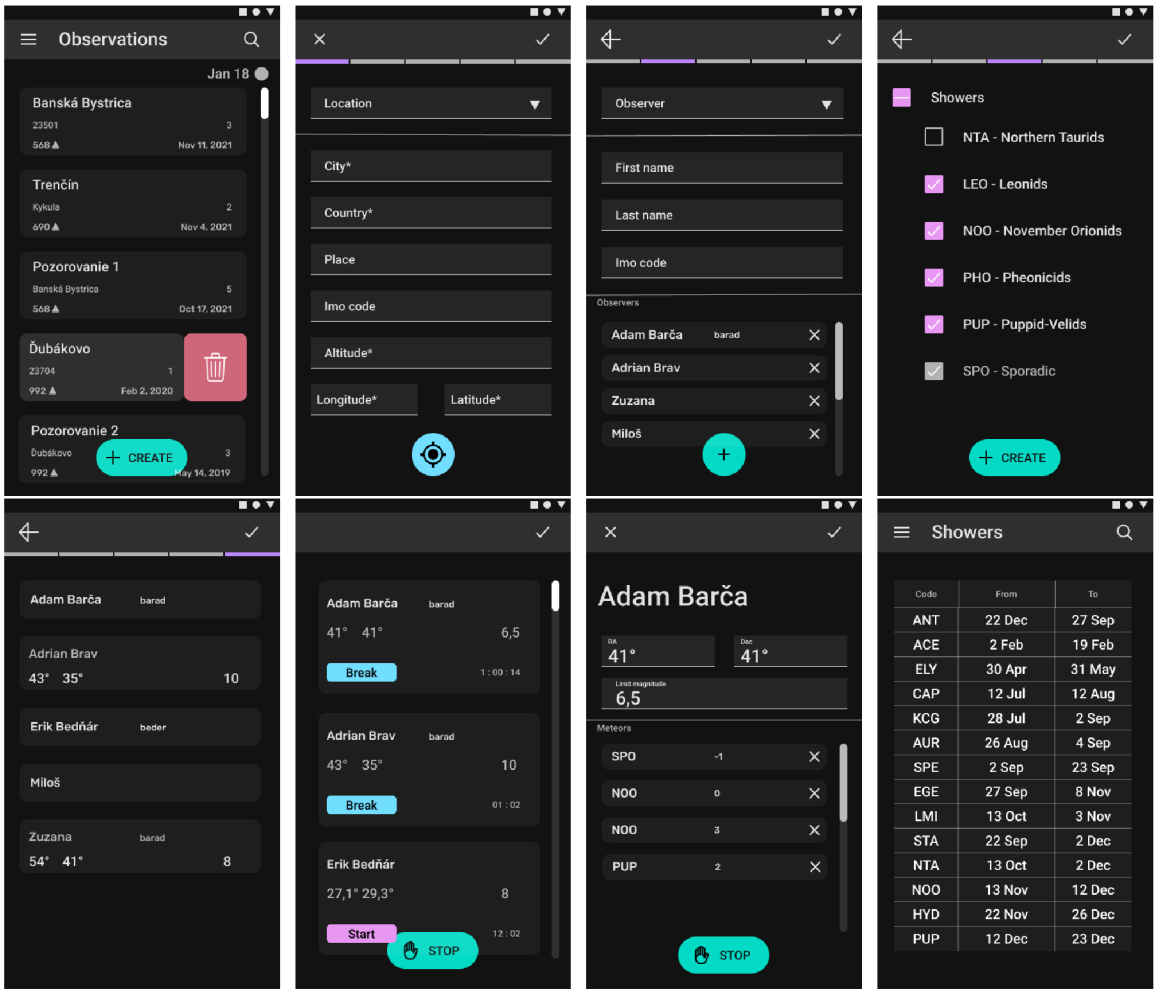
- možnosť paralelného vývoja, pretože každá komponenta je nezávislá na sebe
- ľahšie testovanie jednotlivých časti aplikácie
- udržateľnosť prehľadného kódu toto platí aj u väčších aplikácii
- rýchle úpravy častí aplikácie a doplnenie ich funkcionality



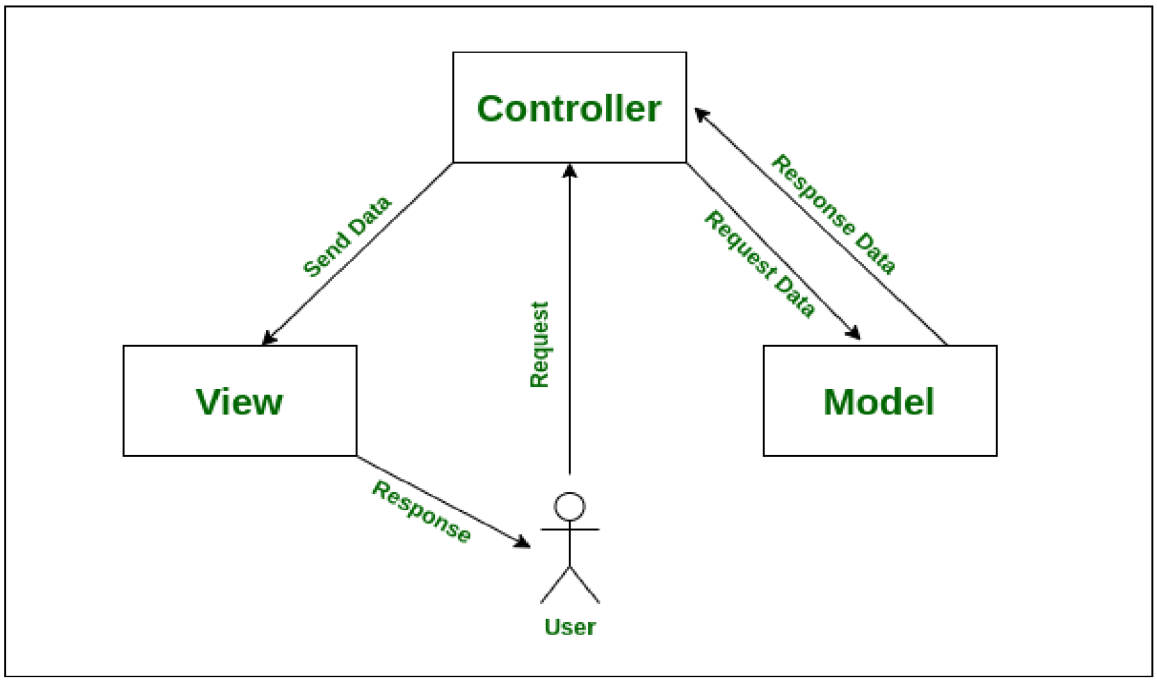
Obrázek 6.1: ER diagram



Obrázek 6.2: Skica aplikácie



Obrázek 6.3: Rámy vysokej presnosti aplikácie



Obrázek 6.4: Architektúra MVC prevzatá z [5]

Kapitola 7

Implementácia aplikácie

V tejto kapitole bude vysvetlená implementácia aplikácie za použitia analýzy z kapitoly 5, návrh z kapitoly 6 a použitých technológii z kapitoly 4. Budú tu vysvetlené jednotlivé fázy implementácie a ich vzájomné prepojenie.

