Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

# Bakalářská práce

# GPS Sledování vozidla

Plzeň 2019 Jan Šťastný

**Místo této strany bude zadání práce.**

# Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 26. června 2019

Jan Šťastný

Rád bych touto cestou vyjádřil poděkování vedoucímu této bakalářské práce panu Ing. Petru Přibylovi za vstřícnost, věnovaný čas, vysokou míru trpělivosti a poskytnutí cenných rad v celém průběhu jejího vypracovávání.

# Abstract

This bachelor thesis deals with the design and implementation of a mobile application designed to simplify the record of the driver's work performed in a particular location. Important task of this mobile application is to record all coordinates of the driver's locations from a remote server. The main activity of the system is to monitor the movement of the mobile device in real time, and to inform the user about the events. This information is derived from the entries and exits of the individual geographic locations. After the user verifies the events and also records more data about the work performed, the events are sent to the server.



# Abstrakt

Tato bakalářská práce se zabývá návrhem a implementací mobilní aplikace určené ke zjednodušení evidence vykonané práce řidiče v určité lokalitě. Její dílčí úlohou je načtení souřadnic všech lokalit ze vzdáleného serveru. Hlavní činností systému je sledování pohybu mobilního zařízení v reálném čase a informování uživatele o nastalých událostech – vstupech a výstupech do, resp. z jednotlivých lokalit. Tyto události jsou následně – za předpokladu, že je uživatelem jejich správnost potvrzena a jsou rozšířeny o informaci o provedené práci – odesílány na zmíněný server.

# Obsah

# 1 Úvod

V dnešní době existuje na trhu velké množství velmi sofistikovaných zařízení a systémů, které jsou specializovány pro zlepšení, zjednodušení a zautomatizování rozličných druhů profesí a činností v mnoha odvětvích. Tyto systémy jsou však mnohdy pro drobnější podnikatele příliš nákladné, a tak se velmi často hledají dostupnější a levnější varianty, které se svými vlastnostmi originálním řešením co nejvíce podobají.

Typickým příkladem je v současnosti oblast zemědělství, kde se ke zefektivnění práce na polích moderní technologie využívají stále častěji. Konkrétně sledování – a případně i řízení – pohybu zemědělského stoje je často realizováno pomocí přesných GPS lokátorů. Jejich cena – i přes svou mírně sestupnou tendenci v posledních letech – ovšem stále není pro všechny zemědělce přívětivá, a tak tzv. „chytré zemědělství“ nastupuje pomaleji, než modernizace a automatizace jiných odvětví.

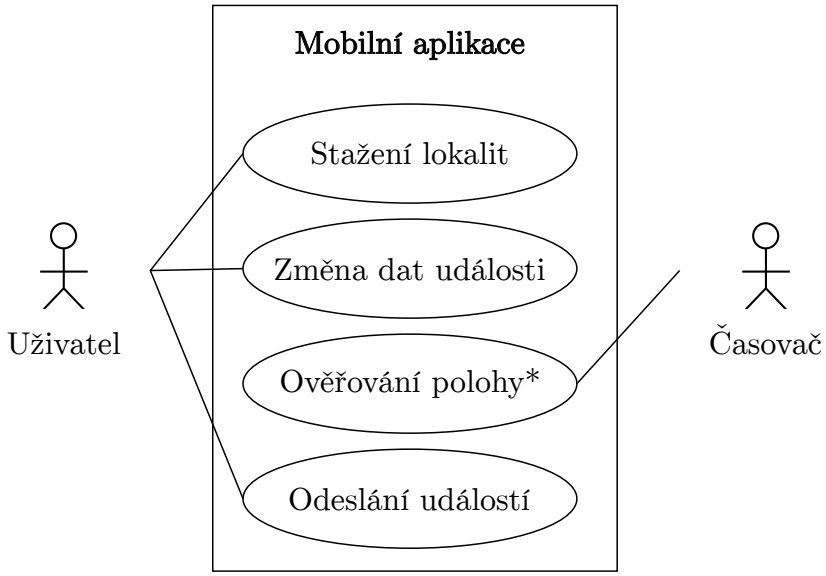
Úkolem, který byl stanoven jako cíl této bakalářské práce, je realizace zjišťování polohy a pohybu zemědělských vozidel. Jako velmi vhodná a cenově dostupná náhrada drahých specializovaných systémů se v tomto případě jeví využití mobilních zařízení, například mobilních telefonů.

Primárním cílem práce je tedy vývoj mobilní aplikace, která bude přítomna při práci zemědělského vozu, s úkolem zastávat některé funkce GPS lokátoru. Aplikace bude s využitím integrovaného GPS modulu v zařízení, ve kterém bude spuštěna, schopna sledovat jeho pohyb napříč lokalitami (poli daného zemědělce) a zaznamenávat aktivity, které byly v daných lokalitách prováděny.

# 2 Specifikace požadavků

Požadovaným výstupem této bakalářské práce je mobilní aplikace, která poskytne řidiči zemědělského vozu jednoduchou evidenci vykonané práce. Tato aplikace tedy musí splňovat několik základních funkčních požadavků. Ty jsou následně zobrazeny v obrázku č. 2.1.

Prvním z nich je stažení dat o lokalitách z externí databáze a jejich následné uložení do databáze v mobilním zařízení. Celá tato akce proběhne vždy na základě podnětu uživatele aplikace. Obsahuje-li interní databáze v zařízení alespoň jeden záznam o lokalitě, je možné spustit aktivitu zjišťování polohy.




*Ověřování, zda se zařízení nenachází v některé z lokalit

Obrázek 2.1: Diagram případů užití

Dalším z požadavků je tedy opakující se ověřování, zda se zařízení v reálném čase nachází v některé z uložených lokalit. Pokud aplikace zaznamená vstup do některé z nich, musí uživatele o této situaci informovat a vytvořit o ní v interní databázi záznam (tzv. událost). V případě výstupu z lokality je aplikace povinna učinit totožný krok. Rozdíl je v tom, že tentokrát je záznam doplněn o další údaj, který je následně uživatel povinen zadat. Konkrétně se jedná o úkon, který v dané lokalitě prováděl.

Posledním funkčním požadavkem je odesílání těchto událostí do totožné (výše zmíněné) externí databáze na žádost uživatele.

# 3 Poloha na povrchu Země

Obsahem následujících sekcí je uvedení do problematiky týkající se získávání a uchovávání dat o poloze objektů na povrchu Země, což je s ohledem na předmět práce velmi důležité. Dále je zde popsán způsob, na jehož základě lze následně vytvořit mechanismus ověřování, zda se zařízení nachází uvnitř či vně lokality.

## 3.1 Souřadnicové systémy

Souřadnice objektů na zemském povrchu lze obecně uchovávat v různých formátech – souřadnicových systémech. Takový systém je definován jako soubor matematických pravidel specifikující způsob, pomocí něhož se v prostoru přiřazují souřadnice jednotlivým bodům. [15]

Každý takový systém tedy zahrnuje:

- název a jedinečný standardizovaný kód,
- referenční plochu (další popis níže), ke které se vztahuje,
- počátek (výchozí bod),
- počet, charakter a orientaci souřadnicových os,
- formát a jednotky souřadnic.

