Vysoká škola ekonomická v Praze
Fakulta informatiky a statistiky

Aplikace pro sledování zvířat

BAKALÁŘSKÁ PRÁCE

Studijní program: Aplikovaná informatika
Studijní obor: Aplikovaná informatika

Autor: Karel Douda
Vedoucí práce: Ing. David Král

Praha, květen 2020

Prohlášení

Prohlašuji, že jsem bakalářskou práci *Aplikace pro sledování zvířat* vypracoval samostatně za použití v práci uvedených pramenů a literatury.

V Praze dne DD. měsíc RRRR .....

Podpis studenta

Poděkování

Poděkování.

Abstrakt

Abstrakt.

Klíčová slova

animal tracking, mobilní aplikace, react native, expo

Abstract

Abstract.

Keywords

animal tracking, mobile app, react native, expo

Obsah

Seznam obrázků

Seznam tabulek

Seznam procedur

Seznam použitých zkratek

IoT Internet of Things
GSM Global System for Mobile Communications
GPS Global Positioning System
HTTP HyperText Transfer Protocol
CSS Cascading Style Sheets
JS JavaScript
SDK Software Development Kit
NPM Node Package Manager
ES EcmaScript
TS TypeScript
RN ReactNative
JSON JavaScript Object Notation

Úvod

S rozvojem IoT technologí dochází ke stále další a další miniaturizaci autonomních off-the-grid zařízení. Jedním z mnoha oborů, které z tohoto vývoje těží, je zoologie. Obory zoologie zabývající se výzkumem migrací, studiem životních cyklů, ochrany a výzkum vlivu lidské činnosti na zvířata díky tomuto vývoji využívají stále dostupnější trackovací zařízení nasazovaná na zvířata. Nasazená zařízení komunikují především pomocí GSM technologií a sbírají telemetrická data o životních funkcích zvířete, pozici, vzdálenosti k zájmovým bodům a podobně.

Cílem této práce je vytvoření aplikace pro sledování a kontrolu divokých zvířat v terénu, konkrétněji ptáků. Primární funkcí aplikace je zobrazení posledních pozic vybraných zvířat v mapě. O vybraných zvířatech se ukládají metainformace, které mohou sloužit k dodatečné identifikaci v terénu. Pro aplikaci je kritická možnost fungování bez internetového připojení, kterou současné řešení nepodporuje. Současné řešení taktéž není vhodné pro použtí na mobilu z hlediska použitelnosti. Toto téma je řešeno z důvodu absence efektivního řešení tohoto problému.

Zvolený způsob řešení je multiplatformní aplikace vyvíjená v prostředí React Native s možností prací offline. React Native umožňuje vytváření mobilních aplikací pro platformy Android i iOS bez nutnosti psát dvě separátní aplikace. React Native je souborem JS knihoven postavených nad frontendovým frameworkem React od společnosti Facebook. Pro zrychlení vývoje bez nutnosti podbroného testování aplikací na obou platformách byla použita nástavba Expo, která dále abstrahuje od platformně závislého kódu. Zdrojem dat je ornitologická platforma Anitra, která mimo funkce datového uložiště podporuje sdílení dat a udržování metainformací o žvířatech.

Tento dokument se skládá z X kapitol. První kapitola ve zkratce popisuje problémovou oblast, současný stav a základní koncepty nutné pro pochopení zbytku práce související s kontextem ornitologie a telemetrických zařízení pro ornitologii. V druhé kapitole se objasní stávající řešení využívaní k docílení určitých požadavků, které objasní následující kapitoly a důvod, proč existující řešení nejsou vhodná. Samotným vývojovým procesem se tento dokument zabývá od kapitoly Analýza požadavků, ve které se uvedou funkční požadavky na mobilní aplikaci. Kapitola technologie popisuje technologie, které pro řešení tohoto problému šlo využít, jaká byla vybrána a základní informace o jednotlivých částech technologie. Kapitola návrh popíše návrh aplikačních architektury a návrh uživatelského rozhraní. Kapitola implementace popíše proces vytváření samotné aplikace a věnuje se specifickému řešení problémů z předchozí kapitoly. Předposlední kapitolou je testování a nasazení, kde je popsán způsob, jakým byla aplikace testována a publikována. Na závěr je zhodnocení splnění cílů práce a možnost dalšího rozvoje.

1. Problémová oblast

Sledování a popis životního cyklu ptáků, vědecká disciplína nazývaná ornitologie, je poměrně starou vědeckou disciplínou s kořeny ve starověku. Jedním z prvních dochovaných textů zabývajícím se popisu života ptáků je Aristotelova *Historia Animalium* v roce 350 př. n. l. Ornitologie se postupně rozvíjela – zaváděly se taxonomie ptactva, studovala se ptačí anatomie, ale spousta otázek spojena s chováním ptáků zůstavala nezodpovězena. V roce 1805. Americký ornitolog John James Audubon se snažil prokázat, že se každým rokem na jeho farmu vrací stejný jedinec druhu *Sayornis phoebe* přivázáním stříbrného lanka na nohu. Tímto nepřímo položil základy pro tzv. kroužkování, pasivní označení jedinců kovovým kroužkem s vyraženým sériovým identifikátorem kroužku. Systém kroužkování zavedl dánský ornitolog a učitel Hans Christian Cornelius Mortensen v roku 1899. Na území České republiky se kroužkování ujalo roku 1910, přibližně rok po rozšíření systému v Anglii a Německu. Dle informací Společnosti spolupracovníků kroužkovací stanice Národního muzea je dnes registrováno 480 spolupracovníků, kteří ročně okroužkují kolem 175 000 ptáků.

Kroužkování ptáků slouží k mnoha účelům – mapování migračních tras (zimoviště, návrat na stejné lokace), populační studie (např. rozšiřování jednotlivých druhů, roční úhyn nebo naopak nárůst), životní cyklus ptactva (např. délka života). Kroužkování je dodnes základem práce ornitologů díky jeho nízké cenové náročnosti a mezinárodní kolaboraci ornitologů, záchranných stanic a dobrovolníků.

Hlavní nevýhodou kroužkování je samozřejmě pasivní povaha této metody – o ptácích se zjišťují pouze kusé informace o jejich přibližné poloze, které musí nahlásit pozorovatel. Pozorování samotné je záležitostí s prvkem nejistoty - kroužky nemusí být dostatečně čitelné pro kompletní identifikaci, pozorovatel tedy může informovat např. jen o pozorování určitého druhu ptáka. Kroužkování závisí také na vysoké angažovanosti dobrovolníků a disciplinované administrativě související s osazením a nahlášením pozorování ptáků s kroužky. Dále taktéž neřeší jiné úkoly ornitologů, které souvisí např. s kontrolou hnízd – musí se zkontrolovat rozsáhlý počet hnízd, což s sebou přináší logistické i administrativní problémy.

1.1 Vývoj v oblasti telemetrie

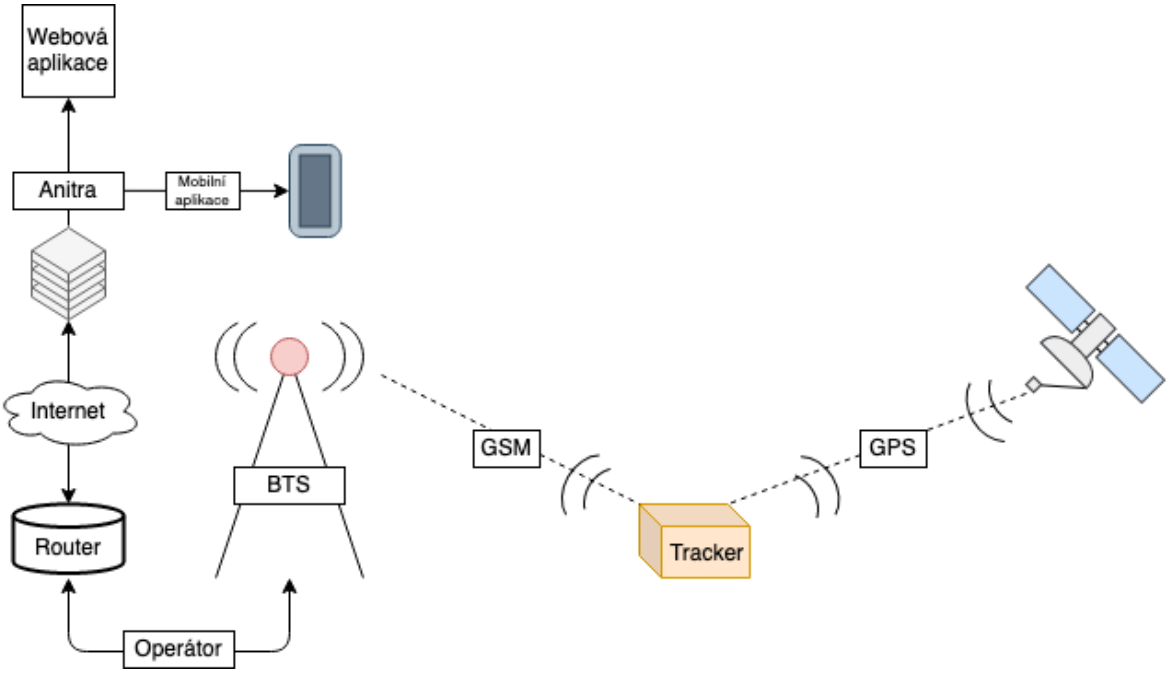
Miniaturizací elektronických součástek, především akumulátorů, se v 60. letech 20. století začínají rozšiřovat aktivní telemetrická zařízení. Jednoduché radiolokátory umožnily v terénu přesně určit pozici zvířete sledováním intenzity signálu. Radiolokátory mohou pomocí elektromagnetických pulsů přenášet určitá data (např. identifikátor radiolokátoru – zvířete) za použití určité modulace signálu. Tímto se zjednodušilo např. hledání hnízd, čímž se taktéž usnadnilo hledání a označení mladých jedinců, kteří ještě nebyli vyvedeni z hnízda.

Ornitologie těžila i z rozvoje kosmických programů. Signály z dostatečně výkonných radio-

lokátorů mohou být přijaty speciálními družicemi v kosmu a za pomocí Dopplerova jevu lze spočítat přibližnou polohu daného jedince. Tato řešení nebyla zpočátku pro ornitologii příliš vhodná z hlediska hmotnosti radiolokátorů. Příchodem GSM sítí v 90. letech a uvolňování restrikcí na použití GPS se situace pro ornitology zásadně změnila. Na trhu se objevily výrazně lehčí (desítky až jednotky gramů) GPS-GSM trackery vhodné i pro malé druhy ptáků. Data z těchto trackerů se průběžně sbírají a odesílají do systémů výrobců zařízení, případně přímo majiteli zařízení.

Rapidní nástup webových technologií po roce 2000 umožnil výrobcům zařízení jednoduše uživatelům poskytovat navazující služby ke svým zařízením, jmenovitě jednoduchou konfiguraci zařízení, základní visualizaci dat, exporty dat, vedení metainformací k zařízení (např. na jakém zvířeti zařízení je) a další. Většina systémů se omezuje pouze na trackery od jednoho výrobce a funkce mimo množinu konfigurace zařízení jsou primitivní, až nedostatečné. V roce 2017 vznikla česká ornitologická platforma Anitra, která poskytuje komplexní nástroje pro správu zařízení a visualizaci dat od širokého spektra výrobců trackerů. Platforma Anitra taktéž poskytuje správu metainformací o zvířetech, správu zájmových bodů, nahrávání příloh a komplexní metody sdílení dat. Pro tuto práci byla platforma Anitra vybraná z důvodu autorovy možnosti vytvářet API na míru mobilní aplikaci a již existující uživatelské základny.

Pro základní představu celého systému je níže uvedeno schématické znázornění vztahů mezi jeho prvky.



Obrázek 1.1: Schéma systému GPS-GSM trackerů

2. Existující řešení

Tato kapitola se věnuje popisu již existujících řešení problémů nastíněných v předchozí kapitole. V této kapitole budou zhodnoceny i další problémy, např. administrativního rázu a problémy s bezpečností dat. Za konkrétní problémy se zde řadí: uchovávání informací o jedincích (morfometricé údaje, obrázky, místo označení), uchování pozic z GPS-GSM trackeru, uchování jiných geografických informací (např. lokace hnízdiště), přehlednost dat, přenositelnost a sdílení dat. Jednotlivá řešení budou popsána, zhodnocena z pohledu vydefinovaných kritérií a na konci bude vynesen verdikt, proč bylo rozhodnuto pro vývoj mobilní aplikace nad platformou Anitra.

2.1 Nestrukturovaný zápis

ruční poznámky - obrázek od D.

Nestrukturovaná data je obecné označení pro data, která se zpravidla nedají popsat exaktním formálním schématem. Nejčastěji se jedná o textové dokumenty, obrázky či jiná multimédia. Pro lidské zpracování bývají nestrukturované dokumenty přirozenější, než strukturované dokumenty s exaktním schématem. Autory nestrukturovaných dat nemusí být pouze lidé, ale pro tento kontext budou uvažovány pouze artefakty lidské činnosti.

Metoda nestrukturovaného ručního zápisu spočívá v uchovávání papírových dokumentů vznikající při činnosti ornitologů. Je možné zapsat např. pozorování jedinců, kontroly hnízd, manipulace (např. nasazení kroužku), morfometrické údaje (délky křídel), obecné poznámky. Schéma zápisu nemusí být normalizované, je zde tedy nejjednoduší možnost schéma upravit pro potřeby ornitologa, případně ornitologické organisace.

Kritérium	Hodnocení
Informace o jedincích	1 – absence vynuceného schématu umožňuje uložit jakékoliv inf
Práce s pozicemi z trackerů	4 – informace musí být ručně vloženy
Práce s uživatelsky vloženými pozicemi	3 – informace musí být ručně vloženy, ale schéma není pevně da
Přehlednost dat	3 – v datech se obtížně hledá, závisí primárně na zapisovateli, n
Přenositelnost a sdílení dat	4 - data jsou složitě přenositelná, mohou být i obtížně pochopit

Tabulka 2.1: Hodnocení Nestrukturovaný zápis

Výhody

- možnost kdykoliv upravit schéma,
- nízká náročnost na technologie,
- rychlost zápisu,

Nevýhody

- obtižné vkládání fotodokumentace, o
- data jsou složitě přenositelná, obvykle pouze ručním přepisem do jiného systému.