7.1 Back-end

V tejto sekcii bude popísaná serverová časť aplikácie, ktorá bola vytvorená pomocou Spring Boot, Lombok, QueryDsl, Hibernate, MupStruct a JPA. Budú tu vysvetlené pojmy, ktoré sú špecifické pre Spring ako je entita, služba a kontrolér.

Databáza

Databáza je neoddeliteľnou súčasťou aplikácie a k jej implementácii by sme sa mali venovať ako k prvej. Databáza bola implementovaná za použitiu ER diagramu 6.1, ktorá je skutočnou kópiou tohto návrhu. Databáza je implementovaná pomocou objektovo-relačného databázového systému PostgreSQL. K vytvoreniu tabuliek k aplikácii slúži skript **schema.sql**, ktorý má cestu `server-spring-boot/src/resources`. Pomocou tohto skriptu sa najprv vymažú všetky tabuľky a indexy, ktoré budú vytvorené pomocou tohto skriptu. Pred začatím vytvárania tabuliek a pridanie dát do nich sa musíme uistiť, že pred vytvorením tabuliek neexistovala žiadna tabuľka s tým istým menom. Potom sa vytvoria tabuľky a indexy k nim. Kód 7.1 ukazaje ako pomocou PostgreSQL sa vytvorí tabuľka lokalizácie. K naplneniu tabuliek dátami slúži skript s názvom **data.sql**, ktorý má cestu `server-spring-boot/src/resources`. Tento skript vytvorí admina a jedného používatela. Ďalej vytvorí všetky meteorické roje definované IMO a tiež aj oblasti, ktoré slúžia na určenie medznej hviezdnej veľkosti viac v sekcii 2.2. Pretože chceme, aby sa schéma databázy implementovala za pomocí skriptu a nie, aby ju implementoval Hibernate tak musíme do súboru **application.properties**, ktorý ma cestu `server-spring-boot/src/resources` vložiť vlastnosť:

```
spring.jpa.hibernate.ddl-auto = none
```

K tomu, aby sa databáza vždy pri štarte aplikácie naplnila dátami zo skriptu data.sql musíme do súboru **application.properties** pridať vlastnosť:

```
spring.sql.init.mode = always
```

Súbor application.properties je špecificky súbor, ktorý využíva Spring Boot na definovanie vlastnosti potrebných k behu aplikácie. V tomto súbore sú aj jednotlivé vlastnosti k pripojeniu aplikácie k databázy, ako je url databázy a prístupové údaje.

```
CREATE TABLE location (
    id              SERIAL          PRIMARY KEY,
    city            VARCHAR(50)     NOT NULL,
    country_name    VARCHAR(50)     NOT NULL,
    place           VARCHAR(50),
    altitude        SMALLINT        NOT NULL,
    longitude       NUMERIC(7, 4)   NOT NULL,
    latitude        NUMERIC(7, 4)   NOT NULL,
    created_by      INT             NOT NULL
        REFERENCES app_user(id) ON DELETE CASCADE
);
```

Výpis 7.1: Príklad vytvorenia tabuľky lokalizácia

Entita

Entita predstavuje tabuľku uloženú v databáze a každá inštancia tejto entity je jeden riadok v tabuľke. Databáza je napojená pomocou tried entít, ktoré sú vytvorené pomocou anotácie **@Entity**, ktoré sú namapované na jednotlivé tabuľky databázy tiež nazývané ako ORM (Object-Relation Mapping). Mapovanie prebieha automaticky a je implementované pomocou JPA (Java Persistence Api) pomocou anotácii **@Column**, **@Id** atď. Všetky implementované entity majú cestu `server-spring-boot/src/main/java/sk/adambarca/serverspringboot/model/entity`. Tieto entity sú implementované ako JavaBeans, čo nám vytvára príliš veľa metód, ktoré majú za úlohu len pridávanie a odoberanie hodnoty z atribútov. Kvôli tomu je kód zbytočne veľký a tiež menej priehľadný. Toto rieši knižnica Lombok, ktorá nám dáva možnosť pridanie anotácie **@Getter** a **@Setter**, ktoré automaticky vygeneruje metódy pre všetky atribúty danej triedy. Vďaka tomu je naša entita omnoho priehľadnejšia. Tiež u niektorých entít sú použité validačné podmienky pomocou anotácii **@Size**, ktoré nám obmedzujú a spresňujú veľkosť alebo hodnotu atribútu. Tieto anotácie je možné vidieť v kóde 7.2.

```
@Entity
@Getter
@Setter
@NoArgsConstructor
public class Location {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Max(50)
    @Column(nullable = false)
    private String city;

    @Max(50)
    @Column(nullable = false)
    private String countryName;
```

```
    @Max(50)
    private String place;

    // max altitude on the Earth (Mount Everest)
    @Size(max = 8849)
    @Column(nullable = false)
    private short altitude;

    @Size(min = -180, max = 180)
    @Column(nullable = false)
    private float longitude;

    @Size(min = -180, max = 180)
    @Column(nullable = false)
    private float latitude;

    @ManyToOne(fetch = FetchType.EAGER, cascade = CascadeType.MERGE)
    @JoinColumn(name = "created_by", nullable = false)
    private User createdBy;
}
```

Výpis 7.2: Príklad implementácie entity lokalizácia

Repozitár

Pokiaľ je databáza úspešne pripojená k serverovej časti a jej tabuľky sú namapované na jednotlivé entity môžeme vytvoriť repozitáre nad danými entitami. Repozitár slúži na dopytovanie, úpravu, pridávanie a mazanie dát z tabuliek. Repozitáre boli vytvorené buď pomocou rozhrania **JpaRepository**, ktoré nám poskytuje jednoduché CRUD (Create, Read, Update, Delete) metódy, ako sú **findAll()**, **getById(Id id)** atď. Všetky repozitáre vytvorenie pomocou rozhrania JpaReposity majú na konci svojho mena názov repository a sú uložené v `server-spring-boot/src/main/java/sk/adambarca/serverspringboot/model/repository`. Druhá možnosť, ako je vytvoriť repozitár je pomocou anotácie **@Repository**, ktorá hovorí Springu, že daná trieda pracuje s databázou. Tieto triedy sa nazývajú ako **DAO** (Data Access Object) a obsahujú zložitejšie dotazy. Každá táto trieda musí obsahovať atribút EntityManager, ktorá sa stará o životný cyklus všetkých entít. Všetky DAO vytvorené v rámci projektu majú cestu `server-spring-boot/src/main/java/sk/adambarca/serverspringboot/model/dao`. V DAO triedach sa dopytujem pomocou knižnice **QueryDsl**, ktorá generuje metadata entít a tým nám umožňuje vytvárať prehľadnejší a hlavne typovo bezpečnejší dotaz. Príklad DAO entity je možné vidieť v kóde 7.3. Každá trieda, ktorá má anotáciu @Entity má pri vytváraní aplikácie vytvorenú takzvanú **Q-types triedu**. Tieto triedy priamo súvisia so svojimi entitami a umožňujú nám prístup k metadátam entity. Všetky vygenerované triedy pomocou QueryDsl majú cestu `server-spring-boot/target/generated-sources/java/sk/adambarca/serverspringboot/model/entity`.

```
@Repository
public class LocationDAO extends AbstractDAO {

    public List<Tuple> getAllWithObservationsCount() {
        JPAQueryFactory queryFactory = new JPAQueryFactory(em);
```

```
        QLocation qLocation = QLocation.location;
        QObservation qObservation = QObservation.observation;

        JPAQuery<Tuple> query = queryFactory
                .select(qLocation, JPAExpressions
                                    .select(qObservation.count())
                                    .from(qObservation)
                                    .where(qObservation.location.eq(qLocation)))
                .from(qLocation)
                .leftJoin(qObservation).on(qObservation.location.eq(qLocation))
                .groupBy(qLocation.id);

        return query.fetch();
    }
}
```

