VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

INFORMAČNÝ SYSTÉM PRE ORGANIZÁTOROV ZÁVODOV PLACHETNÍC

INFORMATION SYSTEM FOR ORGANIZERS OF SAILING RACES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PETER VAŇO**
AUTHOR

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

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

155957

Ústav:	Ústav informačních systémů (UIFS)
Student:	**Vaňo Peter**
Program:	Informační technologie
Název:	**Informační systém pro organizátory závodů plachetnic**
Kategorie:	Informační systémy
Akademický rok:	2023/24

Zadání:

1. Seznamte se s tvorbou webových a mobilních aplikací a prostudujte potřebná vývojová prostředí.
2. Analyzujte požadavky na informační systém organizátorů závodů plachetnic zahrnující kromě základní funkcionality i podporu pro vhodné rozestavení bójek na trati včetně GPS navigace na základě aktuálního větru a zobrazení různých statistik.
3. Na základě požadavků a po konzultaci s vedoucím navrhněte informační systém.
4. Navržený systém implementujte a otestujte jeho funkčnost na vhodném vzorku dat.
5. Zhodnoťte dosažené výsledky a další možnosti pokračování tohoto projektu.

Literatura:

- Žára, O.: JavaScript - Programátorské techniky a webové technologie, Computer Press, 2015. ISBN: 978-80-251-4573-9.
- Welling, L., Thomson, L.: PHP a MySQL: Kompletní průvodce vývojáře. CPress, 2017.

Při obhajobě semestrální části projektu je požadováno:
SEP již byl abslovován.

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

Vedoucí práce:	**Bartík Vladimír, Ing., Ph.D.**
Vedoucí ústavu:	Kolář Dušan, doc. Dr. Ing.
Datum zadání:	1.11.2023
Termín pro odevzdání:	9.5.2024
Datum schválení:	30.10.2023

Abstrakt

Cieľom tejto práce je vytvoriť informačný systém pre organizátorov pretekov plachetníc. Systém je spravený ako webová aplikácia s PWA v jazyku Java a TypeScript s použitím frameworkov Spring a Angular. Hlavnou funkciou je správa pretekov a registrácia pretekárov. Súčasťou systému je podpora rozostavenia trate a následná navigácia k bójam. Pri práci bol dôraz na rozšíriteľnosť, znovu použiteľnosť a jednoduchosť.

Abstract

The aim of this thesis is to create an information system for organizers of sailboat races. The system is created as a web application with PWA in Java and TypeScript using the Spring and Angular frameworks. The main function of the system is the management of races and the registration of participants. The system also includes support for placing the race course and navigation to buoys afterward. During the work, importance was placed on expandability, reuse, and simplicity.

Kľúčové slová

Informačný systém, Regata, Preteky plachetníc, Trojvrstvová architektúra, Java, Spring, Angular, REST API, HTML, CSS, TypeScript, OpenLayers, Mapy, Projekcie, Navigácia, Trate, PWA, Docker

Keywords

Information system, Regatta, Sailboat races, Three-tier architecture, Java, Spring, Angular, REST API, HTML, CSS, TypeScript, OpenLayers, Maps, Projections, Navigation, Race courses, PWA, Docker

Citácia

VAŇO, Peter. *Informačný systém pre organizátorov závodov plachetníc*. Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vladimír Bartík, Ph.D.

Informačný systém pre organizátorov závodov plachetníc

Prehlásenie

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

.....
Peter Vaňo
15. mája 2024

Poďakovanie

Ďakujem pánovi Ing. Vladimírovi Bartíkovi Ph.D. za zastrešenie mojej bakalárskej práce, za voľnosť a ústretovosť, ktorú mi pri vypracovaní poskytol. Taktiež chcem poďakovať hlavnému kapitánovi Bc. Miroslavovi Hrivňákovi a dôstojníkovi pre komunikáciu Martinovi Škorupovi z Hlavného kapitanátu vodných skautov (HKVS) Slovenska za konzultácie potrieb podpory pre skautské regaty.

Obsah

Zoznam obrázkov

Kapitola 1

Úvod

Táto práca sa zaoberá tvorbou informačného systému pre organizátorov regát – závodov plachetníc. Cieľom praktickej časti je jednoduchý, prehľadný, rozšíriteľný systém pre organizátorov a pretekárov, s podporou pre rozostavovanie bójí vyznačujúcich trať na štýl olympijského trojuholníka (tzv. karuselový pretek). Motiváciou práce je potreba podpory pre organizáciu regát Slovenských vodných skautov v rámci rozvoja vodného skautingu na lokálnej a celoštátnej úrovni.

Na začiatok v kapitole 2 rozoberiem definície informačných systémov podľa predstaviteľov rôznej odbornosti. Rozdelím ich podľa najčastejšie používaného delenia, čím sa dostanem k ich základnému deleniu. Predstavím najznámejšie architektúry informačných systémov a spomeniem niektoré ich výhody a nevýhody.

Následne v kapitole 3 predstavím základné trate pretekov plachetníc, ktoré sa vyznačujú bójami, vrátane olympijského trojuholníka, ktorý aktuálne používajú slovenskí vodní skauti. Ďalej sa budem venovať rôznym projekciám máp a ich výhodám a nevýhodám, ako sa mapy uchovávajú v digitálnej forme a ako sa s nimi pracuje. Kapitolu ukončím základmi lokalizácie a vytyčovania trasy.

V kapitole 4 predstavím získané požiadavky na systém, ktoré som vyformoval opakovanými konzultáciami s predstaviteľmi slovenských vodných skautov. Potom definujem diagram prípadov použitia a vysvetlím, prečo je podstatný pre vývoj informačných systémov. Odprezentujem svoj diagram, ktorý som vytvoril na základe získaných požiadaviek. Na záver, v kapitole 4.3 uvediem svoj prehľad existujúcich technológií, ktorý som si zostrojil pred návrhom a implementáciou informačného systému.

Kapitola 5 sa zaoberá samotným návrhom systému. Najprv predstavím architektúru, ktorú som vybral pre môj informačný systém. Následne vyberiem technológie, ktoré budem používať a zdôvodním svoj výber. Vysvetlím čo je to ER diagram a odprezentujem svoj diagram. Na záver kapitoly rozoberiem svoj návrh používateľského rozhrania aj s ukážkou obrazoviek.

V kapitole 6 popisujem ako som postupoval pri implementácii informačného systému. Postupne prejdem implementáciou všetkých častí systému. Na záver kapitoly sa viacej venujem niektorým zaujímavým implementačným detailom.

Na koniec zhrniem v kapitole 7 ako prebiehalo testovanie aplikácie popri vývoji a vo finálnej verzii.

Kapitola 2

Informačné systémy

Čo je to informačný systém?

- Definícia podľa ChatGPT:

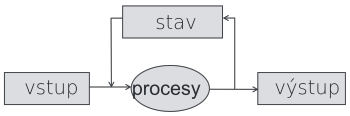
  > „Informačný systém je zložitá sústava zariadení, softvéru, procesov a ľudí, ktorá zhromažďuje, uchováva, spracováva a poskytuje informácie na podporu rozhodovania, riadenia a iných činností v organizácii. Ide o nástroj, ktorý pomáha spravovať a využívať informácie efektívne a účinne.“ [8]

- Definícia podľa Wikipédie:

  > „Informačný systém je systém na zber, udržiavanie, spracovanie a poskytovanie informácií. Počítačový informačný systém je systém zložený z ľudí a počítačov, ktorý spracováva alebo interpretuje informácie.“ [47]

- Definícia podľa doc. Ing. Radeka Burgeta, Ph.D.[1]:

  > „Systém lze chápat jako množinu prvků a vazeb mezi nimi, které jsou definovány na nějakém nosiči. Informační systém je otevřený systém, jehož nosič používá konceptuální zdroje – informace.“ [16]



Obr. 2.1: Schéma informačného systému. Prevzaté z [16]

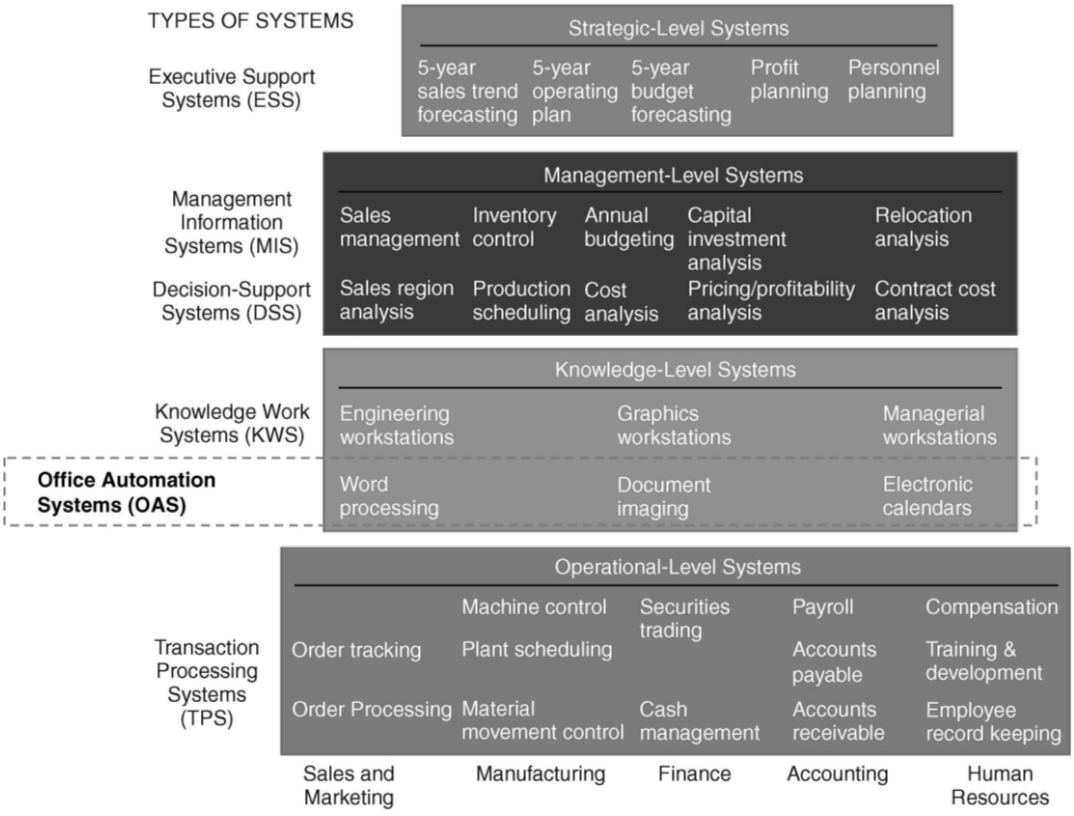
Tieto definície reprezentujú pohľad na informačné systémy podľa umelej inteligencie, polo odbornej verejnosti a odborníka na ne v tom istom poradí. Definície sú si navzájom podobné, čo značí o ustálenosti a všeobecnom povedomí v spoločnosti. Laikmi sú vnímané ako celok vrátane používateľov a vnímajú ich viacej ako čiernu skrinku, kde nepoznajú jej vnútorné fungovanie. Pohľad odborníka je zameraný na systém, jeho vnútorné fungovanie

[1] doc. Ing. Radek Burget, Ph.D. – hlavný výskumník Výskumnej skupiny informačných a databázových systémov a koordinátor predmetu Informačné systémy na Fakulte informačných technológií Vysokého učení technického v Brně

a používatelia v ňom figurujú ako prvky mimo systému, ktoré ovplyvňujú jeho vstupy. Informačné systémy sú, podľa prieniku definícií, systémy spracúvajúce informácie na základe vstupov používateľov a aktuálneho stavu systému a poskytujú spracované, alebo transformované informácie ako svoj výstup – obrázok 2.1.

2.1 Delenie informačných systémov

Podľa viacerých článkov a blogov zameraných na popularizáciu a vzdelávanie v oblasti informačných systémov ich delíme do šiestich skupín podľa ich zamerania a hierarchie v rámci organizácie. Skupiny sú nasledovné: Transakčné systémy, Kancelárske automatizačné systémy, Vedomostno-pracovné systémy, Manažérske informačné systémy, Systémy na podporu rozhodovania, Výkonné[2] informačné systémy. [28, 36, 41]



Obr. 2.2: Typy informačných systémov. Prevzaté z [22]

Transakčné systémy – po anglicky Transaction Processing Systems (TPS), sú systémy, v ktorých je potrebné vykonať celý vybraný proces, alebo ho odmietnuť bez zavedenia zmien do systému. Môže ísť o internetový obchod, registračný systém v knižnici, bankové aplikácie atď. V prípade internetových obchodov nesmú byť stiahnuté peniaze z účtu, a potom systém zruší objednávku, lebo je tovar nedostupný. V prípade transakčných systémov, transakcia nie je len finančný, alebo komerčný termín, ale zároveň môže byť aj databázový,

[2] výkonné – z pohľadu riadenia a hierarchie firmy, napr. výkonné riaditeľstvo

alebo procesný termín. Po zlyhaní časti transakcie sa systém musí vrátiť do stavu, v akom bol pred jej začatím.[4]

Kancelárske automatizačné systémy – po anglicky Office Automation Systems (OAS), sú systémy používané na zjednodušenie práce, komunikácie a procesov v rámci spoločnosti, alebo skupiny. Ide o komunikačné systémy, systémy pre spravovanie dokumentov, systémy sledujúce work-flow[3] organizácie a pod.[7]

Vedomostno-pracovné systémy – po anglicky Knowledge Work Systems (KWS), v minulosti známe aj ako expertné systémy. Pre fungovanie týchto systémov a efektívne vykonávanie ich funkcií je potrebné využívať špecifické odborné vedomosti z danej problematiky. Do tejto skupiny patria aj systémy implementujúce, alebo využívajúce umelú inteligenciu, jej princípy alebo základy.[45]

Manažérske informačné systémy – po anglicky Management Information Systems (MIS), slúžia pre manažérov v spoločnosti na štúdium a prehľad štatistík firmy, pomoc pri bežnom nízko-úrovňovom rozhodovaní. Zahŕňajú väčšinou iba interné dáta, ako históriu výkonu firmy, alebo sumarizované informácie z firemného TPS.[40]

Systémy na podporu rozhodovania – po anglicky Decision Support Systems (DSS), pomáhajú manažérom pri špecifických rozhodnutiach, najmä keď problém nie je vopred úplne definovaný, a tým pádom ani postup nie je vopred celý známy. Okrem interných informácií dostupných pre MIS, obsahujú aj sadu externých informácií, ako napr. ceny na burze, u konkurenta.[40]

Výkonné informačné systémy – po anglicky Executive Support Systems (ESS), slúžia pre vrcholové vedenie firmy na podporu rozhodovania smerovania celej organizácie. Okrem sumarizovaných výstupov z MIS a DSS obsahujú aj podporu pre externé udalosti, ako napr. plánované zmeny zákonov.[40]

So zameraním na fungovanie týchto systémov, je možné ich rozdeliť do dvoch základných skupín. Systémy, pre ktoré je podstatné mať k dispozícii aktuálne dáta hneď po ich zmene (napr. zmenu množstva tovaru na sklade po vykonaní predaja, zmeny v spoločne upravovanom dokumente). Prvú základnú skupinu nazývame **OLTP – Online Transaction Processing**. Patria sem TPS a OAS. Pre druhú skupinu je podstatnejšia analýza a nepotrebujú zmeny registrovať hneď, ako sa udejú – je tolerované opozdenie. Táto skupina sa označuje **MIS – Management Information Systems**. V tomto prípade prekladáme ako Informačné systémy pre podporu riadenia, keďže sa jedná o základnú skupinu obsahujúcu podporné informačné systémy. Do tejto skupiny sa radia KWS, MIS[4], DSS, ESS. Analytické systémy bez priameho napojenia na štruktúru organizácie sa označujú **OLAP – Online Analytical processing** a v kontexte OLTP a MIS patria pod skupinu MIS. Bez hierarchie organizácie sú práve OLAP druhou základnou skupinou spolu s OLTP.[16]

2.2 Architektúra informačných systémov

V tejto kapitole sú rozoberané najčastejšie architektúry informačných systémov, ich výhody a nevýhody.

[3] work-flow, doslovne pracovný tok – ustálené pracovné postupy, procesy používané v organizácii, alebo skupine, pre zefektívnenie práce

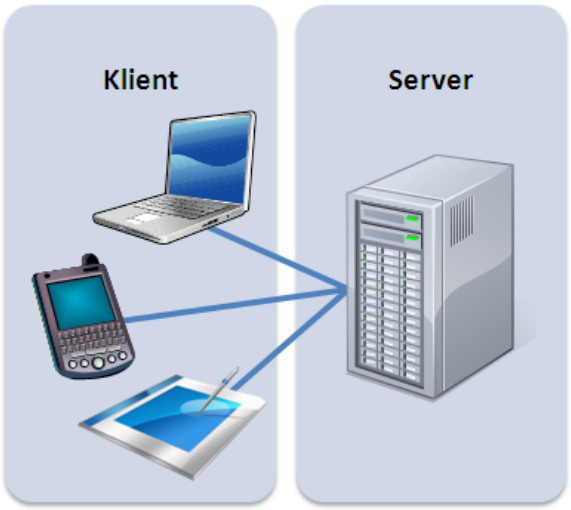
[4] MIS – v kontexte skupiny Manažérskych informačných systémov

2.2.1 Klient – Server architektúra

Pokiaľ nie je napísané inak, informácie uvedené v tejto kapitole čerpajú z [23]. Klient – Server architektúra je tiež známa aj ako dvojvrstvová architektúra. Vrstva je v tomto kontexte myslená ako fyzické oddelenie častí systému na klientskú časť a serverovú časť.