2.2 Strukturovaný zápis

Označení *strukturovaná* se přisuzuje datům, které explicitně zachycují fakta, atributy, objekty, u kterých je významným rysem existence elementů dat. Strukturované ukládání dat je strojově zpracovatelné počítači a ukládá se často v databázových systémech nebo tabulkových procesorech. V této podkapitole bude popsán zápis pomocí tabulkového procesoru.

Na rozdíl od ručního zápisu je zde možné vynutit schéma, přehlednost i přenositelnost dat je tedy nesrovnale vyšší. Změny schématu pro zaznamenání nového druhu informací mohou být komplikací, např. v případu rozdělení datového elementu na dva, nutnosti změnit procesy či procedury zapisování záznamu. Problematické je vkládání a zobrazování některých telemetrických dat z trackerů umístěných na zvířatech. Ačkoliv např. zobrazení numerických telemetrických dat (teplota, napětí na akumulátoru) je jednoduché a dá se v tabulkovém procesoru zobrazit jednoduše, geografické pozice nikoliv. Problém také nastává při přibývání dat s výkonem. Problém zde může nastat při spolupráci více uživatelů najednou, synchronizace dat se musí kontrolovat, aby nedošlo ke ztrátě dat.

Kritérium	Hodnocení
Informace o jedincích	2 – způsob dostačuje, ale změna schématu nemusí vždy být flexibil
Práce s pozicemi z trackerů	3 – informace musí být ručně vloženy, případně integrovány službo
Práce s uživatelsky vloženými pozicemi	3 – informace musí být ručně vloženy, ale schéma není pevně dané
Přehlednost dat	2 – v datech se dá vcelku dobře hledat, ale některé metriky je obtí
Přenositelnost a sdílení dat	3 - data jsou přenositelná do jiných formátů, ale díky absenci stanc

Tabulka 2.2: Hodnocení Strukturovaný zápis

Výhody

- možnost kdykoliv upravit schéma,
- nízká náročnost na technologie,
- rychlost zápisu,
- integrované visualizační nástroje,
- tabulková podoba dat srozumitelná pro vědce.

Nevýhody

- závislost na přístupu k souborům tabulkového procesoru,
- obtížné či neefektivní zachycení nestrukturovaných dat (obrázků, volných textů),
- data jsou stále z větší části vkládaná ručně,
- data jsou složitě přenositelná, obvykle pouze ručním přepisem do jiného systému.

2.3 Geografické informační systémy

Geografický informační systém je označení pro sadu hardware, software a geografických dat pro sbět, správu, analýzu a zobrazení všechn způsobů geograficky referencovaných informací, neboli také prostorových dat. Z geografické povahy dat tedy vyplývá, že některé GIS softwary ref by mohly být pro ornitologickou aplikaci vhodné.

2.3.1 Google MyMaps

Google MyMaps je produktem z portfolia mapových produktů společnosti Google. Google MyMaps slouží především k uchování bodů, čar a polygonů v mapě. Pro ukládání dat tabulkového charakteru má aplikace podporu pouze pro objekty viditelné v mapě ve formě atributu, výhodou je ale jim možnost definovat schéma. Software ze své obecné charakteristiky má vysokou míru personalizace a obecnosti. Nespornou výhodou je napojení na ekosystém Google Drive, který umožňuje nahradit nedostatky této aplikace jinými. Správa příloh je zde taktéž vyřešena velmi dobře.

Kritérium	Hodnocení
Informace o jedincích	5 – veškeré informace se vztahují k bodům, lze kategorizovat po
Práce s pozicemi z trackerů	3 – pozice musí být ručně vloženy ve formátu KML
Práce s uživatelsky vloženými pozicemi	1 – vysoká míra personalizace, v podstatě universální
Přehlednost dat	2 – mapová data lze ukládat vysoce přehledně za použití person
Přenositelnost a sdílení dat	2 - data lze jednoduše vyexportovat do standardního formátu K

Tabulka 2.3: Hodnocení Google MyMaps

Výhody

- vysoká míra personalizace,
- jednoduchost softwaru,
- standardizovaný formát pro export i import dat (KML),
- napojení na ekosystém Google Drive.

Nevýhody

- zaměřeno především na objekty v mapě,
- složité vytváření struktur nad objekty v mapě (taxonomie, metainformace),

2.4 Specializovaný software

Specializované softwarové produkty vyvíjené přímo pro doménu ornitologie mohou kombinovat přístupy GISových aplikací i tabulkových procesorů, případně i dokumentových nástrojů.

2.4.1 Movebank

Movebank je specializovanou aplikací vyvinutou Max Planck institutem v Německu. Aplikace se zabývá ukládáním dat o zvířatech, ať již pozičních, senzorických i vygenerovanými uživateli (např. pozorování). Aplikace není vyvinuta primárně pro ornitologické potřeby, ale její datový model obsahuje desítky datových položek vhodných pro ornitology. Značnou výhodou je podpora výrobců trackerů pro kontinuální export dat přímo do Movebank a možnost data z Movebank exportovat, či využít v doménově specickém softwaru. Za nevýhodu pro některé ornitology může být nutnost data uveřejnit, případně o ně po 90 dnech přijít. Movebank taktéž obsahuje funkce pro správu projektů a publikování studií a je mezi ornitology často citovaným zdrojem.

Kritérium	Hodnocení
Informace o jedincích	1 –
Práce s pozicemi z trackerů	1 – aplikace přímo optimalizovaná pro tyto účely
Práce s uživatelsky vloženými pozicemi	2 – ?
Přehlednost dat	1 – platforma uzpůsobena pro prohlížení dat
Přenositelnost a sdílení dat	2 – data jsou vysoce přenositelná a dobře sdílená, problémem však

Tabulka 2.4: Hodnocení Movebank

Výhody

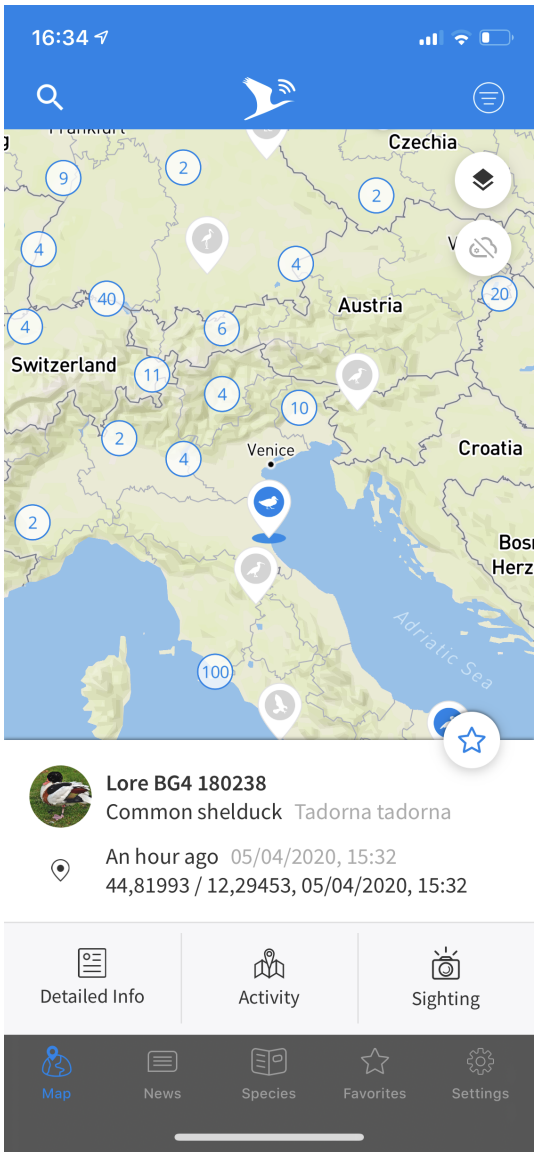
- doménová specializace,
- dlouholetý vývoj aplikace – komplexní datový model,
- komplexní možnosti publikování dat,
- dobrá integrace s doménově specifickými nástroji

Nevýhody

- nutnost data vždy v určité formě publikovat.

2.4.2 Movebank Animal tracker

Od tvůrců aplikace Movebank pochází i mobilní aplikce *Animal tracker*, která umožňuje zobrazení tras od jednotlivých trackerů i zobrazení informací o zvířeti, co tracker nosí. Aplikace nevyžaduje žádné přihlašovací údaje a umožňuje veřejnosti nahlásit pozorování konkrétních jedinců. Za největší slabinu aplikace se dá považovat nutnost připojení k internetu, aplikace si neukládá informace do mezipaměti. Pro hledání jedinců v oblasti nižší kvality signálu tedy aplikace není vhodná. Aplikace je ale jinak velmi přehledná i rychlá. Pro aplikaci nebude uvedeno vlastní hodnocení, jelikož se jedná o rozšíření zmíněné aplikace Movebank pro mobilní zařízení.



Obrázek 2.1: Screenshot z aplikace Movebank Animal Tracker

2.4.3 Anitra

Anitra je českou ornitologickou značkou zabývající se vývojem a produkcí vlastních GPS-GSM trackovacích zařízení a poskytování softwarové platformy pro zobrazení dat a správu projektů. Rozdílem oproti platformám jiných výrobců je od základu jiná koncepce platformy. Platforma poskytovaná firmou Anitra průběžně integruje data z GPS-GSM trackerů od více výrobců a umožňuje nad daty dělat analýzy, automaticky je zpracovat dle předem stanovených pravidel, spravovat informace o jedincích, zadávat do aplikace vlastní body a přílohy. Od předem zmiňovaného Movebanku se aplikace liší především v možnosti ochranně vlastnictví dat, v aplikaci Anitra není povinnost data kdykoliv publikovat a majiteli dat zůstávají uživatelé. Aplikace taktéž umožňuje generovat uživatelsky přívětivé výstupy pro veřejnost či filtrovaná nebo plná data jednoduše sdílet mezi uživateli. Za podstatnou nevýhodu a nutnost využívat alternativní způsob zápisu v poli je především fakt, že platforma Anirta je řešena jako webová

aplikace. V mobilních zařízeních je tedy obtížněji použitelná, ale hlavní nevýhodou je nutnost být stále připojen k internetu.

Kritérium	Hodnocení
Informace o jedincích	1 – detailní schéma vhodné pro ukládání informací o zvířatech
Práce s pozicemi z trackerů	1 – aplikace přímo optimalizovaná pro tyto účely
Práce s uživatelsky vloženými pozicemi	1 – modul aplikace věnovaný pro vytváření těchto objektů
Přehlednost dat	1 – data lze zobrazit v mapě, tabulce, vyexportovat
Přenositelnost a sdílení dat	1 – vlastník může vždy data vhodně vyexportovat

Tabulka 2.5: Hodnocení Movebank

Výhody

- doménová specializace,
- dynamický rozvoj aplikace,
-

Nevýhody

- webová aplikace není vhodná pro použití v terénu,
- platforma není mezi ornitology příliš rozšířená,
- proprietární a nezdokumentované API.

3. Analýza požadavků

Kapitola analýza požadavků se věnuje první fázi ve vývoji softwarového projektu. Analýza požadavků se skládá z různých fází dle metodiky, ale vždy je cílem od relevantních stran získat seznam požadavků, co aplikace musí splňovat, aby naplnila cíle, kvůli kterým se k vývoji SW projektu rozhodlo.

3.1 Identifikace stakeholderů

Identifikace stakeholderů je první fází v analýze požadavků. Účelem této fáze je identifikovat všechny relevantní zůčastněné strany, aby mohly být zapojeny do fáze sběru požadavků. V případě nedostatečného zapojení stakeholderů je vysoká pravděpodobnost, že software nebude uspokojovat potřeby všech zúčastněných stran.

Za stakeholdery byly identifikovány následující subjekty: stávající uživatele webové aplikace Anitra a majitel firmy Anitra System s.r.o.

Stávající uživatelé aplikace již požadovali funkce pro práci v terénu a jejich požadavky byly zaevidovány do pořadníku funkcí pro webovou aplikaci. Tyto požadavky byly konzultovány se zakladatelem firmy Anitra a byly vybrány k řešení. Zakladatel firmy je projektovým manažerem a komunikuje se zákazníky na denní bázi, má tedy dostatečný přehled o požadavcích zákazníků a jejich priorit. Cílem zakladatele firmy je poskytnout uživatelům nástroje pro efektivní práci v terénu i pro rychlou kontrolu dat z mobilních zařízení, což by mohlo být bráno jako konkurenční výhoda, jelikož na trhu nebyla nalezena podobná aplikace.

3.2 Sběr požadavků

Pro zakladatele firmy Anitra je cílem mobilní aplikace doplnění ekosystému značky Anitra o vhodné řešení zobrazení a zadávání dat v terénu. Na začátku psání práce byla pouze dostupná webová aplikace, která měla pro práci v terénu následující nevýhody:

- data po odpojení z internetu nebyla dostupná, ve většině případů ani již načtená data,
- aplikace byla příliš náročná na energii,
- mapové podklady se nezobrazovaly po odpojení ze sítě,
- aplikace byla na datový přenos příliš náročná,
- navigace v GUI byla pro malá zařízení příliš složitá.

Tyto podněty byly sebrány od uživatelů webové aplikace Anitra v období 2018-2020, ale mělo by se jednat o dostatečný základ pro zařazení mobilní aplikace do portfolia ekosystému Anitra, jelikož se jedná o obecné předpoklady pro software tohoto typu.

Z těchto zkušeností uživatelů vyplynuly klíčové funkční i nefunkční požadavky na funkcionalitu aplikace. Jednotlivé části jsou rozebrány níže. Další funkční požadavky byly vydefinovány zakladatelem firmy Anitra (zde uvedeny ve zkrácené formě):

- zobrazení dat ze zařízení,
- zaznamenat trasu, bod,
- možnost přiložit obrázek k zvířeti,
- notifikace o nových datech,
- zobrazení vlastní pozice na mapě.

3.3 Funkční požadavky

Funkční požadavky popisují očekávané chování systému, ale nevěnují se technickým detailům, jak má systém fungovat. Funkční požadavky slouží pro splnění potřeby uživatelů a lze je vyjádřit ve formě případů užití nebo diagramu případu užití.

// TODO: Use case diagram z Astah :)

Jednotlivé případy užití jsou rozebrány v kapitole uvedené níže.

3.3.1 Diagram případů užití

Diagram případu užití je diagramem ze stadardní sady UML využívané v softwarovém inženýrství.