Výpis 7.3: DAO trieda vytvorená pomocou anotácie @Reposity za použitiu QueryDsl na dopytovanie dát z databázy

Služba

Pokiaľ sú vytvorené všetky repozitáre s ich metódami, ktoré budeme využívať v našej aplikácii tak je dobrou praktikou vytvoriť vrstvu služieb, ktorá implementuje všetku business logiku. Každá takáto služba musí mať anotáciu **@Service**, ktorou oznamujeme Springu, že táto trieda implementuje business logiku. Všetky služby implementované v aplikácii sú na adrese `server-spring-boot/src/main/java/sk/adambarca/serverspringboot/service`. Služba môže pristupovať k rôznym tabuľkám za použitia viacerých repozitárov alebo služieb. Tieto repozitáre sú namapované pomocou anotácie **@Autowired** a v tom prípade nie je nutné tieto objekty inicializovať či už v konštruktore alebo v nejakej inej metóde. Príklad služby, ktorá sa stará o business logiku lokalizácie je možné vidieť v kóde 7.4. Tieto metódy môžu byť jednoduché alebo zložité. Pri jednoduchých metódach zavoláme príslušnú operáciu repozitára a pre mapujeme objekt z databázy na požadovaný výstup, ktorý bude späť poslaní klientovi tieto objekty sa volajú **DTO** (Data Transfer Objects). Všetky DTO objekty, ktoré sa v aplikácii používajú majú adresu `server-spring-boot/src/main/java/sk/adambarca/serverspringboot/service/dto`. K premapovaniu týchto objektov slúži knižnica **MapStruct**, v ktorej definujeme rozhranie a knižnica sama vytvorí konkrétnu implementáciu mapovača. Vďaka tomu je kód priehľadnejší a aj ľahšie udržiavateľný, pretože keď sa vymaže nejaký atribút na entite nemusíme ho tiež vymazávať v mapovači stačí, keď triedy znovu vygenerujeme počas kompilácie. Kód 7.5 ukazuje jednoduchosť mapovača vytvoreného pomocou MapStruct. Všetky rozhrania mapovača vytvorené pre túto aplikáciu majú cestu `server-spring-boot/src/main/java/sk/adambarca/serverspringboot/mapper` a ich vygenerované triedy majú cestu `server-spring-boot/target/generated-sources/annotations/sk/adambarca/serverspringboot/mapper`.

```
@Service
public class LocationService {

    @Autowired private LocationRepository locationRepository;
    @Autowired private LocationMapper locationMapper;
    @Autowired private LocationDAO locationDAO;
```

```
    public List<LocationDTO> getAll() {...}

    public List<LocationWithObservationsCountDTO> getAllWithObservationsCount()
        {...}

    public LocationDTO getById(Long id) {
        Location location = locationRepository.getById(id);

        return locationMapper.location2LocationDTO(location);
    }

    public LocationDTO create(LocationDTO locationDTO) {...}

    public LocationDTO update(LocationDTO locationDTO) {...}

    public boolean deleteById(Long id) {...}

    private LocationWithObservationsCountDTO
        mapToLocationWithObservationsCountDTO(Tuple unsolvedTuple) {...}
}
```

Výpis 7.4: Príklad implementácie služby pre entitu lokalizácia

```
@Mapper(componentModel = "spring")
public interface UserMapper {

    @Mapping(target = "password", ignore = true)
    UserDTO master(User user);

    User userCreatedDTO2User(UserCreateDTO userCreateDTO);
}
```

Výpis 7.5: Príklad rozhrania pre mapovač za použitia knihovny MapStruct

Kontrolér

Implementácia každého kontroléra musí na začiatku obsahovať Spring anotáciu **@Controller** alebo **@RestController**. Anotácia @RestController oznamuje Springu, že chceme používať rozhranie API (Application programming interface) **Spring RESTful**, ktoré vracia údaje späť klientovi zvyčajne pomocou reprezentácie **XML** (Extensible Markup Language) alebo **JSON** (JavaScript Object Notation). Kontroléry obsahujú tiež anotáciu **@RequestMapping("URI")**, ktorá určuje hlavnú cestu ku kontroléru. Každý kontrolér pracuje s jednou službou, ktorá je automaticky inicializovaná pomocou anotácie **@Autowired**. Metódy kontroléra sú mapované podľa ich URI (Uniform Resource Identifier) a podľa ich HTTP (Hypertext Transfer Protocol) metódy. Na toto slúžia anotácie **@GetMapping**, **@PostMapping**, **@PutMapping** a **@DeleteMapping**. Tieto anotácie majú dobrovoľný parameter s názvom **value**, ktorý nám umožňuje spresniť novú URI k danej metóde kontroléra. Kontrolér, ktorý spracováva požiadavky ohľadom meteorických rojov je možné vidieť v kóde 7.6. Každá metóda zavolá požadovanú metódu zo služby, ktorú implementuje a buď vráti požadovaný objekt DTO alebo vráti chybu s odpovedajúcim stavovým HTTP kódom a chybovou správou. Kontroléry, ktoré boli vytvorené pre túto aplikáciu sa nachá-

dzajú v balíčku ktorý ma adresu `server-spring-boot/src/main/java/sk/adambarca/serverspringboot/controller`.

```
@RestController
@RequestMapping("/showers")
@CrossOrigin
public class ShowerController {

    @Autowired private ShowerService showerService;

    @GetMapping
    public ResponseEntity<List<ShowerDTO>> getAll() {
        return new ResponseEntity<>(showerService.getAll(), HttpStatus.OK);
    }

    @GetMapping("/current")
    public ResponseEntity<List<ShowerDTO>> getCurrentShowers() {
        return new ResponseEntity<>(showerService.getCurrentShowers(),
            HttpStatus.OK);
    }
}
```