Souřadnice konkrétního bodu představují uspořádanou $n$-tici čísel, která vyjadřuje jeho vztah k danému systému. Počet těchto čísel je podmíněn prostorem, ve kterém se body nachází. [18]

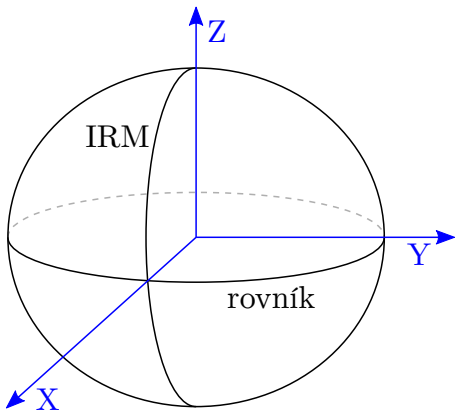
Souřadnicový systém je vždy vztažen k tzv. referenční ploše. To je označení pro matematicky definovatelné těleso, které co nejvíce odpovídá skutečnému tvaru Země či její určité části. Často je pro tyto účely využíváno tzv. rotačních elipsoidů. Pro velmi malá území okrouhlého tvaru o ploše asi 200 $km^2$ (kruh o poloměru 8 $km$) lze dokonce zemský povrch považovat za rovinu (pro méně přesné výpočty může jít o plochu až přibližně 700 $km^2$, tedy kruh s poloměrem 15 $km$). [6]

### 3.1.1 WGS-84

Nejrozšířenějším souřadnicovým systémem je v dnešní době systém zvaný WGS-84 *(World Geodetic System 1984)*. Tento geodetický systém je založený na referenčním elipsoidu se shodným názvem WGS-84, který představuje matematickou aproximaci tvaru planety Země. Jeho střed leží v těžišti planety Země.

Referenčním (nultým) poledníkem je pro souřadnicovou soustavu (zobrazenou na obrázku 3.1) na tomto elipsoidu tzv. *IERS*[1] *Reference Meridian* (IRM), který od historického poledníku *Greenwich Prime Meridian* leží východněji o *5,31″*. Počátek soustavy je shodný se středem elipsoidu, směr os je popsán v následujícím seznamu:

- **X** – je definována průsečnicí referenčního poledníku a roviny rovníku,
- **Z** – směřuje k severnímu pólu Země, zároveň je rotační osou elipsoidu,
- **Y** – doplňuje systém na tzv. *pravotočivý pravoúhlý souřadnicový systém*, směr kladné části osy leží v rovině rovníku o 90° východně vzhledem k ose X. [17]




Obrázek 3.1: Souřadnicová soustava na elipsoidu WGS-84.

Další – častější – možnost určení přesných souřadnic v tomto systému vychází z běžných zeměpisných souřadnic. Polohu je tedy možné určit pomocí zeměpisné délky, šířky a výšky. Šířka nabývá hodnot od $-90°$ do $90°$, kde $0°$ představuje rovník, délka pak nabývá hodnot od $-180°$ do $180°$,

[1] **IERS** (International Earth Rotation and Reference Systems Service) je služba, jejíž primárním cílem je poskytování astronomickým a geodetickým komunitám data a standardy související s rotací Země a referenčními plochami.

kde 0° je IRM. Libovolně zvolený bod v území České republiky tak bude definován následujícím způsobem.

| 49,1428 N, 15,0021 E |
|---|

V tomto případě jde o bod, jehož poloha je $49,1428°$ severně od rovníku a $15,0021°$ východně od IRM.

### 3.1.2 S-JTSK

Jedním z dalších mnoha souřadnicových systémů je Systém jednotné trigonometrické sítě katastrální (S-JTSK). Vychází z tzv. Křovákova zobrazení, proto jeho celý název zní *S-JTSK / Krovak, South West*.

Křovákovo zobrazení slouží k transformaci zeměpisných souřadnic z tzv. Besselova elipsoidu (referenční plocha) do pravoúhlého souřadnicového systému. Transformace probíhá v několika krocích, kdy po tzv. Gaussovo konformním zobrazení Besselova elipsoidu na kouli nastává konformní zobrazení na kuželovou plochu v obecné poloze.

Zavedení tohoto souřadnicového systému umožňuje zobrazit bývalou Československou republiku v prvním kvadrantu soustavy. Počátek soustavy je ve vrcholu kužele. Kladná část osy $X$ tohoto souřadného systému je směřována k jihu, kladná část osy $Y$ k západu. Obě tyto souřadnice jsou tedy na celém území současného Česka i Slovenska kladné a zároveň pro každý bod ležící uvnitř těchto států platí, že $Y < X$.

Geodetický základ tohoto systému tvoří tzv. *Jednotná trigonometrická síť katastrální*. Jedná se o soubor pevně stanovených bodů, které rozdělují povrch států na trojúhelníky.

Vzhledem k metodě vytvoření systému *S-JTSK* nelze použít univerzální transformační klíč pro celé území republik. Vykazuje totiž nepravidelné zkreslení. [17]

## 3.2 Zjišťování polohy v mobilních zařízeních

V dnešních mobilních zařízeních je možné pro určování přesné polohy využít tzv. *Global Navigation Satellite System* (GNSS). Jedná se o systémy, které za pomoci družic na oběžné dráze poskytují pokrytí celého zemského povrchu signálem, který zjištění konkrétní polohy zprostředkuje.

### 3.2.1 Princip

Struktura všech GNSS je založena na velmi podobném principu. Výše zmiňované družice představují tzv. *kosmický segment*. Ten zajišťuje pomocí vhodné konstelace určitého počtu družic (pohybujících se po různých drahách) optimální pokrytí Země.

Dalším prvkem každého družicového navigačního systému je tzv. *řídicí segment*. Ten obsahuje soustavu monitorovacích stanic, které měří přesnou polohu navigačních družic a kvalitu navigačních signálů. Monitorovací stanice jsou na zemském povrchu rozmístěny tak, aby každá navigační družice mohla být trvale monitorována. Naměřená data z monitorovacích stanic jsou pak předávána do řídicí stanice. Zde dochází k výpočtu parametrů celého kosmického segmentu a opravě navigačních signálů (dat), které navigační družice distribuují uživatelskému segmentu. Opravená data jsou bezprostředně vysílána k jednotlivým družicím. [13]

Pomocí rádiových vln jsou z družic přenášeny signály, které zachycují přijímače v tzv. *uživatelském segmentu*. Na základě těchto signálů je možné určit vzdálenost přijímače od dané družice (někde na povrchu koule o poloměru této vzdálenosti se zařízení nachází). Průnikem získaných oblastí minimálně ze čtyř družic, které se nachází v danou chvíli na obzoru, lze dosáhnout určení polohy zařízení (zeměpisná šířka a délka, nadmořská výška a čas). Pro zvýšení přesnosti je potřeba přijmout signál z více družic najednou.

### 3.2.2 Vývoj

První pokusy o využití družic pro určování polohy se datují do období šedesátých let minulého století. Tento rozvoj zpočátku probíhal především z důvodu vojenského využití – o první funkční systém tohoto typu se zasloužily americké vzdušné síly a námořnictvo. Jde o systém *Navstar GPS* (Global Positioning System), který se až na přelomu tisíciletí rozšířil i do civilní sféry a v dnešní době je stále nejrozšířenějším GNSS. Pomocí tohoto systému lze zaměřit polohu s přesností do 5 metrů. Navstar GPS pracuje právě se zmiňovaným souřadnicovým systémem WGS-84 (jehož popis se nachází v sekci 3.1.1 na straně 10).

S mírným zpožděním za americkým systémem byl vyvinut ruský (dříve sovětský) systém *GLONASS* (Globální Navigační Satelitní Systém). Přesnost tohoto systému v civilní sféře je částečně omezena, ovšem (a to zejména na severní polokouli) dosahuje přesnějších hodnot, nežli Navstar GPS – až 2,8 m.