3.4 Nefunkční požadavky

Nefunkční požadavky udávají kvalitu systému, ale ne jeho chování. Tyto požadavky často přináší omezení do implementací funkčních požadavků a je někdy obtížné tyto požadavky sladit. Jako příklad se často uvádí požadavky spojené s komfortem uživatelů (např. přístupnost, výkonnost, lokalizace), provozem aplikace (přenositelnost, tolerance chyb) a bezpečnost. Do nefuknčních požadavků se řadí i požadavky související s možností dalšího vývoje systému.

Z výše uvedeného seznamu požadavků vyplynulo několik klíčových nefunkčních požadavků především ve vztahu k uživatelům. Pro úspěch aplikace je nutné, aby grafické rozhraní aplikace bylo dostatečně jednoduché a podstatné funkce byly rychle přístupné bez složitých navigací, jelikož uživatel nemusí mít mnoho času pro práci s aplikací v terénu. S tímto souvisí i nutnost odezvy v jednotkách sekund.

Neuvedeným požadavkem je multiplatformnost aplikace, jelikož část uživatelů stávající aplikace využívá mobilní operační systém iOS a část Android. Podstatným požadavkem je i

rozšiřitelnost aplikace, jelikož se s vývojem aplikace počítá i po dokončení této práce.

3.5 Časové aspekty

Aplikace má být vyvinuta a nasazena nejpozději do konce května, ideálně již v dubnu. V tuto dobu probíhá nejvyšší počet ornitologických operací spojených s nasazováním trackerů, kontrolou hnízd a dalších. Aplikaci nebylo možné začít vyvjíet dříve než v polovině února kvůli návaznosti na API již existující webové aplikace. Pro prvotní verzi aplikace je kritické aby spolehlivě zobrazovala poslední body v mapě a fungovala bez internetového připojení.

4. Technologie

Kapitola technologie se zabývá využitými technologiemi a jakým způsobem byly vybrány. Jednotlivé technologie jsou představeny pro porozumění následující kapitole návrh, který byl proveden s ohledem na vybrané technologie.

Pro výběr technologií byla stanovena následující kritéria:

- podpora platforem iOS a Android,
- rychlost vývoje,
- velikost komunity a aktuálnost dokumentace,
- programovací jazyk,
- jednoduchost vytváření UI,
- dostupnost potřebných komponent pro aplikaci.

Jednotlivá kritéria jsou rozebrána níže.

4.1 Kritéria výběru

4.1.1 Podpora více platforem

Podpora platforem iOS a Android je kritickým požadavkem aplikace vycházející z analýzy požadavků. Technologie musí podporovat, ideálně platformně nezávisle abstrahovat, přístup k zdrojům souborového systému, přístup k vzdáleným zdrojům pomocí protokolu HTTP, umožnit vytvářet mapové aplikace, bezpečně uložit uživatelské přihlašovací údaje. Technologie by neměla vyžadovat platformně závislý nativní kód.

4.1.2 Rychlost vývoje

Rychlost vývoje je obtížně kvantifikovatelnou veličinou. Do rychlosti vývoje lze zahrnout mnoho aspektů a záleží, z jakého pohledu se na problematiku pohlíží. Z pohledu řízení projektu se může např. jednat o počet dostupných vývojářů a doba adaptace na technologii. Z pohledu programátora se může jednat o kvalitu a dostupnost vývojových nástrojů, dostupnost knihoven, přehlednosti kódu, dostupné funkce programovacího jazyka technologie. Z pohledu údržby se může jednat o očekávanou životnost technologie nebo dostupnosti testovacích nástrojů. Kombinací těchto aspektů může vzniknout více či méně subjektivní hodnocení, jak rychlý by měl vývoj s pomocí danou technologií být. Neexistuje žádná konkrétní metodika, která by se snažila tento jev kvantifikovat, ale existují názory mezi odbornou veřejností, které technologie se považují za vhodné pro rapidní vývoj. Pro tuto práci, z hlediska časového, bude nutné vybrat technologii, která naplňujne co nejvíce kritérií pro rapidní vývoj.

4.1.3 Velikost komunity

Velikost komunity určuje dvě podstatné věci – životnost technologie a množství dostupné aktuální dokumentace a komunitních návodů. Pro dlouhodobé hledisko je potřeba vybrat technologii, která se dle současného stavu zdá dlouhodobě perspektivní a ne na úpadku. Dostupná kvalitní oficiální i komunitní dokumentace je taktéž podstatným faktorem, který technologii zpřístupňuje pro nové projekty. Tato veličina je již kvantifikovatelná např. analýzou trendů či sledováním open-source projektů v daných technologiích. Pro určení této veličiny bude provedeno srovnání dle Google Trends a hlavních repozitářů

4.1.4 Programovací jazyk

Programovací jazyk technologie je taktéž podstatný faktor. Standardní knihovny některých jazyků mají mnoho dostupných funkcí, které urychlují nebo komplikují vývoj. např. již předpřipravené metody pro HTTP komunikaci. Některé jazyky mohou těžit i z vlastností dané svým paradigmatem, např. funkcionální jazyky mohou být vhodnější pro paralelní zpracování. Aspekty paralelního nebo asynchronního zpracování jsou pro mobilní aplikace podstatné z důvodu nezamknutí vykreslovacího vlánka pro uživatelské rozhraní z důvodu uživatelského komfortu.

4.1.5 Vytváření uživatelského rozhraní

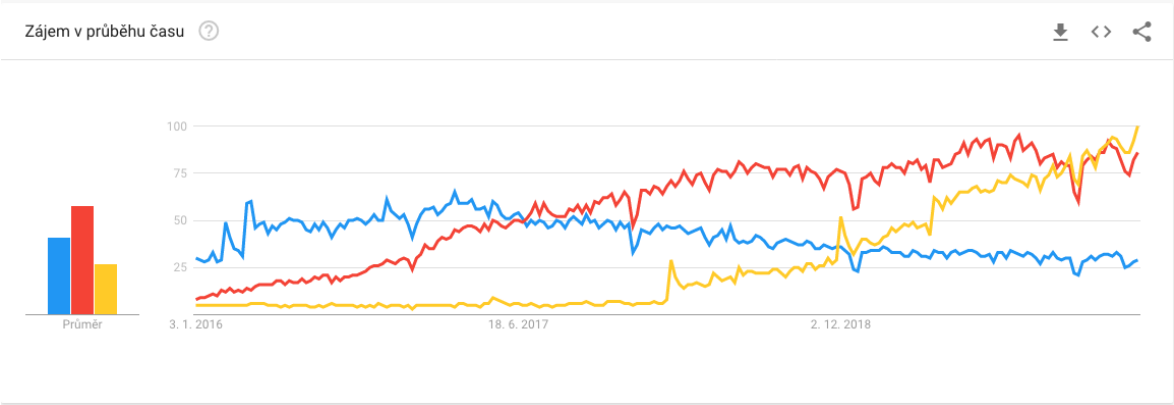
Aspekty vytváření uživatelského rozhraní spočívají především v možnostech technologie. Technologie by měla umožnit vytvářet reaktivní a moderně vypadající uživatelská rozhraní, podporovat animace, reagovat na uživatelské interakce. Na těchto principech je založen například Material Design od společnosti Google. Výhodou nástroje pro vytváření uživatelského rozhraní je moderní přístup k vytváření uživatelského rozhraní např. s tzv. *flexboxem*, nástrojem pro jednodimenzionálně řazená uživatelská rozhraní s rožšířenou podporou zarovnání (*Basic Concepts of Flexbox*, 2019). Nativní podpora stylování v podobě webových CSS je taktéž vyhodou, jelikož ji nezanedbatelná vývojářů do jisté míry rozumí.

4.1.6 Hotové komponenty

Hotové komponenty umožňují se při vývoji věnovat pouze řešení potřebných doménových problémů, namísto vytváření od základu nových komponent a s tím spojené testování. Důsledkem je využívání specifického portfolia komponent, na kterých může vývojář nabrat znalosti a použít je na dalším projektu využívající stejnou technologii, případně vývojář již zkušenosti mít může, tím vývoj výrazně urychlit. Hotové komponenty jsou obrovskou výhodou v rapidním prototypování, protože mohou aplikaci již ve fázi prototypu poskytnout určitou představu o finální funkčnosti.

4.2 Výběr technologie

Pro užší posouzení byly vybrány následující technologie: React Native, Xamarin a Flutter. Technologie jsou ve zkratce popsány níže a zhodnoceny.



Obrázek 4.1: Srovnání trendů v období 1. 1. 2016 až 11. 4. 2020; červená křivka – React Native, modrá křivka – Xamarin, žlutá křivka – Flutter; – Zdroj: Google Trends

4.2.1 Hodnocení React Native

React Native je framework pro vytváření nativních mobilních aplikací s JS frameworkem React. React Native je uveden v další kapitole. Tento framework byl vybrán k posouzení především kvůli obrovské komunitě a prověřenosti. Nespornou výhodou frameworku využití jazyku JavaScript, který je třetím nejvíce vyhledávaným programovacím jazykem. JavaScript (resp. EcmaScript) je uveden v následujících kapitolách a nebudou zde jeho výhody rozvedeny. Pod pojmem React Native je zde brána v potaz nadstavba Expo, která dále abstrahuje od nativního kódu.

Kritérium	Hodnocení	Poznámka
Podpora platforem	2	plně podporuje obě mobilní platformy, ale některá API j
Rychlost vývoje	1	kombinace jednoduchého jazyka, dobrých nástrojů a dos
Velikost komunity a dokumentace	1	z obrázku 4.1 vyplývá obrovská popularita frameworku v
Programovací jazyk	1	
Vytváření UI	2	
Hotové komponenty	1	24299 balíčků [1]

Tabulka 4.1: Hodnocení React Native

4.2.2 Hodnocení Xamarin

Xamarin je rozšíření platformy .NET pro vývoj multiplatformních aplikací nejen pro mobilní zařízení. Xamarin je postavený nad návrhovým vzorem Model-View-ViewModel, který odděluje uživatelské rozhraní, data a aplikační logiku do separátních vrstev. Uživatelské rozhraní

je možno vytvářet deklarativně v jazyku XAML, případně v přímo v jazyku C#. Změny v uživatelském rozhraní se zajišťují pomocí tzv. data bindingu, tedy navázání proměnné v uživatelském rozhraní s proměnnou ve view modelu. Za Xamarinem stojí technologický gigant Microsoft a je z vybraných technologií nejstarší.

Kritérium	Hodnocení	Poznámka
Podpora platforem	2	
Rychlost vývoje	3	
Velikost komunity a dokumentace	3	
Programovací jazyk	2	
Vytváření UI	2	
Hotové komponenty	4	

Tabulka 4.2: Hodnocení Xamarin

4.2.3 Hodnocení Flutter

Flutter je SDK od společnosti Google sloužící pro vytváření "nádherných, rychlých zážitků pro mobil, web a desktop s jedním zdrojovým kódem". Flutter je z vybraných technologií nejnovější, ale výrazně nejrychleji roste. Za obrovskou výhodu Flutteru se dá považovat vývojové prostředí s funkcemi live-reload, které na rozdíl od vývojových prostředí pro Xamarin a React Native má výrazně lepší zapamatování stavu. Flutter byl přímo navržen pro aplikace tohoto typu a obsahuje tedy mnoho optimalizací. Flutter aplikace jsou taktéž výrazně menší na stáhnutí, než aplikace React Native nebo Xamarin. Za nevýhodu Flutteru se dá považovat pouze jazyk Dart, který na rozdíl od C# nebo JS není stále dostatečně viditelný ve vývojářské komunitě. Podstatnou nevýhodou je nízký počet komponent, vývojáři si tedy musí značnou část komponent psát sami.

Kritérium	Hodnocení	Poznámka
Podpora platforem	1	
Rychlost vývoje	1	
Velikost komunity a dokumentace	3	
Programovací jazyk	4	
Vytváření UI	1	
Hotové komponenty	3	8450 balíčků [2]

Tabulka 4.3: Hodnocení Flutter

Výhody

- dynamická technologie,
- vysoce kvalitní vývojářské nástroje,
- vyvíjeno stejnou společností, která stojí za OS Android,
- vysoká míra optimalizace,

- možnost vytvářet i webové aplikace.

Nevýhody

- relativně obskurní jazyk Dart,
- nížký počet aktuálních návodů,
- nížký počet relevantních komponent,
- obtížnější zprovoznění + velikost SDK,
- komunita nemá stále stejnou velikost jak komunita React a React Native.

4.2.4 Vybraná technologie

Pro řešení práce byla vybraná technologie React Native (resp. Expo), jelikož ze srování vyšla nejlépe. Dále bylo rozhodnuto o využití nadstavby TypeScript na JS a správce stavů MobX. Popis technologie je uveden v další kapitole.

4.3 React Native

Technologie React Native je postavena na mnoha dílčích částech, které se musí uvést pro komplexní pochopení React Native samotného.

4.3.1 JavaScript a EcmaScript

JavaScript je multiparadigmatický interpretovaný slabě typovaný programovací jazyk určen primárně pro webové prohlížeče. JavaScript vznikl v roce 1995 Brendanem Eichem pod firmou Netscape. Cílem bylo přidat interaktivitu do čistě statických webových stránek a reagovat tedy na interakci uživatele i jinak, než jen přenačtením celé stránky. S obrovským rozvojem domacích počítačů přišel i rozvoj internetu a internetových prohlížečů. Od již zmíněného roku 1995 do roku 2004 byl trh webových prohlížečů dominován verzemi prohlížeče Internet Explorer od firmy Microsoft, která přišla s vlastní implementací JavaScriptu, tzv. JScriptem. Kvůli nespolupráci mezi vývojáři různých implementací vznikaly v jazyku nekonzistence mezi prohlížeči, ale i přesto se jazyk určitým směrem ubíral. V roce 2005 byl zaveden termín AJAX, který znamenal obrovskou změnu v interaktivitě webových aplikací. Zkratka AJAX, neboli Asynchronous JavaScript and XML byla revoluční ve změně komunikace serveru s JavaScriptem. JavaScript dříve obdržoval fragmenty HTML od serveru a tím aktualizoval stránku, případně se stránka po každé významné interakci uživatele překreslila sama. AJAX zavedl možnost jak na pozadí získat z určitých vstupů určité výstupy v lépe strojově zpracovatelném formátu. S tímto revolučním nápadem začaly vznikat první JavaScript knihovny a frameworky pro interaktivní webové *aplikace*, např. dodnes používaná knihovna jQuery. Dalším podstaným krokem byl vývoj webového prohlížeče Google Chrome, resp. jeho JavaScript interpreteru V8. V8 je open-source JS interpreter s just-in-time kompilací, který byl revoluční

primárně ve své rychlosti. Tento interpreter byl také zařazený i do aplikací mimo webové prohlížeče, čímž se případy užití pro jazyk výrazně rozšířily. Zde za zmínku stojí node.js (vznik datovaný do roku 2009) a jeho ekosystém NPM, který je rozveden níže. Dalším milníkem podstatným pro tuto práci je rozšíření frameworků pro single-page aplikace, které emulují chování běžných nativních aplikací v prohlížeči. Aplikace fungují na komponentovém systému a aktualizují se při změně dat. Aplikace jsou tedy uživatelsky i programátorsky komfortnější a jejich vývoj je značně rychlejší. Aplikace tohoto typu se rozšiřují od roku 2010.