Výpis 7.6: Príklad jednoduchého kontroléru pre spracovanie meteorických rojov

Zabezpečenie

Zabezpečenie aplikácie bolo implementované pomocou rámca Spring spolu s jeho nadstavbou **Spring Security**. Spring Security používa rozhranie UserDetailsService, ktoré obsahuje metódu loadUserByUsername pre vyhľadávanie používateľov podľa ich používateľského mena. Toto rozhranie je implementované v službe UserService, kde je prepísaná metóda loadUserByUsername. Táto metóda načítava používateľa z databázy vo forme entity User a tá je následne namapovaná spolu aj s jej rolami na požadovanú triedu UserDetails, ktorá slúži na reprezentáciu autentizovaného používateľa. Pre autorizáciu a autentizáciu je použitý **JWT** (JSON Web Token), ktorý nám umožňuje bezpečný prenos pomocou tokenu, ktorý je vo formáte **JSON** (JavaScript Object Notation). Tento token sa vytvára pomocou knižnice **JJWT** v metóde generateJwtToken(UserDetails userDetails) a je zašifrovaný pomocou algoritmu **HS512**. **Autentifikácia** je proces, pri ktorom overujeme identitu používateľa za účelom či má právo používať našu aplikáciu. Pre autentifikáciu používateľa sa používa jeho *prihlasovacie meno* a *heslo*, ktoré sa šifruje pomocou algoritmu **BCrypt**. Z údajov o používateľovi je vytvorený objekt triedy UsernamePasswordAuthenticationToken, ktorý je použitý ako parameter pre autentifikáciu pomocou triedy AuthenticationManager. Pokiaľ autentifikácia používateľa prebehla v poriadku metóda vráti meno používateľa, jeho role a vygenerovaný JWT token, ktorý je potrebný pridať do každej hlavičky volanie pre jeho autorizáciu. **Autorizácia** je proces, pri ktorom sa overuje či má používateľ oprávnenie vykonávať danú úlohu, ktorú žiada. Autorizácia aplikácie je implementovaná v triede JwtTokenFilter s adresou `server-spring-boot/src/main/java/sk/adambarca/serverspringboot/security/filter` v jej jedinej metóde *doFilterInterval*. V tejto metóde sa skontrolujú hlavičky každého volania pričom najprv hľadá či hlavička obsahuje JWT token a pokiaľ áno tak či užívateľ, ktorého reprezentuje daný token vôbec existuje. Ak všetko prebehlo v poriadku nastaví používateľa aj s jeho právami do bezpečnostného kon-

textu aplikácie. Definovanie autorizácie k jednotlivým koncovým bodom aplikácie sú definované v triede **WebSecurityConfig** s adresou `server-spring-boot/src/main/java/sk/adambarca/serverspringboot/security/config` v metóde *configure(HttpSecurity http)*, kde je definované, ktoré role majú prístup k danému koncovému bodu.

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.cors().and().csrf().disable();
    http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
    http.authorizeRequests()
            .antMatchers("/login").permitAll()
            .antMatchers("/users").hasAnyRole(ADMIN_ROLE)
            .antMatchers("/roles").hasAnyRole(ADMIN_ROLE)
            .antMatchers("/*").hasAnyRole(USER_ROLE, ADMIN_ROLE)
            .anyRequest().authenticated();

    http.addFilterBefore(jwtTokenFilter,
        UsernamePasswordAuthenticationFilter.class);
}
```

Výpis 7.7: Metóda configure pre autorizáciu používateľa

Ako môžeme vidieť v kóde 7.7 k prihláseniu má prístup úplne každý používateľ zatiaľ, čo k informáciám o registrovaných používateľoch, ako sú samotný používatelia alebo role, ktoré sú implementované rámci aplikácie majú prístup iba správcovia a admin. K ostatným koncovým bodom servera má prístup každý používateľ, ktorý je autentifikovaný.

7.2 Front-end

V tejto sekcii bude popísaná klientska časť aplikácie, ktorá bola vytvorená pomocou Ionicu a Angularu. Budú tu vysvetlené pojmy, ktoré sa týkajú hlavne Angularu a Ionicu. Ionic používa ako hlavný nástroj na vývoj aplikácii Ionic CLI (command-line interface). Vďaka tomu môžeme vytvárať jednotlivé časti aplikácie pomocou príkazového riadka a tiež jednoducho spúšťať a kompilovať aplikáciu.

Služba

Služba je trieda, ktorá nám poskytuje nejakú konkrétnu funkcionalitu, ktorú môžeme od našej aplikácie očakávať. Služby sa najviac používajú, ako prostriedok na komunikáciu so serverom pričom nám poskytujú dáta o komunikácii, ale môže sa využívať aj na iné procesy, čo aplikácia potrebuje. Angular tak isto ako aj Spring využíva pre služby **DI (dependency injection)**. To má za následok, že sa nestaráme o inicializáciu tohto objektu, iba oň požiadame a Angular sa postará o to, aby nainicializoval tento objekt pomocou injektora. Službu môžeme vytvoriť príkazom `ionic generate service`, pričom parametrami sú cesta služby alebo názov služby. Tento príkaz registruje službu v koreňovom injektore, ktorý sa vyznačuje ako **@Injectable({providedIn: 'root'})** v takom prípade Angular vytvorí jednu zdieľanú inštanciu služby, ktorú poskytne každému kto si o ňu požiada. Triedy musia o tieto služby požiadať v ich konštruktore. Príklad služby, ktorá sa stará o žiadanie dát zo servera je možné vidieť v kóde 7.8. Všetky služby vytvorené v tomto projekte majú cestu `ui-ionic\src\app\service`.

```
@Injectable({
  providedIn: 'root'
})
export class MeteorService {

  private resourceUrl = environment.api.API_URL + '/meteors';

  constructor(
    private httpUtils: HttpUtils
  ) { }

  create(meteor: MeteorDTO, intervalId: number): Observable<HttpResponse> {
    return from(Http.post(this.httpUtils.getHttpOptionsWithData(this.resourceUrl
        + `/intervals/${intervalId}`, meteor)));
  }

  getByProtocolId(protocolId: number): Observable<HttpResponse> {
    return from(Http.get(this.httpUtils.getHttpOptions(this.resourceUrl +
        `/protocols/${protocolId}`)));
  }