Mezi další dnes již rozšířené GNSS systémy patří projekt *Galileo*, který byl vyvinut v posledních letech pod záštitou Evropské unie. Dalším globálním funkčním systémem je čínský *BeiDou Navigation Satellite System*. [13]

Při určování polohy a navigaci pomocí mobilních zařízení byla tato zařízení donedávna odkázána pouze na systém Navstar GPS. Moduly pro určování přesné polohy – v dnešní době integrované do běžně používaných mobilních zařízení – však již mají možnost přijímat signál z družic různých (výše zmíněných) systémů. Díky kombinaci získaných dat tedy může určená poloha dosahovat vyšší přesnosti.

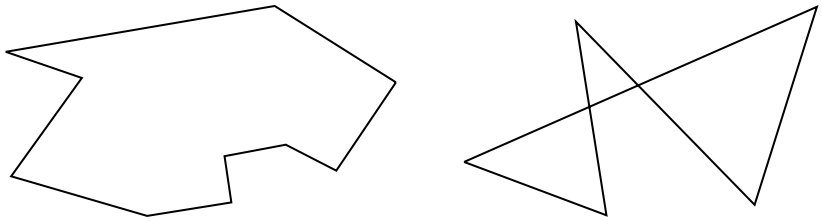
Aplikace pro mobilní zařízení využívající signály z GNSS se v určitých situacích mohou potýkat s nízkou úrovní signálu. To je dáno konstrukcí zařízení a prostředím, ve kterém jsou mobilní zařízení používána. Problém s úrovní signálu nastává běžně například uvnitř budov či v zalesněných oblastech.

## 3.3 Ověřování výskytu bodu v polygonu

Základem pro následné vybudování aparátu pro ověřování, zda se mobilní zařízení nachází v některé z lokalit, je následující algoritmus. Jedná se o způsob, pomocí něhož lze potvrdit či vyvrátit výskyt určitého bodu v daném polygonu.

Polygon (mnohoúhelník či $n$-úhelník) je matematicky definován jako uzavřená rovinná oblast, jejíž hranicí je jednoduchá uzavřená lomená čára s $n$ vrcholy, kde $n \in \{n \in \mathbb{N} \mid n \geq 3\}$. Zároveň se zpravidla předpokládá, že žádné dvě sousední strany lomené čáry neleží na téže přímce. [11]

Pro potřeby tohoto systému musí být zároveň zaručeno, že žádné dvě strany polygonu se nebudou protínat. Příklady povoleného a zakázaného polygonu jsou uvedeny na obrázku 3.2.



Obrázek 3.2: Příklad povoleného (vlevo) a zakázaného (vpravo) polygonu v tomto systému.

Předem bylo stanoveno, že bod, který leží na jedné ze stran polygonu, bude algoritmem označen za bod, který leží v polygonu.

### 3.3.1 Algoritmus

Pro ověřování výskytu bodu v polygonu se často využívá tzv. *paprskový algoritmus*. Jde o řešení, které bylo popsáno již v roce 1962 M. Shimratem a následně opraveno R. Hackerem. Pro studium tohoto algoritmu vhodně posloužila ukázka konkrétní implementace na webu *GeeksforGeeks.org* [7], a to i přes to, že obsahovala podstatné nedostatky – vůbec neobsahovala řešení problémové situace popsané v níže zmíněném kroku č. 1.

Do tohoto algoritmu tedy vstupují dva parametry: polygon splňující výše zmíněnou definici a bod $P[x_P, y_P]$, o jehož výskytu v daném polygonu se bude rozhodovat.

Základní myšlenka tohoto řešení je popsána v následujících bodech.

1. **Nalezení extrémního bodu**

   V tomto kroku je potřeba nalézt tzv. *extrémní bod* $E[x_E, y_E]$, který splňuje tato kritéria:

   - bod nesmí ležet uvnitř polygonu,
   - úsečka $PE$ nesmí procházet žádným z bodů tvořících polygon.

   Dodržení druhého kritéria je klíčové, neboť v případě, kdy by úsečka $PE$ jakýmkoli bodem polygonu procházela, by mohlo dojít k situacím, ve kterých by následující postup ověřování (počínaje krokem č. 2) nebyl platný.

2. **Nalezení počtu průsečíků**

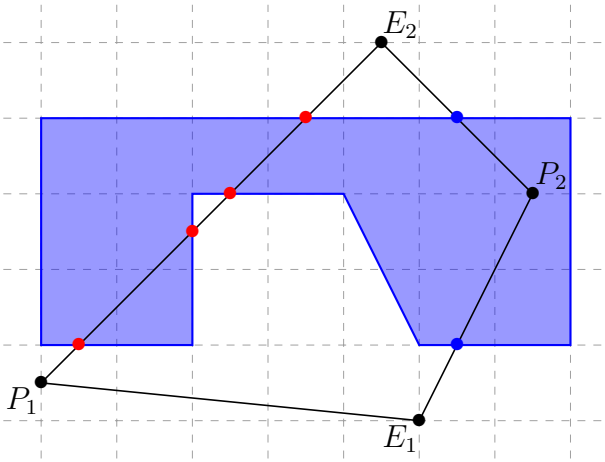
   Nad každou dvojicí sousedních vrcholů polygonu (tvořících stranu polygonu) proběhne dvojí ověření.

   - Zjistí se, zda bod $P$ neleží přímo na této straně. Pakliže ano (tj. platí-li předpoklad, že body tvořící danou stranu leží s bodem $P$ v jedné přímce a zároveň pro ně platí vztah 3.1 na straně 18), celý algoritmus je ukončen a výstupem je tvrzení, že bod $P$ leží v polygonu.
   - Provede se ověření, zda ji protíná úsečka $PE$. V případě, že ano, dojde ke zvýšení hodnoty čítače průsečíků $i$. Následuje ověřování u další hrany.

3. **Vyhodnocení počtu průsečíků**

   Za předpokladu, že všechny iterace algoritmu proběhnou a nenastane jeho předčasné ukončení, nastane kontrola hodnoty $i$. Bude-li hodnotou čítače liché číslo, bod v polygonu leží. V jiném případě (včetně $i = 0$) bod v polygonu neleží.

V obrázku 3.3 je tato myšlenka znázorněna pro různé případy. Modrá plocha představuje vstupní polygon, $P_1$ a $P_2$ jsou potenciální vstupní body a $E_1$ a $E_2$ body, které algoritmus může zvolit jako extrémní. Například tedy pro bod $P_2$ v případě, že extremním bodem bude zvolen $E_1$, platí, že počet průsečíků vzniklé úsečky $P_2E_1$ se stranami polygonu je lichý a bod tedy leží uvnitř.




Obrázek 3.3: Myšlenka ověřování výskytu bodu v polygonu. (Bod $P_1$: při zvolení jakéhokoli extrémního bodu $E_k$ je počet průsečíků (vyznačeny červenou barvou) úsečky $P_1E_k$ se stranami polygonu vždy sudý nebo roven nule, bod $P_2$: počet průsečíků (vyznačeny modrou barvou) $P_2E_k$ se stranami je vždy lichý.

### 3.3.2 Dílčí problémy

Nechť body $A[x_A, y_A]$, $B[x_B, y_B]$ a $C[x_C, y_C]$ $D[x_D, y_D]$ leží v rovině $xy$ a úsečky $AB$, $BC$ a $CD$ jsou zkonstruovány právě z těchto bodů. S jejich pomocí budou vysvětleny dílčí problémy, které výše zmíněný algoritmus řeší.