Klientská časť, alebo aj front-end, je dostupná na zariadení používateľa vo forme internetovej stránky, alebo inštalovateľnej aplikácie. Všetky aktuálne verzie klientskej časti komunikujú s tou istou serverovou časťou, čo zabezpečuje konzistentnosť dát. Jedným z parametrov definície klientskej časti je jej tzv. hrúbka. Hrúbka vyjadruje množstvo funkcionality implementovanej v klientskej časti. Logika na jednej strane zväčšuje veľkosť a zložitosť, na druhej strane znižuje záťaž na serverovú časť.

Serverová časť, alebo aj back-end, poskytuje pre klientskú časť spracované dáta vo vopred určenej štruktúre. Klientská časť pošle požiadavku na dáta, serverová časť ju spracuje, získa potrebné dáta, spracuje ich na odpoveď a tú odošle naspäť. V prípade zmeny logiky na serverovej časti nie je potrebné robiť zmenu na klientskej časti, pokiaľ sa nezmení štruktúra dát prenášaná medzi časťami. Rovnako to funguje aj v opačnom prípade. To zabezpečuje lepšiu dostupnosť a menší počet potrebných aktualizácií konkrétnej časti.



Obr. 2.3: Znázornenie Klient – Server architektúry. Prevzaté z [26]

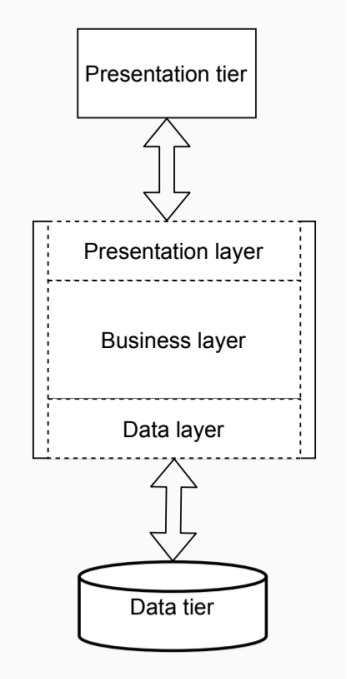
2.2.2 Trojvrstvová architektúra

Trojvrstvová architektúra je rozšírenou dvojvrstvovou, kde je serverová časť rozdelená na aplikačnú vrstvu a databázovú vrstvu. Databáza môže byť aj na tom istom zariadení ako aplikačná vrstva, ale beží nezávisle od nej. Aplikačná vrstva rieši logiku spracovania dát a sprostredkuje komunikáciu medzi databázovou časťou a klientskou, ktoré na priamo nekomunikujú. [24]

Aplikačnú vrstvu je možné ešte rozdeliť na tri logické vrstvy, pre lepšiu organizovanosť a udržateľnosť kódu. Jednotlivé vrstvy sú: prezentačná vrstva, biznis vrstva, dátová vrstva. V tomto logickom členení, prezentačná vrstva zabezpečuje komunikáciu s klientom, prípadne sa pod ňu môže zaradiť aj definícia rozhrania komunikácie. Biznis vrstva býva väčšinou najrozsiahlejšia a implementuje logiku aplikácie a spracovanie dát medzi formátmi klientskej

vrstvy a databázovej vrstvy. Dátová vrstva zabezpečuje komunikáciu s databázou na základe potrieb biznis vrstvy. [17]

Tak, ako bolo možné rozdeliť serverovú časť na aplikačnú vrstvu a databázovú vrstvu, tak je možné pokračovať v delení ostatných a novovzniknutých častí. Vrstvy sa môžu deliť ďalej podľa špecifických potrieb informačného systému. Takúto architektúru nazývame n-vrstvová architektúra (angl. n-tier architecture). [24]



Obr. 2.4: Znázornenie Trojvrstvovej architektúry s logickým členením. Prevzaté z [17]

2.2.3 Mikroservisy

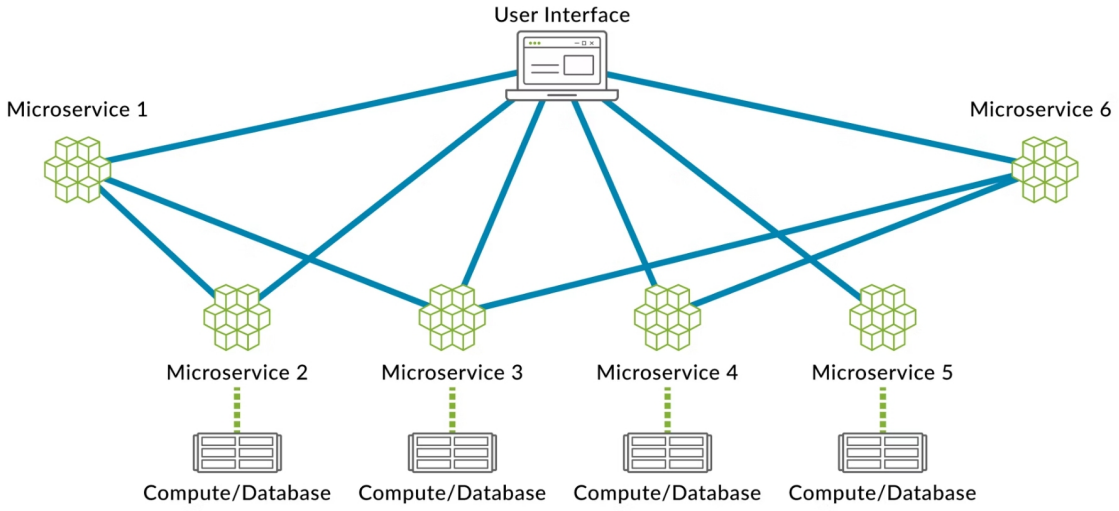
Mikroservisy sú zo spomenutých architektúr najmladšou. Riešia problém nafukovania a komplexnosti systémov počas ich vývoja. Mikroservisy bežia a sú nasadzované nezávisle od seba. Komunikujú medzi sebou výhradne po sieti podľa dohodnutého rozhrania. Snahou mikroservisy je dostať sa k pravidlu jednej zodpovednosti[5] – servis má vykonať iba to, čo má na starosti a nemá robiť nič navyše. S úrovňou členenia a izolácie sa zväčšujú rovnako ich výhody, ako aj nevýhody. [30]

Medzi najväčšie výhody patrí heterogenita – vďaka komunikácii medzi servismi po sieti, každý môže používať inú technológiu, ktorá je najvhodnejšia pre jeho funkciu. Pri správnom výbere to môže viesť k lepšiemu výkonu systému ako celku, môžu sa používať špecializované databázy pre potreby konkrétneho servisu, skúšať a používať nové technológie s omnoho menším dopadom a množstvom zmien potrebných pre nasadenie oproti monolitným systémom. Ďalšou výhodou je izolovanosť problémov. Pokiaľ nedôjde ku kaskádovému zlyhaniu, zlyhanie servisu zostane izolované a systém môže fungovať aj naďalej. Voľba hardvéru môže byť špecializovaná podľa potrieb servisov. Nie je potrebné zabezpečiť výkonný hardvér pre celý systém, ale len pre servisy, ktoré ho naozaj potrebujú. Ostatné môžu fungovať

[5]Pravidlo jednej zodpovednosti – známe aj pod anglickým názvom single responsibility principle

aj na slabších zariadeniach, prípadne s menším úložiskom a pod. S tým súvisí aj výhoda aktualizácií a nasadzovania. Pre zmeny v jednom servise, nie je potrebné znova nasadzovať celý systém, ale iba konkrétny modul. [30]

Architektúra mikroservisov sa nezaobíde bez nevýhod. Hneď prvou je ich komplexnosť. Množstvo servisov výrazne zvyšuje komplexnosť celej architektúry a jej správy. Miesto jednej aplikácie je potrebné nasadzovať a spravovať za každý modul jeden mikroservis, čo sa ľahko môže počítať na desiatky. Keďže komunikácia medzi servismi prebieha po sieti, vznikajú všetky problémy s tým spojené – strata informácií, poruchy siete, spojenia, zabezpečenie voči útokom, formát posielania a prijímania dát, atď. Testovanie a hľadanie chýb je taktiež omnoho náročnejšie. Pri testovaní je potrebné simulovať komunikáciu medzi servismi, alebo vstupné dáta. Hľadanie chýb väčšinou znamená nazretie do logov niekoľkých servisov, kým je zrejmá príčina chyby. Taktiež vzniká riziko kaskádového zlyhania, kedy prestane fungovať niekoľko, alebo všetky servisy v dôsledku chyby v jednom z nich. [35]



Obr. 2.5: Znázornenie architektúry mikroservisov. Prevzaté z [29]

Kapitola 3

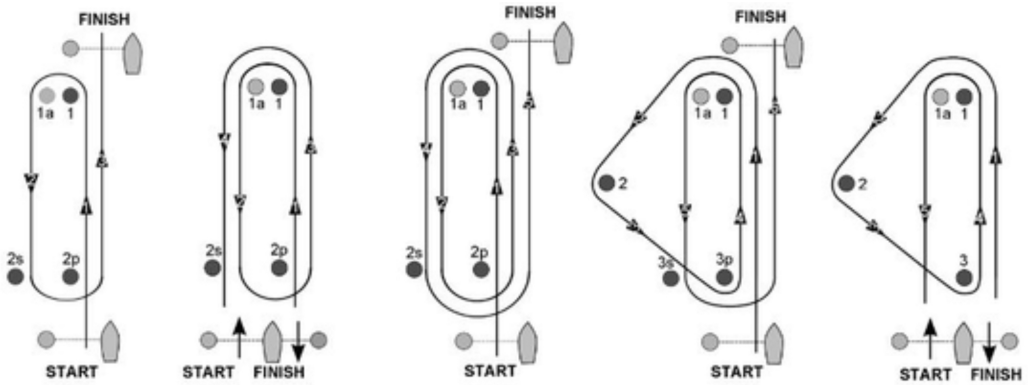
Trate a navigácia

Táto kapitola sa zaoberá informáciami o základných tratiach pretekov plachetníc a základmi navigácie z pohľadu informačných technológií, ako aj základnými typmi reprezentácie Zeme na mapy.

3.1 Trate

S rozvojom jachtingu a jeho popularizáciou sa vyvíjali aj trate, na ktorých sa preteky uskutočňovali. V roku 1996 bolo plachtenie zaradené medzi olympijské športy a v roku 2000 na olympiáde v Sydney sa prvý-krát aj pretekalo. Trate sú navrhnuté, aby preskúšali pretekára pri plachtení s rôznym smerom vetra. [32]

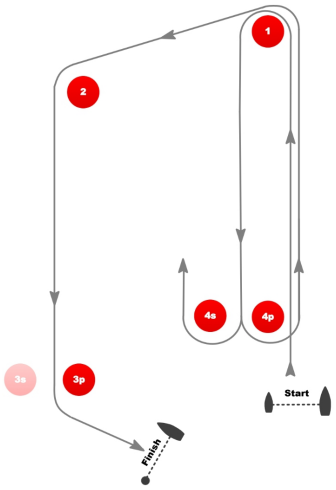
Náveterná – záveterná trať je najjednoduchším typom trate vyznačenej bójami. Bóje sú orientované v smere vetra. Pretek začína buď v strede medzi bójami, kde je štartovacia čiara vyznačená dvomi rozhodcovskými loďami, alebo pri záveternej bóji, kde je čiara vyznačená bójou a rozhodcovskou loďou. Plachetnice smerujú k náveternej bóji – plachtia proti vetru, bóju oboplávajú a vrátia sa k záveternej – plachtia v smere vetra. Po jej oboplávaní sa musia ešte raz vrátiť k náveternej bóji, kde je buď vyznačená cieľová čiara samotnou bójou a rozhodcovskou loďou, alebo je štartovacia čiara v prípade záveternej bóje a rozhodcovskej lodi zároveň aj cieľovou. Trať je možné predĺžiť viacerými kolami okolo bójí. [38] Pre schému tratí pozri obrázok 3.1 – tri ľavé trate.



Obr. 3.1: Schémy dvoj-bójových (ľavé tri) a troj-bójových (pravé dve) tratí. Prevzaté z [34]

Olympijský trojuholník, tiež známy ako Karuselový pretek, pozostáva z troch bójí rozmiestnených vo vrcholoch rovnostranného trojuholníka, ktorého jedna strana je súbežná so smerom vetra. Plachetnica štartuje pri záveternej bóji, pláva k náveternej a následne sa vracia späť k záveternej okolo bočnej. Nasleduje znovu plavba k náveternej, potom k záveternej a do cieľa pri náveternej bóji. [38] Popísaná konfigurácia je znázornená na obrázku 3.1 ako druhá sprava. Je zároveň aj najčastejšie používanou na pretekoch českých vodných skautov a aktuálne jedinou používanou slovenskými vodnými skautmi, preto som tento typ trate zvolil pre implementáciu v praktickej časti. Pôvodne sa používala aj na olympijských hrách pre väčšinu kategórií, v dnešnej dobe sa však už používa lichobežníková trať. Existujú aj podobné konfigurácie so štartom a cieľom na iných miestach, prípadne väčším počtom kôl.

Lichobežníková trať je trať aktuálne používaná na olympiáde vo väčšine kategórií a na značnom množstve iných národných a medzinárodných pretekov. Základne lichobežníku sú súbežné so smerom vetra. Pretek začína pri záveternej bóji dlhšej základne, pretekári oboplávajú náveternú bóju dlhšej základne, ďalej náveternú kratšej základne a potom okolo záveternej kratšej základne do cieľa. Trať sa predlžuje kolom okolo dlhšej, alebo kratšej základne lichobežníka – v prípade olympijskej trate kolom okolo dlhšej základne. [38]



Obr. 3.2: Schéma lichobežníkovej trate. Prevzaté z [38]

Okrem tratí vyhradených bójami, existujú ešte preteky medzi miestami. Cieľom takýchto pretekov je doplaviť sa z jedného miesta do ďalšieho, prípadne pri dlhých pretekov prejsť cez sériu miest.

3.2 Mapy

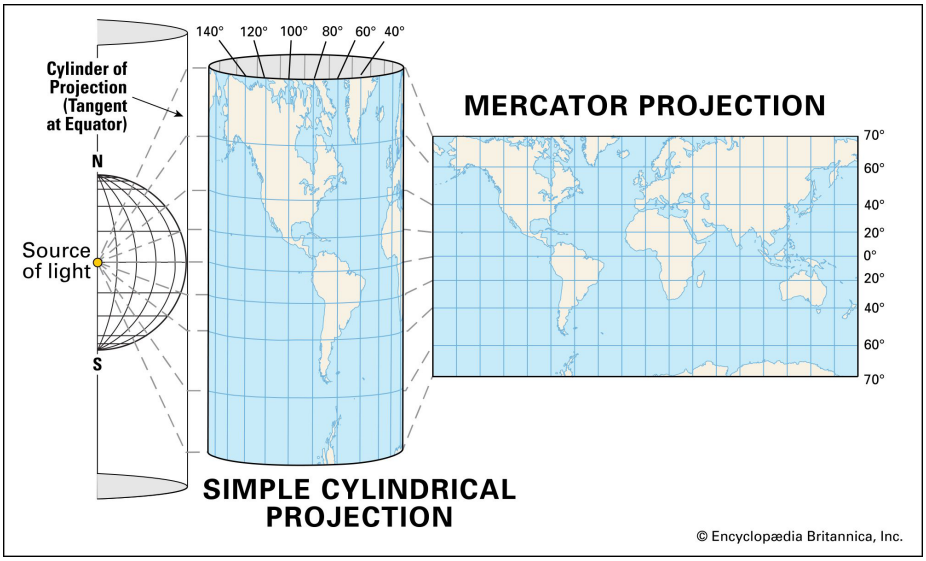
Pochopenie máp a základných princípov ich tvorby je podstatné pri akejkoľvek práci s nimi. Či už pri odčítaní vzdialenosti na papierových mapách, alebo pri samotnom výbere mapy pre danú oblasť môže vzniknúť množstvo problémov a chýb, pokiaľ sa nezamyslíme nad niektorými princípmi kartografie.

3.2.1 Projekcia

Mapy sú tvorené projekciou Zeme na dvojrozmernú plochu, väčšinou na povrch jednoduchšieho telesa. Špeciálnym prípadom je napríklad glóbus, ktorý je projekciou na povrch gule.

Rôzne projekcie majú svoje výhody a nevýhody, a tak aj keď je jedna vhodná na daný problém, v inom môže byť nepoužiteľná kvôli skresleniu.

Valcová projekcia je jedným z najbežnejších a najstarších zobrazení. Najznámejším predstaviteľom je celosvetovo používaná Mercatorová projekcia[1], kde sa pomyselný valec o nekonečnej výške umiestní okolo zeme tak, aby jeho os bola zhodná so Zemskou osou a dotýkal sa rovníka. Následne sa každé miesto premietne na povrch valca pomyselnou polpriamkou začínajúcou v strede zeme a prechádzajúcou cez dané miesto na priesečník s povrchom valca. Čím je miesto ďalej od rovníka, tým podlieha väčšiemu skresleniu, dokonca samotné póly nie je možné premietnuť. Vzdialenosti v oblasti rovníku sú relatívne presné, s malým skreslením, narozdiel od pólov, pri ktorých sa javia omnoho väčšie než v skutočnosti sú. Z praktických dôvodov sa preto projekcia nerobí nad určitú zemepisnú šírku. Projekcia zobrazuje poludníky ako rovnobežné čiary a pri rovnobežkách zachováva ich rovnobežnosť. Taktiež zachováva kolmosť poludníkov na rovnobežky a preto je vhodná pre držanie kurzu, určovanie azimutu[2]. Úsečka v tomto zobrazení nepredstavuje najkratšiu vzdialenosť medzi dvomi bodmi. Na malé vzdialenosti je rozdiel zanedbateľný, avšak napríklad pri ceste z Európy do Ameriky je rozdiel už niekoľko stoviek kilometrov. Výnimkou sú úsečky na rovníku, alebo orientované zo severu na juh – úsečky, ktoré by po predĺžení predstavovali kružnice so stredom v strede Zeme. Projekcia sa používa napríklad na mapu sveta. [5] Princípy tejto projekcie je možné aplikovať aj s rôzne orientovaným valcom pre menšie skreslenie iných oblastí.