  deleteAll(meteors: MeteorDTO[]) {
    return from(Http.put(this.httpUtils.getHttpOptionsWithData(this.resourceUrl +
        "/deleteAll", meteors)));
  }
}
```

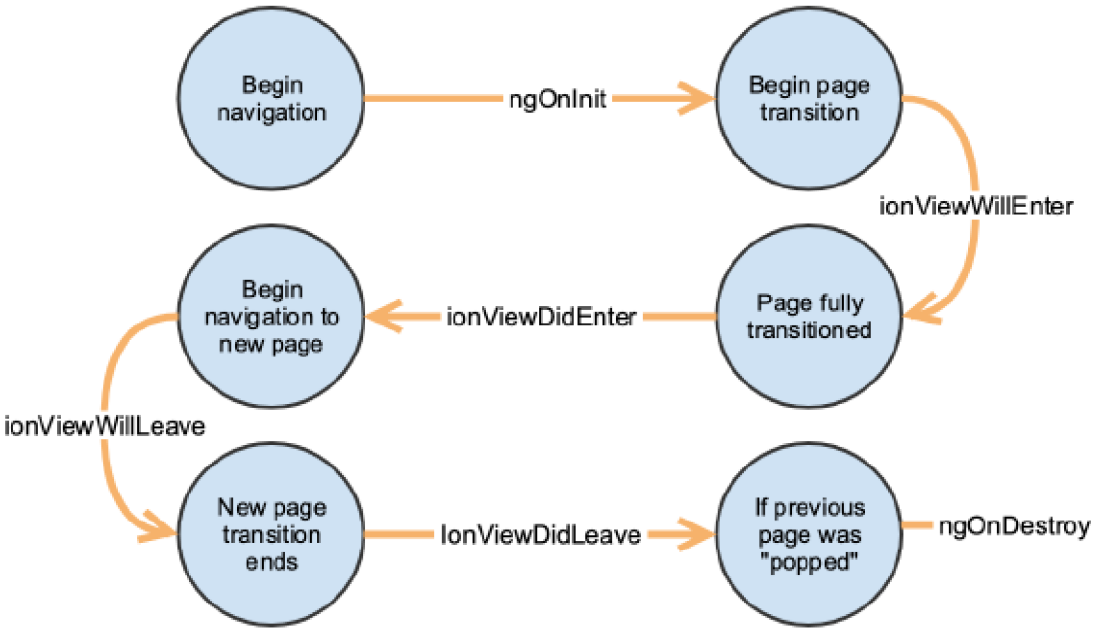
Výpis 7.8: Príklad jednoduchej služby pre spracovávanie meteorov zo servera

Stránka

Stránka je špeciálny blok, ktorý sa týka Ionicu a predstavuje jednu stránku obrazovky. Ionic stránka je nadstavbou Angular komponentu a mala by sa používať na vyjadrenie celej jednej stránky a nie len nejakej určitej časti, ako to je pri komponentoch. **Komponenty** poskytujú dáta a logiku, ktorá bude použitá v šablónach pomocou dátovými väzbami. Pomocou komponent môžeme rozdeliť aplikáciu na niekoľko funkčných blokov vďaka čomu je náš kód priehľadnejší. Tieto blogy môžu byť napríklad hlavička, formulár alebo tlačidlo. Ionic poskytuje nové **udalosti životného cyklu**, ktoré je možné použiť v stránke oproti pôvodným Angular udalostiam, ktoré je možné použiť v komponentoch. Pokiaľ Ionic načíta novú stránku tak tu starú stránku neodstráni iba ju schová za tu novú. Toto má pár výhod, a to takých, že môžeme zanechať stav starej stránky, ako sú jej dáta alebo pozícia v rolovaní a tiež nám to poskytuje plynulejší priebeh, pretože stránku už nemusíme znovu vytvárať. Udalosti a ich postupný priebeh je ilustrovaný na obrázku 7.1.
Udalosti životného cyklu Ionovej stránky:

- **ngOnInit** spustí sa iba raz, a to pri inicializácii komponenty. Používa sa na načítavanie údajov zo služieb, ktoré nevyžadujú obnovenie pri každej návšteve stránky.
- **ionViewWillEnter** spustí sa vždy pri zobrazení stránky. Táto udalosť je dobrá na načítanie údajov zo služieb, u ktorých vyžadujeme, aby sa aktualizovali pri každom zobrazení tejto stránky.

- **ionViewDidEnter** spustí sa až, keď je stránka na čítaná, čiže až vtedy, čo užívateľ už uvidel stránku. Môže pomôcť s výkonom aplikácie pokiaľ máme problémy s načítavaním údajov z udalosti ionViewWillEnter.
- **ionViewWillLeave** spustí sa pred animáciou novej stránky, ktorá sa ma načítať. Používa sa na vyčistenie, ako je napríklad odhlásenie odberu z pozorovateľných položiek.
- **ionViewDidLeave** táto udalosť sa spustí už, keď je nasledujúca stránka úplne načítaná. V tejto udalosti sa rieši logika, ktorá sa nemusí riešiť, keď je stránka viditeľná.
- **ngOnDestroy** používa sa, keď aplikácia zničí stránku. Môže slúžiť, ako dodatočné čistenie stránky, ktoré nebolo implementované v udalosti ionViewWillLeave.



Obrázek 7.1: Životný cyklus ionovej stránky prevzatý z [11]

Stránku môžeme vytvoriť pomocou príkazu `ionic generate page`, kde parametrami sú buď cesta súboru alebo jeho názov. Tento príkaz pridá novú trasu s názvom novej vygenerovanej stránky do hlavného navigačné súboru **app-routing.module.ts** a tiež vytvorí šesť súborov:

- **[názov stránky]-routing.module.ts** súbor pre definovanie trás našej aplikácie
- **[názov stránky].module.ts** súbor pre zaistenie importovania komponent, smerovania a poskytovateľov rámca
- **[názov stránky].page.html** definuje HTML šablónu pre stránku
- **[názov stránky].page.scss** definuje CSS štýly pre stránku
- **[názov stránky].page.spec.ts** definuje jednotlivé testy pre stránku
- **[názov stránky].page.ts** definuje logiku stránky

Šablóna

Šablóna definuje obsah komponentu, ktorý je napísaný v jazyku HTML. Pre lepšiu prácu so šablónou a jej logikou využívame **dátové väzby**, ktoré nám poskytuje Angular. Pomocou dátovej väzby môžeme priradzovať hodnotu premennej zo súboru, kde máme napísanú logiku komponenty do našej šablóny vďaka čomu môžeme dynamicky meniť obsah a vzhľad stránky. Cieľom dátovej väzby môže byť vlastnosť, udalosť alebo názov atribútu.
Dátové väzby je možné deliť na tri typy podľa ich smeru dátového toku:

- **od zdroja k šablóne** táto väzba sa používa na vyjadrenie hodnoty atribútu, štýl triedy atribútu a na interpoláciu. Interpolácia textu slúži na dynamickú zmenu toho, čo sa zobrazuje v aplikácii, ako je napríklad názov stránky. Cieľ väzby, ktorý je iný, ako interpolácia musíme ohraničiť hranatými zátvorkami `[]` a pri interpolácii musíme použiť dvojité zložené zátvorky `{{}}`. Príklad jednoduchej väzby zo zdroja k šablóne je možné vidieť v kóde 7.9.

```
<ion-item [routerLink]="page.url" routerDirection="forward"
    *ngIf="hasAnyAuthorities(page.roles)">
    <ion-icon [name]="page.icon" slot="start" color="white"></ion-icon>
    {{page.title}}
</ion-item>
```

Výpis 7.9: Vzorový kód datovej väzby od zdroja k šablóne

- **od šablóny k zdroju** táto väzba sa používa na udalosti komponentov alebo HTML elementov k cieľu. Pokiaľ máme udalosť, pri ktorej chceme zavolať nejakú funkciu použijeme presne túto väzbu. Táto väzba sa zaistí použitím guľatých zátvoriek `()`, ako je ukázané v kóde 7.10.

```
<ion-fab vertical="bottom" horizontal="center" slot="fixed">
    <ion-fab-button color="button-secondary" (click)="getCurrentLocation()">
        <ion-icon name="locate-outline"></ion-icon>
    </ion-fab-button>
</ion-fab>
```

Výpis 7.10: Vzorový kód datovej väzby od šablóny k zdroju

- **obojstranná väzba** táto väzba kombinuje obidve väzby zmienené vyššie. Slúži predovšetkým na to, keď chceme počúvať udalosti a zároveň aktualizovať hodnotu medzi nadradenými a podradenými komponentami. Syntax obojstrannej väzby je kombinácia hranatých a guľatých zátvoriek `[()]` a je ju možné vidieť vo vzorovom kóde 7.11.