#### Vektorový součin

Vektorovým součinem vektorů $u(x_u, y_u, z_u)$ a $v(x_v, y_v, z_v)$ je nazýván vektor

$$w\left(\begin{vmatrix} y_u & z_u \\ y_v & z_v \end{vmatrix}, \begin{vmatrix} z_u & x_u \\ z_v & x_v \end{vmatrix}, \begin{vmatrix} x_u & y_u \\ x_v & y_v \end{vmatrix}\right).$$

Pro vektorový součin platí následující věty.

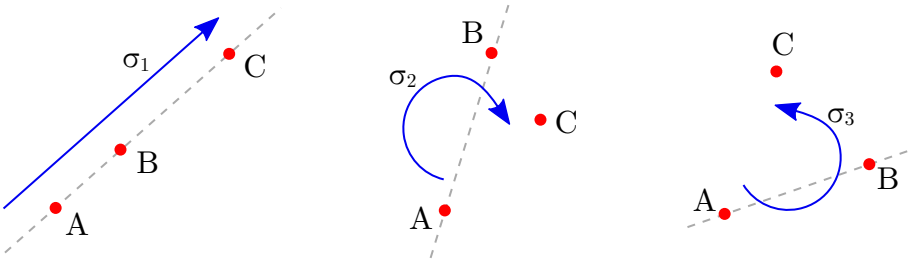
- Vektorový součin dvou lineárně závislých vektorů je nulový vektor.
- Vektorový součin dvou lineárně nezávislých vektorů je kolmý k oběma daným vektorům. [19]

### Orientace tří bodů

Pro potřeby následného zjišťování, zda se protínají dvě úsečky, je potřeba nejprve umět zjistit tzv. orientaci $\sigma$ tří bodů $A$, $B$ a $C$ v rovině. Pro ověření, zda body leží v jedné přímce je možné využít porovnání směrnic dvou přímek, z nichž jedna je tvořena body $A$ a $B$ a druhá body $B$ a $C$. Pro další kroky algoritmu je však potřeba – v případě, že body v přímce neleží – dále zjistit, ve které polorovině tvořené přímkou danou prvními dvěma body leží bod třetí, a tak pouhé porovnání směrnic nestačí.

Závisle na vstupním pořadí bodů jsou definovány následující tři typy jejich orientace $\sigma$, které jsou dále znázorněny v obrázku 3.4 (zde jde o případ, kdy body vstupují v pořadí: $A$, $B$, $C$):

- body v přímce (v obrázku $\sigma_1$),
- body ve směru hodinových ručiček (v obrázku $\sigma_2$),
- body proti směru hodinových ručiček (v obrázku $\sigma_3$).




Obrázek 3.4: Typy orientace tří bodů v rovině (v pořadí: $A$, $B$, $C$).

Zjištění je možné provést pomocí vektorového součinu vektorů $\overrightarrow{AB}$ a $\overrightarrow{BC}$. Proto, že vektorový součin je definován ve trojrozměrném prostoru, všem třem vstupním bodům je přidána třetí nulová složka $z$. Body tedy získají tuto podobu:

$$\begin{aligned} A[x_A, y_A, z_A] &= A[x_A, y_A, 0], \\ B[x_B, y_B, z_B] &= B[x_B, y_B, 0], \\ C[x_C, y_C, z_C] &= C[x_C, y_C, 0]. \end{aligned}$$

Vytvoření potřebných vektorů je pak možné následujícím způsobem:

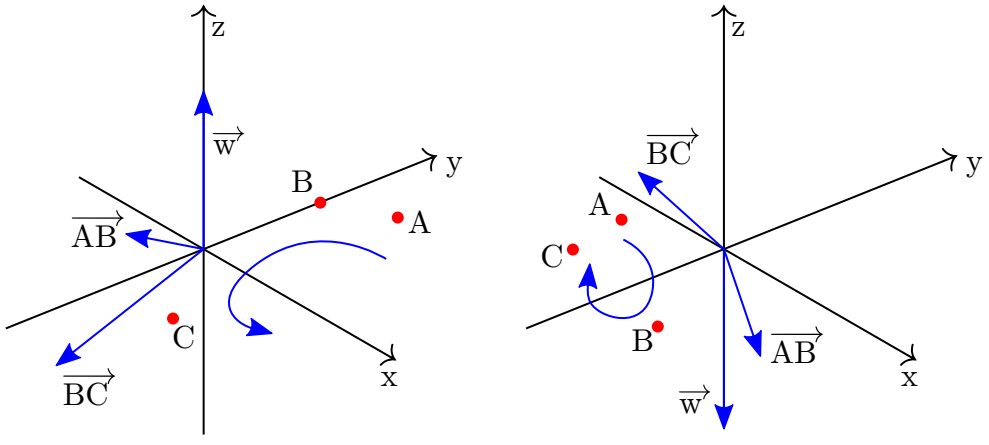
$$\begin{aligned} \overrightarrow{AB} &= (x_B - x_A, y_B - y_A, z_B - z_A) = (x_B - x_A, y_B - y_A, 0), \\ \overrightarrow{BC} &= (x_C - x_B, y_C - y_B, z_C - z_B) = (x_C - x_B, y_C - y_B, 0). \end{aligned}$$

Vektorový součin má pak vždy tento tvar:

$$\overrightarrow{AB} \times \overrightarrow{BC} = \overrightarrow{w} = (0, 0, (x_B - x_A) * (y_C - y_B) - (x_C - x_B) * (y_B - y_A)).$$

Dle směru vektoru $\overrightarrow{w}$ (konkrétně tedy hodnoty jeho třetí souřadnice $z_w$) je následně možné určit orientaci vstupních bodů (obrázek 3.5).

- $z_w = 0$ – lineární kombinace vektorů, body jsou v přímce,
- $z_w > 0$ – proti směru hodinových ručiček,
- $z_w < 0$ – ve směru hodinových ručiček.




Obrázek 3.5: Určení orientace tří bodů v rovině $xy$ v pořadí $A$, $B$, $C$ – proti směru hodinových ručiček vlevo, ve směru vpravo.

V případě, že trojice bodů $A$, $B$ a $C$ leží v jedné přímce, platí, že $C \in AB$ právě tehdy, když

$$\begin{aligned} x_C \leq \max(x_A, x_B) \wedge x_C \geq \min(x_A, x_B) \wedge \\ y_C \leq \max(y_A, y_B) \wedge y_C \leq \min(y_A, y_B). \end{aligned} \tag{3.1}$$

**Protínání dvou úseček**

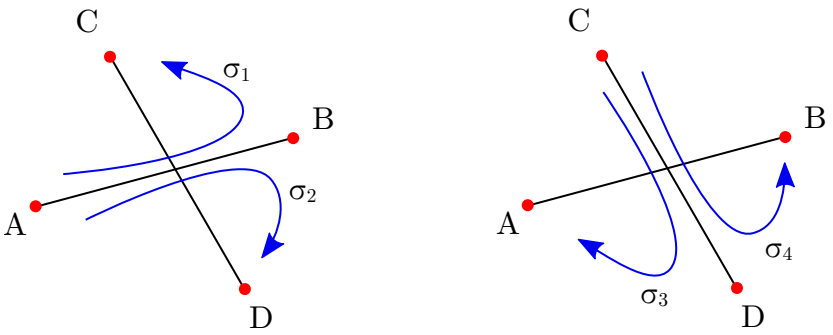
Pro ověření, zda se úsečky $AB$ a $CD$ protínají, je třeba zjistit orientace bodů ve čtyřech následujících případech, z nichž každý kombinuje úsečku a jeden bod z úsečky druhé:

1. kombinace $AB$ a $C$ – orientace $\sigma_1$ bodů v pořadí $A$, $B$ a $C$,
2. kombinace $AB$ a $D$ – orientace $\sigma_2$ bodů v pořadí $A$, $B$ a $D$,
3. kombinace $CD$ a $A$ – orientace $\sigma_3$ bodů v pořadí $C$, $D$ a $A$,
4. kombinace $CD$ a $B$ – orientace $\sigma_4$ bodů v pořadí $C$, $D$ a $B$.