EcmaScript je specifikací jazyka JavaScript, kterou musí každé prostředí implementovat. EcmaScript definuje pouze jazyk, ale ne objekty kontextu, např. ve webových prohlížečích není definována interakce s DOM. JavaScript je ale dnes používán i na serverech, embedded zařízeních či jako skriptovací jazyk mnoha programů, bylo nutno od těchto specifikací abstrahovat, aby základ jazyka zůstal stejný. Tímto ale vznikají jisté nekonzistence mezi implementacemi v různých prohlížečích, obzvláště napříč verzemi, kde se jádro interpreteru již neaktualizuje. Pro řešení těchto problémů byly mimojiné zavedeny tzv. *polyfilly*, které doplňují určité funkce z nové verze zpětně do staré a *transpilátory* (např. Babel), které některé nové jazykové funkce umí převést do verze ES kompatibilní se starým interpreterem. V čase psaní této práce je aktuální verzí EcmaScript 2018.

S příchodem node.js ekosystému vznikl v JS světě koncept NPM balíčků. Balíčky jsou publikovány autorem do centrálního repozitáře balíčků a každý programátor si tento balíček může nainstalovat. Balíčky jsou typicky knihovny, malého či velké rázu, frameworky, nástroje, samostatné programy či kompletní projekty. Koncept balíčků výrazně urychluje vytváření projektů a instalaci knihoven. Příklady práce s NPM jsou uvedeny v kapitolách React a React Native.

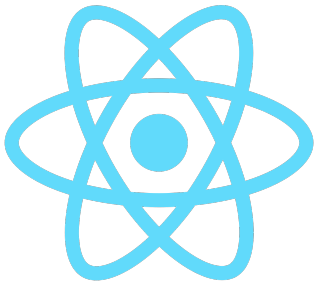
4.3.2 TypeScript

TypeScript je nadstavbou jazyka ECMAScript. TypeScript rozšiřuje jazyk o kontrolu typů a běžné konstrukty známé z jiných jazyků (např. rozhraní). Jazyk je plně kompabitilní s ES, ale samotný TypeScript není přímo většinou interpeterů spustitelný. Při sestavování projektů s TypeScriptem se využívá tzv. transpilace, převedení z jednoho programovacího jazyka do druhého. TypeScript vyšel v roce 2012 pod křídly firmy Microsoft a je aktivně vyvíjen dodnes.

TypeScript se běžně využívá ve větších projektech především kvůli prevenci chyb a zvýšení přehlednosti kódu. Vývojová prostředí taktéž lépe rozumí typovaným jazykům a tímto se může i zrychlit vývoj. Zaintegrovat TypeScript do existujícího projektu také není problém, jelikož jakýkoliv validní kód v ES je validní kód v TypeScript.

4.3.3 React

React je JS knihovna pro vytváření uživatelských rozhraní (ref) od společnosti Facebook. React je deklarativní, component-based knihovna s velkým spektrem potenciálních využití.



Obrázek 4.2: Logo React – zdroj: Facebook Inc.

Jako jiné moderní UI knihovny v JS je taktéž reaktivní, při změně dat se překreslí uživatelské rozhraní pouze tam, kde musí. Deklarativnost Reactu spočívá právě v zakládání si na komponentách. Uživatelské rozhraní se skládá z předdefinovaných, uživatelem vytvořených nebo komunitou vytvořených komponentách, které se do sebe skládají. Každá komponenta si uchovává vlastní data (popř. stav v atributu props) a může komunikovat s nadřazenou či podřazenou komponentou pomocí událostí nebo předáváním dat. Komponenty jsou v Reactu definovány pomocí speciální kombinace JS a HTML, tzv. JSX. JSX je vždy obaleno elementem, který může být HTML elementem či jinou komponentou. V JSX se na rozdíl od běžného HTML může vyskytovat aplikační logika, uvozena v složených závorkách. Níže je uvedena kompletní komponenta z oficiální dokumentace (ref!), která při využití této komponenty `<HelloComponent name="world"/>` vypíše do div tagu Hello world.

```
1
2  class HelloMessage extends React.Component {
3    render() {
4      return (
5        <div>
6        Hello {this.props.name}
7        </div>
8      );
9    }
10 }
```

Procedura 4.1: React komponenta

Případně pro ilustraci podmíněné logiky lze uvést následující příklad:

```
1
2  class ConditionalComponent extends React.Component {
3    render() {
4      return (
5        <div>
6          {this.props.sayHello?
7            <span>
8                Hello {this.props.name}
9            </span>
10           :<span>
```

```
11                Goodbye {this.props.name}
12            </span>
13          }
14        </div>
15      );
16    }
17  }
```

Procedura 4.2: React komponenta s podmínkou

Příklad využívá slabého typování JS, stačí tedy pouze aby sayHello bylo pravdivé (případně *truthy* – aby se alespoň evaluovalo na pravdu) a provede se správná část JSX. Podmínka je zde zapsána ve formě ternárního operátoru, klasická if struktura by musela být ve vlastní funkci.

Komponenty mohou samozřejmě být výrazně komplexnější, např. mohou obsahovat funkce vracející JSX a tím být výrazně přehlednější. Mimo JSX mohou obsahovat metody a atributy pro práci s daty, např. pro získání dat ze serveru nebo k vyfiltrování dat. Tyto funkce mutují stav komponenty pomocí funkce `setState`, která mění stav komponenty (`this.props.*` v ukázce) a zároveň zajistí její překreslení, pokud je to nutné. Změna jména by tedy mohla být docílena i zavoláním `this.setSate({name: 'svete'});` z metody v komponentě.

React aplikce se ale nemusí skládat jen z komponent, modulární systém nového JS umožňuje jednoduše zakomponovat do aplikace i kód mimo komponenty. Komponenty ale stále zůstanou centrálním bodem aplikace a je tedy výhodné aplikaci strukturovat do většího počtu malých komponent kvůli přehlednosti, testovatelnosti a znovupoužitelnosti.

4.3.4 React Native

React Native je framework pro vytváření mobilních aplikací postavený nad knihovnou React, opět od společnosti Facebook. React Native, na rozdíl např. od progresivních webových aplikací se chová jako nativní aplikace – aplikace je vykreslovaná nativním kódem a nemá omezení typická pro webové aplikace, např. v přístupu k určitým API. Rozdílem od čistého Reactu je omezený výběr komponent a výrazně odlišný způsob jejich stylování. Zatímco v React je možno využít plného rozsahu CSS a všech HTML elementů, v React Native jsou v základu poskytnuty jen určité komponenty, nejpodstnější z nich jsou View, Text a Image. View je kontejnerovým prvkem, který nemá žádné specifikované využití. Používá se často pro držení jiných komponentů či stylování. Text je jediná komponenta, která může obsahovat text. Image je komponenta obsahující obrázek z lokálního nebo vzdáleného zdroje. Jsou poskytnuty i další komponenty pro formulářová pole (TextInput, Button, Picker), indikátory aktivity a seznamy. Jiné komponenty se dají složit z těchto základních komponent, jejich událostí a jejich stylů.

React Native umožňuje psát nativní kód a mnoho komponent této skutečnosti využívá. Nevýhodou tohoto přístupu je, že nativní kód se musí psát v Javě pro Android a v jazyce Swift

pro iOS. Tímto vzniká prakticky vzato duplicita kódu a dvojnásobná nutnost testovat.

4.3.5 MobX

MobX je knihovna pro reaktivní správu stavu aplikace (ref) . Knihovna zavádí koncept pozorovatelného stavu, tedy že se na objekt či kolekci objektů naváže pozorovatel, který automaticky zajišťuje změnu stavu pro React při změně pozorovaného. Tímto se výrazně zjednodušuje vývoj React Native aplikací tím, že se pro každou změnu stavu nemusí volat setState a nemusí se využívat pouze proměnné props. Na rozdíl od častěji využívané knihovny Redux je knihovna výrazně úspornější, přičemž obě knihovny zajišťují, že data tečou ze zdroje k cíli pouze jedním směrem a že na správné změny komponenty reagují.

4.3.6 Expo

Expo je součástí nástrojů a služeb postavených nad React Native a nativními platformami sloužící k vývoji, sestavení, nasazení a rychlé iteraci iOS, Android a webových aplikací s jednotným zdrojovým kódem (ref). Expo je abstrakce oddělující React Native kompletně od nativního kódu a tedy od nutnosti vyvíjet i ReactNative aplikace ve třech jazycích (JS pro ReactNative, Java pro Android a Swift pro iOS). Expo poskytuje většinu API pro běžné mobilní aplikace (podstatnou výjimkou je Bluetooth a nákupy v aplikaci), nástroje pro rapidní vývoj a cloudovou infrastrukturu pro sestavení artefaktů, není tedy nutné pro sestavení např. iOS aplikace vlastnit stroj s operačním systémem MacOS. Nástroje pro rapidní vývoj jsou obrovské plus platformy Expo – aplikační kód se odešle se na servery Expo jedním příkazem (viz ukázka kódu 4.4), uživateli se vygeneruje QR kód pro oskenování mobilní aplikací a aplikaci je možné spustit na jakémkoliv zařízení za pomocí speciální aplikace. Z tohoto zařízení je v realném čase možné sledovat běh událostí a ladících zpráv na stroji vývojáře.

Mezi nevýhody Expo se udává následující:

- nepodporuje všechny API,
- nejsou podporované všechny způsoby běhu na pozadí,
- aplikace jsou výrazně větší (min. 25 MB),
- push notifikace jsou možné pouze přes infrastrukturu Expo,
- minimální podporovaná verze je Android 5 a iOS 10,
- zdlouhavé čekaní na sestavení ve veřejné infrastruktuře.

Mezi chyby se uvádí i nemožnost využívat nativní knihovny, ale zde těžko posoudit, zda se jedná o výhodu nebo nevýhodu. Expo slibuje některé z těchto problémů vyřešit, ale jedná se o dlouhodobé problémy a žádné odhadované datum řešení není stanoveno.

Při vývoji čistě React Native aplikací a Expo aplikací není poznat téměř žádný rozdíl, jediný rozdíl je v použitých knihovnách – knihovny nesmí využívat nativní kód a pro práci s systémovými API se využívá API poskytované balíčkami Expo.

Pro vytvoření Expo aplikace stačí zadat následující kód do příkazové řádky:

```
npm install --global expo-cli
expo init mobilni-aplikace
```

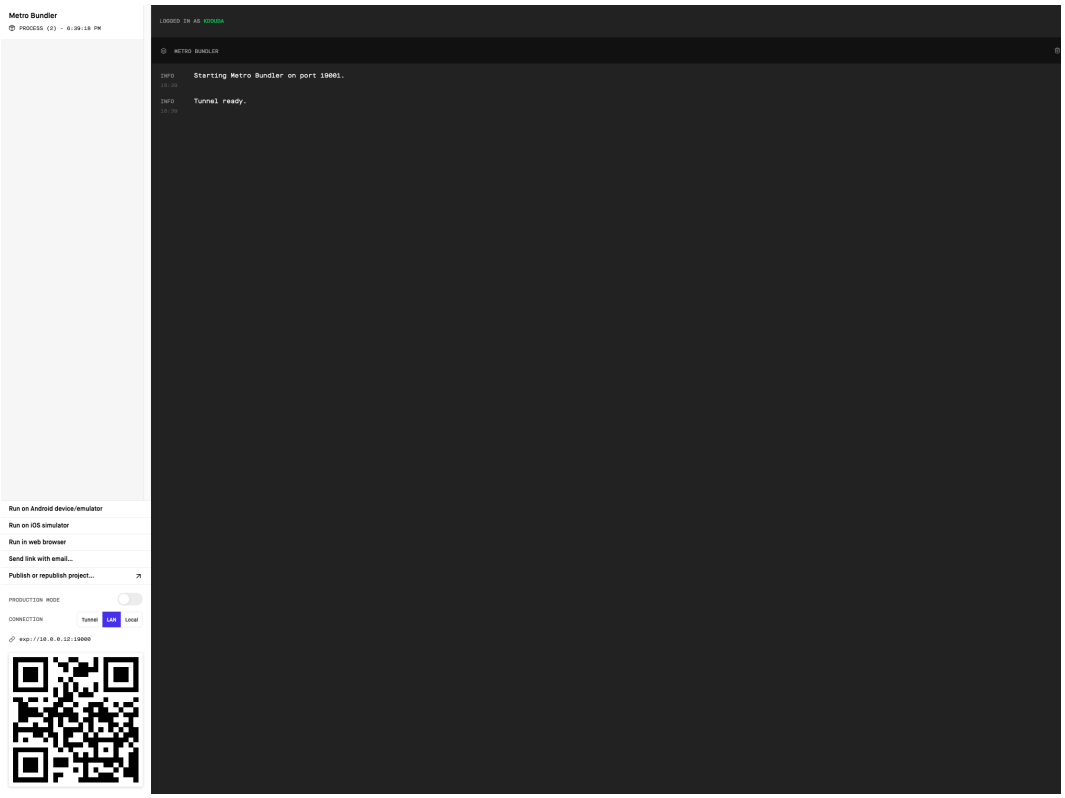
Procedura 4.3: Vytvoření základní struktury aplikace

Expo automaticky vytvoří doporučovanou adresářovou strukturu a ukázkové komponenty. Většina projektů má tedy stejný základ a vývojáři by se v nich měli lépe vyznat. Pro spuštění sestavovacích nástrojů a live-reload funkce stačí do příkazové řádky zadat:

```
expo start --android #platforma Android
expo start --ios #platforma iOS
```

Procedura 4.4: Spuštění mobilní aplikace na virtuálním stroji nebo připojeném zařízení

Po spuštění příkazu se načte ve webovém prohlížeči obrazovka obsahující log sestavovacího nástroje (Metro Bundler), ladící výstupy aplikace, seznam proveditelných akcí a QR kód pro spuštění aplikace na jakémkoliv mobilním zařízení s klientskou aplikací Expo. Z této obrazovky je taktéž možné projekt publikovat přes cloudové nástroje Expo. Alternativní způsob sestavení spustitelného artefaktu pro mobilní zařízení je popsán v následující kapitole.