```
<app-navigation title="Users" navType="search" [(searchText)]="searchText">
</app-navigation>
```

Výpis 7.11: Vzorový kód obojstrannej datovej väzbu

Direktíva

Direktíva je funkcia, ktorá sa vykoná vždy, keď ju kompilátor Angular nájde v **DOM** (Document Object Model). Direktívy rozširujú syntax jazyka HTML a tým nám poskytujú

lepšiu manipuláciu s DOM čím môžeme meniť vzhľad aplikácie.
Angular delí direktívy do troch typov:

- **komponenty** o komponentoch je viac popísané v sekcií 7.2. Táto direktíva je výnimočná tým, že má šablónu vďaka čomu určuje svoj vzhľad a nemení ho ostatným DOM elementom, ako je to u ďalších direktív.
- **direktívy atribútov** sa používajú na zmenu správania a vzhľadu DOM. Pomocou direktív atribútov môžeme dynamicky meniť vzhľad a správanie elementu pomocou premennej. Trieda musí mať anotáciu **@Directive** a definovaný **selektor**, ktorý označuje miesto, kde má Angular kompilátor zavolať funkciu. Direktívu môžeme vytvoriť pomocou príkazu `ionic generate directive` pričom parametrami sú cesta direktívy alebo jej názov. Každá direktíva by mala mať prístup k svojmu hostiteľskému elementu, ktorý chceme upravovať pomocou importovania **ElementRef** do kontruktora triedy. ElementRef poskytuje priamy prístup k hostiteľskému prvku DOM prostredníctvom jeho nativeElement vlastnosti [1]. Vzorový kód ukazujúci direktívu, ktorá mení farbu elementu na žltú 7.12 a jej následne použitie 7.13.

```
@Directive({
  selector: '[appHighlight]'
})
export class HighlightDirective {
    constructor(private el: ElementRef) {
       this.el.nativeElement.style.backgroundColor = 'yellow';
    }
}
```

Výpis 7.12: Atribútová direktíva na zmenu farby DOM elementu na žltú

```
<p appHighlight>Highlight me!</p>
```

Výpis 7.13: Vzorové použitie atribútovej direktívy 7.12

Angular ponúka tiež pár stavaných atribútových direktív, najpoužívanejšie sú:

  - **NgClass** pomocou tejto direktívy môžeme pridávať alebo odstraňovať CSS triedy.
  - **NgStyle** pomocou tejto direktívy môžeme pridávať alebo odstraňovať sadu HTML štýlov, ako je napríklad veľkosť písma alebo jeho font.
  - **NgModel** pridá obojstrannú dátovú väzbu k elementu formulára HTML.

- **štrukturálne direktívy** menia rozloženie DOM pridaním alebo odstraňovaním DOM elementov. Táto direktíva je špecifická tým, že sa pred jej názov píše hviezdička *. Tiež ako atribútová direktíva sa, aj táto vytvára pomocou príkazu `ionic generate directive`. Táto direktíva, ale musí mať v konštruktore definované premenné typu **TemplateRef** a **ViewContainerRef**. Pomocou premennej typu TemplateRef môžeme získať obsah DOM elementu a pomocou ViewContainerRef môžeme pristupovať ku kontajneru pohľadu, vďaka ktorému môžeme vytvoriť nový DOM element [2]. Angular nám poskytuje niekoľko vstavaných štrukturálnych direktív:

  - **NgIf** táto direktíva dokáže vytvárať alebo odstraňovať DOM elementy zo šablóny pomocou podmienky. Vzorový kód, ktorý popisuje direktívu NgIf je možné vidieť v 7.14 jej použitie 7.15.

```
@Directive({ selector: '[appCustomIf]'})
export class CustomIfDirective {

  constructor(
    private templateRef: TemplateRef<any>,
    private viewContainer: ViewContainerRef) { }

  @Input() set appCustomIf(condition: boolean) {
    if (condition) {
      this.viewContainer.createEmbeddedView(this.templateRef);
    } else {
      this.viewContainer.clear();
    }
  }
}
```

Výpis 7.14: Vzorová štrukturálna direktíva NgIf