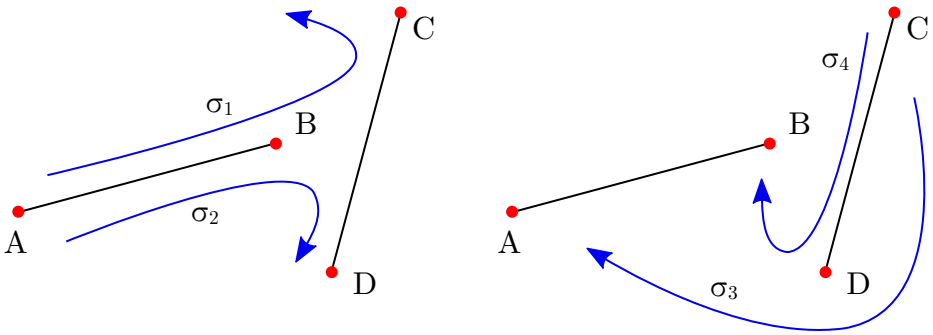
Pro úsečky $AB$ a $CD$ vždy platí, že se protínají právě tehdy, když

$$\sigma_1 \neq \sigma_2 \wedge \sigma_3 \neq \sigma_4. \tag{3.2}$$

To je demonstrováno na obrázku 3.6, kde jsou zobrazeny všechny situace, které mohou při průběhu algoritmu nastat. Modrou barvou jsou naznačeny orientace pro jednotlivé trojice bodů. Jiným situacím je zabráněno díky dodržení kritérií pro získávání extrémního bodu.




(a) Protínající se úsečky: $\sigma_1 \neq \sigma_2$ a $\sigma_3 \neq \sigma_4$.



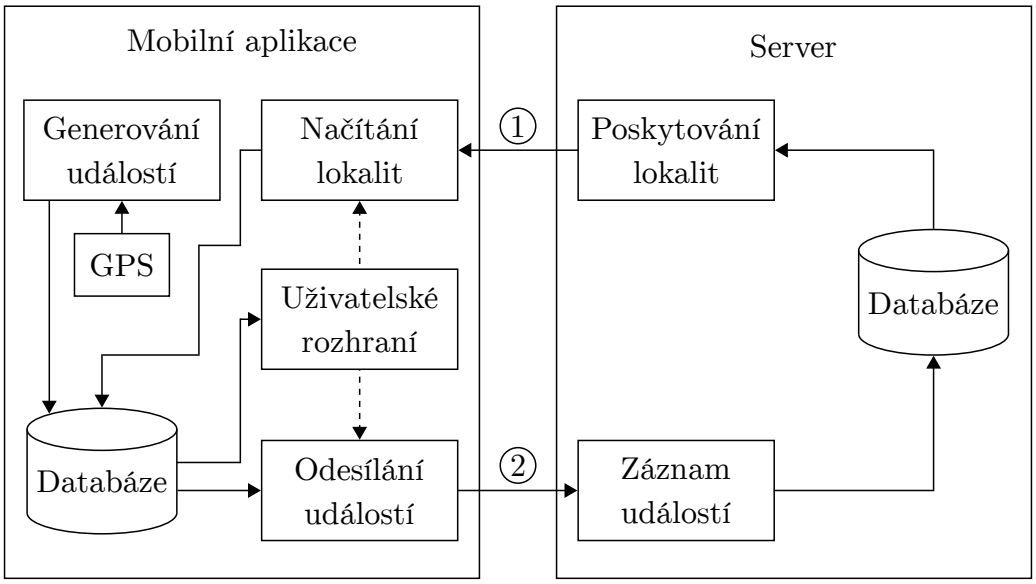

(b) Neprotínající se úsečky: $\sigma_1 \neq \sigma_2$ a $\sigma_3 = \sigma_4$.

Obrázek 3.6: Názorná ukázka platnosti výrazu 3.2.

# 4 Návrh a implementace systému

## 4.1 Architektura systému

Pro řešení zadaného problému bylo žádoucí a vyhovující využít architekturu *klient–server* a rozdělit tak systém na dva samostatně pracující celky, které spolu musí být schopny dle předem stanovených pravidel vzdáleně komunikovat.




Obrázek 4.1: Schéma architektury systému

První ze dvou zmíněných částí systému je server. Ten poskytuje služby, které zprostředkovávají přenos dat mezi ním a klientskými aplikacemi. Obsahuje databázi lokalit, jejichž seznam s potřebnými podrobnostmi umožňuje na požádání klientům distribuovat (vazba č. 1 v obrázku 4.1). Jeho další funkcí je schopnost data v daném formátu od klientů přijímat (vazba č. 2 v obrázku 4.1) a v této databázi je uchovávat. Detailní popis serverové části práce lze nalézt v sekci 4.3 na straně 23.

Druhou – klíčovou – část systémového celku tvoří samotná klientská aplikace, která je určena pro mobilní zařízení. Tato aplikce zabezpečuje detekci pohybu zařízení. Zjišťuje, zda její uživatel překročil hranici některé z lokalit.

Seznam těchto lokalit, který poskytuje na žádost serverová část, se ukládá v zařízení. V případě, že k překročení hranice dojde, má aplikace za úkol vygenerovat událost a uložit ji také v interní databázi zařízení. Prostřednictvím této aplikace je dále možné upravit některé detaily těchto událostí a následně je odeslat na server. Veškeré bližší informace týkající se mobilní aplikace jsou k nalezení v sekci 4.4 na straně 28.

## 4.2 Vstupní data

### 4.2.1 Lokality

Obsahem databáze nacházející se na serverové straně systému jsou informace o lokalitách. Takovou lokalitou se rozumí ohraničená část zemského povrchu. Vnější hranice této plochy určuje tzv. *hraniční polygon*. Informace o přesné podobě lokalit jsou popsány níže v sekci *Formát*.

Kromě dalších atributů definujících lokality (podrobně popsaných v sekci 4.3.1 na straně 23) jsou zde uloženy body (souřadnice), ze kterých je daná lokalita tvořena.