Obrázek 4.3: Expo – zdroj: autor

4.3.7 TurtleCLI

TurtleCLI je nástroj pro sestavení Expo aplikací z příkazové řádky. Nástroj je určen pro uživatele, kteří si nepřejí využívat infrastruktury poskytované od Expo pro sestavování vlastních aplikací, ale přejí si aplikaci sestavit například na vlastním stroji nebo na vlastním nástroji kontinuální integrace.

Možnosti sestavení aplikací Expo jsou tedy dvojího typu, cloudové služby poskytované od Expo a TurtleCLI na vlastní infrastruktuře. Expo poskytuje cloudové služby zdarma ve variantě Community a placené prioritní zpracování ve variantě nazývané Priority.

Autor práce vybral tuto technologi z důvodu vysoké vytíženosti veřejné infrastruktury společnosti Expo (varianta nazývaná Community), kde jeho build ve frontě čekal v jednotkách hodin, než se dostal na řadu. Společnost Expo nabízí placenou variantu svých služeb, ve kterých avizuje výrazně vyšší rychlost odbavení, než u varianty zdarma a nástroje pro týmovou práci. Za značnou nevýhodu se taktéž může brát fakt, že kód je vždy odesílán třetí straně k sestavení, což pro mnoho firem může být nepřípustné. Oproti tomu provozování na vlastní infrastruktuře skýtá mnoho potenciálních výhod, za zmínku stojí napojení na firemní CI procesy, testovací infrastrukturu, rapidní nasazení mnoha paralelních větví do testovacích kanálů, jednodušší napojení sestavení na jiné akce. Níže je uvedena tabulka popisující podstatné rozdíly mezi jednotlivými způsoby sestavení Expo aplikací.

Kategorie	Community	Priority	Vlastní infrastruktura
Cena	zdarma	29 USD / měs.	cena výpočetního výkonu
Čekací doba na začátek sestavení	cca. 1h	minuty	žádná
Složitost zprovoznění	součástí expo-cli	triviální	dle zvolené varianty, nepříliš náročné
Provázanost s Expo	vždy	vždy	žádná

Tabulka 4.4: Srovnání možností sestavení Expo aplikace

5. Návrh

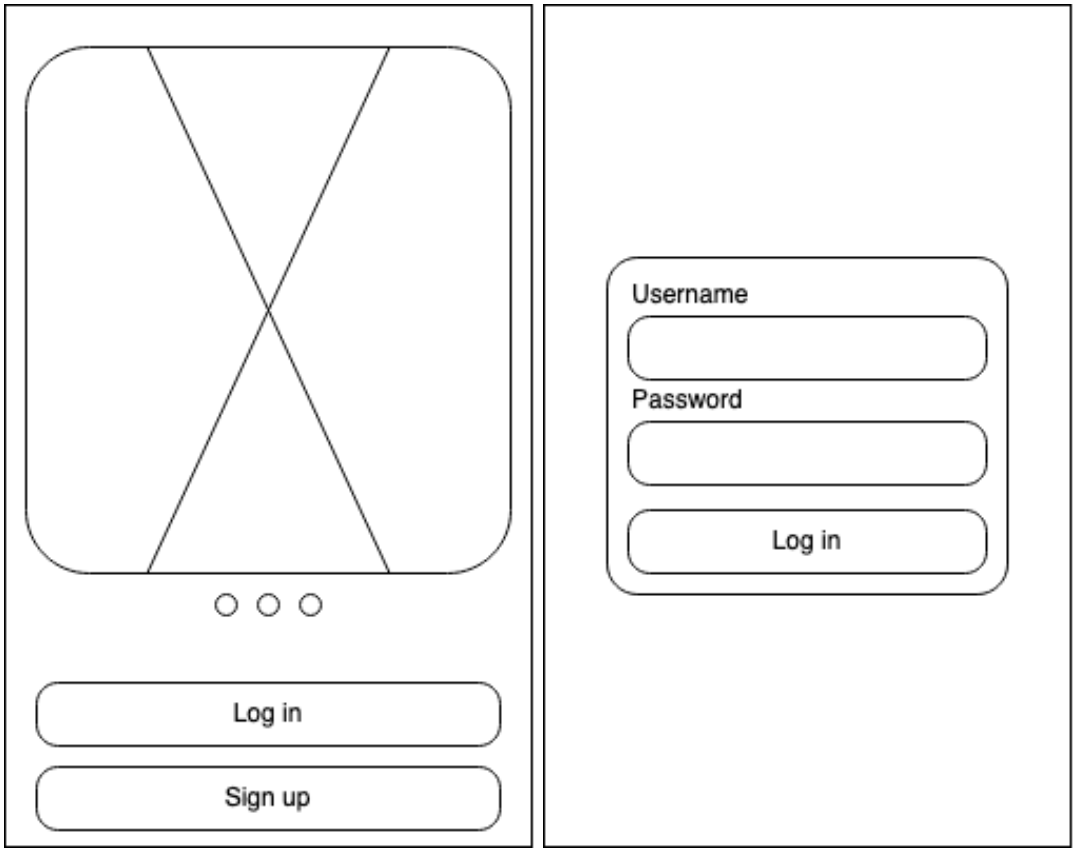
Kapitola návrh se věnuje dvou tématům. V první části se věnuje návrhu uživatelského rozhraní a přechodů mezi obrazovkami. V druhé části se věnuje tradičnějšímu chápání slova návrh ve vývoji, a to softwarové architektuře. Cílem návrhu je vytvořit postup, jakým se má při implementaci postupovat.

5.1 Uživatelské rozhraní

Návrh uživatelského rozhraní je disciplínou softwarového inženýrství, která se věnuje návrhu uživatelského rozhraní na koncepční úrovni. Při vytváření uživatelského rozhraní je nutno brát v potaz požadavky uživatelů, aby nebylo ubíráno na efektivitě aktivity uživatele v aplikaci. Cílem je navrhnout uživatelské rozhraní pro uživatele rychlo uchopitelné a přehledné. V disciplíně návrhu uživatelského rozhraní se využívá mnoho modelovacích technik, např. prototypování a drátěné modely. Pro správné posouzení efektivnosti uživatelského rozhraní je nutno tyto modely konfrontovat s potenciálními uživateli a sledovat jejich reakce a přpomínky a reflektovat je v následujících iteracích návrhu uživatelského rozhraní.

Pro tuto práci byla zvolena technika drátěných modelů (tzv. wireframe), schematickým nákresem prvků na obrazovce a jakým způsobem do sebe zapadají (ref). Tyto návrhy jsou ušetřeny komentářů, které součástí wireframů někdy bývají. V rámci této práce nebylo možno provádět studie mezi potenciálními uživateli, byl zvolen alternativní způsob inspirace známými UI elementy, které jsou přítomny v jiných aplikacích.

5.1.1 Onboarding, přihlášení, registrace



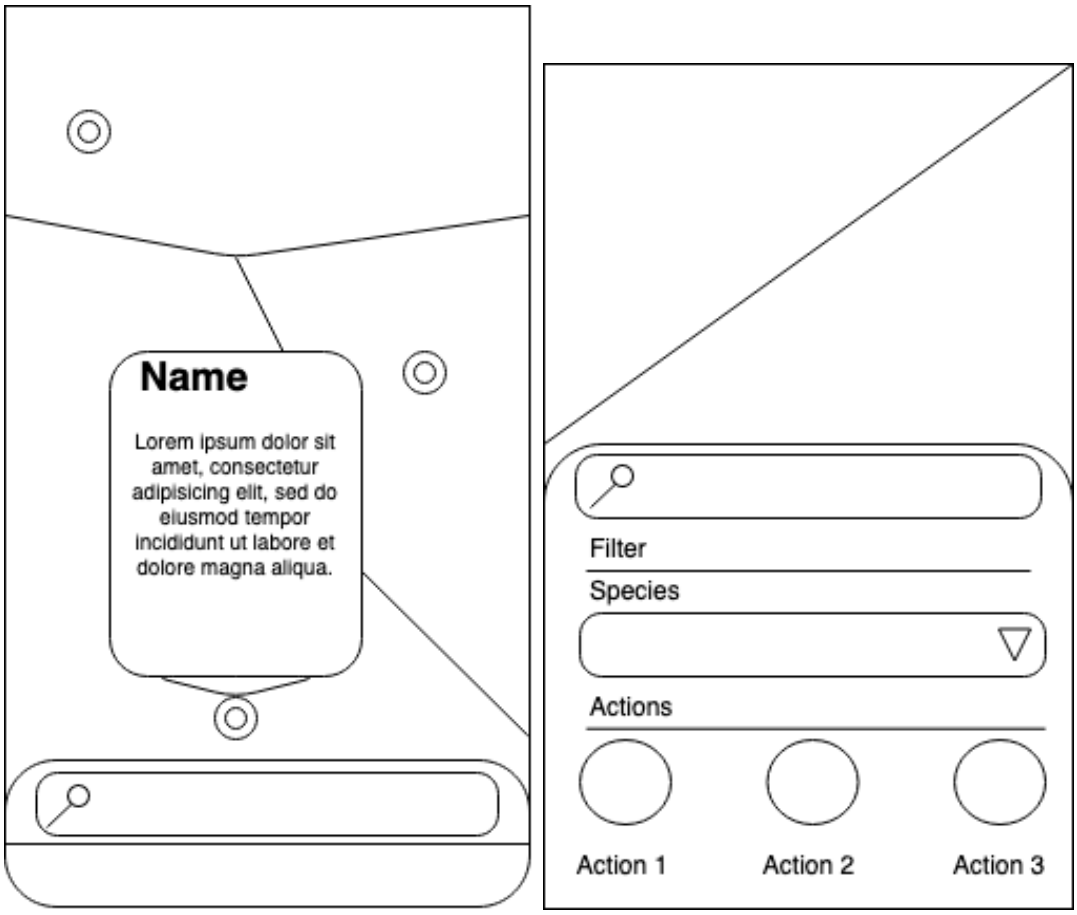
Obrázek 5.1: Wireframe onboarding a přihlašovací obrazovky – zdroj: autor

Pro přihlašovací obrazovku byl zvolen koncept tzv. onboardingu, praktiky popisující funkce aplikace a výhody založení účtu. Onboarding lze také chápat jako náhradu aplikačního návodu. Onboarding může být prováděn mnoha způsoby, jako nejsnadněji řešitelný byl vybrán tzv. carousel, seznam posouvatelných panelů, kde každý z nich obsahuje určité informace o mobilní aplikaci. Pro přihlašovací obrazku byl zvolen očekávatelný klasický formát přihlašovacího dialogu. Pro registraci byl využit tzv. web view, komponenta, co zobrazí webový prohlížeč svázaný s aplikací. Registrace se tedy provádí přes stávající webovou aplikaci. Důvodem tohoto rozhodnutí je absence API pro vytváření uživatelských účtu, ale zároveň i předpoklad, že uživatel mobilní aplikace bude i zároveň uživatelem aplikace webové.

5.1.2 Mapová koncepce

Hlavní obrazovkou aplikace byla zvolena obrazovka s mapou, aby odpovídala očekávání uživatelů webové aplikace Anitra, kde hlavní obrazovkou je taktéž mapa. Veškeré kontrolky, které by se daly zobrazit jako separátní stránky, se zobrazují v modálních oknech nad obrazovkou, což je opět koncepce vycházející z již hotové webové aplikace. Aplikace tedy rovnou splní jeden ze zmíněných případů užití při zapnutí, a to kontrolu posledních pozic trackerů

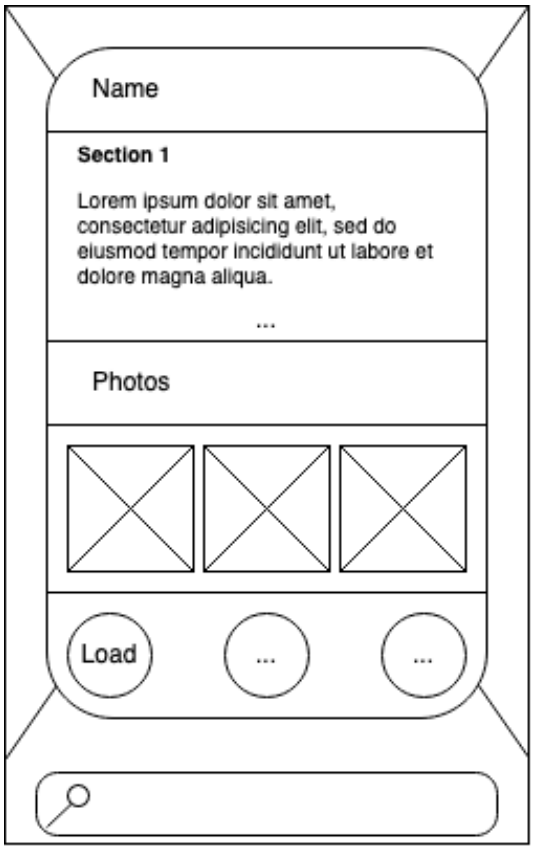
a případné odfiltrování pro uživatele nezajímavých trackerů způsoben srovnatelným s aktuálním řešením, čímž by se dalo očekávat zkrácení doby učení uživatelů aplikace. Na mapě se zobrazují individuální poslední pozice trackerů uživatele, aktuálnost poslední pozice je odstupňována dle času a barvy ikony, konzistentně s aktuálním řešením. V poslední pozici se zobrazí základní informace o trackeru – název, lokalita poslední pozice. Po klepnutí na poslední pozici se zobrazí detail objektu, ze kterého uživatel může spouštět další akce.