```
<p *appCustomIf="true">Show me!</p>
```

Výpis 7.15: Použitie vzorovej štrukturálnej direktívy prevzatej z 7.14

– **NgFor** nám vytvára elementy koľko krát chceme. Táto direktíva sa používa na vytváranie zoznamov prvkov.
– **NgSwitch** nám umožňuje zobraziť jeden prvok z niekoľkých možných možností na základe podmienky.

Kapitola 8

Testovanie

Poslednou fázou vývoja aplikácie bolo testovanie. Testovanie je veľmi dôležitou fázou vďaka, ktorej môžeme zistiť mnoho chýb a vylepšení. Testovanie môžeme deliť do niekoľkých kategórii pričom tu budú popísané dva typy, a to testovanie vývojárom a testovanie užívateľmi. Testovanie aplikácii je zdĺhavý a náročný proces pričom jeho vynechanie môže mať za následok zhoršenú kvalitu produktu a celkového dojmu z aplikácie, preto by sa nemalo rozhodne podceňovať.

8.1 Testovanie vývojárom

Každú časť aplikácie je nutné dôkladne otestovať predtým než začneme robiť na ďalšej. Napríklad implementáciu databázy, komunikácia so serverom a rôzne formuláre, z ktorých sa primárne skladá naša aplikácia. Pokiaľ máme všetky časti aplikácie otestované a už nemáme žiadnu ďalšiu funkcionalitu, ktorú by sme chceli do aplikácie pridať je čas otestovať aplikáciu ako celok. Na otestovanie celej aplikácie som vytvoril pozorovania pomocou protokolu na IMO stránke [10], kde som zadal približne reálne údaje a úlohou bolo vytvoriť presne to isté pozorovanie v aplikácii. Testovacie dáta som navrhol na základe problematických častí aplikácie, kde by sa dalo očakávať nesprávny výstup aplikácie.

Testovací protokol č.1

Tento test sa zameriava na problematiku, kde je treba poslať protokol na účet pozorovateľa pokiaľ takýto pozorovateľ s týmto menom je registrovaný na stránke IMO. V tomto testovacom pozorovaní sa zaznamenal jeden sporadický meteor v čase od 23:00 do 23:30. Výsledné pozorovanie je možné vidieť v tabuľke 8.1. Skript, ktorý vytvorý dané pozorovanie má cestu `imo-protocol-tests/test1-data.sql`.

Testovací protokol č.2

V tomto teste bol vytvorený protokol pozorovateľom, ktorý nemá založený účet na IMO stránke. V tomto teste sa zameriavame na vytváranie meteorov a intervalov pozorovaní pričom boli vytvorené 4 intervaly, ktoré začali v jednom dni a skončili v druhom. Tiež sa tento test sústredí na vytváranie meteorov, ktoré majú hodnotu magnitúdy končiacu na 0.5 alebo ak počas intervalu nebol zaznamenaný ani jeden meteor z daného meteorického roju. Skript, ktorý vytvorí toto pozorovanie, ktoré je možné vidieť v tabuľke 8.2 má cestu `imo-protocol-tests/test2-data.sql`.

Observer: Barca Adam
Session Date UT: 2022-04-10
Location: Trenčín, Slovakia (lat: 48.88°, lng:18.03°)

Count distribution

Start date	End date	RA	Dec	Teff	F	Lm	SPO
2022-04-10 23:00	2022-04-10 23:30	60.00	75.00	0.50	1.11	3.08	1 (C)

Magnitude distribution

Start date	End date	Shower	0
2022-04-10 23:00	2022-04-10 23:30	SPO	1.0

Comments:
can delete 12.4.2022 20:50

Tabulka 8.1: Testovací protokol č.1

8.2 Testovanie užívateľmi

Poslednou fázou testovania bolo otestovať aplikáciu v reálnej situácii, kde bolo treba zistiť funkčnosť aplikácie a tiež zistiť názory a dojmy ostatných užívateľov. Skúšobné pozorovanie trvalo približne 30 minút a boli zaznamenané len 2 meteory. Ale aj počas tohto testovania sa objavila chyba, ktorá mala za následok poslania IMO protokolu v zlej forme. Problém bol v tom pokiaľ sme mali interval, ktorý má 0 meteorov tak sa tento interval nezaznamenáva do IMO protokolu, ale indexy intervalov som nechal také isté. To malo za následok, že prvý interval, ktorý mal nejaké meteory začínal napríklad indexom 1, čo IMO stránke nevyhovovalo, a tak keď prehľadávala dáta a nenašla žiadny interval s indexom 0 tak ďalej ani neprehľadávala ostatné intervaly. Výsledný protokol, ktorý bol vytvorený počas pozorovania je možné vidieť na obrázku 8.3. Druhou zásadnou vecou, ktorá sa objavila počas testovania bolo, že užívateľovi nebolo na prvý pohľad jasné prečo nevidí žiadnych pozorovateľov vo svojich dátach. Problém bol v tom, že žiadnych pozorovateľov nemal vytvorených, čiže sa mu nemohli ani žiadny zobraziť. Tento problém sa rieši tak, že užívateľa informujeme o skutočnosti, že dáta nie sú vytvorené a zoznam je prázdny namiesto toho, aby videl len prázdny zoznam.

Observation submitted in the name of: Adam123 Barca123
Submitted by: Barca Adam
Session Date UT: 2022-04-10
Location: Brno, Czechia (lat: 49.20°, lng:16.61°)

Count distribution

Start date	End date	RA	Dec	Teff	F	Lm	ANT	SPO
2022-04-10 22:00	2022-04-10 22:50	60.00	60.00	0.83	1.11	3.08	3 (C)	1 (C)
2022-04-10 22:50	2022-04-10 23:10	65.00	65.00	0.33	1.11	3.18	0 (C)	1 (C)
2022-04-10 23:50	2022-04-11 00:06	70.00	70.00	0.38	1.11	3.57	1 (C)	0 (C)
2022-04-11 00:13	2022-04-11 01:42	75.00	75.00	1.48	1.11	3.74	2 (C)	2 (C)

Magnitude distribution

Start date	End date	Shower	0	1	2	3
2022-04-10 22:00	2022-04-10 22:50	ANT	1.0	0.5	1.0	0.5
2022-04-10 22:50	2022-04-10 23:10	ANT	-	-	-	-
2022-04-10 23:50	2022-04-11 00:06	ANT	1.0	-	-	-
2022-04-11 00:13	2022-04-11 01:42	ANT	2.0	-	-	-
2022-04-10 22:00	2022-04-10 22:50	SPO	1.0	-	-	-
2022-04-10 22:50	2022-04-10 23:10	SPO	1.0	-	-	-
2022-04-10 23:50	2022-04-11 00:06	SPO	-	-	-	-
2022-04-11 00:13	2022-04-11 01:42	SPO	0.5	1.5	-	-

Comments:
13.04.2022 02:30

Tabulka 8.2: Testovací protokol č.2

Observer: Jozef Drga
Session Date UT: 2022-04-29
Location: Brno, Česko (lat: 49.24°, lng:16.51°)

Count distribution

Start date	End date	RA	Dec	Teff	F	Lm	ANT	SPO
2022-04-29 21:46	2022-04-29 22:11	290	0	0.417	1.0	6.25	1 (C)	1 (C)

Magnitude distribution

Start date	End date	Shower	3	4	5
2022-04-29 21:46	2022-04-29 22:11	ANT	0.5	0.5	-
2022-04-29 21:46	2022-04-29 22:11	SPO	-	-	1.0

Comments:

Tabulka 8.3: Testovacie pozorovanie

Kapitola 9

Záver

Cieľom tejto bakalárskej práce bolo vytvoriť mobilnú aplikáciu na operačný systém Android, ktorá umožní používateľovi počas pozorovania meteorov zapisovať dáta o pozorovaní. Z týchto dát by malo byť možné vytvoriť protokoly podľa štandardu IMO (International Meteor Organization) a tie následne prepojiť s databázou IMO VMDB. Po naštudovaní teórie a ako prebieha pozorovanie meteorov bolo tiež nutné zistiť, ako vyzerá IMO protokol a ktoré dáta sa v ňom zaznamenávajú. V tejto aplikácii boli výlučne použité dáta, ktoré pochádzajú z oficiálnych stánok IMO, ako sú napríklad meteorické roje alebo oblasti na oblohe. Ďalej sa tu vysvetlené rozdiely medzi typmi mobilných aplikácii a ich výhody a nevýhody. Sú tu zmienené technológie, ktoré boli použité k tvorbe tejto práce či už za účelom programovania alebo dizajnu. V analýze aplikácie som sa najviac zameral na protokol IMO, kde boli vysvetlené jednotlivé časti, z ktorých sa protokol skladá a tiež možnosti, ktorými sa daný protokol dá vyplniť. Návrh aplikácie či už databázy alebo užívateľského rozhrania boli prekonzultované s mojim vedúcim práce pánom Mgr. Jozefom Drgom, ktorý pomáhal navrhovať funkcionality aplikácie a tým zlepšiť celkovú aplikáciu. Počas návrhu používateľského rozhrania sa využili moderné princípy návrhu dizajnu, ktoré boli implementované do tejto práce. V implementačnej časti sú ukázané časti kódu, ktoré sú prevzaté z tejto práce a ktoré vysvetľujú jednotlivé fázy implementácie, ktoré je nutné poznať pre úspešnú implementáciu tejto práce pomocou jednotlivých technológii, ktoré boli použité v tejto práci. Testovanie aplikácie pomohlo overiť funkčnosť aplikácie, ako aj objaviť nové chyby poprípade rozšírenia. Výsledkom tejto práce je teda plne funkčná mobilná aplikácia, ktorá je zverejnená v Google Play[1] pre jej jednoduchú inštaláciu.