#### Původ

V dnešních geografických systémech pracujících s mapovými podklady týkajícími se území České republiky se využívá převážně souřadnicový systém s názvem *S-JTSK / Krovak, East North*. Jedná se o tzv. zápornou projekci systému S-JTSK / Krovak, South West (popsaného v sekci 3.1.2 na straně 11). Souřadnice jsou tentokrát vedeny v pořadí $X, Y$. Mezi souřadnicemi „záporného“ $(X_{SW}, Y_{SW})$ a „kladného“ $(Y_{EN}, X_{EN})$ systému platí následující vztahy:

$$X_{SW} = -Y_{EN} \text{ a } Y_{SW} = -X_{EN}. \text{ [1]}$$

Zdrojem dat pro tento systém (a tedy pro vytvoření seznamu polygonů) je veřejný registr půdy zvaný *LPIS* (Land Parcel Identification System) spravovaný Ministerstvem zemědělství České republiky. Jedná se o geografický informační systém, který je primárně tvořen evidencí využití zemědělské půdy. Jeho hlavním účelem je ověřování údajů v žádostech o dotace ze zdrojů Evropské unie i z národních dotačních programů ve vazbě na zemědělskou půdu. [5]

Konkrétní data z tohoto portálu lze v různých, předem definovatelných, strukturách získat prostřednictvím veřejné webové služby v datovém formátu XML. Souřadnice bodů tvořících polygony lokalit jsou v tomto výstupu

zaznamenány právě ve zmiňovaném „záporném“ systému. Tím dochází ke sporu, neboť GPS moduly v dnešních běžně používaných mobilních zařízeních pro definici polohy využívají souřadnicový systém WGS-84 (popsaný v sekci 3.1.1 na straně 10). Pro urychlení následné výpočetní práce bylo tedy rozhodnuto, že veškeré souřadnice budou v tomto projektu figurovat právě v systému WGS-84.

#### Formát

Jelikož se při rozdělování dotací zohledňuje výměra zemědělské půdy, kterou v tomto systému představují zmiňované lokality, nutně musí existovat možnost uvnitř jejich hraničního polygonu vytvořit polygony další – vnitřní. Ty představují tzv. „díry“ a vymezují tak v lokalitě prostor, který k ní nepatří (například les či budova). Tyto situace jsou jak v tomto, tak i v systému LPIS zohledněny.

Každá lokalita je tak v systému definována jako uspořádaná $n$-tice polygonů $(P_0, ..., P_{n-1})$, kde $n \in \mathbb{N}$. Pro každý polygon $P_i$, kde $i \in \mathbb{N}_0$, pak platí:

$$P_i \text{ je } \begin{cases} \text{hraniční polygon} & \text{pro } i = 0, \\ \text{vnitřní polygon} & \text{pro } i > 0. \end{cases}$$

Pro všechny vnitřní polygony platí, že se nikdy nemohou vzájemně překrývat a zároveň zasahovat za hranice polygonu vnějšího.

Dále pro každou dvojici z množiny všech lokalit z jejich charakteru vyplývá, že se dané lokality nikdy nemohou překrývat.

Bližší podrobnosti týkající se nahrávání dat o lokalitách do databáze jsou popsány v sekci 4.3.1 *(Pořizování dat o lokalitách do databáze serveru)* na straně 25.

### 4.2.2 Poloha

Dalšími daty, které vstupují do tohoto systému, jsou údaje o aktuální poloze zařízení, v němž je spuštěna hlavní aktivita aplikace. Jedná se o GPS souřadnice, které jsou získávány pomocí volně dostupné externí knihovny (více v sekci 4.4.5 – *Kontinuální zjišťování polohy* – na straně 38). Potřebné jsou pro tento systém pouze složky zeměpisné šířky a délky, neboť se na lokality pohlíží jako na rovinné útvary (sekci 3.1 na straně 9) a nadmořská výška ve výpočtech tedy nehraje žádnou roli.

## 4.3 Server

Oddělenou součást tohoto projektu představuje serverová část sestávající ze samostatné databáze a serverové aplikace. Nutné je poznamenat, že tato část není součástí zadání práce a vytvořena byla pouze pro testovací účely. Jedná se o velice jednoduché řešení, nicméně pro funkčnost systému plně dostačuje.

Klíčovou úlohou serverové aplikace je poskytovat klientské aplikaci jednotný přístup k databázi a zajistit tak vhodnou a bezproblémovou komunikaci. Pro jednoduchý způsob komunikace byla zvolena webová služba. Více informací o tomto typu komunikace se nachází sekci 4.3.2 *(Webová služba)* na straně 26).

### 4.3.1 Databázový systém

Úkolem databázového systému je uchovávání dat týkajících se zejména dvou hlavních oblastí (zmíněných v sekci 4.1 na straně 20), které systém pokrývá. První oblast představují data, která definují vstupní lokality, druhou oblast ta, jenž popisují události zaznamenané a zaslané prostřednictvím klientské aplikace.

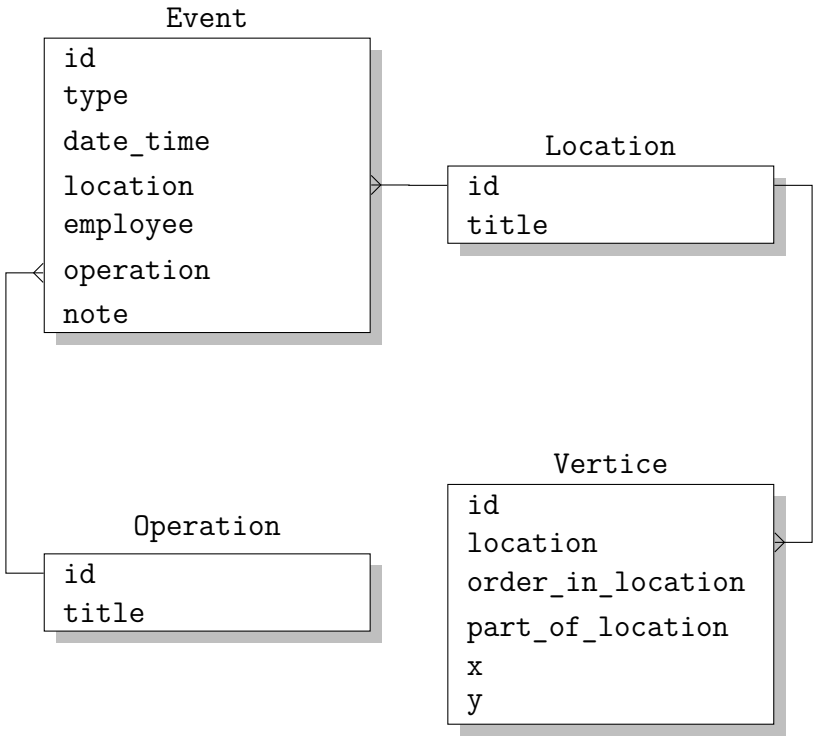
#### HyperSQL databáze

Rozsah databázového systému z hlediska požadavků na počet tabulek není nikterak velký – jedná se konkrétně o 4 tabulky, jejichž konkrétní podoba je definována v následující sekci *(Databázový model)*.

Díky své jednoduchosti byla pro tento systém vybrána HyperSQL databáze (HSQLDB). Jedná se o moderní relační databázový (transakční) software. Často je využíván jak pro vývoj a testování, tak i pro nasazení v běžném provozu. Tento systém zároveň zahrnuje výkonný SQL příkazový řádek a jednoduché uživatelské rozhraní pro vykonávání databázových operací. [2]

#### Databázový model

Pro tento databázový systém byl navržen jednoduchý datový model. Popis tohoto modelu je pro přehlednost zobrazen v tzv. entitně relačním (ER) diagramu (na obrázku 4.2). Tento typ diagramu je standardem pro popis datových modelů. Jeho základními prvky jsou entity (objekty reálného světa) a relace (vazby mezi nimi). Každá entita je dále popsána pomocí atributů.




Obrázek 4.2: ER diagram databáze.

V následujícím seznamu jsou popsány všechny tabulky implementované databáze. Primárním klíčem každé tabulky je atribut `id`, který slouží k jednoznačné identifikaci jakékoli entity v systému.