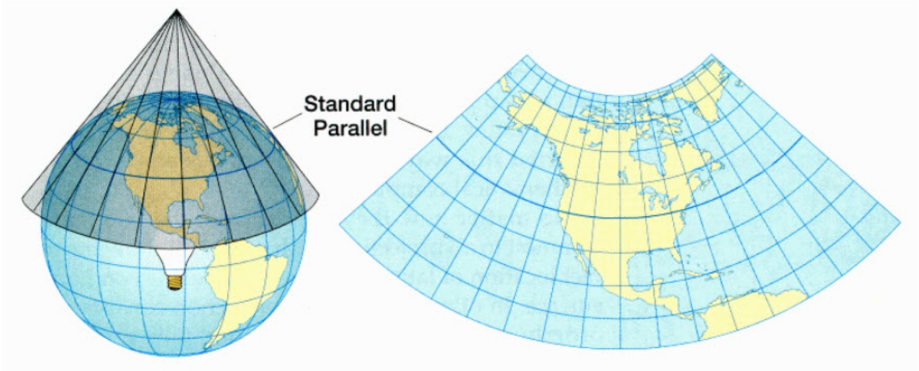
Obr. 3.3: Ukážka Mercatorovej projekcie. Prevzaté z [5]

Kužeľová projekcia umiestni pomyselný kužeľ okolo Zeme, ktorého povrch pretína Zem na jednej, alebo na dvoch kružniciach. Body na Zemi sa premietajú rovnako ako pri valcovej projekcii – zo stredu Zeme. Vzdialenosti na týchto kružniciach sú zachované a smerom od nich sa skreslenie zväčšuje. Skreslenie je konštantné pozdĺž rovnobežiek, a preto

[1] Mercatorová projekcia – pomenovaná podľa autora projekcie, kartografa Gerardusa Mercatora žijúceho v 16. storočí

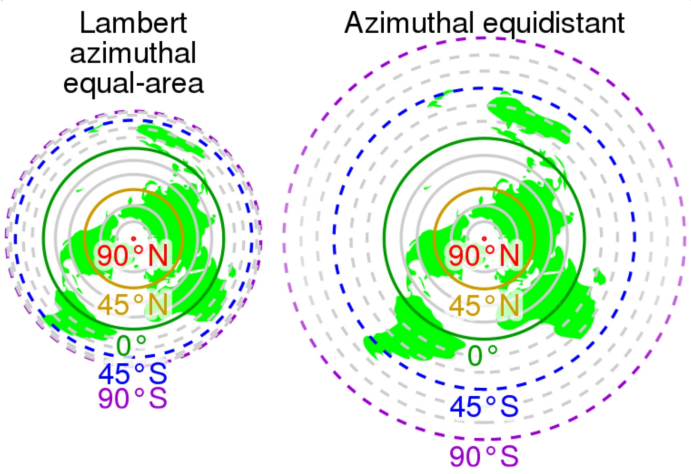
[2] azimut – uhol (označovaný aj pochodový uhol) používaný v geografii, ktorého ramená sú definované severom a daným miestom, vrchol je miesto z ktorého je uhol meraný, uhol sa meria v stupňoch od severu v smere hodinových ručičiek (v smere na východ)

je vhodné pre mapy veľkých oblastí v smere východ – západ, prípadne porovnávanie skutočných veľkostí štátov v rovnakej zemepisnej šírke, alebo ležiacich na kružniciach, ktorými kužeľ pretína Zem. Tento typ projekcie sa najčastejšie používa napríklad na mapu Spojených Štátov Amerických. [13]



Obr. 3.4: Ukážka kužeľovej projekcie. Prevzaté z [42]

Azimutová projekcia premieta povrch Zeme na kruh. Uhly zo stredu projekcie sú zachované, a tak všetky úsečky zo stredu do ľubovoľného bodu predstavujú najkratšiu vzdialenosť. V rámci tohto typu existujú aj projekcie bez deformácie vzdialenosti, alebo bez deformácie plochy. Naopak tvary a uhly sú všade okrem zo stredu projekcie deformované. Touto projekciou je možné zobraziť aj celú Zem, pričom bod presne na opačnej strane Zeme sa stane kružnicou na kraji celej projekcie – bude po celom okraji, ľubovoľným azimutom sa dostaneme zo stredu projekcie ku nemu. Tento typ projekcie je vhodný na meranie najkratšej vzdialenosti, smeru dvoch vzdialených miest, prípadne porovnávanie plochy dvoch oblastí, pri ktorých by kužeľová projekcia nebola praktická. [20]



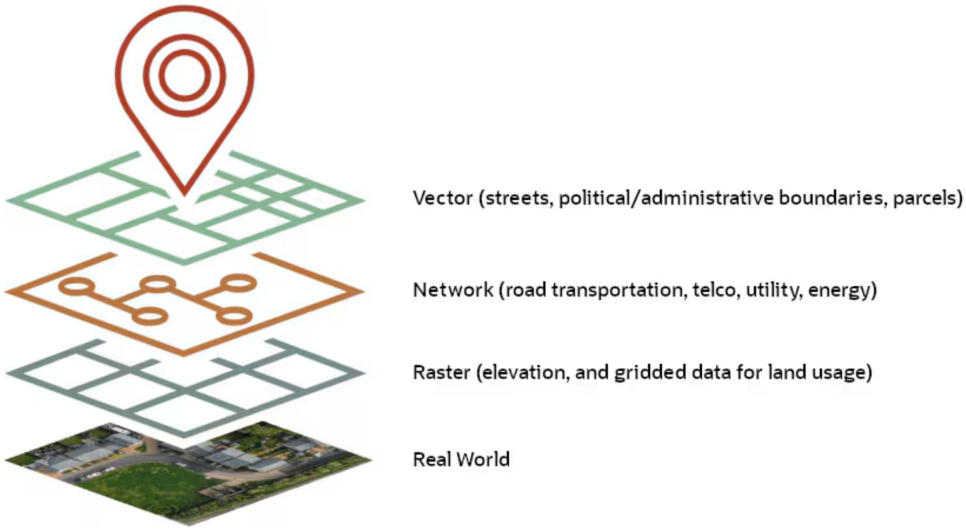
Obr. 3.5: Ukážka azimutovej projekcie. Prevzaté z [48]

Okrem spomenutých základných typov projekcie existujú aj projekcie na iné telesá prípadne hybridné, spájajúce metódy a telesá iných zobrazení. Je podstatné vybrať správnu projekciu na základe miesta, veľkosti oblasti a úlohy, ktorú má mapa riešiť.

3.2.2 Digitálne mapy

Existujú dva základné princípy uchovávania máp v elektronickej forme. Prvým spôsobom je ako rastrovú grafiku, kde sú definované farby jednotlivých pixlov a mapa je prakticky obrázok so statickou grafikou. Problém nastáva pri menení úrovne priblíženia, kedy na mape môže byť príliš veľa detailov, alebo naopak, detaily sú rozmazané. To je možné vyriešiť inými zdrojmi mapy pre rôzne úrovne priblíženia. Tento princíp je náročnejší na uchovávanie, ale rýchlejší na vykreslenie u používateľa. V dnešnej dobe ide skôr do úzadia. Druhým spôsobom je vektorová grafika, kde sú objekty a oblasti na mape definované pomocou bodov, čiar, mnohouholníkov. Takéto vykreslenie má výhodu ostrosti a presnosti pri ľubovoľnom priblížení, avšak grafika väčšinou vyzerá jednoduchšie. [9]

Je taktiež možné kombinovať obidva typy spolu. Ako podklad použiť napríklad satelitné snímky v rastrovej grafike a tento podklad prekryť objektmi definovanými vo vektorovej grafike – ulice, názvy, vrstevnice a pod. Je tak možné maximalizovať výhody a minimalizovať nevýhody obidvoch typov. Väčšina digitálnych máp obsahuje niekoľko vrstiev s rôznymi druhmi informácií v rastrovej, alebo vektorovej grafike. Poradie vrstiev určuje prekrývanie objektov na mape (napr. ulica prekrýva vrstevnicu). Jedná sa o veľké množstvo rôznorodých informácií, ktoré je potrebné uchovávať v databáze a s ktorými je potrebné efektívne pracovať na základe ich definovanej polohy. Toto zabezpečujú špecializované geografické databázy (angl. Geospatial databases). Tie zabezpečujú efektívny výber údajov pre rôzne vrstvy na základe polohy, vzdialenosti, smeru a pod., a taktiež podporujú špecializované dopyty na databázu na základe geografických parametrov. [31]

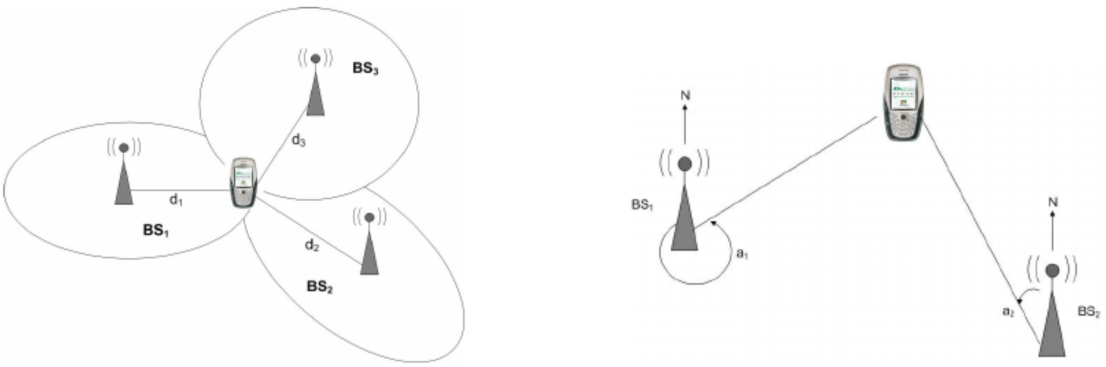


Obr. 3.6: Ukážka rôznych vrstiev mapy. Prevzaté z [31]

3.3 Navigácia

Navigácia má za cieľ vytvorenie trasy z jedného bodu – väčšinou aktuálnej polohy, do druhého a sledovanie trasy medzi týmito bodmi. Pre navigáciu je podstatné poznať aktuálnu polohu, polohu cieľa a trasu medzi nimi. Podľa typu navigácie je vhodné aby poznala aj prípadné prekážky na trase a v jej blízkosti a vhodným spôsobom na ne upozorňovala.

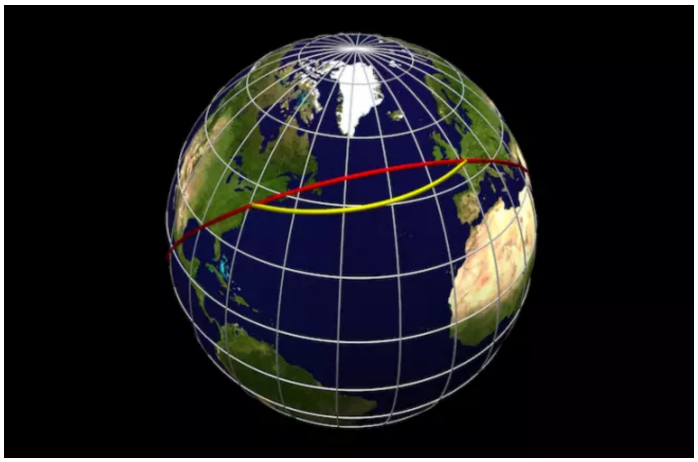
Získanie aktuálnej polohy zariadenia prebieha na základe informácií z okolitých zariadení so známou polohou – prístupový bod WLAN, stanica GSM, GPS družica a pod. Zariadenie osloví dostupné zariadenia vybraného typu, alebo typov v okolí a na základe smeru signálu, dĺžky odozvy, prípadne sily signálu vypočíta aktuálnu pozíciu. Podľa vybraného algoritmu hľadá priesečníky priamok (smer signálu), kružníc (dĺžka odozvy), kruhov (sila signálu). Zariadenie môže použiť niektorú metódu na získanie približnej polohy a inú na jej upresnenie. Proces získavania polohy sa volá triangulácia. [37]



Obr. 3.7: Ukážka triangulácie za pomoci dĺžky odozvy (vľavo) a smeru signálu (vpravo). Prevzaté z [37]

Vytýčenie trasy je možné rozdeliť do dvoch skupín. Trasa vzdušnou čiarou je najjednoduchší princíp, kedy je podstatné len smerovanie. Na krátke vzdialenosti sa vypočíta uhol a vzdialenosť medzi aktuálnou pozíciou a cieľom používajúc Mercatorovú projekciu (viď. kapitola 3.2.1 časť o valcovej projekcii), keďže azimuty sú na nej zachované (vykreslené ako rovné čiary). Priamku s daným azimutom prenesenú na mapu, alebo na guľu vo forme krivky nazývame loxodróma. Čím je však vzdialenosť medzi dvomi bodmi väčšia, tým sa vzdialenosť podľa azimutu viacej odchyľuje od reálnej najkratšej vzdialenosti, to neplatí pre špecifické prípady vysvetlené v projekcii. V prípade väčších vzdialeností je potrebné vypočítať tzv. ortodrómu. Ortodróma je najkratšia vzdialenosť medzi dvomi bodmi na povrchu gule, je to časť kružnice prechádzajúcej obidvoma bodmi so stredom zhodným so stredom gule. Ortodróma sa na mape vyrobenej podľa Mercatorovej projekcie javí ako časť sínusoidy. Rozdiel medzi ortodrómou a loxodrómou je možné vidieť na obrázku 3.8. Tento prístup je vhodný najmä pre lodnú a leteckú dopravu a pre orientáciu v teréne mimo turistických chodníkov. [12]

Druhou skupinou je navigácia po mape využívajúca dopravné siete. Táto skupina je značne zložitejšia a náročnejšia na realizáciu. Je vhodná pre navigáciu v mestách, cestnú dopravu, riečnu dopravu. Pre navigáciu je potrebné zostrojiť vážený graf s možnými cestami do cieľa. Následne sa pomocou vybraného algoritmu vyberie najkratšia trasa do cieľa a tá sa premietne podľa grafu naspäť na mapu. Existuje množstvo trasovacích algoritmov, pričom každý má svoje výhody a nevýhody. Medzi najznámejšie patrí Dijkstrov algoritmus, A-star algoritmus, prehľadávanie grafu do šírky, do hĺbky. Niektoré algoritmy skončia v momente, kedy nájdu cieľový vrchol, iné musia prejsť všetkými uzlami, aby mohli vyhodnotiť najkratšiu cestu. [39]



Obr. 3.8: Ukážka rozdielu loxodrómy (žltá) a ortodrómy (červená). Prevzaté z [12]

Kapitola 4

Analýza systému a technológií

Jedným z prvých krokov pri tvorbe informačného systému je analýza požiadaviek na neho a prehľad dostupných technológií pre danú problematiku. Táto kapitola predstaví požiadavky na systém získané konzultáciou so slovenskými vodnými skautmi a tie prevediem na formu diagramu prípadov použitia. Na záver uvediem prehľad technológií pre všetky časti systému.

4.1 Požiadavky na systém

Systém má slúžiť najmä pre organizátorov pretekov plachetníc. Tí majú mať možnosť vytvárať preteky, spravovať svoje preteky, určovať trasu. Ďalej majú mať možnosť prijímať a zamietať posádky pretekárov do svojich pretekov a nahrávať výsledky pre dané posádky. Ďalšími používateľmi sú pretekári. Pretekári majú možnosť registrovať svoje lode, spravovať ich a prihlasovať ich na pretek. Taktiež sa môžu nahlasovať ako členovia posádky do iných lodí, pričom schvaľuje ich členstvo majiteľ lode.

Pretekár

Pod pretekárom sa rozumie ľubovoľný registrovaný používateľ systému. Užívateľ sa najprv registruje, pričom na registráciu je potrebné používateľské meno a heslo, pod ktorými sa bude neskôr prihlasovať. Po schválení registrácie administrátorom, pretekár môže pristupovať do systému. Môže pridať svoje skutočné meno a e-mail a meniť už zadané informácie.

Pretekár môže registrovať svoje lode na základe mena a prípadne registrácie, ak ňou loď disponuje. Pretekár môže bez obmedzení meniť tieto informácie o svojich lodiach.

Organizátor

Organizátorom sa stáva ľubovoľný používateľ, ktorého administrátor určí po dohode. Organizátor môže vytvárať preteky, upravovať informácie o pretekoch. Informácie o pretekoch sú v rozsahu:

- názov
- miesto konania ako slovný údaj
- dátum konania
- dátum ukončenia registrácie

- krátky popis

Organizátor môže určiť svoj pretek ako súkromný, alebo verejný, čím určí, či je potrebné schvaľovať posádky pri registrácii na pretek. Po skončení pretekov môže organizátor nahrať výsledky jednotlivým posádkam v rozsahu miesto a čas. Zároveň môže vykonávať všetky činnosti ako pretekár, vrátane registrácie lodí.