Každá aplikácia sa dá zlepšiť či už opravou nejakej chyby alebo pridanie poprípade úprava nejakej časti aplikácie. Aj, keď táto aplikácia splňuje, že sa dá pomocou nej vytvoriť IMO protokol to neznamená, že toto je jediný protokol, ktorý sa dá vyplniť počas pozorovania. K meteorom sa dajú pridať ďalšie veličiny, ktoré je možné zaznamenávať počas pozorovania, ako je napríklad kvalita alebo nadmorská výška. Aplikáciu je možné rozšíriť pomocou notifikácii, ktoré by nás upozorňovali na isté špecifické situácie. Týmito situáciami mám na mysli upozornenie, že si má dať pozorovateľ prestávku, aby sa nezhoršila jeho kvalita pozorovania alebo upozornenie, že pozorovateľ už mal dostatočne dlhú prestávku a je možné pokračovať v pozorovaní.

[1] aplikácia IMO dostupná v obchode Google Play https://play.google.com/store/apps/details?id=sk.adambarca.imo

Literatura

[1] Angular. *Attribute directives* [online]. 2022 [cit. 2022-04-16]. Dostupné z: https://angular.io/guide/attribute-directives.

[2] Angular. *Writing structural directives* [online]. 2022 [cit. 2022-04-16]. Dostupné z: https://angular.io/guide/structural-directives.

[3] Drga, J. *VIZUÁLNE POZOROVANIA METEOROV AKO ALTERNATÍVNE LABORATÓRNE CVIČENIA Z FYZIKY* [online]. 2015 [cit. 2022-04-04]. Dostupné z: https://ufv.science.upjs.sk/_projekty/smolenice/pdf_15/10_drga_petrik.pdf.

[4] Fandom. *CSS* [online]. 2022 [cit. 2022-03-23]. Dostupné z: https://htmlcss.fandom.com/wiki/CSS.

[5] Geeks, G. for. *Benefit of using MVC* [online]. 2021 [cit. 2022-03-31]. Dostupné z: https://www.geeksforgeeks.org/benefit-of-using-mvc/.

[6] Griffith, C. *What is Hybrid App Development?* [online]. 2022 [cit. 2022-03-25]. Dostupné z: https://ionic.io/resources/articles/what-is-hybrid-app-development.

[7] IBM. *Use-case diagrams* [online]. 2021 [cit. 2022-03-29]. Dostupné z: https://www.ibm.com/docs/en/rational-soft-arch/9.6.1?topic=diagrams-use-case.

[8] IBM. *Three-Tier Architecture* [online]. 2022 [cit. 2022-03-25]. Dostupné z: https://www.ibm.com/cloud/learn/three-tier-architecture.

[9] IMO. *The Observation* [online]. 2019 [cit. 2022-04-04]. Dostupné z: https://www.imo.net/files/data/brno/withLM/chart05_LM.pdf.

[10] IMO. *Add an Observation Session* [online]. 2022 [cit. 2022-03-27]. Dostupné z: https://www.imo.net/members/imo_observation/add_observation.

[11] Ionic. *Ionic Page Life Cycle* [online]. 2022 [cit. 2022-04-16]. Dostupné z: https://ionicframework.com/docs/angular/lifecycle.

[12] Jacobs, B. *Six Common Questions About Xcode Answered* [online]. 2022 [cit. 2022-03-24]. Dostupné z: https://cocoacasts.com/six-common-questions-about-xcode-answered.

[13] Jun, S. *Proč je Figma dar z nebes?* [online]. 2021 [cit. 2022-03-24]. Dostupné z: https://www.simonjun.cz/blog/proc-je-figma-dar-z-nebes.

[14] King, B. *RIGHT ASCENSION & DECLINATION: CELESTIAL COORDINATES FOR BEGINNERS* [online]. 2019 [cit. 2022-04-04]. Dostupné z: https://skyandtelescope.org/astronomy-resources/right-ascension-declination-celestial-coordinates/.

[15] Koďousková, B. *ARCHITEKTURA MVC: DEFINICE, STRUKTURA, FRAMEWORKY* [online]. 2021 [cit. 2022-03-31]. Dostupné z: https://www.rascasone.com/cs/blog/architektura-mvc-struktura-frameworky.

[16] mania, M. *Three-tier architecture* [online]. 2015 [cit. 2022-03-25]. Dostupné z: https://managementmania.com/en/three-tier-architecture.

[17] mania, M. *Natívna aplikácia* [online]. 2017 [cit. 2022-03-24]. Dostupné z: https://managementmania.com/sk/nativna-aplikacia-native-application.

[18] mDevelopers. *What is the Angular framework?* [online]. 2022 [cit. 2022-03-23]. Dostupné z: https://mdevelopers.com/blog/what-is-the-angular-framework.

[19] Moqups. *Low Fidelity vs High Fidelity Wireframes: Which should you use?* [online]. 2021 [cit. 2022-03-30]. Dostupné z: https://moqups.com/blog/low-fidelity-vs-high-fidelity-wireframes/.

[20] Mroczkowska, A. *What Is a Mobile App?* [online]. 2021 [cit. 2022-03-24]. Dostupné z: https://www.thedroidsonroids.com/blog/what-is-a-mobile-app-app-development-basics-for-businesses.

[21] NASA. *What Is a Meteor Shower?* [online]. 2018 [cit. 2022-04-03]. Dostupné z: https://spaceplace.nasa.gov/meteor-shower/en/.

[22] Oracle. *What Is a Database?* [online]. 2022 [cit. 2022-03-23]. Dostupné z: https://www.oracle.com/database/what-is-database/.

[23] Peter, B. *Základy pozorovania nočnej oblohy* [online]. 1994 [cit. 2022-04-03]. Dostupné z: http://astro.begi.sk/publ/zaknoc/zaklpoz.htm.

[24] Sharma, K. *What is PostgreSQL and why do enterprise developers and start-ups love it?* [online]. 2021 [cit. 2022-03-23]. Dostupné z: https://ubuntu.com/blog/what-is-postgresql.

[25] Team, I. E. *What Is a Web Application? How It Works, Benefits and Examples* [online]. 2021 [cit. 2022-03-24]. Dostupné z: https://www.indeed.com/career-advice/career-development/what-is-web-application.

[26] TechTerms. *Frontend* [online]. 2020 [cit. 2022-03-23]. Dostupné z: https://techterms.com/definition/frontend.

[27] Wickramasinghe, S. *The Spring Framework Beginner's Guide: Features, Architecture & Getting Started* [online]. 2021 [cit. 2022-03-23]. Dostupné z: https://www.bmc.com/blogs/spring-framework/.

[28] Wikipedia. *Android software development* [online]. 2022 [cit. 2022-03-24]. Dostupné z: https://en.wikipedia.org/wiki/Android_software_development.

Příloha A

Obsah priloženého pamäťového média

- **server-spring-boot/** zdrojové súbory serverovej časti
- **ui-ionic/** zdrojové súbory klientskej časti
- **tests/** testy pre overenie funkčnosti aplikácie
- **IMO.apk** inštalačný súbor aplikácie
- **README.md** popis pre sprevádzkovanie aplikácie
- **documentation/** zdrojové súbory textovej práce
- **xbarca04-documentation.pdf** táto textová práca vo formáte PDF