Obrázek 5.2: Wireframe poslední pozice a filtrování mapy – zdroj: autor

Pro navigaci v aplikaci je potřeba zavést základní menu, které uživateli umožní nejpodstatnější akce docílit v rozmezí pár uživatelských interakcí. Do aplikace bylo přidáno menu ve formě vyjížděcí kontrolky (realizované knihovnou rn-sliding-up-panel) s full-text vyhledávačem v hlavičce. Po zadání textu do vyhledávače se prozkoumají názvy všech trackerů a na mapě (či v dalších přehledových kontrolkách) zůstanou pouze relevantní trackery. Po vysunutí menu nahoru se zobrazí další možnost filtrování, a to klíčový výběr podle druhu. Filtry fungují v kombinaci ve formě logické spojky A. Pod filtrem druhu se nachází seznam akcí, které uživatel může provést.

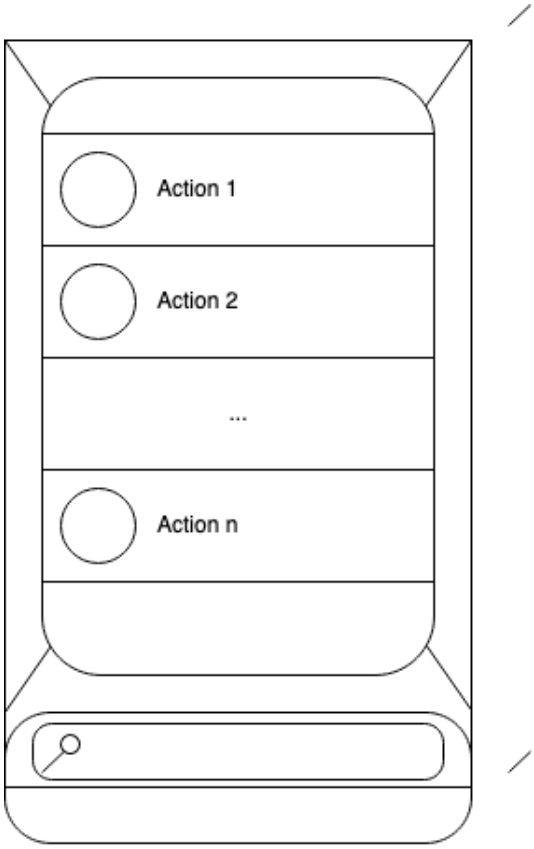
5.1.3 Detail trackeru



Obrázek 5.3: Wireframe detailu trackeru – zdroj: autor

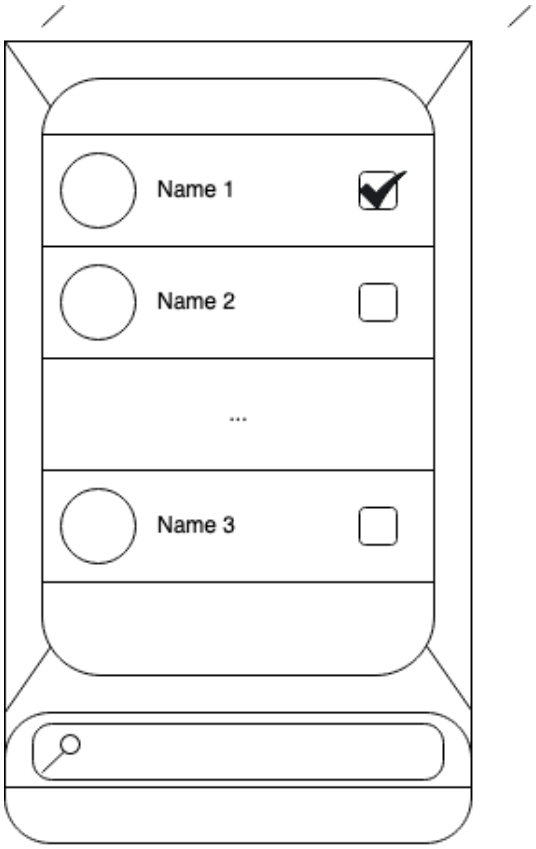
Po vybrání trackeru v mapě nebo v seznamu trackerů (viz dále) se uživateli zobrazí modální okno s popisem, základními informacemi, fotogalerií a seznamem akcí, které uživatel může s trackerem provést, v prvotní verzi aplikace pouze načíst trasu za posledních 30 dní (nebo do limitu 100 pozic). Ze stejného okna lze načtenou trasu skrýt.

5.1.4 Kontextové menu a podobrazovky



Obrázek 5.4: Wireframe kontextového menu – zdroj: autor

Kvůli způsobu provedení aplikace nebylo mnoho míst, kam přehledně uložit ovládací prvky. Některé ovládací prvky jsou také provázány s kliknutím do mapy. Z tohoto důvodu byl zvolen koncept kontextového menu nad mapou. Po dlouhém klepnutí na hlavní mapu se zobrazí seznam veškerých dostupných akcí. V seznamu jsou i globální (tedy nekontextové) akce, např. odhlášení, zobrazení seznamu trackerů, výběr mapových podkladů, stáhnutí off-line map. Kontextové menu se zobrazí v klasickém modálním okně a dá se odvolat kliknutím mimo okno nebo na jakýkoliv ovládací prvek vně.



Obrázek 5.5: Wireframe seznamu trackerů – zdroj: autor

Pro seznam trackerů byla zvolena seznamová komponenta s hlavičkami oddělující jednotlivé druhy, kvůli konzistenci s webovou aplikací. Kliknutím na položk u v seznamu se zobrazí detail trackeru a kliknutím na zaškrtávací tlačítko se načte trasa zařízení.

Stáhnutí off-line regionu map se aktivuje po kliknutí na položku v kontextovém menu, pokud je uživatel dostatečně přiblížen v mapě. Pro uživatele nemá smysl stahovat regiony výrazně větší než 1 km^2, což lze omezit právě např. minimálním tzv. *zoom levelem* mapy. Běžné rasterové (tedy především satelitní) mapy jsou klientksým softwarům indexovány v trojrozměrné struktuře ve formátu *z-x-y*, kde z udává zoom level a x a y 2D souřadnice daného mapového čtverce. Tato struktura v podstatě odpovídá datové strutkuře *quadtree*, roste tedy exponenciálně počet čtverečků ve kterých se vyskytuje vybraná plocha, zatímco velikost jednotlivých čtverečků v bytech je v podstatě srovnatelná. Pro ušetření místa na klientovi byl vybrán tedy přístup s omezením vybrané plochy nastavením minimálního zoom levelu. Uživatel tak může prakticky stahovat maximálně několik stovek mapových čtverečků, čímž by nemělo dojít k nedostatku místa na straně uživatele. Před stáhnutím bude uživateli zobrazen dialog, ve kterém se uživateli zobrazí počet čtverečků ke stáhnutí a odhadovaná velikost v bytech. Po potvrzení ze strany uživatele se zobrazí indikátor zobrazující počet státhnutých čtverečků k celkovému počtu. Při zadávání se také uživateli zobrazí na mapě již stáhnuté regiony označené červeným čtvercem.

5.2 Aplikační architektura

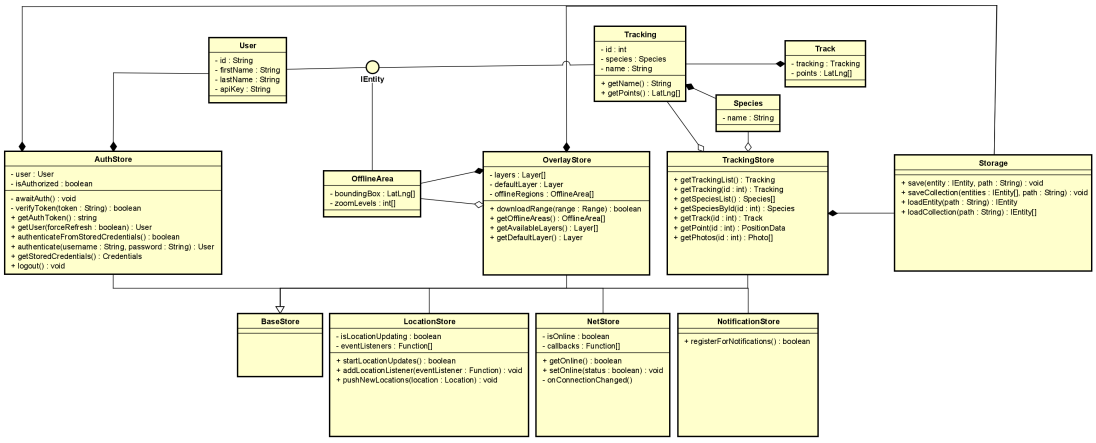
Aplikační (resp. softwarová) architektura popisuje strukturu softwarového systému. Struktura softwarového systému obsahuje softwarové elementy, jejich vztahy a vlastnosti elementů a jejich vlastností (ref). Přístupy k této disciplíně se liší dle metodiky vývoje softwaru, rigozórní metody preferují architekturu dopředu pevně formulovat, zatímco více agilní metodiky pravý opak. V rámci této práce softwarová architektura nebyla řešena příliš do podrobna, jelikož se jedná o velmi malou aplikaci a úmyslem bylo dodat produkt co nejdříve. Cílem bylo navrhnout koncept bez zbytečných duplikacích kódu, možností centralizovaného čtení a zápisu do cache, volání API z jednoho místa a přehlednost kódu.

5.2.1 Základní koncepce

Pro řešení aplikace byly zvoleny architektonické vzory, tedy typické architektury, které se v návrhu softwarových architektur vyskytují. Aplikace byla navrhnutá jako dvouvrstvá klient-server aplikace, kde první vrstva zajišťuje operaci s daty (modelová vrstva, podkapitola Datové zdroje a entity) a druhá vrstva správné zobrazení (prezentační vrstva, podkapitola Komponenty). Tento koncept byl zvolen díky plnému splnění architektonických nároků malé aplikace a komponentové architektuře React, přičemž bylo zajištěno, že je dodržen princip toku dat (z modelů do komponent) a událostí (z komponent do modelů).

5.2.2 Datové zdroje a entity

Aplikace je ze své podstaty klientem webové služby poskytované platformou Anitra, data z trackerů jiným způsobem optimálně získat nelze. Komunikace se vzdáleným serverem probíhá pomocí Hyper Text Transfer Protocolu, za pomocí tzv. REST API formátu, přičemž zprávy ze strany serveru jsou serializovány do formátu JSON, které v aplikaci odpovídají entitám (vyměněné zprávy jsou většinou počtem atributů nadmnožinou, mobilní aplikace nepotřebuje všechna data). Pro každou doménovou oblast bylo v aplikaci vytvořeno tzv. uložiště – třída, která plní určité business požadavky, přičemž prezentační vrstva nepotřebuje vědět, jakým způsobem byla data získána či předaná data zpracována. Název uložiště byl zvolen právě z důvodu, že třídy zajišťují i cachování objektů.



Obrázek 5.6: Class diagram – zdroj: autor

Pro aplikaci byly navrhnuty následující datové zdroje:

- BaseStore – základní třída pro společnou funkcionalitu,
- AuthStore – třída poskytující metody pro autentifikaci uživatele,
- LocationStore – třída zpracovávající získávání a ukládání GPS pozic,
- NetStore – třída zpracovávající události spojené s připojením a odpojením k internetu,
- NotificationStore – třída zpracovávající registraci k notifikacím,
- OverlayStore – třída zajišťující mapové podklady,
- TrackingStore – třída zajišťující akce spojené s jednotlivými trackery.

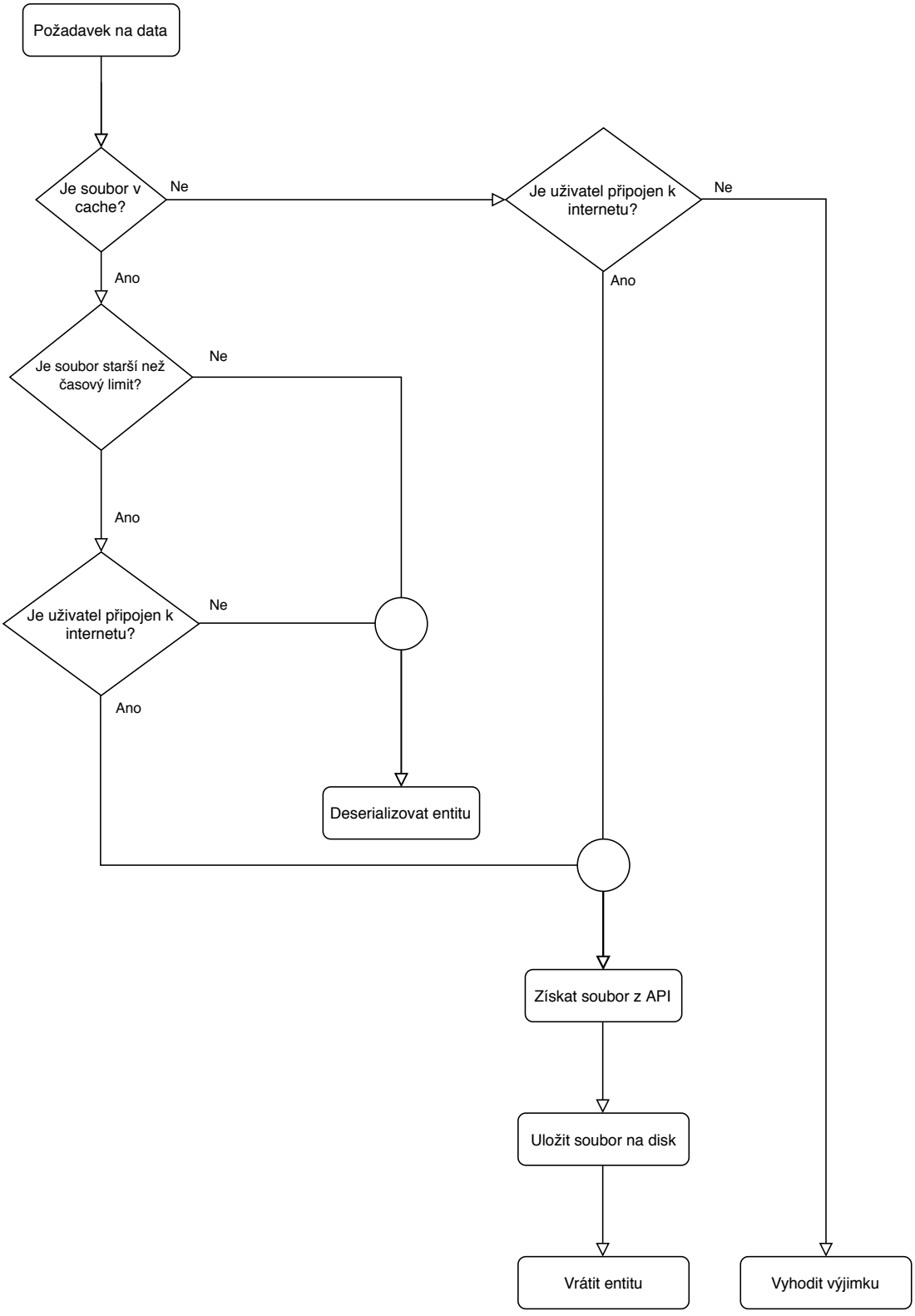
Na datové zdroje navazují entity, objekty, se kterými komponenty i datové zdroje pracují. Nejpodstatnější datové zdroje jsou uvedeny níže:

- Tracking – informace o trackeru,
- Species – kontejner pro názvy druhu,
- Track – informace o trase zařízení,
- User – uživatel aplikace,
- OfflineRegion – informace o stáhnutých mapových podkladech.