Administrátor

Posledným aktérom systému je administrátor. Administrátor má za úlohu pomôcť spravovať údaje ostatých používateľov, lode, preteky a vykonávať schvaľovaco – zamietacie akcie vysvetlené neskôr v kapitole 4.1 o prihlasovaní na pretek. Administrátor môže spravovať role ostatných užívateľov vrátane seba. Administrátor schvaľuje nové registrácie udelením role pretekára.

Prihlasovanie na preteky

Prihlasovanie na preteky prebieha na dvoch úrovniach. Na úrovni pretekárov a na úrovni posádok. Prihlásenie je možné najneskôr v deň ukončenia registrácie, ak nie je zadaný, tak najneskôr v deň konania pretekov.

Na úrovni posádok môže ľubovoľný pretekár registrovať ľubovoľné množstvo svojich lodí na pretek. Jedna loď môže byť v jedných pretekoch iba jeden-krát. Lode sa zároveň stávajú aj identifikátorom posádok. Ak je pretek verejný, registráciu nie je potrebné schvaľovať. Ak naopak je pretek označený ako súkromný, musí byť posádka dodatočne schválená organizátorom daných pretekov, alebo administrátorom. Organizátor daných pretekov, administrátor a majiteľ lode na ktorú je posádka registrovaná, môžu hocikedy účasť posádky v pretekoch zrušiť.

Na úrovni pretekárov sa prihlasujú konkrétny pretekári do vytvorených posádok ako ich členovia. Ich členstvo v posádke schvaľuje majiteľ lode, alebo administrátor. Pretekár môže svoje členstvo v posádke zrušiť, čím zruší aj svoju registráciu na preteky. Jeho členstvo môže zrušiť aj majiteľ lode a organizátor. Pretekár môže byť registrovaný iba v jednej posádke v jedných pretekoch.

Trať

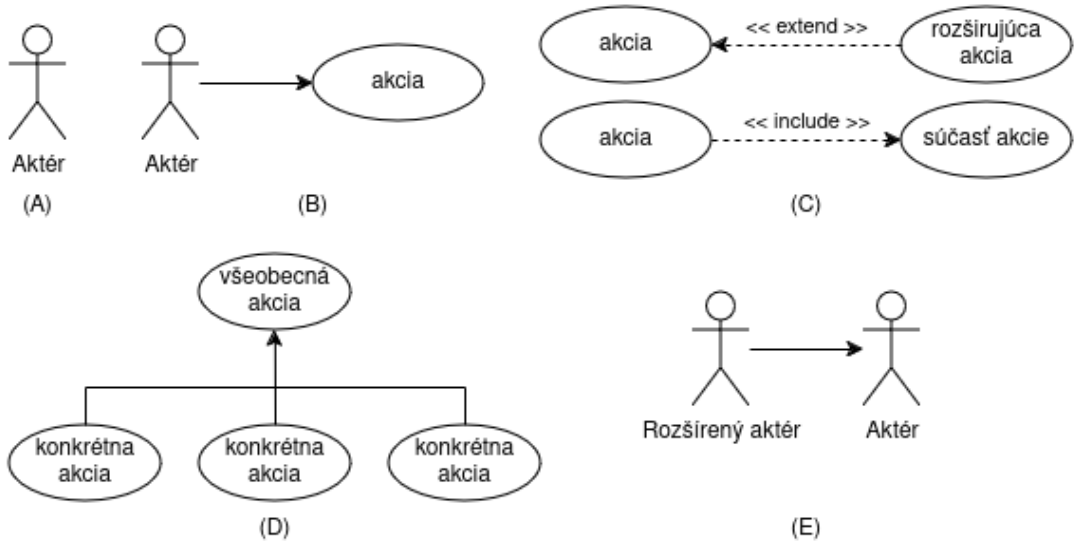
Slovenskí vodní skauti aktuálne používajú pre preteky plachetníc jedinú trať vyznačenú bójami. Formát trate je karuselový trojuholník, známy aj ako olympijský trojuholník (pozri kapitolu 3.1). Pretek môže ale nemusí mať vopred určenú trať tohoto typu. Ak ju obsahuje, pretekár, organizátor aj administrátor si trať môžu zobraziť na mape. Pri zobrazení je mapa zameraná na stred trate s vhodnou úrovňou priblíženia.

Systém umožňuje organizátorovi daných pretekov, alebo administrátorovi pridať trať ku pretekom. Po zvolení možnosti umiestnenia trate, môže používateľ zadať azimut, z ktorého aktuálne fúka vietor, prípadne sa predpokladá, že vietor bude fúkať v čase pretekov. Systém automaticky natočí predlohu trate podľa zadaného smeru vetra. Pri pokladaní predlohy na mapu, systém ukazuje aktuálnu vzdialenosť medzi jednotlivými bójami trate. Organizátor, alebo administrátor môžu rovnakým spôsobom meniť a premiestňovať trať.

Administrátor a organizátor pretekov môžu spustiť navigáciu podľa azimutu (pozri kapitolu 3.3). Navigácia po získaní aktuálnej pozície zobrazí pozíciu zariadenia na mape, jeho smerovanie a vhodným spôsobom trasu a vzdialenosť ku jednotlivým bójam. Pohľad mapy je zameraný na aktuálnu polohu zariadenia.

4.2 Diagram prípadov použitia

Diagram prípadov použitia, po anglicky use-case diagram, je podstatným nástrojom analýzy systému. Správne vypracovaný diagram reprezentuje funkčné požiadavky na systém. Pod funkčnými požiadavkami rozumieme všetky možné akcie, ktoré je možné vykonávať, alebo sa automaticky vykonávajú vrámci systému. Diagram taktiež identifikuje všetky typy aktérov (používateľov) vystupujúcich v systéme. Aktérom jasne definuje právomoci a prístup ku konkrétnym akciám. Diagram má grafickú formu, takže je možné ho použiť ako prehľadnú predlohu pri následnej implementácii systému. Nedefinuje priamo detaily implementácie ani vzhľad systému, preto slúži skôr ako osnova pre potrebné metódy rozhrania medzi serverovou časťou a používateľskou časťou systému. [44]



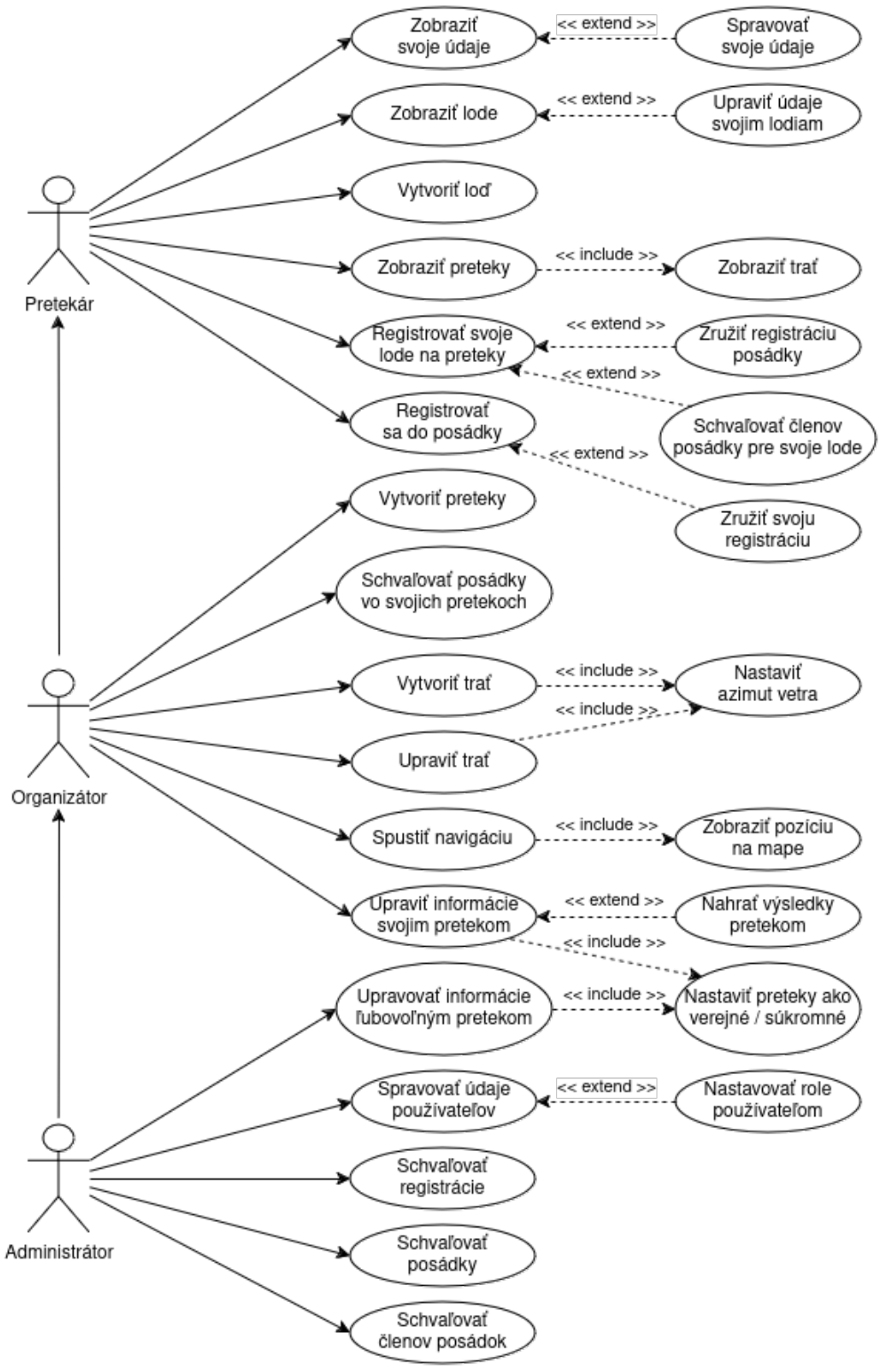
Obr. 4.1: Prvky diagramu prípadov použitia. (A) – aktér; (B) – možná akcia; (C) – rozšírenie (hore) a súčasť (dole) akcie; (D) – zovšeobecnenie akcie; (E) – rozšírenie aktéra

Diagram používa jazyk UML – Unified Modeling Language. Jazyk bol vyvinutý špeciálne na modelovanie a vizualizáciu systémov a ich súčastí. Obsahuje znaky a symboly vyjadrujúce rôzne závislosti a vzťahy rôznych prvkov systému. Pre diagram prípadov použitia sa používajú nasledovné z nich (pozri obrázok 4.1):

- *aktér* – označuje typ používateľa v systéme. Značí sa ako panáčik s názvom.
- *možná akcia* – definuje akciu, ktorú môže aktér vykonávať. Značí sa oválom s názvom akcie, ku ktorému smeruje šípka od daného aktéra.
- *rozšírenie akcie* – definuje akciu, ktorú aktér môže, ale nemusí vykonať spolu s akciou, ktorú rozširuje. Značí sa rovnako ako akcia, ale šípka je prerušovaná a smeruje od rozširujúcej akcii k akcii, ktorú rozširuje s doplnkovým textom « extend ».
- *súčasť akcie* – definuje akciu, ktorá sa vykonáva ako súčasť akcie. Značí sa ako akcia, ale šípka je prerušovaná a smeruje od akcie, ktorej je súčasťou, ku danej akcii s doplnkovým textom « include ».

- *zovšeobecnenie akcie* – definuje všeobecnú akciu, ktorú môže aktér vykonávať a súhrn konkrétnych, ktoré zahŕňa (napr. všeobecnou akciou je požičanie lode a konkrétnymi je požičanie motorového člnu, požičanie vodného skútra, požičanie plachetnice). Všeobecná akcia sa značí ako normálna akcia a konkrétne ako akcie, ktorých šípky sa zjednotia do jednej a smerujú ku všeobecnej.
- *rozšírenie aktéra* – rozširuje akcie aktéra o všetky akcie iného aktéra. Značí sa ako šípka od rozšíreného aktéra k aktérovi, ktorého akcie môže použiť.

Získané požiadavky som spracoval a na ich základe som vytvoril diagram prípadov použitia. Vytvorený diagram je možné vidieť na obrázku 4.2. Na diagrame je možné vidieť troch aktérov a akcie, ktoré boli spomenuté v požiadavkách na systém (pozri kapitola 4.1). Pri zostavovaní diagramu som začal identifikáciou aktérov. Po ich identifikovaní, som znova postupne prešiel požiadavky a vypisoval som akcie v nich spomenuté. Následne som akcie priradil k aktérom a sebe navzájom (rozšírenia a súčasti akcií), a tým som definoval ich vzájomné vzťahy.



Obr. 4.2: Vytvorený diagram prípadov použitia na základe požiadaviek na systém.

4.3 Dostupné technológie

Táto kapitola sa zaoberá prehľadom dostupných technológií pre tvorbu informačného systému podľa zadaných požiadaviek. Okrem spomenutých technológií existuje ešte veľa iných, no kvôli rozsiahlosti sa kapitola venuje iba niektorým. Technológie sú rozdelené do kategórií podľa ich použitia.

4.3.1 Front-end technológie

Pojem front-end technológie zahŕňa technológie zabezpečujúce interakciu užívateľa so systémom. Technológie zabezpečujú dizajn, tlačidlá, menu, obrázky, atď. Zobrazujú informácie zo systému používateľovi, spracovávajú rôzne typy vstupov a reagujú na ne. Medzi základné a najbežnejšie jazyky front-end technológií patrí HTML – HyperText Markup Language, CSS – Cascading Style Sheets, JavaScript a TypeScript.

HTML je štandardný značkovací jazyk, ktorý používa rôzne značky na identifikáciu a definíciu obsahu. Pomocou neho je definovaný dokument, ktorý následne iné aplikácie (napr. webové prehliadače spracujú) spracujú a výsledok zobrazia. V HTML funguje vrstvenie značiek – väčšina značiek môže obsahovať ďalšie v sebe, a preto každá značka má svoju začínajúcu a ukončujúcu časť. Ukončujúca časť môže byť aj súčasťou začínajúcej, ak neobsahuje iné značky. Príklad značiek: *<html>* – otváracia časť, *</html>* – ukončujúca časť, *<html/>* – začínajúca časť spolu s ukončujúcou. Základná štruktúra dokumentu obsahu značku *html* obsahujúcu značky *head* a *body*. *html* označuje, že sa jedná o dokument definovaný v HTML a vyznačuje zároveň že všetko medzi značkami je súčasťou dokumentu. Všetko zahrnuté v značke *head* sú pomocné informácie pre správne zobrazenie dokumentu, aj keď samotný obsah sa priamo v spracovanom výsledku nezobrazuje. Môže tu byť napríklad pomenovanie záložky. Značka *body* ohraničuje priamo zobraziteľný obsah dokumentu. Napríklad môže obsahovať *<h1>IS Regatta</h1>*, čo značí v dokumente nadpis najvyššej úrovni s textom „IS Regatta“. [19]

CSS je jazyk zameraný na formátovanie dokumentov. Pomocou referencie typov objektov, tried, ich usporiadania, identifikácie definuje pravidlá zobrazovania pre jednotlivé skupiny. Referencia môže byť platná buď iba pre jeden dokument, pre viacero, alebo aj pre všetky dokumenty. Môže definovať pozíciu prvku, jeho farbu, pozadie, veľkosť, atď. CSS môže byť definované v osobitnom súbore, potom HTML dokument obsahu referenciu na tento súbor. Ďalšou možnosťou je pomocou značky *style* v značke *head*. Posledný spôsob je priamo v začínajúcej časti značky pomocou *style="..."*. Napríklad *<p style="position: absolute; top: 5px;»...</p>* umiestni paragraf (značka *p*) na vzdialenosť päť pixlov od horného kraja obrazovky, alebo rodičovskej značky, ak má nastavenú pozíciu ovplyvňujúcu umiestnenie. [49]

JavaScript je najpoužívanejší programovací jazyk pre front-end. Je to slabo typovaný na objektoch založený skriptovací jazyk slúžiaci na tvorbu dynamických stránok. Kód je priamo v HTML dokumente, alebo v osobitnom súbore, na ktorý je odkazované z HTML dokumentu. Kód je vykonaný pri načítaní stránky a nie je potrebný dodatočný kompilátor. [18]

TypeScript je na rozdiel od JavaScriptu objektovo-orientovaný jazyk, so silným typovaním. Jeho zdrojové súbory sú v samostatných súboroch (nie sú priamo v HTML súboroch) a pre ich vykonanie je potrebný kompilátor, ktorý ho spracuje na JavaScript kód. TypeScript má vďaka silnému typovaniu lepšiu podporu už počas vývoja a veľa chýb sa môže prejaviť už pri písaní kódu a kompilovaní a nie až priamo za behu, ako je tomu pri JavaScripte. [18]

Frameworky

Medzi najznámejšie a najpopulárnejšie front-end frameworky patrí React, Angular a Vue.js. Všetky tri frameworky nie sú úplne nové, a preto sú aj v povedomí ľudí a majú svoje komunity. Pri robení prehľadu som hľadal najmä tie, ktoré majú rozvinutú podporu. Porovnania v tejto časti sú podľa [10].

React[1] bol vytvorený aby riešil problémy s udržiavaním Facebooku pri neustálom pridávaní nových prvkov a funkcií. Z toho vyplýva aj jeho výnimočná vhodnosť pre jednostránkové aplikácie s veľkým počtom používateľov. Jeho výhodou je znovu použiteľnosť napísaných komponentov a dostupnosť vývojárskych nástrojov. React používa JavaScript. Keďže sa framework neustále vyvíja, jeho dokumentácia nie je v ideálnom stave. Je náročnejší na naučenie pre nových programátorov. Framework používa napríklad Netflix, DropBox, Pinterest.