- **`Location`** – Obsahem této tabulky jsou jednotlivé lokality. Každá lokalita má kromě primárního klíče název uložený v atributu `title`.
- **`Vertice`** – Zde se nachází jednotlivé body všech lokalit z tabulky `Location`. Každý bod náleží právě jedné lokalitě a patří vždy do právě jednoho z jejích dílčích polygonů (tuto informaci nese atribut `part_of_location`; popis v sekci 4.2.1 *(Formát)* na straně 22). V atributu `order_in_location` má zároveň každý z bodů své pořadí vztahující se k celé lokalitě nehledě na polygon, ve kterém se nachází. Dále je daný bod pochopitelně definován souřadnicemi `x` a `y` (uloženými konkrétně v systému WGS-84 s přesností na 7 desetinných míst).
- **`Event`** – Tato tabulka obsahuje přijaté události z klientské aplikace. Každá tato událost má následující vlastnosti: typ (vjezd nebo výjezd do, resp. z lokality), datum a čas (kdy se událost odehrála), identifikátory uživatele, lokality a operace (který uživatel ve které lokalitě událost způsobil a co zde případně prováděl) a textovou poznámku.

- `Operation` – Tabulka `Operation` představuje číselník. Jedná se o seznam primárních klíčů a k nim přiřazeným názvům operací, které mohou být provedeny při dané události v lokalitě.

#### Pořizování dat o lokalitách do databáze serveru

Data, která jsou potřebná pro správný chod tohoto systému, jsou dvojího druhu. Jedná se o data týkající se lokalit (tedy tabulek `Location` a `Vertice`) a operací (tedy tabulky `Operation`).

Vzhledem k tomu, že konkrétní získávání dat o lokalitách není předmětem této bakalářské práce, nebyl zadavatelem projektu nikterak specifikován způsob jejich nahrávání do databáze. S ohledem na tuto skutečnost nebyl import lokalit a operací vůbec programově řešen.

Funkční způsob, jak lze tato data do databáze vložit, je popsán v *Uživatelské příručce* (v sekci C.1 na straně 58).

### 4.3.2 Serverová aplikace

Serverová aplikace má v systému úlohu poskytovatele přístupu k databázi. Zabezpečuje jednak sběr a zpracovávání dat s účelem uložit je do serverové databáze a jednak poskytování dat v databázi uložených.

#### Využité technologie

V této sekci se nachází stručný popis technologií, které dohromady tvoří funkční celek serverové aplikace.

Základem serverové aplikace je webový server *Jetty*. Při návrhu systému bylo potřeba počítat s tím, že v rámci komunikace aplikace serveru s relační databází bude potřeba zohlednit rozdílnost objektově orientovaného a relačního přístupu.

K tomu slouží tzv. *Java Persistence API* (JPA). To je standard, který popisuje rozhraní pro správu relačních dat v aplikacích vytvořených v programovacím jazyce Java. Využívá k tomu *objektově relační mapování* (ORM). Definuje tzv. entity, které typicky představují tabulky v databázi, přičemž každá instance dané entity odpovídá řádku v dané tabulce. Obdobu tabulky tak v Javě představuje tzv. entitní třída. [4]

Samotné ORM je obecně technika, která zjednodušuje a automatizuje přemostění mezi dvěma výše zmíněnými paradigmaty – objektově orientovaným

a relačním. Jeho cílem je tedy upřednostnění objektového přístupu k tabulkám v databázi. V JPA se konkrétní mapování atributů entitní třídy na atributy tabulky implementuje pomocí *Java anotací*.

Jednou z nejrozšířenějších implementací JPA je *Hibernate*, který byl v tomto systému využit. Jedná se o výkonný nástroj který prostřednictvím výše zmíněného ORM zabezpečuje persistenci dat v systému. [14]

Řízení procesu sestavování aplikace je realizováno pomocí nástroje *Apache Maven*, který zabezpečuje instalaci a spouštění externích knihoven.

### Webová služba

Webová služba je obecně softwarový standard poskytující prostředky pro vzájemnou komunikaci různých aplikací běžících na různých strojích (i platformách) v síti. Pro tento projekt byla zvolena varianta komunikace s webovou službou typu *REST* (Representational State Transfer). Jedná se o architekturu takové aplikace, jejíž část je vykonávána na jiném stroji v síti. Komunikace probíhá prostřednictvím HTTP (Hypertext Transfer Protocol) protokolu s využitím tzv. CRUD operací, kterými jsou: Create (vytvořit), Read (číst), Update (editovat) a Delete (smazat).

Pro usnadnění vývoje aplikací (v programovacím jazyce Java) využívající architekturu REST, existuje rozhraní, které se nazývá Java API for RESTful Web Services, zkráceně JAX-RS. Pomocí JAX-RS anotací se ve zdrojových kódech definují zdroje a akce, které lze s těmito zdroji provádět. [3]

V tomto projektu bylo využito právě jedné z nejrozšířenějších implementací JAX-RS – *RESTEasy*.

Komunikace typu REST podporuje přenos dat ve formátech XML, JSON (JavaScript Object Notation) či prostého textu. Pro tento systém se jeví jako nejpříhodnější formát JSON, neboť by měl zajistit mírnou úsporu přenášených dat. Implementovaná webová služba tedy z pohledu klienta poskytuje dvě základní operace.

V prvním případě jde o **stažení dat o lokalitách**. HTTP požadavek typu GET musí mít následující podobu.

```
ADRESA_SERVERU:PORT/locations
```

Odezvou serveru je v případě úspěchu seznam všech dostupných lokalit včetně jejich bodů ve formátu JSON.

V případě druhém se jedná o **odeslání dat o události**. Tuto operaci, jejíž volání je zobrazeno níže, je možné vyžádat s pomocí požadavku typu POST.

```
ADRESA_SERVERU:PORT/newEvent
```

Zároveň je s touto zprávou tedy nutné odeslat data o události ve formátu JSON. Pro správný chod systému musí tato data tyto atributy v následujícím pořadí:

- **`type`** – typ události, může nabývat hodnot `ARRIVAL` (vstup do lokality) a `DEPARTURE` (výstup z lokality),
- **`dateTime`** – datum a čas události ve vybraném formátu mezinárodní normy *ISO 8601* (pro konkrétní příklad by datum 12. dubna 2019, 21 hodin, 53 minut a 11 sekund na území České republiky bylo odesláno takto: `2019-04-12T21:53:11+02:00`),
- **`employee`** – jedinečný identifikátor uživatele, který způsobil událost,
- **`location`** – jedinečný identifikátor lokality, ve které se stala událost,
- **`operation`** – jedinečný identifikátor operace, která byla v lokalitě provedena (v případě události vstupu nastaveno na hodnotu `-1`),
- **`note`** – textová poznámka k události (v případě události vstupu nastaveno na hodnotu `null`).

Příkladem události vstupu do lokality může být následující zpráva.

```
{
   "type":"ARRIVAL",
   "dateTime":"2019-04-21T07:10:00+02:00",
   "employee":13,
   "location":1,
   "operation":-1,
   "note":null
}
```

Příklad opačné události – výstupu z lokality – je zobrazen v této ukázce.

```
{
   "type":"DEPARTURE",
   "dateTime":"2019-04-21T11:37:00+02:00",
   "employee":21,
   "location":3,
   "operation":2,
   "note":"5 hodin prace, 30 minut pauza"
}
```

Tato webová služba je implementována zcela bez zabezpečení, neboť je součástí serveru, jehož účelem je pouhé testování komunikace. Přístup ke stahování seznamu lokalit a odesílání událostí má tedy kdokoli. Přenášený identifikátor uživatele `employee` tedy není nikterak závazný a v mobilní aplikaci je možné jej nastavit na libovolné kladné číslo.