Cache

Uložiště pro každý požadavek provádí následující sekvenci rozhodnutí: pokusí se načíst výsledek (entitu) z cache a prověřit, že není starší než daný časový limit, aby byla dodržena aktuálnost dat. V případě že je stáří entity přes časový limit a uživatel je připojený k internetu, uživateli se vrátí čerstvá odpověď z API. V případě že uživatel připojený k síti není, vrátí se uložená odpověď, případně chyba, pokud žádná není. Chyba je odchycena v uživatelském rozhraní a uživateli se zobrazí hláška, že tento zdroj nebyl uložen do aplikační cache. V případě že soubor v cache není a uživatel je připojen k internetu, z API se entita získá a uloží na disk. Tímto se zajistí dostupnost aktuálních dat v případě že uživatel připojený

k internetu je a dostupnost dat v případě že není. V obrázku 5.7 je celý proces vyznačen vývojovým diagramem.

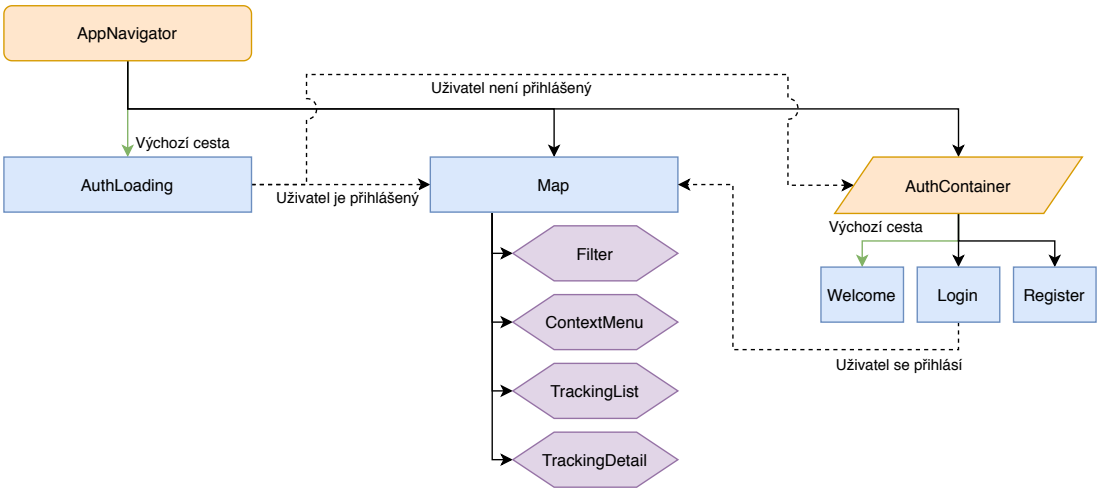


Obrázek 5.7: Vývojový diagram cache – zdroj: autor

5.2.3 Komponenty

V předchozích kapitolách byl zmíněn základ komponentového systému React, který aplikace využívá. Komponenty jsou samostatné (a často znovupoužitelné) části aplikace s datovými vstupy a reagující na události, ať už uživatelského rozhraní či dceřiných komponent. Komponenty v aplikaci lze rozdělit do dvou typů dle funkcí, které plní. První skupinou jsou obrazovky sdružují komponenty neobrazovkového typu (dále subkomponenty) do funkčního celku a předávají jim data a reagují na jejich události. Subkomponenty zpravidla slouží k zobrazování určitých dat nebo uživatelské interakci. Subkomponentám jsou předávána data a s obrazovkami komunikují přes události, je zde tedy zajištěná obousměrná komunikace mezi obrazovkou a komponentou. Předáním dat se zpravidla subkomponenta překreslí bez nutnosti překreslit nesouvisející (sub)komponenty, čímž je splněna tzv. reaktivita aplikace (zajištěna předem zmíněnou knihovnou MobX).

Komponenty jsou strukturovány do složek dle svého typu. Obrazovky jsou uloženy ve složce *src/screens*, subkomponenty se nachází ve složce *src/components*.



Obrázek 5.8: Strom komponent – zdroj: autor, klíč níže

Obrazovky

Obrazovky v aplikaci se skládají do tzv. navigátorů. Navigátor je komponentou knihovny *react-navigation*, která umožňuje přepínat mezi obrazovkami v reakci na uživatelské interakce nebo události v kódu (ref). Princip práce s navigátory je daný historií knihovny React a v nomenklatuře i chování odpovídá principům webových stránek. Každá obrazovka je přirovnatelná k webové stránce, navigátor mezi nimi umí přesměrovat a fungují zde koncepty navigačních tlačítek zpět i vpřed. Navigátory se rozlišují dle typů uživatelských interakcí pro přenavigování, v aplikaci se používají pouze dva z mnoha typů. Navigátory lze skládat do sebe a lze se navigovat napříč obrazovkami jiných navigátorů.

Prvním použitým typem je *SwitchNavigator*, který nemá prostředky pro navigaci mezi obrazovkami na základě uživatelských interakcí (např. tlačítko zpět, gesto přetáhnutí po obrazovce), ale pouze z kódu. Tento navigátor je vhodný pro úvodní stránku s rozhodnutí, zda má uživatel být přesměrován na přihlášení nebo do aplikace. V obrázku 5.8 je uveden v zaobleném obdélníku nahoře pod názvem AppNavigator.

Druhým využitým navigátorem je *StackNavigator*, který funguje na principu zásobníku (ref). Každá nová otevřená obrazovka se přiřadí na vrchol zásobníku a uživatel se může z této obrazovky vždy vrátit na předchozí. Tento navigátor se hodí např. na registraci a přihlášení. Na obrázku 5.8 je uveden v kosodélníku vpravo pod názvem AuthContainer.

Aplikace bude celá obalena v jednom SwitchNavigatoru, pro jednodušší správu autentizované a neautentizované části aplikace. Výchozí obrazovkou aplikace je *AuthLoading*, která rozhodne, zda se uživatel má přesměrovat do neautentizované části či do autentizované. Neautentizovaná část (pod navigátorem *AuthContainer*) se dále dělí na stránky onboarding (Welcome), přihlášení (Login) a registrace (Register). Mezi neautentizovanými stránkami lze přepínat gestem či tlačítkem zpět. Autentizovaná část (Map) je postavena na mapové koncepci aplikace a není využitý žádný navigátor, ale modální okna tvořená subkomponentami.

5.2.4 Subkomponenty

Subkomponenty vycházejí z předešlých částí návrhu (konkrétněji wireframe) a pro každou komponentu z wireframe bude vytvořena vlastní React komponenta. Pro často používané opakující se fragmenty v GUI lze také využít vlastní komponenty, o tomto bude rozhodnuto v implementaci. Subkomponenty nebyly předem plánovány do stejné úrovně jako obrazovky a mimo subkomponenty definovaé z wireframů byly vytvářeny ad hoc. Některé subkomponenty jsou vyznačeny v obrázku 5.8 v fialových šestiúhelnících pod obrazovkou Map.

6. Implementace

Kapitola implementace se věnuje konkrétním akcím vedoucím k vytvoření spustitelného aplikačního artefaktu. V jednotlivých podkapitolách budou uvedeny využité prostředky, konkrétní kroky a jejich výstupy. Některé z těchto kroků jsou použitelné pro všechny Expo projekty a některé úsudky lze využit při vytváření vlastních Expo projektů. Za zmínku zde také stojí kapitola Implementace off-line map, ve které je popsán způsob jak zajistit funkční off-line mapy pro Expo projekty, pro co neexistují zatím žádné komunitní knihovny nebo Expo-nativní způsoby řešení.

6.1 Vývojové prostředí

Vývojové prostředí je kolekce procedur a nástroj pro vývoj, testování a ladění aplikací nebo programů (ref). Termín vývojové prostředí se používá v synonymu s termínem IDE, integrovaným vývojový prostředím, což je softwarový nástroj pro psaní, stestavení, testování a ladění programů. Do tohoto termínu taktéž lze zahrnout i nástroje co IDE samotné využívá pro setavení artefaktů.

Pro vývoj aplikace byly zvoleny následující nástroje vývojového prostředí:

- VisualStudio Code jako nástroj IDE,
- Git jako verzovací nástroj,
- GitHub jako nástroj verzování i projektového řízení,
- AndroidStudio pro správu Android virtuálních strojů,
- XCode pro správu iOS virtuálních strojů (pouze na MacOS).

Další klíčový softwarový nástroj je taktéž instalace Node.js a balíčkovacího nástroje NPM, bez kterých by vývoj Expo aplikce nemožný. Text této práce se jejich instalaci nevěnuje a předpokládá, že již nainstalované jsou. Instalace na běžných a aktuálních verzích OS není problémem.

Do pojmu vývojové prostředí se zahrnují i izolovaná prostředí, ve kterém běží testovací nebo produkční instance aplikace (resp. datového zdroje), v tomto případě platformy Anitra. Pro vývoj této aplikace nebylo toto rozlišení nutné a vyvílejo se přímo na produkční verzi platformy Anitra.

6.1.1 Vytvoření projektu

V ukázce kódu 4.3 bylo již zmíněno, jakým způsobem vytvořit Expo projekt. Pro vytvoření expo projektu stačí zadat následující příkazy do příkazové řádky.

```
1 npm install --global expo-cli
```

```
expo init mobilni-aplikace
```

Procedura 6.1: Vytvoření nového projektu

První řádek nainstaluje *expo-cli*, nástroje pro vytváření, správu, spouštění a sestavení Expo projektů. Druhý příkaz vytvoří nový expo projekt *mobilni-aplikace* v aktuální složce uživatele.

Z předpřipraveného kódu je dobré zminit vstupní bod aplikace *App.tsx*, ve kterém je připravená ukázková komponenta. Z tohoto vstupního bodu se implementují navigátory zmíněné v kapitole Implementace navigátorů.

6.2 Implementace uložišť a entit

Implementace uložišť a entit byla vytvořena dle návrhu aplikace. Bylo vytvořeno více entit, než bylo plánováno, aby celá aplikace komunikovala za pomocí typovaných objektů a tím se zjednodušil proces ladění v aplikaci i urychlil vývoj. Pro entity bylo využito rozhraní, aby entity pro funkce ukládání měly stejnou signaturu a bylo je možné používat ve stylu OOP bez složitých konstrukcí. V ukázce kódu uvedené níže je ukázáno rozhraní, které musí implementovat všechny entity a entity, které mají být uložitelné na disk.

```
export interface IEntity {
  id?: number;

  synchronized: boolean;
  lastSynchronized?: Date;
};

export interface ISerializableEntity extends IEntity {
  toJson() : object;

  toJsonString() : string;

  fromJson(json: any): IEntity;
};
```

Procedura 6.2: Ukázka entity

Implementace uložišť samotných spočívá v implementaci tří částí – v komunikaci se serverem, transformaci dat od serveru a uložení těchto dat na disk. Pro komunikaci se serverem pomocí HTTP Rest API byla zvolena knihovna *axios*, která se běžně používá v JavaScriptových projektech pro abstrakci rozdílů mezi jednotlivými klienty. Knihovna dále umožňuje jednodušší přístup např. k hlavičkám HTTP dotazu a je tedy jednoduché se např. oproti vzdálenému serveru autentifikovat. Axios očekává na vstupu objekt obsahující výčet parametrů a na výstupu vrací tělo dotazu zformátované do správného formátu (např. když server vrací JSON odpověď, Axios z vráceného textu správně složí JSON).

Transformace dat ze serveru spočívá v převodu klíčů a hodnot vrácených ze serveru do předem zmíněných entit. Entity v aplikaci neobsahují celá data, ale pouze výčet, který aplikace skutečně potřebuje. Tímto se šetří nejen místo na disku, ale i doba serializace a deserializace entit při čtení z disku. V API nejde stanovit, jaká pole dotazovatel vyžaduje a filtrování je tedy nutno provést až na koncovém bodu procesu, tedy v mobilní aplikaci.

Pro uložení dat byla využita součást Expo *expo-file-system*, která abstrahuje od jednotlivých souborových systémů operačních systémů. Nevýhodou tohoto systému je neobratnost při zakládání složek, složky se musí vytvářet dopředu a nelze je vytvořit při zápisu souboru. Podstatnější nevýhodou je nutnost serializovat obsah souboru do textu, nejedná se tedy o příliš vhodnou metodu ukládání např. obrazových dat kvůli nutnosti převést binární data na textový formát např. Base64. Alternativní řešení pro Expo neexistuje. Ukázka práce se souborovým systémem je v následující ukázce kódu.

```
1 import * as FileSystem from 'expo-file-system';
2
3 let string = await FileSystem.readAsStringAsync(path);
```

Procedura 6.3: Ukázka entity

Do proměnné string se asynchronně zapíše obsah souboru v cestě, v případě chyby čtení, např. způsobené neexistujícím souborem, se vyhodí výjimka. Soubor lze dále zpracovat především např. textu na JSON nebo na XML, případně jakýkoliv jiný formát, který aplikace využívá. V aplikaci se nad těmito funkcemi staví třída PersistenStorage, která poskytuje obecné metody pro uložení a získání kolekcí či jednotlivých entit.

6.3 Implementace obrazovek

Implementace obrazovek byla nejdelší částí tvorby aplikace a nejnáročnější na testování. Obrazovky jsou React komponenty umístěny ihned pod navigátorem. V jejich životním cyklu se vytváří subkomponenty, získávají data z uložišť a pomocí událostí komunikují s uložišti. Obrazovky jsou spolu se subkomponentami zodpovědné za správu interakcí s uživatelem.