Angular[2] je najrozšírenejší framework používajúci výhradne TypeScript. Bol vyvinutý Googlom. Angular obsahuje veľké množstvo už implementovaných prvkov, a tak znižuje náročnosť na množstvo kódu. Obsahuje automatickú detekciu zmien ako zo zobrazenia na dáta (model), tak aj v opačnom smere. Má dobrú podporu pre znovu používanie komponentov na rôznych miestach. Dokumentácia je síce rozsiahla, ale často zmätočná, to však vyrovnáva veľká komunita s množstvom vlastného materiálu a fór. Angular je náročnejší na naučenie najmä kvôli svojej komplexnosti. Framework používa napríklad BMW, Xbox, Forbes.

Vue.js[3] využíva JavaScript, ale podporuje aj TypeScript. Je značne jednoduchší oproti predchádzajúcim dvom a je vhodný na široké spektrum typov aplikácií. Vďaka tomu je jednoduchý na naučenie a napomáha tomu aj dobrá dokumentácia. Komunita je značne menšia a veľa pluginov je písaných v čínštine, čo naopak môže byť problém pri učení a práci s ním. Framework používa napríklad 9gag, Reuters a Xiaomi.

4.3.2 Back-end technológie

Pojem back-end označuje časť aplikácie, ktorú používateľ nevidí a nemá k nej prístup. Back-end zabezpečuje fungovanie aplikácie, logiku spracovania a ukladania dát, databázové systémy, atď. Front-end spracuje vstup používateľa a pre informácie komunikuje práve s back-endom, ktorý požiadavku spracuje, vykoná potrebné kroky a vráti spracované dáta ako odpoveď, ktorú môže front-end zobraziť v reakcii na vstup. Často sa označuje aj ako jadro aplikácie.

C# je silne typovaný objektovo orientovaný programovací jazyk. Syntax má podobnú ako C, C++, JavaScript a Java. Vďaka tejto podobnosti sa ho ľahko naučia programátori, ktorí niektorý z týchto jazykov ovládajú. C# nie je obmedzený na jednu, ale môže bežať na viacerých platformách, no používa sa prevažne na Windows aplikácie. Bol vytvorený firmou Microsoft, ktorá ho dodnes zastrešuje. Vďaka podpore veľkej spoločnosti má rozsiahlu dokumentáciu a komunitu. [46]

PHP[4] znamená „PHP: Hypertext Preprocessor“, a teda sa jedná o rekurzívnu skratku. PHP je skriptovací jazyk, prevažne používaný na tvorbu internetových stránok a aplikácií. Je možné používať ho priamo v definíciách HTML dokumentu, podobne ako JavaScrpit, no na rozdiel od neho, je vykonávaný na strane servera a nie internetovým prehliadačom

[1] React – dostupné na https://react.dev/
[2] Angular – dostupné na https://angular.io/
[3] Vue.js – dostupné na https://vuejs.org/
[4] PHP – dostupné na https://www.php.net/

u používateľa. Front-end zobrazuje už spracovaný HTML dokument, ktorý obsahuje výstupy z PHP, avšak priamo kód PHP nemá k dispozícii. [15]

Java[5] je objektovo orientovaný jazyk so širokými možnosťami použitia. Je vhodná aj na rýchle spracovanie rozsiahleho množstva dát, pre viac platformové aplikácie a je jedným z najpoužívanejších jazykov pre tvorbu mobilných aplikácií. Obsahuje veľké množstvo zabudovanej funkcionality a nie je preto potrebné veľa funkcií písať od nuly. Java má rozsiahlu dokumentáciu a komunitu a je preto jednoduchšia na naučenie pre začínajúcich programátorov aj napriek svojej komplexnosti. Java má jednu z najviac rozvinutých bezpečností medzi programovacími jazykmi, a tak sa značne znižuje riziko infekcie systému škodlivým softvérom. Java kombinuje prístupy kompilátorov a interpreterov vo virtuálnom stroji (JVM – Java Virtual Machine), v ktorom sa vykonáva kód. [3]

Frameworky

Medzi back-end frameworkami predstavím zástupcov pre spomenuté jazyky. Každý jazyk má množstvo iných, ktoré však nebudem rozoberať do podrobností.

.NET[6] je najznámejší a najpoužívanejší framework používajúci C#. Je taktiež spravovaný a udržiavaný Microsoftom. Vďaka tomu má rozsiahlu dokumentáciu a komunitu. Komunita vytvorila veľké množstvo voľne dostupných knižníc, ktoré môžu napomôcť vývoju. Rýchlo vykonáva dopyty na databázu a prípravu šablón pre front-end. Je vhodný najmä pre aplikácie fungujúce na Windows, ale je možné ho použiť aj na iných platformách. [27]

Laravel[7] je framework zameraný na tvorbu webových aplikácií. Používa architektúru MVC[8]. Je zároveň najpoužívanejším PHP frameworkom a je možné ho používať aj na tvorbu front-endu pomocou predlôh a technológie jedno-stránkových aplikácií. Projekt v Laraveli obsahuje .env súbor v ktorom je možné ľahko konfigurovať podstatné údaje závislé od prostredia (vývojárske, produkcia a pod.). Laravel používa pre databázu najmä SQLite, MySQL a PostgreSQL. Medzi ďalšie známe PHP frameworky patrí Symfony, CodeIgniter a CakePHP. [6, 21]

Spring[9] je najznámejší Java framework. Jeho súčasťou je množstvo projektov, ktoré je možné ale nepovinné použiť a každý zabezpečuje podporu pre nejakú funkcionalitu. Slúži na zastrešenie veľkého množstva funkcionality, aby programátor písal hlavne aplikačnú logiku. Podporuje použitie aj iných frameworkov, napríklad Hibernate pre komunikáciu s databázou. Súčasťou Springu je aj napríklad bezpečnosť a automatické formuláre pre prihlasovanie, odhlasovanie používateľov. Spring je možné použiť nielen pre webové aplikácie, ale aj pre množstvo iných ako napríklad samostatne stojace aplikácie. Všetky dostupné funkcionality je jednoduché ovládať pomocou anotácií a konfiguračných súborov. [43] Ďalšie frameworky sú napríklad Strutrs, Google Web Toolkit a Hibernate.

4.3.3 API

API – Application Programming Interface, definuje štruktúru, prípadne protokoly pre komunikáciu medzi dvomi časťami aplikácie. Pomáha napájať rôzne front-endy k back-endu, využívať cudzie služby vo vlastných aplikáciách, zjednodušuje používanie cudzieho kódu (napr. komunitné knižnice, moduly zabezpečujúce nejakú funkcionalitu). Je zároveň pod-

[5] Java – dostupné na https://www.java.com/
[6] .NET – dostupné na https://dotnet.microsoft.com/
[7] Laravel – dostupné na https://laravel.com/
[8] MVC – Model Viev Controller, pozri https://en.wikipedia.org/wiki/Model-view-controller/
[9] Spring – dostupné na https://spring.io/

statným nástrojom pri vyvíjaní rôznych častí aplikácie rôznymi tímami. Dnes najpoužívanejšou skupinou sú tzv. REST API (Representational State Transfer API) kvôli ich flexibilite a jednoduchej syntaxi. [2]

4.3.4 Databázy

Databáza je softvérový systém pre ukladanie, organizáciu, správu a získavanie dát. Databázy majú rôzne fungovanie a zameranie, vrátane takých, ktoré majú slúžiť pre všeobecné použitie a teda nemajú konkrétne zameranie. Databázy delíme podľa ich štruktúry na hierarchické, relačné, nerelačné a objektovo orientované. Hierarchické databázy sú najstaršie z vymenovaných. Sú štrukturované ako informatické stromy. Sú rýchle na vyhľadávanie, avšak navigácia v nich je o to komplexnejšia kvôli štruktúre rodič – potomok. Relačné databázy využívajú jazyk SQL (Structured Query Language) pre operácie v nich. Dáta uchovávajú v tabuľkách, pričom záznamy môžu obsahovať referenciu na dáta v iných tabuľkách vo forme cudzieho kľúča. Objektovo orientované databázy ukladajú priamo objekty a zachovávajú referencie medzi nimi. Odrážajú väčšinou priamo štruktúru back-endu a tried objektov v nich definovaných. Nerelačné databázy zastrešujú všetky ostatné databázy, kde špecifické požiadavky neumožňujú použiť, alebo by bolo nepraktické použiť niektorú z predchádzajúcich skupín. Patria sem napríklad aj geografické databázy spomenuté v kapitole 3.2.2.

4.3.5 Mapy

V tejto kapitole spomeniem tri z najznámejších frameworkov pre prácu s mapami, ich výhody a nevýhody.

Leaflet[10] je najpoužívanejšou open-source knižnicou pre používanie elektronických máp. Pri tvorbe bol použitý JavaScript a v ňom sa aj používajú. Knižnica je malá, ale obsahuje funkcionalitu pre väčšinu potrieb aplikácií pracujúcich s mapou. Snaží sa zameriavať na jednoduchosť používania a efektivitu fungovania. [1]

OpenLayers[11] sú rozsiahlou knižnicou pre prácu s mapami. Knižnica je taktiež open-source. Obsahuje množstvo funkcionality a podporujú rastrové aj vektorové mapy (pozri kapitolu 3.2.2). Je ich možné používať v JavaScripte a aj v TypeScripte. Zameriavajú sa najmä na výkonnosť, a tak zvládajú aj náročnejšie operácie. Je možné použiť ľubovoľné mapové podklady, vrátane vlastných. Majú rozsiahlu dokumentáciu a veľkú knižnicu príkladov použitia s ukážkami a zdrojovými kódmi, takže je jednoduché sa naučiť pracovať s nimi aj od nuly. [33]

Google Maps Platform[12] sú komerčnou platformou pre prácu s mapami. Umožňujú používať navigáciu, trasovanie, aktuálnu dopravnú situáciu, rôzne mapové podklady, referencie na miesta a veľa ďalších. Cena sa odvíja od množstva použití, prvých 200$ mesačne je zdarma. Vďaka spravovaniu veľkou firmou má platforma dobrú dokumentáciu a množstvo použiteľných projektov vytvorených komunitou. [14]

[10] Leaflet – dostupné na https://leafletjs.com/
[11] OpenLayers – dostupné na https://openlayers.org/
[12] Google Maps Platform – dostupné na https://developers.google.com/maps/

Kapitola 5

Návrh systému

Návrh systému začína výberom vhodnej architektúry. Nasleduje výber jednotlivých technológií pre dané časti architektúry. Po výbere technológií, vrátane technológie pre ukladanie dát je potrebné ešte navrhnúť štruktúru ukladaných dát. Poslednou časťou návrhu, je návrh používateľského rozhrania.

5.1 Výber architektúry

Pre svoj systém som zvolil trojvrstvovú architektúru (pozri kapitolu 2.2.2). Prezentačná vrstva bude relatívne úzka a bude mať minimum implementovanej logiky v sebe. Výnimkou bude časť vykonávajúca navigáciu, ktorá bude implementovať celkovú logiku umiestňovania trate a navigácie ku bójam, aby som minimalizoval zaťaženie back-endu a sieťovej komunikácie medzi back-endom a front-endom. Takto bude zaťaženie ponechané na zariadení klienta. Aplikačnú vrstva sa bude držať logického členenia na prezentačnú, biznis a dátovú vrstvu. Komunikácia medzi klientskou a aplikačnou vrstvou bude definovaná pomocou API (pozri kapitolu 4.3.3). Dátová vrstva bude relačný databázový server obsahujúci potrebné tabuľky.

5.2 Výber technológií

Táto kapitola postupne prejde výberom technológií pre časti trojvrstvovej architektúry navrhnutej v predchádzajúcej kapitole. Ku každej vybranej technológii bude uvedené aj krátke zdôvodnenie jej výberu. Pre referencie na jednotlivé technológie, pozri kapitolu 4.3.

Pre tvorbu front-endu som vybral framework Angular, používajúci jazyky HTML, CSS a TypeScript. Zvolil som ho najmä pre množstvo materiálu pomáhajúceho s vývojom, jeho silné typovanie a odhaľovanie chýb už počas vývoja. Keďže rátam s rozširovaním svojho systému v budúcnosti, zohľadnil som aj jeho podporu pri znovu používaní rôznych komponentov.

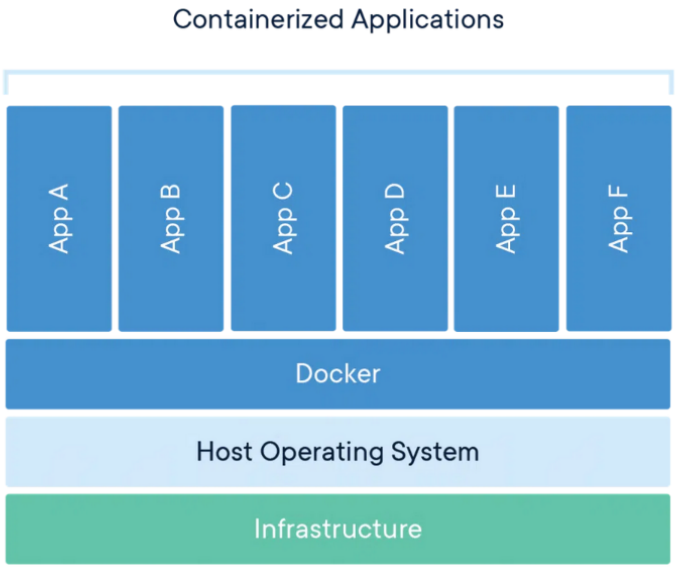
Ďalej som zvolil knižnicu OpenLayers najmä pre jej výkon a pre rozsiahlu funkcionalitu. Výkon bol podstatným parametrom, keďže zaťaženie umiestňovania bójí a navigácia ku nim bude na zariadení používateľa. Ďalším dôvodom bola možnosť používať rôzne mapové podklady a jednoduchá integrácia s námornou a riečnou mapou. Námorná a riečna mapa obsahujú lodné trasy, majáky, vyhradené oblasti na vodných tokoch a plochách a používateľ musí tieto objekty zohľadňovať pri umiestňovaní trasy. Keďže som zvolil Angular,

bola pre mňa rozhodujúcou aj podpora TypeScriptu. Zavážilo aj množstvo materiálov s príkladmi a dokumentácia samotnej knižnice.

Pre aplikačnú vrstvu som zvolil framework Spring používajúci jazyk Java. Hlavným dôvodom bola existencia podpory pre webové aplikácie a možnosť pridávania modulov s funkcionalitou podľa potreby. Pre preklad som si vybral Maven[1]. Maven je softvér pre manažovanie projektu, získavanie závislostí a vytváranie spustiteľného programu. Je jedným z najznámejších nástrojov pre pomoc s vývojom Java aplikácií. Spring som si vybral aj pre veľa možností znovu používania kódu. Pre komunikáciu s databázou som zvoli framework Hibernate, keďže je podporovaný aj samotným Springom a do značnej miery zjednodušuje komunikáciu s databázou a dopyty na ňu.

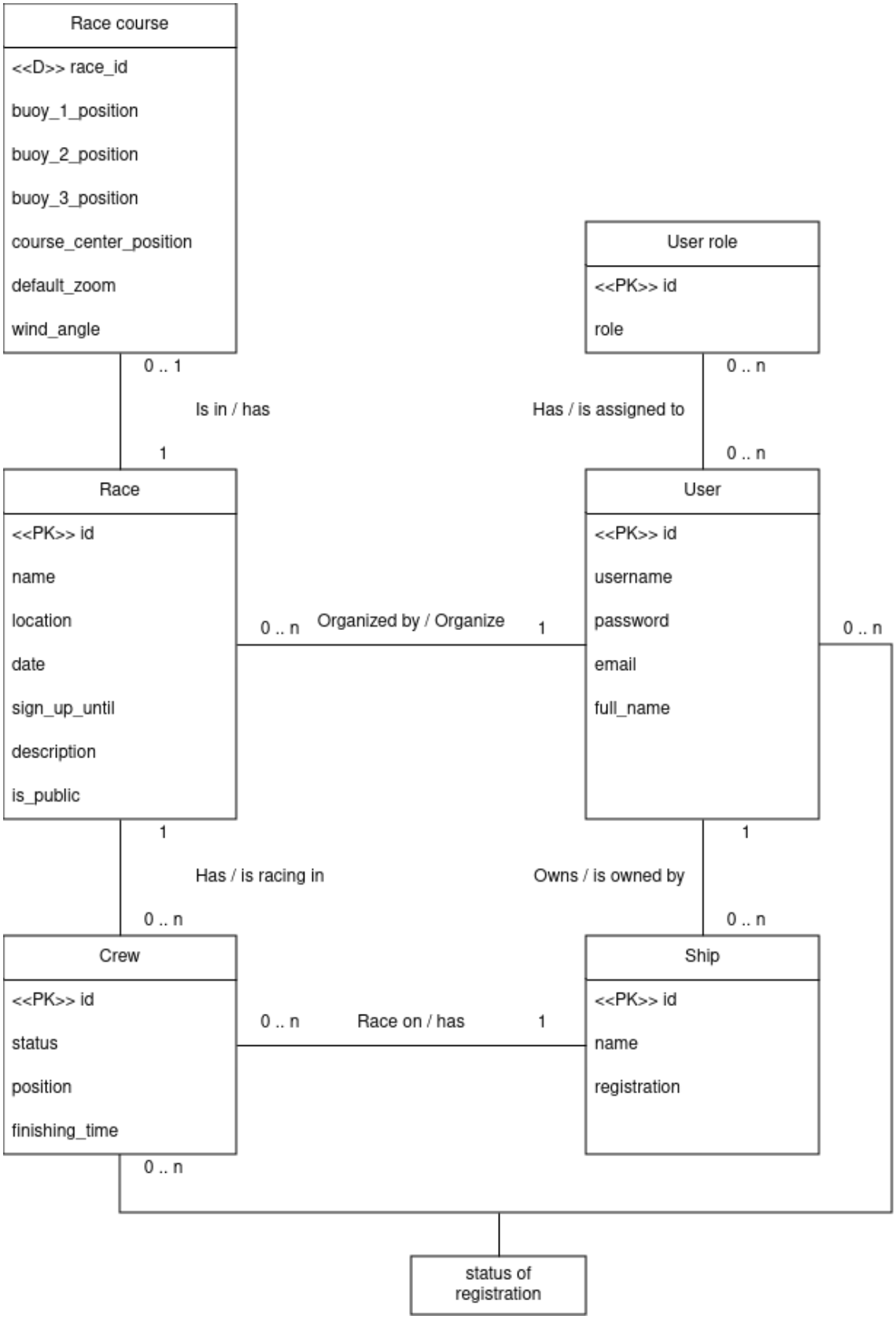
Pre definíciu komunikácie medzi klientskou a aplikačnou vrstvou som zvolil REST API, konkrétne jazyk OpenAPI[2]. Maven obsahuje aj modul na preklad OpenAPI definície do Java súborov vo forme rozhraní a tried, čo zjednodušuje implementáciu prezentačnej vrstvy v aplikačnej vrstve.

Pre databázovú vrstvu som zvolil MariaDB[3] server, najmä kvôli rozsiahlej dokumentácii s príkladmi a kvôli predchádzajúcim skúsenostiam. Databázový server som zaobalil do kontajneru pomocou Dockeru[4]. Kontajner je štandardizovaná jednotka softvéru obsahujúca všetky závislosti potrebné pre jeho fungovanie. Kontajner beží pomocou virtualizácie na systémovej úrovni. Poskytuje tak výbornú prenositeľnosť softvéru bežiaceho v ňom a ďalšiu vrstvu ochrany pred útokmi a škodlivým softvérom.



Obr. 5.1: Ukážka architektúry Dockeru a kontajnerov. Prevzaté z [11]

[1]Maven – dostupné na https://maven.apache.org/
[2]OpenAPI – dostupné na https://www.openapis.org/
[3]MariaDB – dostupné na https://mariadb.org/
[4]Docker – dostupné na https://docs.docker.com/



Obr. 5.2: Vytvorený ER diagram systému.

5.3 ER diagram

ER diagram (Entity Relation diagram) je diagram entít a vzťahov medzi nimi. Je v jazyku UML, rovnako ako diagram prípadov použitia (pozri kapitolu 4.2). Používa sa pre návrhy relačných databáz a slúži ako ich predloha. Každá entita na grafe obsahuje aj prvky, ktoré sú požadované. Vzťahy sú označené početnosťou, ktorá určuje aký je medzi nimi vzťah. Väzby sú typu many-to-one (viacero prvkov je obsiahnutých v jednom), many-to-many (viacero prvkov je vo viacerých prvkoch) a one-to-one (Jeden prvok je obsiahnutý v jednom). Pri väzbe one-to-one je potrebné zamyslieť sa, či sa nejedná vlastne o tú istú entitu. Väzbu many-to-many je potrebné mapovať v osobitnej tabuľke a môže mať taktiež svoje vlastnosti. [25]

V požiadavkách z kapitoly 4.1 som identifikoval entity, ktoré v systéme figurujú. Následne som identifikoval medzi ktorými entitami existujú väzby. Vzťahom som určil ich početnosť a pomenoval ich slovesami, ktoré ich vystihujú. Výsledný diagram je možné vidieť na obrázku 5.2.

Používateľ

Používateľ, v diagrame *User*, symbolizuje všetkých používateľov bez ohľadu na ich funkciu v systéme. V momente registrácie sa vytvorí nový záznam. Používateľ obsahuje unikátny identifikátor, používateľské meno, aktuálne heslo, e-mailovú adresu a celé civilné meno.

Používateľská rola

Používateľská rola, v diagrame *User role*, symbolizuje číselník obsahujúci všetky možné role. Každá rola má unikátny identifikátor a slovné označenie. Každý používateľ môže vlastniť ľubovoľný počet rolí a ľubovoľná rola môže byť udelená ľubovoľnému množstvu používateľov. V databáze je vzťah medzi rolami a používateľmi nutné spraviť pomocou doplnkovej tabuľky.

Pretek

Pretek, v diagrame *Race*, symbolizuje jednotlivé preteky plachetníc. Každé preteky obsahujú unikátny identifikátor, názov pretekov, miesto konania pretekov, dátum konania pretekov, dátum ukončenia registrácie, krátky popis a identifikátor, či sa jedná o verejné preteky. Preteky sú organizované vždy práve jedným používateľom – organizátorom. Používatelia môžu organizovať ľubovoľné množstvo pretekov. Toto je možné reprezentovať v databáze uložením identifikátoru používateľ ku daným pretekom (cudzí kľúč).

Trasa pretekov

Trasa pretekov, v diagrame *Race course*, symbolizuje uloženú trasu ku pretekom. Identifikuje sa preto identifikátorom pretekov (cudzím kľúčom), namiesto svojho vlastného. Okrem identifikátora obsahuje pozície všetkých troch bóji, pozíciu stredu trate, ktorá bude v strede zobrazenej mapy pri zobrazení trate, úroveň priblíženia, ktorá sa taktiež má nastaviť zobrazeniu a uhol z ktorého má fúkať vietor v čase pretekov pre natočenie trate.

Loď

Loď, v diagrame *Ship*, symbolizuje lode, ktoré môže vlastniť ľubovoľný používateľ. Loď nemôže mať viacerých vlastníkov, a ani nemôže byť bez vlastníka, má vždy práve jedného. Značí sa to rovnako ako pri pretekoch, cudzím kľúčom – identifikátor používateľa uložíme ku danej lodi. Loď má unikátny identifikátor, názov a registračnú značku.

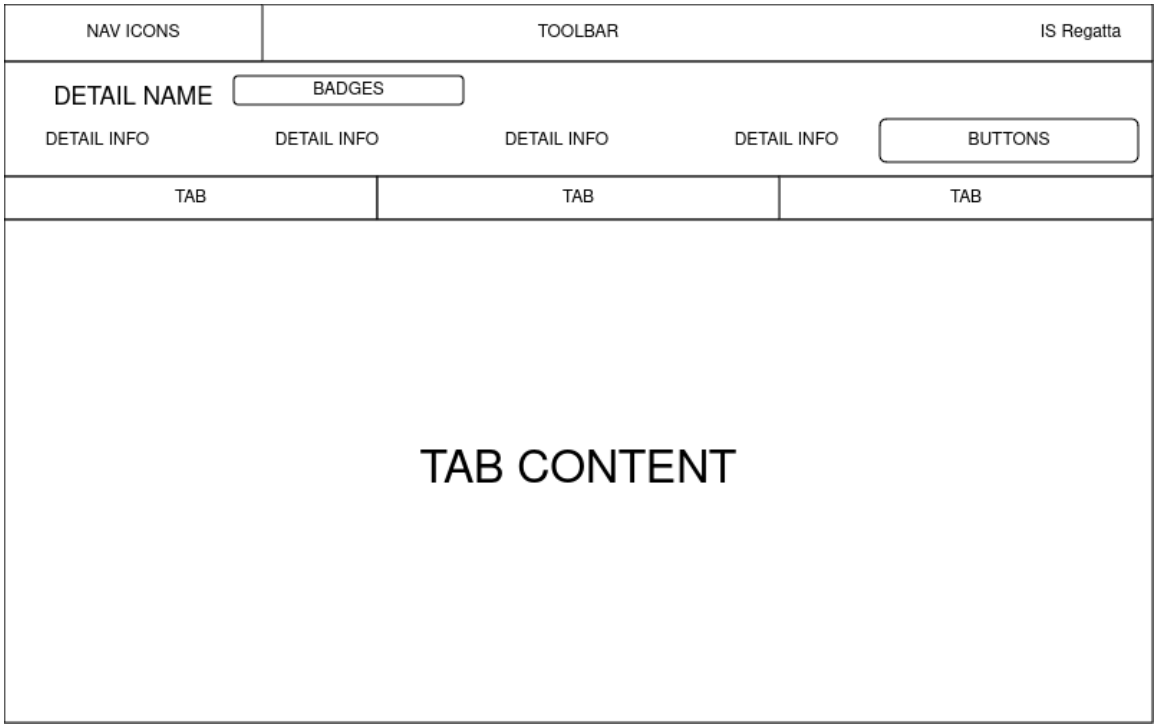
Posádka

Posádka, v diagrame *Crew*, symbolizuje posádky, ktoré sa zúčastňujú pretekov. Posádka obsahuje svoj unikátny identifikátor, stav registrácie na preteky a výsledky z pretekov v rozsahu pozícia a čas. Posádka preteká na práve jednej lodi a na práve jedných pretekoch. Každá loď môže mať ľubovoľný počet posádok a v každých pretekoch môže byť ľubovoľný počet posádok. Reprezentované to bude uložením cudzích kľúčov lode a pretekov v posádke. Ďalej každá posádka má ľubovoľný počet členov – používateľov a každý používateľ môže byť v ľubovoľnom počte posádok. Tento vzťah vyjadríme pomocnou tabuľkou v databáze s cudzími kľúčmi používateľov a posádok. Taktiež do pomocnej tabuľky pridáme stav registrácie do posádky, v diagrame *status of registration*.

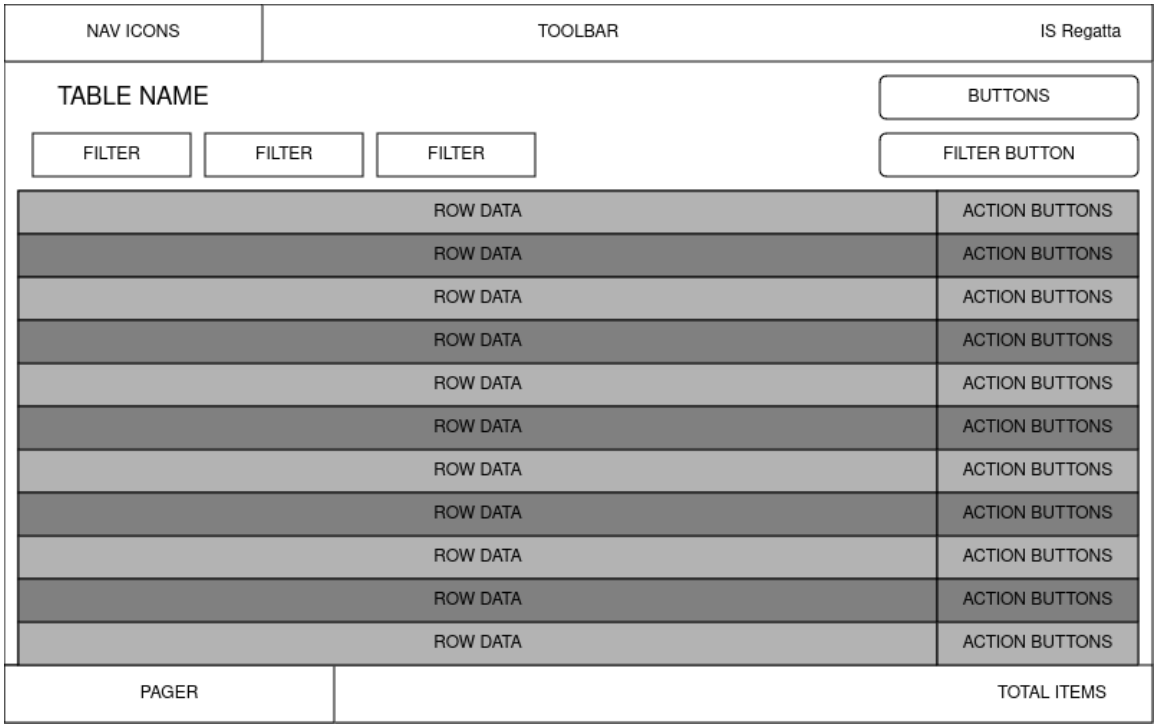
5.4 Návrh UI

Posledným krokom návrhu informačného systému ju návrh stránok, ktoré sa budú zobrazovať používateľom. Tento krok sa nazýva návrh používateľského rozhrania, skrátene návrh UI (z anglického User Interface). Samotný návrh môže odhaliť chýbajúce definície funkcionality systému, prípadne odhaliť nové. Uľahčuje prácu vývojárom, aby sa mohli sústrediť na implementáciu a nemuseli vymýšľať dookola aké prvky majú kde byť a či niektoré nezabudli umiestniť na danú obrazovku.

Pri mojom návrhu UI som sa zameral na jednotný formát všetkých stránok a znovu použiteľnosť kódu. Snažil som sa pristupovať k rozloženiu uniformne, aby som pri implementácii po spravení formátu ho už iba napĺňal potrebnými informáciami. Vznikli mi dva typy obrazoviek. Obrazovky pre detail záznamu (pozri obrázok 5.3) a obrazovky, ktoré slúžia ako tabuľky pre zobrazovanie záznamov (pozri obrázok 5.4). V zobrazení záložky na obrazovke detailu môže byť aj tabuľka s rovnakým formátovaním ako je ukázané na obrázku 5.4. Formuláre budú fungovať ako vyskakujúce okná, okrem prihlasovacieho a registračného formuláru, ktoré budú mať samostatnú stránku.



Obr. 5.3: Návrh dizajnu UI pre zobrazenie detailu.



Obr. 5.4: Návrh dizajnu UI pre zobrazenie tabuliek.

Kapitola 6

Implementácia

Po navrhnutí systému nasleduje samotná implementácia. Zdrojové súbory sú rozdelené podľa častí architektúry popísanej v kapitole 5.1. Priečinok *fe-angular* obsahuje zdrojové súbory ku klientskej vrstve, priečínok *be-spring* obsahuje zdrojové súbory k aplikačnej vrstve, priečinok *rest-api* obsahuje definície komunikačného rozhrania medzi klientskou a aplikačnou vrstvou a posledný priečinok *docker* obsahuje definíciu kontajneru pre MariaDB databázu. Pri spomenutých priečinkoch je súbor pom.xml. Tento súbor používa Maven pre zostrojenie celého informačného systému.

6.1 REST API

Implementáciu som začal definíciou rozhrania. Aj keď sa mi nepodarilo definovať všetko na začiatku, poskytlo mi to dobrý začiatok a doplniť ďalšie informácie do API neskôr počas implementácie nebol problém. Vytvoril som súbor *rest-api/pom.xml*. Na začiatku súboru je jeho identifikácia vrátane odkazu na rodičovský artefakt. Nasleduje definícia závislostí. Následne sa pre plugin *openapi-generator-maven-plugin* definujú súbory, v ktorých sú definície rozhrania pomocou OpenAPI a z ktorých má vytvoriť rozhrania a triedy v Jave.

V priečinku *rest-api/src/main/resources* sú súbory s definíciou rozhrania. Každý súbor sa drží nasledovnej štruktúry:

```
openapi: 3.0.0
info:
  ...
servers:
  ...
paths:
  /example/url:
    method:
      ...
components:
  schemas:
    exampleDto:
      ...
```

Časť *info* tvorí hlavičku identifikujúcu rozhranie a *server* parametre rozhrania. V sekcii *paths* sú definované url a parametre dopytov a očakávané odpovede. Posledná sekcia *components* definuje DTO (Data Transfer Object) používané v rozhraní.

6.2 Back-end – Spring

Implementáciu serverovej časti som začal vytvorením Spring aplikácie pomocou nástroja na inicializáciu projektov – Spring Initializr[1]. Ako počiatočné závislosti som zvolil niektoré, ktoré poskytovali funkcionalitu web aplikácií, bezpečnosť, komunikáciu s databázou. To vytvorilo aj súbor *be-spring/pom.xml*, ktorý definuje závislosti a zabezpečuje preklad aplikácie Mavenom.

Pri implementácii serveru som postupoval zhora nadol. Pri tomto spôsobe začne implementácia od controllerov a postupne napĺňa ich funkcionalitu do hĺbky. Controllery som definoval pomocou rozhraní vygenerovaných REST API. Pre mapovanie tried použitých v Jave a DTO vygenerovaných z REST API medzi sebou, som použil funkcionalitu Mapstructu[2]. Ten umožňuje pomocou anotácií definovať mapovanie objektov medzi sebou a napríklad automaticky mapuje položky s rovnakým názvom.

Controllery využívajú servisy a perzistentné servisy, ktoré spolu zabezpečujú celú aplikačnú logiku. V mojej implementácii majú servisy na starosti biznis logiku a perzistentné servisy sa starajú o formátovanie dát získaných z databázy. Obidva typy servisov majú anotáciu *@Sevice* a *@Slf4j*. Prvá zmienená zabezpečuje aby aplikácia rozpoznala triedu ako servis a vytvárala jej inštancie tam kde to je potrebné. Druhá anotácia zabezpečuje podporu logovacích záznamov v metódach a je použitá aj pri controlleroch.

Perzistentné servisy používajú repozitáre pre komunikáciu s databázou. Repozitáre sa definujú pomocou rozhrania, ktoré rozširuje *JpaRepository*, prípadne *JpaSpecificationExecutor*. Hibernate vytvára implementujúce triedy, ktoré priamo komunikujú s databázou. To poskytuje abstrakciu zľahčujúcu prácu s databázou. Dopyty na databázu sa definujú názvom metódy, alebo v zložitejších prípadoch pomocou anotácie *@Query(value = "...")*.