Níže je uvedena ukázková implementace obrazovky AuthLoading, která rozhoduje o přesměrování uživatele po zapnutí aplikace, pokud je přihlášen či ne.

```
1  import React from 'react';
2  import { StyleSheet, Text, View } from 'react-native';
3  import { MaterialIndicator } from 'react-native-indicators';
4
5  import AuthStore from '../store/AuthStore';
6  import Theme from "../constants/Theme.js";
7
8  export default class AuthLoading extends React.Component {
9    verifyAuth = async () => {
10     await AuthStore.awaitAuth();
11     if (AuthStore.isAuthorized) {
```

```
12        this.props.navigation.navigate("Map");
13      } else {
14        this.props.navigation.navigate("AuthContainer");
15      }
16    }
17
18    componentDidMount() {
19      this.verifyAuth();
20    }
21
22    render () {
23      return (
24      <View style={styles.container}>
25        <View>
26          <MaterialIndicator color={ Theme.colors.brand.primary }/>
27        </View>
28      </View>
29      );
30    }
31  }
32
33  const styles = StyleSheet.create({
34    container: {
35      flex: 1,
36      backgroundColor: Theme.colors.default.background,
37      alignItems: 'center',
38      justifyContent: 'center',
39      alignSelf: 'stretch',
40    }
41  });
```

Procedura 6.4: Ukázka implementace obrazovky

V ukázce lze vidět několik podstatných částí. V první části jsou importy závislostí, kde veškeré obrazovky potřebují naimportovat alespoň závislost *React* z knihovny React. Na druhém řádku lze vidět import typických ovládacích prvků a utilit pro obrazovku či subkomponentu. StyleSheet je způsob zapsání stylu pro ReactNative, ukázka způsobu zápisu stylů se nachází v objektu *styles* na konci ukázky kódu. Text je komponenta, která umožňuje zobrazení stylovaného textu a jako jediná může obsahovat čistý text. Komponenta View rámcově odpovídá funkci tagu *div* v HTML a je pouze prázdným kontejnerem pro ostatní komponenty, ale může být stylována. Následně je vidět import *MaterialIndicator* z knihovny *react-native-indicators*, externí komponenty, která zobrazuje načítací spinner, aby UI nevypadala neresponzivně. Následně je importováno uložiště pro autentifikaci a seznam konstant obsahující styl aplikace.

Samotná obrazovka začíná na řádku 8, kde se nachází definice třídy komponenty. Na řádku devět se nachází metoda komponenty, ve které se volá uložiště Auth pro ověření, zda je uživatel autorizovaný a případně se přenaviguje na správnou obrazovku. Vysoce podstatná je metoda *componentDidMount* na řádku 18, která se spustí po načtení komponenty a volá již zmiňovanou funkci na ověření přihlášení. Metoda render vrací JSX obrazovky, tedy strukturu všech subkomponent. Na konci je již zmiňovaný objekt styles, obsahující styly.

Tato obrazovka je reprezentativní pro všechny obrazovky, s výjimkou hlavní mapy.

6.3.1 Implementace hlavní mapy

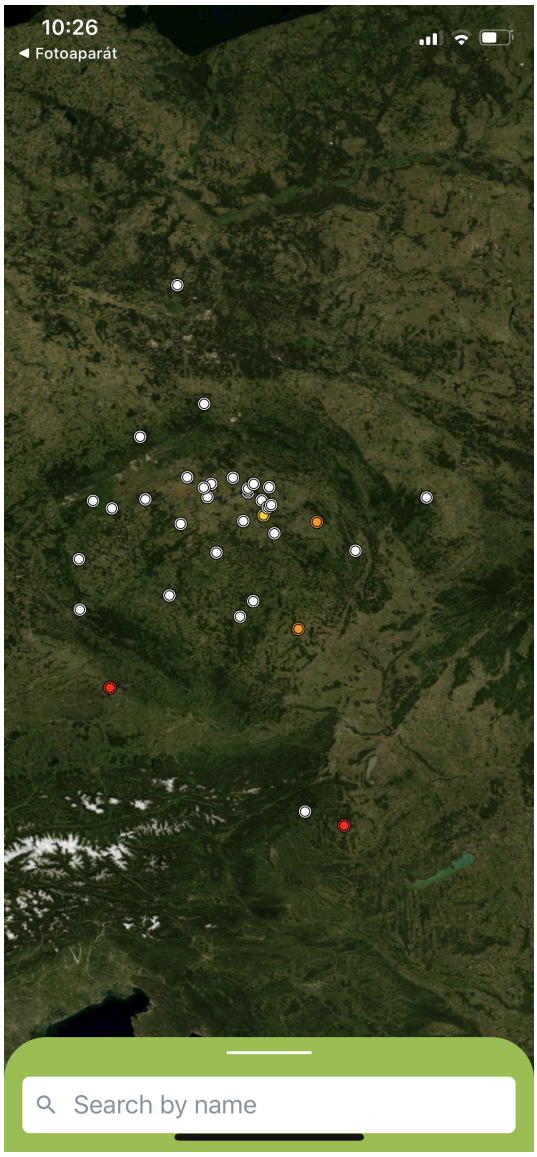
Pro hlavní část této obrazovky, tedy mapy, byla využita knihovna *react-native-maps* z důvodu absence alternativ pro Expo. Knihovna však splňuje veškeré požadavky aplikace. Do mapy lze např. vkládat objekty typu trasa, bod i tyto body následně stylovat. Objekty do mapy se přidávají jako individuální komponenty, např. bod s tzv. infoboxem (bublinou nad bodem) se vytvoří následujícím způsobem.

```
<MapView>
  <Marker
    coordinate={ { latitude: point.lat, longitude: point.lng } }
    icon={image}
    image={image}
    zIndex={zIndex}
  >
    <Callout>
      <MarkerPosition tracking={track.tracking} id={point.id}/>
    </Callout>
  </Marker>
</MapView>
```

Procedura 6.5: Ukázka implementace obrazovky

Ukáza ilustruje způsob umístění v mapě (parametr coordinate), způsob vybrání ikony pro obě platformy (parametr icon a image) a nastavení zIndexu, který určuje pořadí vykreslování komponent vzájemně se překrývajících. Komponenta Callout obsahuje již zmiňované infowindow, které obsahuje subkomponentu MarkerPosition.

Pro implementaci vyjížděcího menu pro filtry byla využita komponenta *rn-sliding-up-panel*, která plně splina zadání z kapitoly návrh. Detaily implementace lze zobrazit v příloze, jelikož jsou příliš dlouhé pro ukázky v textu této práce.



Obrázek 6.1: Výsledek implementace hlavní mapy – zdroj: autor

Výsledek implementace mapy je srovnatelý s drátěným modelem představeným v kapitole návrh.

6.4 Implementace subkomponent

6.4.1 Implementace off-line map

Pro implementace off-line map bylo nutno vytvořit dvě komponenty. První komponentou jsou off-line mapy samotné, druhou komponentou způsob, jak vybrat rozsah, který má být stáhnut.

Off-line schopnost map je umožněna díky následujícím klíčovým možnostem: možnost načítat tzv. tile mapy (mapy uložené ve formátu obrázků s jednoznačným indexem v dimenzích x, y a z) a URI, které využívá Expo i pro ukládání souborů. Pro mapy je tedy jedno, zda načítají jednotlivé mapové čtverečky z disku, či přímo ze vzdáleného mapového serveru.

```
import { UrlTile } from 'react-native-maps';
<MapView..>
  <UrlTile urlTemplate="..."/>
</MapView>
```

Procedura 6.6: Off-line mapy

Do urlTemplate se místo vzdáleného zdroje doplní část kódu, která obsahuje parametry pro jednotlivé dimenze mapy ve formátu *{z}/{x}/{y}.png*. Každý mapový čtvereček je následně načítán z disku. Toto chování je vhodné zapnout pouze v případě že uživatel není připojený k síti, či na přímé vyžádání uživatele.

Z časových nedostatků nebylo možno vytvořit komfortní uživatelský nástroj pro výběr off-line map, ale pouze rudimentární. Z kontextového menu uživatel zapne dialog zadávání bodů, čímž se uloží příznak do aplikace, že uživatel vybírá off-line region. Uživatel je následně instruován, aby klikal do mapy.

6.5 Implementace navigátorů

Pro implementaci navigátorů byla vybrána již zmiňovaná standardní knihovna *react-navigation*. Pro instalaci je využitý balíčkovací nástroj NPM.

```
npm install react-navigation
npm install react-navigation-stack
```

Procedura 6.7: Instalace react-navigation

Po instalaci je nejprve ve vybrané React komponentě nejprve naimportovat dané knihovny.

```
import { createSwitchNavigator, createAppContainer,
    NavigationContainerComponent, NavigationActions } from 'react-navigation';
import { createStackNavigator } from 'react-navigation-stack';
```

Procedura 6.8: Import knihoven pro hlavní navigátor

Importováním těchto závislostí půjde vytvořit switch navigátor, hlavní navigační kontejner aplikace a stack navigátor. Následně je nutno naimportovat komponenty obrazovek následujícím způsobem.

```
import Welcome from './src/screens/auth/Welcome'; // naimportuje komponentu
    obrazovky Welcome
import Login from './src/screens/auth/Login';
import Register from './src/screens/auth/Register';
import MapScreen from './src/screens/in/Map';
```

Procedura 6.9: Import obrazovek pro hlavní navigátor

Navigátory lze následně sestavit modulárním způsobem vkládání do sebe, kde listy grafu navigátorů jsou jednotlivé komponenty obrazovek.

```
const AuthContainer = createStackNavigator({
  Welcome: Welcome,
  Login: Login,
  Register: Register
}, {
  headerMode: 'none'
});

const AppNavigator = createSwitchNavigator({
  AuthLoading: AuthLoading,
  Map: MapScreen,
  AuthContainer: AuthContainer
}, {
  "initialRouteName": "AuthLoading" // nazev vychozi cesty, resp. obrazovkove
     komponenty
});
```

Procedura 6.10: Implementace navigátorů

Navigátory se následně obalí aplikačním kontejnerem, ze kterého se vytvoří hlavní komponenta aplikace, tedy její vstupní bod.

```
const AppContainer = createAppContainer(
  AppNavigator
);

export default class App extends React.Component
{
  render () {
    return (
        <React.Fragment>
          <AppContainer ref = { setNavigatorRef }/>
          <FlashMessage position="top" />
        </React.Fragment>
    )
  }
}
```

Procedura 6.11: Vytvoření aplikačního kontejneru

V ukázce kódu se vytváří komponenta App, která je vstupním bodem aplikace. V komponentě App se nachází fragment, který je kolekcí různých React komponent a sám nemá v uživatelském rozhraní žádnou funkci. Následně se vytvořený AppContainer využije přímo v komponentě a jeho vykresleným obsahem se stávají různé obrazovky specifikované v předchozích ukázkách kódu. Ačkoliv se nejedná přímo o funkce navigátoru, na stejné úrovni se také nachází komponenta FlashMessage, která zajistí, že se tzv. flash zprávy vygenerované v aplikaci zobrazují napříč všemi obrazovkami z jednoho místa (tedy bez duplikace kódu). Nejedná se tedy o funkci navigátoru, ale pouze o komponentu, kterou je vhodno umístit na

nejvyšší úroveň kvůli prevenci duplikaci kódu a jedná se o vhodnou ukázku, co lze na stejnou úroveň jako kontejner navigátorů vložit.

Pro přenavigování lze využít v jakékoliv komponentě umístěné pod AppContainerem (resp. pod jakýmkoliv navigátorem) funkcí uvedenou níže.

```
this.props.navigation.navigate("Map");
```

Procedura 6.12: Přenavigování

Všem komponentám je předán odkaz na navigátor a pomocí klíče (názvu) obrazovky se na ni lze přenavigovat kdekoliv uvnitř komponenty, např. při reagování na kliknutí z tlačítka či doběhnutím vnitřní události. Navigace mimo komponentu ale tímto způsobem není možná, proto byl autorem aplikace v App.tsx vytvořena možnost, jak tuto funkci z komponenty získat. V komponentě AppContainer byl specifikován atribut ref. Atribut ref v Reactu slouží k získání instance určité komponenty. Komponenta se předá funkci setNavigatorRef, který do lokální proměnné vloží instanci navigátoru. Aplikace nad touto lokální instancí vytváří funkci navigate, ve které volá událost navigace a je tedy možné z kódu mimo komponenty volat funkce navigace.

```
let instanceRef: NavigationContainerComponent;

function setNavigatorRef(instance: NavigationContainerComponent) {
  instanceRef = instance;
}

function navigate(routeName, params) {
  instanceRef.dispatch(
    NavigationActions.navigate({
      routeName,
      params,
    })
  );
}
```

Procedura 6.13: Přenavigování

6.6 Implementace push notifikací

Jednou z mnoha výhod platformy Expo je jednoduchost napojení aplikačních push notifikací. Není potřeba využívat různé poskytovatele pro různé platformy, Expo zajistí doručení na jakékoliv zařízení, které se k notifikacím registruje. Výhodou také je jednoduchost celého systému, kde stačí aby se zařízení pouze oznámilo serveru svým identifikátorem a následně lze notifikace volat za pomocí REST API Expo. Celý proces je velmi jednoduchý na implementování v mobilní aplikaci i ve zdroji notifikací. Získání notifikací na straně aplikace lze docílit následujícím způsobem.

```
import { Notifications } from 'expo';
```

```
import * as Permissions from 'expo-permissions';
import Constants from 'expo-constants';


const { status } = await Permissions.askAsync(Permissions.NOTIFICATIONS);

if (finalStatus !== 'granted') {
  return;
}

const token = await Notifications.getExpoPushTokenAsync();
```

Procedura 6.14: Expo notifikace

Pro notifikace je nutno získat uživatelovo svolení. Pokud svolení dá, Expo vrátí textový token, který se následně dá použít pro poslání notifikace na konkrétní zařízení. Pro zjednodušení práce s větším počtem stejných notifikací je možnost vytvářet skupiny příjemců, aplikace tohoto nevyužívá. Kód pro generování zpráv na straně serveru není součástí této práce a je pouze obsažen ve webovém backendu aplikace Anitra.

Závěr

Seznam použité literatury

Basic Concepts of Flexbox: CSS Flexible Box Layout, 2019. Mountain View: MDN Web Docs. Dostupné také z: `https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox`.

Přílohy