Modely sú definované na základe ER diagramu (pozri kapitolu 5.2) pomocou anotácií *@Entity* a *@Table*. Použil som projekt Lombok[3] pre definovanie get, set metód a konštruktérov.

Priečinok *be-spring/src/main/java/com/xvanop01/isregatta/base/congig* obsahuje podstatné konfiguračné súbory pre bezpečnosť, nastavenie zdieľania zdrojov mimo aplikácie (CORS), MVC. Definuje sa tu prístup k stránkam a metódam, autorizácia a podobne. Konfiguračné súbory pre jednotlivé profily (rôzne prostredia) sú v priečinku *be-spring/config*. Definujú napríklad pripojenie na databázu a použitie flyway skriptu. Profily sa používajú podľa parametrov pri spustení.

Priečinok *be-spring/src/main/resources/db/mariadb/migration* obsahuje flyway skripty. Počas mojej implementácia som vytvoril iba jeden inicializačný skript obsahujúci všetky definície tabuliek a naplnenie číselníka rolí.

6.3 Front-end – Angular

Implementáciu front-endu som taktiež začal vygenerovaním kostry projektu. Pre generovanie Angular projektu sa používa príkaz v Termináli

```
$ npx -p @angular/cli ng new isregatta-angular
```

Začal som implementáciou základného rozloženia stránok podľa UI návrhu (pozri kapitolu 5.4). Pokračoval som tvorbou pomocných komponentov, ktoré sa používajú v ostat-

[1] Spring Initializr – dostupné na https://start.spring.io/
[2] Mapstruct – dostupné na https://mapstruct.org/
[3] Lombok – dostupné na https://projectlombok.org/

ných. Tieto komponenty sú v priečinku *fe-angular/src/app/core*. Ďalej som vytvoril súbor *fe-angular/dev/proxy.conf.json*, kde som definoval formát volaní na back-end pre správne rozlíšenie dopytov.

Moduly v Angulari som vytvoril zhodne k položkám navigačného menu. Modul *races* je najrozsiahlejším a obsahuje aj traťovú a navigačnú časť. Pri ich tvorbe som sa snažil dbať na znovu použiteľnosť všetkých komponentov.

Front-end je spustiteľný samostatne na osobitnom porte, kde sa prejavujú zmeny v kóde v reálnom čase. Aby nebola nutnosť spúšťať front-end osobitne, ale zároveň aby po nasadení, keď nemusí nutne bežať na tom istom zariadení ako back-end, som pomocou Mavenu vytvoril použiteľný front-end pre monolitnú aplikáciu a front-end tak beží zároveň s back-end na jeho porte.

Pri používaní aplikácie so samostatne spusteným Angularom som implementoval aplikáciu vo forme PWA (Progressive web app). PWA je stiahnuteľná aplikácia založená na webových technológiách distribuovaná cez webové prehliadače. Aplikácia využíva rovnakú definíciu front-endu ako internetové prehliadače a používa aj rovnako komunikáciu s back-endom.

6.4 Implementačné detaily

Táto kapitola sa zaoberá niektorými zaujímavými implementačnými detailami, ktorým sa chcem venovať do väčších detailov, lebo sa venujú netradičným veciam, alebo pristupujú k problému netradične. Niektoré detaily sú podporné a slúžia pre znovu použiteľnosť a zjednodušenie kódu.

6.4.1 Odpoveď podľa výnimky

Priečinok *be-spring/src/main/java/com/xvanop01/isregatta/base/exception*

Definoval som si vlastnú výnimku, ktorú používam ak back-end narazí na očakávaný problém.

```
public class HttpException extends Exception {
    private final HttpReturnCode errorCode;
    public HttpException(HttpReturnCode errorCode, String message) {
        super(message);
        this.errorCode = errorCode;
    }
    public HttpReturnCode getErrorCode() {
        return this.errorCode;
    }
}
```

HttpReturnCode je číselník obsahujúci číslo reprezentujúce kód HTTP chybovej odpovede. Ďalej som pre výnimku vytvoril triedu, ktorá ju spracuje statickou metódou.

```
public class HttpExceptionHandler {
    public static ResponseEntity resolve(HttpException exception) {
        HttpReturnCode code = exception.getErrorCode();
        if (code == null) {
            code = HttpReturnCode.BAD_REQUEST;
```

```
        }
        return ResponseEntity.status(code.getCode())
            .contentType(MediaType.TEXT_PLAIN)
            .body(exception.getMessage());
    }
}
```

HttpExceptionHandler sa používa v kontroleri na miestach, kde metódy môžu vyhodiť *HttpException*. Metóda sa obalí try – catch blokmi a v catch bloku kontroler iba vráti *HttpExceptionHandler.resolve(e)*. Na front-end sa tak dostaví iba chybová odpoveď s kódom a správou, ktorá je definovaná pri vyhadzovaní podmienky.

6.4.2 Reflexia

Priečinok *be-spring/src/main/java/.../base/support*, súbory začínajúce *TableData*

Táto časť bola vytvorená kvôli znovu použiteľnosti kódu a jednoduchšiemu vytváraniu tabuliek. Z front-endu príde názov servisu a parametre strany. *TableDataServiceProvider* pomocou reflexie získa potrebný servis a podľa preddefinovaných metód vytvorí a vráti odpoveď. Servisy v sebe prepíšu metódy podľa potreby (filtrovanie, špecifický dotaz na databázu a pod.). Reflexia prebieha na základe vlastnej anotácie:

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Component
public @interface TableData {
    String value() default "";
}
```

Tá sa používa pre servis, ktorý zároveň rozširuje predlohu *TableDataService*.

```
@TableData("race-table")
public class RaceTableDataService extends TableDataService<Race,
  RaceRepository, RaceTableDataFilter, RaceDetailDto, RaceMapper> {
    ...
}
```

Spring vie potom na základe rozširovania *TableDataService* a hodnoty uvedenej v anotácii *@TableData* využiť reflexiu a získať inicializovaný servis, ktorý má použiť.

```
public Page<?> getTableData(String name, Pageable pageable,
  List<Filter> filter) throws HttpException {
    TableDataService<?, ?, ?, ?, ?> service =
      context.getBean(name, TableDataService.class);
    return service.getFormated(pageable, service.createFilter(filter));
}
```

6.4.3 Parametrizované triedy

Súbor *be-spring/src/main/java/.../base/support/template/TableDataService.java*

Jedná sa predlohu pre servisy zabezpečujúce dáta pre tabuľky. Predloha je parametrizovanou abstraktnou triedou. Parametre je potrebné určiť pre klasifikáciu a používanie potrebných súčastí v predlohe.

```
public abstract class TableDataService<
  E,
  R extends JpaRepository<E, ?> & JpaSpecificationExecutor<E>,
  F,
  D,
  M extends TableDataResponseMapper<E, D>
  > {
  ...
}
```

Pre názornosť som v ukážke oddelil jednotlivé parametre do osobitných riadkov. Parameter E definuje triedu entity, teda model, ktorý je potrebné získať z databázy. Parameter R definuje triedu repozitáru, ktorý sa má použiť pre získanie záznamov. Keďže sa má jednať o repozitár, ktorý má poskytnúť záznamy triedy E vyžaduje sa, aby stanovená trieda rozširovala *JpaRepository<E, ?>*. Pre zabezpečenie funkcionality špecifikácie, sa vyžaduje aj rozšírenie *JpaSpecificationExecutor<E>*. Parameter F definuje triedu filtra a parameter D definuje triedu DTO, aby servis vedel v akom formáte má poskytnúť odpoveď. Posledným parametrom M je definovaná trieda poskytujúca premapovanie E na D pomocou Mapstructu, preto je vyžadované rozšírenie triedy M šablónou, ktorá túto funkcionalitu zabezpečí. Nasleduje príklad triedy rozširujúcej *TableDataService* rozdelený na riadky rovnakým spôsobom. V komentároch pred triedou je pre názornosť uvedená definícia tried, ktoré rozširujú iné.

```
//RaceRepository extends JpaRepository<Race, Integer>,
//  JpaSpecificationExecutor<Race>
//RaceMapper extends TableDataResponseMapper<Race, RaceDetailDto>

@TableData("race-table")
public class RaceTableDataService extends TableDataService<
  Race,
  RaceRepository,
  RaceTableDataFilter,
  RaceDetailDto,
  RaceMapper
  > {
    ...
}
```

6.4.4 Identifikácia a inicializácia triedy počas runtime

Súbor *be-spring/src/main/java/.../base/support/template/TableDataService.java*

Jedná sa o súčasť podpory pre tabuľky na back-ende. *TableDataService* potrebuje identifikovať triedu servisu a filtru, aby mohol premapovať hodnoty vo formáte string na dané hodnoty vo filtri. Kvôli znovu použiteľnosti je metóda implementovaná v abstraktnej triede, takže je potrebné aby najprv identifikovala samú seba a následne triedu svojho filtra.

```
public F createFilter(List<Filter> filterList) throws HttpException {
  if (filterList != null && !filterList.isEmpty()) {
    ResolvableType resolvableType = ResolvableType.forClass(
      this.getClass()).as(TableDataService.class);
```

```
    Class<F> fClass = (Class<F>) resolvableType.resolveGeneric(2);
    if (fClass != null) {
      try {
        F filter = fClass.newInstance();
        for (Filter f : filterList) {
          if (f.getValue() == null || f.getValue().isEmpty()) {
            continue;
          }
          Field field = fClass.getDeclaredField(f.getColumn());
          Class<?> c = field.getType();
          ...
        }
      return filter;
      } catch (InstantiationException | IllegalAccessException
        | NoSuchFieldException e) {
          throw new HttpException(HttpReturnCode.BAD_REQUEST,
            e.getMessage());
      }
    }
  }
  return null;
}
```

Kód sa vykonáva zo servisu rozširujúceho *TableDataService*. Najprv sa vytvorila inštancia *ResolvableType* pre aktuálnu triedu, ktorej povieme že má formát *TableDataService.class*, lebo vieme, že ju servis rozširuje. Následne dáme zistiť triedu parametru s indexom dva – filter. Pokiaľ sa všetko podarilo, pomocou metódy *newInstance()* vytvoríme inštanciu filtra a postupne napĺňame jeho políčka podľa toho, ako by sa mali volať. V ukážke je preskočená časť, ktorá mapuje reťazec (String) na potrebný typ.

6.4.5 Dialóg

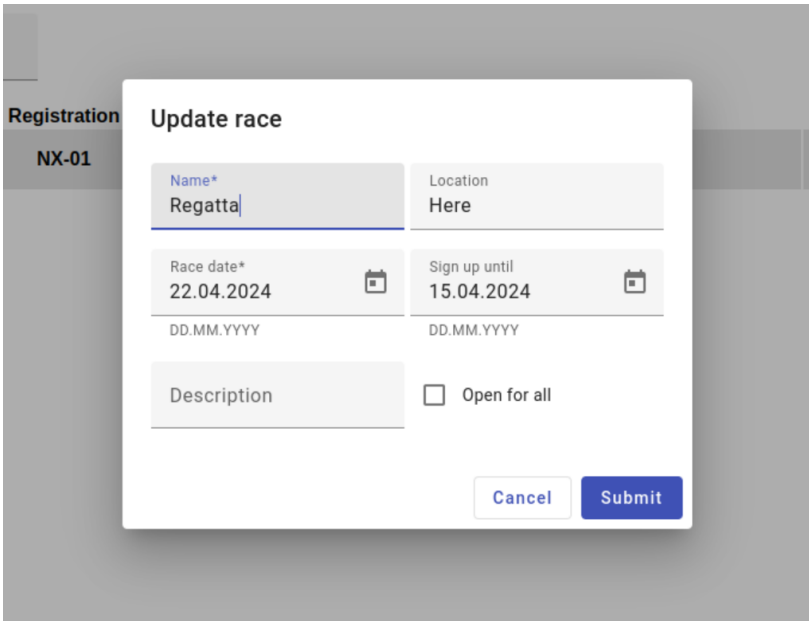
Priečinok *fe-angular/src/app/core/support/dialog*

Súbory v tomto priečinku sú zamerané na znovu použiteľnosť kódu vytváraním a spracovávaním formulárov. Definujú formulár, vstupy rôznych typov, validáciu hodnôt, predvyplnenie, dizajn prvkov. Vo väčšine prípadov sa dialóg zaobalí ešte do ďalšieho komponentu, ktorý ho transformuje do formy vyskakovacieho okna a má na starosti odosielanie formulára. Nasleduje príklad použitia.

```
<app-dialog [title]="raceId ? 'Update race' : 'Create race'" [data]="data"
            (submitButtonClick)="onSubmitButtonClick($event)">
  <app-dialog-field [field]="'name'" [title]="'Name'"
                    [type]="DialogFieldType.STRING" [required]="true">
  </app-dialog-field>
  <app-dialog-field [field]="'location'" [title]="'Location'"
                    [type]="DialogFieldType.STRING">
  </app-dialog-field>
  <app-dialog-field [field]="'date'" [title]="'Race date'"
                    [type]="DialogFieldType.DATE" [required]="true">
```

```
  </app-dialog-field>
  <app-dialog-field [field]="'signUpUntil'" [title]="'Sign up until'"
                    [type]="DialogFieldType.DATE">
  </app-dialog-field>
  <app-dialog-field [field]="'description'" [title]="'Description'"
                    [type]="DialogFieldType.STRING">
  </app-dialog-field>
  <app-dialog-field [field]="'isPublic'" [title]="'Open for all'"
                    [type]="DialogFieldType.BOOLEAN">
  </app-dialog-field>
</app-dialog>
```

Ukážka kódu vo vyskakovacom okne je na obrázku 6.1.



Obr. 6.1: Ukážka vyskakovacieho dialógu.

6.4.6 Tabuľka

Priečinok *fe-angular/src/app/core/support/table*

Súbory v tomto priečinku zabezpečujú kompletnú podporu pre zobrazovanie tabuliek. Zabezpečujú filtrovanie, zoraďovanie, stránkovanie vrátane nastavenia veľkosti strany a podporu pre tlačidlá k záznamom pre akcie nad jednotlivými záznamami. Medzi súbormi sú dve smernice. Prvá – *TableSearchDirective* zabezpečuje definíciu polí filtrovania. Druhá – *TableColumnDirective* definuje zabezpečuje definíciu stĺpcov tabuľky. Nasleduje príklad použitia.

```
<app-table #usersTable [serviceName]="'user-table'"
           (onButtonClick)="buttonClicked($event)">
  <app-table-search [title]="'Username'" [column]="'username'">
  </app-table-search>
  <app-table-search [title]="'Name'" [column]="'name'">
  </app-table-search>
```

```
  <app-table-search [title]="'Email'" [column]="'email'">
  </app-table-search>

  <app-table-column [title]="'Id'" [field]="'id'" [width]="50">
  </app-table-column>
  <app-table-column [title]="'Username'" [field]="'username'">
  </app-table-column>
  <app-table-column [title]="'Name'" [field]="'fullName'">
  </app-table-column>
  <app-table-column [title]="'Email'" [field]="'email'">
  </app-table-column>

  <app-table-column [field]="Action.RedirectToDetail"
                    [icon]="'fas fa-angle-right'" [isButton]="true">
  </app-table-column>
  <ng-container *ngIf="isAdmin">
    <app-table-column [field]="Action.EditUser" [icon]="'far fa-edit'"
                      [isButton]="true">
    </app-table-column>
    <app-table-column [field]="Action.ChangePermissions"
                      [icon]="'fas fa-user-cog'" [isButton]="true">
    </app-table-column>
  </ng-container>
</app-table>
```

Ukážka tabuľky podľa kódu je na obrázku 6.2.

Obr. 6.2: Ukážka tabuľky.

6.4.7 Mapa s navigáciou

Priečinok *fe-angular/src/app/races/map*

Priečinok obsahuje celkovú logiku zobrazovania mapy, vytvárania trate a navigácie ku jednotlivým bójam. OpenLayers poskytuje jednoduchý import vektorových a rastrových vrstiev:

```
const courseLabelLayer = new VectorLayer({
```

```
  source: this.courseLabelVector,
  style: courseLabelStyle
}l);
const seaLayer = new TileLayer({
  source: new OSM({
    attributions: [
      '© <a href="https://www.openseamap.org/">OpenSeaMap</a>',
        ATTRIBUTION
    ],
    opaque: false,
    url: 'https://tiles.openseamap.org/seamark/{z}/{x}/{y}.png'
  })
});
...
this.map = new Map({
  layers: [
    new TileLayer({
      source: new OSM(),
    }),
    seaLayer,
    courseLabelLayer,
    courseLayer,
    navigationLayer
  ],
    target: 'map',
    view: view
});
```

V tejto ukážke som si vytvoril vlastnú vektorovú vrstvu mapy a importoval vrstvu rastrovej námornej mapy. Následne som ich vzájomným spojením a spojením s ďalšími vrstvami definoval mapu. Poradie vrstiev v mape určuje aj to, ako sa budú prekrývať.

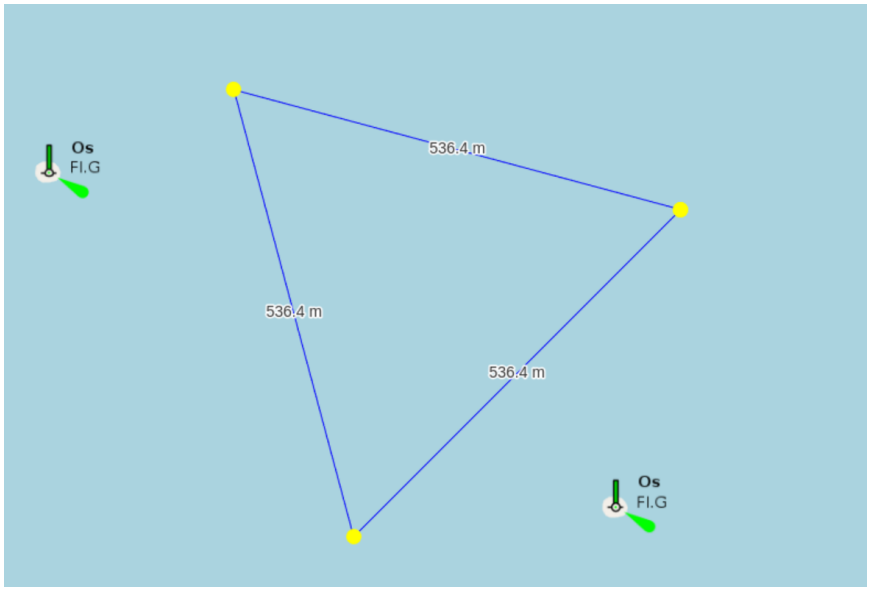
```
this.map.on('moveend', () => {
  this.refresh();
});
```

Táto časť kódu ukazuje ako sa pracuje s udalosťami mapy. Reťazec definuje udalosť, na ktorú ideme definovať reakciu. Definícia môže byť cez lambda funkciu (ako v ukážke), alebo cez referenciou na funkciu.

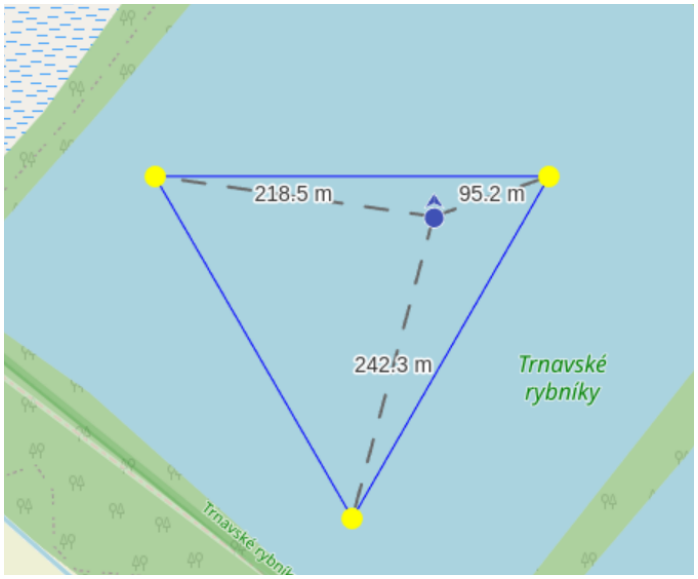
Vykreslenie trate podľa vetra som zabezpečil rozmiestnením bójí okolo stredu mapy do rovnostranného trojuholníka, pričom jedna zo strán je orientovaná zo severu na juh a nachádza sa na pravej strane od stredu. Následne som celý trojuholník rotoval okolo stredu o uhol vetra. Tým sa zabezpečí orientácia trate.

```
const l1: number = radius;
const l2: number = radius/2;
const l3: number = Math.sqrt((radius * radius) - ((radius/2)*(radius/2)));
const c1: Array<number> = [coord[0] + l2, coord[1] + l3];
const c2: Array<number> = [coord[0] - l1, coord[1]];
```

```
const c3: Array<number> = [coord[0] + l2, coord[1] - l3];
const track: Array<Array<number>> = [c1, c2, c3, c1];
this.untransformedCoordinates = track;
const polygon = new Polygon([track]);
if (this.lastValidAngle != 0) {
  polygon.rotate(- this.lastValidAngle * Math.PI / 180, coord);
}
```



Obr. 6.3: Ukážka zobrazovania vzájomnej vzdialeností bójí pri umiesťovaní trate.



Obr. 6.4: Ukážka navigácie.

Kapitola 7

Testovanie

Testovanie je dôležitá súčasť vývoja všetkých aplikácií. Odhaľuje chyby v implementácií, ale aj nesprávne pochopené požiadavky pri testovaní zákazníkom. Overuje teda či systém funguje podľa požiadaviek a či funguje podľa očakávania.

Testovanie pri vývoji

Systém som testoval priebežne počas celého vývoja. Vždy po implementácii funkcionality som preskúšal či funguje správne. Pri väčších zmenách a po implementácii väčšieho množstva funkcionality som vždy preskúšal funkčnosť celého systému, aby som mal istotu, že neprestala nejaká už implementovaná funkcia fungovať správne. Po dokončení som vyskúšal systém aj od nuly – iba zo zdrojových kódov, s potrebou inicializácie všetkého.

Testovanie skautmi

Systém mi priebežne, ale hlavne po jeho finalizácii pomohli pretestovať aj vybraný slovenský vodný skauti. Vykonávali všetky akcie, ako keby systém používali počas reálnych pretekov, vrátane rozostavenia trate a navigácie. Nahlásené chyby som zapracoval a opravil.

Aplikáciu som im sprístupnil len dočasne pomocou tunelovania, keďže som nemal pripravený server. Vďaka tomuto je aplikácia uspôsobená aj na fungovanie cez tunelovanie. Avšak niektoré chyby, ktoré sa ukazovali boli spôsobené práve ním.

Plánujem so slovenskými vodnými skautmi ďalšie veľké testovanie, tento-krát v normálnej prevádzke na plánovaných pretekoch plachetníc AquAeris v septembri tohto roku. Počas tohto testovania už bude aplikácia nasadená na serveri, čím eliminujem chyby spôsobené testovaním cez tunelovanie.

Kapitola 8

Záver

Cieľom tejto bakalárskej práce bolo navrhnúť a implementovať informačný systém pre organizátorov pretekov plachetníc. Systém obsahuje podporu pre rozostavenie trate vzhľadom na vietor vrátane navigácie ku jednotlivým bójam. Systém obsahuje preteky, na ktoré sa používatelia môžu prihlasovať buď ako členovia posádok, alebo môžu prihlasovať na preteky svoje lode. Do lodí sa následne môžu prihlásiť iný používatelia ako posádka, ale aj ich majitelia. Po skončení pretekov je možné nahrať jednotlivým posádkam ich výsledky. Systém implementuje značnú podpornú funkcionalitu zameranú na znovu použiteľnosť, jednoduchosť a prehľadnosť. Systém bol tvorený najmä s dôrazom na jednoduchú rozšíriteľnosť, kvôli plánovanému rozširovaniu systému v budúcnosti.

8.1 Možné rozšírenia

Informačný systém vyhovuje požiadavkám, ktoré boli pre neho definované slovenskými vodnými skautmi. Keďže bol kladený dôraz na rozšíriteľnosť počas jeho vývoja, boli plánované aj niektoré jeho rozšírenia. Systém môže byť rozšírený o ďalšie štatistiky pretekárov a ich grafické zobrazenie. Do systému môže byť pridané sledovanie pretekárov s následným zobrazením trasy a štatistík nazbieraných sledovaním. Ďalším možným rozšírením je podpora nových typov tratí, prípadne vlastného rozostavenia bójí mimo formátu bežných tratí. Ďalšou možnosťou rozšírenia je pridanie kategórií plachetníc a nastavovanie povolených kategórií pre preteky. Systém môže byť rozšírený o fotografie. Pod tým je myslená galéria pre jednotlivé preteky, fotografie daných lodí, profilové fotografie a podobne.

Literatúra

[1] Agafonkin, V. *Leaflet* online. 2024. Dostupné z: https://leafletjs.com/. [cit. 2024-05-01].

[2] AWS. *What is an API (Application Programming Interface)?* online. 2024. Dostupné z: https://aws.amazon.com/what-is/api/. [cit. 2024-05-01].

[3] AWS. *What is Java?* online. 2024. Dostupné z: https://aws.amazon.com/what-is/java/. [cit. 2024-05-01].

[4] Bernstein, P. A. a Newcomer, E. *Principles of Transaction Processing*. 2. vyd. 30 Corporate Drive, Suite 400, Burlington, MA 01803, USA: Morgan Kaufmann Publishers, 2009. 1-5 s. ISBN 978-1-55860-623-4.

[5] Britanica, T. E. o. E. *Mercator projection* online. 26. novembra 2023. Dostupné z: https://www.britannica.com/science/Mercator-projection. [cit. 2024-05-01].

[6] Brotherton, C. *The Most Popular PHP Frameworks to Use* online. 14. decembra 2023. Dostupné z: https://kinsta.com/blog/php-frameworks/#what-are-the-best-php-frameworks. [cit. 2024-05-01].

[7] Cardoso, J. *Office Automation Systems*. Funchal, Portugal: University of Madeira, 2006. Vedecký papier.

[8] ChatGPT. *Informačné systémy* online. 2024. Dostupné z: https://chat.openai.com/share/9d5ecab5-ea20-4d74-a0c3-5f48c620847d. [cit. 2024-05-01].

[9] Dalby, B. *Map File Formats* online. 22. marca 2024. Dostupné z: https://eastwestmapping.ie/map-file-formats/. [cit. 2024-05-01].

[10] Dhaduk, H. *Best Frontend Frameworks for Web Development* online. 1. februára 2024. Dostupné z: https://www.simform.com/blog/best-frontend-frameworks/. [cit. 2024-05-01].

[11] Docker. *Use containers to Build, Share and Run your applications* online. 2024. Dostupné z: https://www.docker.com/resources/what-container/. [cit. 2024-05-01].

[12] Fernbach, C. Navigation, what is the difference between loxodromy and orthodromy? *Boating culture* online, Júl 2022. Dostupné z: https://www.boatnews.com/story/30282/navigation-what-is-the-difference-between-loxodromy-and-orthodromy. [cit. 2024-05-01].

[13] Geography, G. *Conic Projection: Lambert, Albers and Polyconic* online. 14. októbra 2023. Dostupné z: https://gisgeography.com/conic-projection-lambert-albers-polyconic/. [cit. 2024-05-01].

[14] Google. *Google Maps Platform* online. 2024. Dostupné z: https://developers.google.com/maps. [cit. 2024-05-01].

[15] Group, T. P. *What is PHP?* online. 2024. Dostupné z: https://www.php.net/manual/en/intro-whatis.php. [cit. 2024-05-01].

[16] Ing. Radek Burget, P. *Informační systémy: Pojem informačního systému, data, informace* online. Fakulta informačních technologií VUT v Brně, september 2021. Dostupné z: https://www.fit.vutbr.cz/~burgetr/iis/p01_informacni_systemy/. [cit. 2024-05-01].

[17] Ing. Radek Burget, P. *Informační systémy: Architektury informačních systémů* online. Fakulta informačních technologií VUT v Brně, september 2021. Dostupné z: https://www.fit.vutbr.cz/~burgetr/iis/p02_architektury/. [cit. 2024-05-01].

[18] JavaTPoint. *Difference between JavaScript and TypeScript* online. 2021. Dostupné z: https://www.javatpoint.com/javascript-vs-typescript. [cit. 2024-05-01].

[19] JSGSPRODUCTIONS.COM. *HTML5 FOR BEGINNERS AND ADVANCED* online. JSGS PRODUCTIONS. Dostupné z: https://books.google.sk/books?id=t2r0DwAAQBAJ.

[20] Lapaine, M. a Usery, E. L. *Choosing a Map Projection*. 1. vyd. Gewerbestrasse 11, 6330 Cham, Switzerland: Springer International Publishing AG, 2017. ISBN 978-3-319-51835-0.

[21] Laravel. *Getting Started: Installation* online. 2024. Dostupné z: https://laravel.com/docs/11.x. [cit. 2024-05-01].

[22] Laudon, J. P. a Laudon, K. C. *Management Information Systems*. 8. vyd. Prentince Hall, New York: Pearson, 2003. ISBN 978-0131014986.

[23] Learn, O. *Client – Server architecture* online. 17. apríla 2017. Dostupné z: https://www.open.edu/openlearn/science-maths-technology/an-introduction-web-applications-architecture/content-section-1.1. [cit. 2024-05-01].

[24] Learn, O. *Multi – tier architecture* online. 17. apríla 2017. Dostupné z: https://www.open.edu/openlearn/science-maths-technology/an-introduction-web-applications-architecture/content-section-1.2. [cit. 2024-05-01].

[25] Lucidchart. *What is an Entity Relationship Diagram (ERD)?* online. 2024. Dostupné z: https://www.lucidchart.com/pages/er-diagrams. [cit. 2024-05-01].

[26] Mania, M. *Architektúra klient-server* online. 4. novembra 2016. Dostupné z: https://managementmania.com/sk/architektura-klient-server. [cit. 2024-05-01].

[27] Microsoft. *Why Choose .NET?* online. 2024. Dostupné z: https://dotnet.microsoft.com/en-us/platform/why-choose-dotnet. [cit. 2024-05-01].

[28] Mukherjee, S. The 6 Types Of Information Systems And Their Applications. *Emeritus* online. 24. apríla 2024. Dostupné z: https://emeritus.org/in/learn/the-6-types-of-information-systems-and-their-applications/. [cit. 2024-05-01].

[29] Networks, J. *What is a cloud microservice* online. 2024. Dostupné z: https://www.juniper.net/us/en/research-topics/what-is-a-cloud-microservice.html. [cit. 2024-05-01].

[30] Newman, S. *Building Microservices*. 1. vyd. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, Inc., 2015. ISBN 978-1-491-95035-7.

[31] OCI. *What is a geospatial database?* online. 2024. Dostupné z: https://www.oracle.com/autonomous-database/what-is-geospatial-database/. [cit. 2024-05-01].

[32] Olympedia. *Sailing* online. 2023. Dostupné z: https://www.olympedia.org/sports/SAL. [cit. 2024-05-01].

[33] OpenLayers. *OpenLayers* online. 2024. Dostupné z: https://openlayers.org/. [cit. 2024-05-01].

[34] Phillips, J. *Sailing* online. 25. júna 2021. Dostupné z: https://hatterassailing.org/learn-to-sail/basic-race-course-starting-knowledge/. [cit. 2024-05-01].

[35] Reselman, B. a McKenzie, C. *10 disadvantages of microservices you'll need to overcome* online. 16. novembra 2023. Dostupné z: https://www.theserverside.com/answer/What-are-some-of-the-disadvantages-of-microservices. [cit. 2024-05-01].

[36] Roberts, S. Types of Information Systems and Their Uses. *The Knowledge Academy* online. 28. decembra 2023. Dostupné z: https://www.theknowledgeacademy.com/blog/types-of-information-systems/. [cit. 2024-05-01].

[37] Roxin, A.; Gaber, J.; Wack, M. a Nait Sidi Moh, A. Survey of Wireless Geolocation Techniques. In: Université de Technologie de Belfort-Montbéliard (UTBM). *GLOBECOM - IEEE Global Telecommunications Conference*. December 2007. ISBN 978-1-4244-2024-7.

[38] RYA. *Course diagrams* online. 2024. Dostupné z: https://www.rya.org.uk/racing/running-racing/course-diagrams. [cit. 2024-05-01].

[39] Sari, S. *Algorithms for Computing Routes on a Map* online. Apríl 2023. Dostupné z: https://www.baeldung.com/cs/routes-optimal-on-map. [cit. 2024-05-01].

[40] Silva Carvalho Brito, A. E. da. Major types of systems in organizations. *Management Information Systems* online. 2006. Dostupné z: https://paginas.fe.up.pt/~acbrito/laudon/ch2/chpt2-1fulltext.htm. [cit. 2024-05-01].

[41] Simplilearn. The 6 Most Popular Types of Information Systems and Their Applications. *Simplilearn* online. 5. októbra 2023. Dostupné z: https://www.simplilearn.com/types-of-information-systems-and-applications-article. [cit. 2024-05-01].

[42] Skeeter, J. *Understanding Map Projections* online. 25. mája 2023. Dostupné z: https://ubc-library-rc.github.io/map-projections/content/proj-family.html. [cit. 2024-05-01].

[43] Spring. *Projects* online. 2024. Dostupné z: https://spring.io/projects. [cit. 2024-05-01].

[44] srinam. *Use Case Diagrams | Unified Modeling Language (UML)* online. Február 2024. Dostupné z: https://www.geeksforgeeks.org/use-case-diagram/. [cit. 2024-05-01].

[45] Stefik, M. *Introduction to Knowledge Systems*. 1. vyd. 340 Pine Street, Sixth Floor, San Francisco, CA 94104-3205 USA: Morgan Kaufmann Publishers, Inc., 1995. 292-299 s. ISBN 1-55860-166-X.

[46] Wagner, B. *A tour of the C# language* online. 8. mája 2024. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/overview. [cit. 2024-05-10].

[47] Wikipedia. *Informačný systém* online. Marec 2022. Dostupné z: https://sk.m.wikipedia.org/wiki/Informa%C4%8Dn%C3%BD_syst%C3%A9m;. [cit. 2024-05-01].

[48] Wikipedia. *Azimuthal equidistant projection* online. 26. marca 2024. Dostupné z: https://en.m.wikipedia.org/wiki/Azimuthal_equidistant_projection. [cit. 2024-05-01].

[49] York, R. *Beginning CSS: Cascading Style Sheets for Web Design*. 2. vyd. 10475 Crosspoint Boulevard, Indianapolis, IN 46256: Wiley Publishing, Inc., 2007. ISBN 978-0-470-09697-0.

Príloha A

Obsah priloženého pamäťového média

is-regatta-src adresár obsahujúci zdrojové súbory pre celý informačný systém

technicka-sprava-src adresár obsahujúci zdrojové súbory technickej správy

xvanop01.pdf technická správa

readme informácie o súboroch, aplikácii, jej zostavení a spustení