## 4.4 Mobilní aplikace

Obsahem této sekce jsou veškeré informace týkající se návrhu, vývoje a testování cílové mobilní aplikace – hlavního cíle této bakalářské práce.

### 4.4.1 Využité technologie

Vývoj aplikací určených pro dva v dnešní době jednoznačně nejrozšířenější operační systémy na mobilních zařízeních – iOS a Android (dříve i Windows Phone) – probíhal vždy odděleně. Pro každý z těchto systémů existuje tzv. nativní jazyk, ve kterém lze aplikace vyvíjet. Pro Android je tímto jazykem Java, pro iOS je to Swift, dříve Objective-C (pro Windows Phone jím byl C#).

#### Xamarin

Změna v přístupu k vývoji nastala v roce 2011, kdy byl vyvinut framework *Xamarin* (čti: [zæmərɪn]). Ten je od roku 2016 vlastněn společností Microsoft, která jej poskytuje zcela zdarma v rámci vývojového prostředí Visual Studio. Jedná se o nástroj, který umožňuje tvorbu aplikací pro obě dominantní platformy (iOS i Android) s jednou zásadní výhodou – umožňuje mezi

jednotlivými projekty sdílet až 90 % zdrojového kódu, konkrétně psaného ve vysokoúrovňovém objektově orientovaném jazyce C#. Xamarin poskytuje jednotné API pro přístup k běžným zdrojům napříč oběma platfomami, čímž – prostřednictvím sdílení kódu – zásadně redukuje náklady a čas na vývoj. Výstupem kompilování kódu je jsou však totožné soubory, jako při nativním vývoji.

Vývoj aplikace pro obě platformy tak v praxi probíhá ve třech projektech. Jeden – sdílený – ovšem obsahuje převážnou část kódu (na platformě nezávislého) celé aplikace, ostatní dva projekty specifikují pouze ty záležitosti, které sdílený kód nepokrývá a je nutné je vyřešit odděleně.

Nástroj Xamarin využívá tzv. *.NET Base Class Library* (BCL), což je rozsáhlá kolekce tříd, která zahrnuje práci s textem, I/O zařízeními či databází. Dále poporuje například serializaci či síťování a mnoho dalších, běžně i méně často, používaných záležitostí z různých oblastí vývoje.

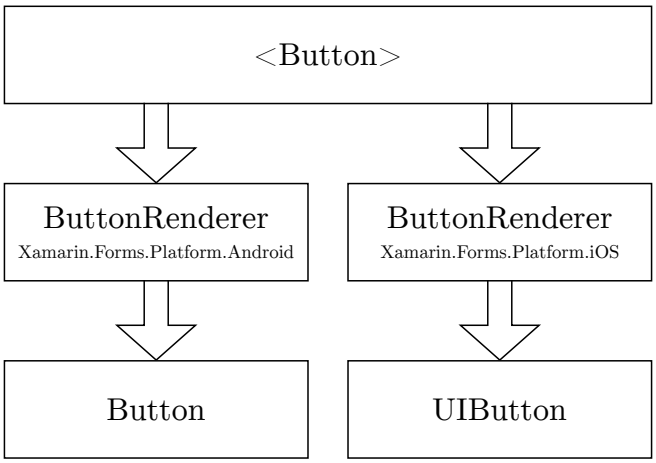
Platforma .NET využívá pro sdílení kódu mechanismu zvaného NuGet. Do projektů je tak možno přidávat externí knihovny v podobě tzv. *NuGet balíčků*. Výhodu pro tento projekt představují zejména balíčky, které umožňují jednotný přístup k určité komponentě zařízení, která bez použití tohoto balíčku vyžaduje nativní přístup.[8]

### Xamarin Forms

Pro usnadnění vývoje grafického uživatelského rozhraní (GUI – Graphical User Interface) existuje nadstavba Xamarinu s názvem *Xamarin.Forms*. Jde o framework obsahující množství abstraktních modelů grafických komponent (jako např. tlačítek či textových polí), pomocí kterých je možné GUI definovat ve sdíleném projektu. Jedná se tedy o rozšíření množiny záležitostí, ke kterým lze přistupovat pomocí sdíleného kódu.

Výše zmíněné abstraktní modely jsou za běhu aplikace mapovány na nativní komponenty (obrázek 4.3). Pro každý model jsou vždy připraveny tzv. renderery, což jsou třídy, které zabezpečují komunikaci s nativními komponentami obou platforem. Díky tomuto mapování – pokud nedojde k záměrné změně programátorem – je zaručeno, že design výsledné aplikace bude stejný jako při nativním vývoji. [9]

Pro zjednodušení návrhu grafického uživatelského rozhraní se v .NET aplikacích často používá značkovací jazyk *XAML* (Extensible Application Markup Language). Jedná se o jazyk založený na XML, který napomáhá k oddělení logiky od čistého designu aplikace.



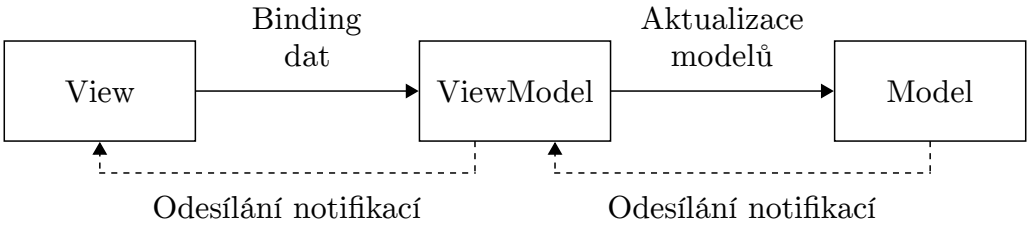

Obrázek 4.3: Příklad mapování abstraktního modelu `<Button>` na nativní tlačítka daného operačního systému. [9]

### 4.4.2 Základní koncept

V projektech často dochází v průběhu vývoje k problémům díky těsnému propojení mezi logikou aplikace a ovládacími prvky uživatelského rozhraní. Aby tyto problémy nenastaly, je nutné dané části aplikace oddělit. Dosažení této separace usnadňuje vývoj, údržbu i testování aplikace.

Jedním z návrhových vzorů zabezpečující přehledné oddělení logiky aplikace od GUI, který byl využit i pro tento projekt, je vzor *Model–View–ViewModel* (MVVM). Jedná se o popis architektury aplikace a o způsob propojení jejích jednotlivých částí.

Aplikace navržená podle tohoto vzoru se rozděluje na tři základní vrstvy, které jsou zobrazeny na obrázku 4.4 včetně jejich vzájemných vztahů.




Obrázek 4.4: Schéma vrstev návrhového vzoru MVVM. [10]

Případné implementační změny se týkají jen jedné z vrstev, a tak jsou jednodušší a méně náročné. Tyto vrstvy jsou blíže představeny v následujícím seznamu.

- **Model** – V této vrstvě se nacházejí třídy popisující data, se kterými aplikace pracuje. Jde o neviditelné třídy, které nemají žádné informace o stavu ovládacích prvků aplikace. Zpravidla se používají při přenosu a práci s daty, typicky při spojení aplikace se službami či různými typy úložišť.
- **View** – Tato vrstva reprezentuje uživatelské rozhraní aplikace. Je tedy zodpovědná za jeho strukturu a vzhled. V ideálním případě je definováno pomocí souborů v jazyce XAML, které v přidružených souborech obsahují omezené množství kódu (již v jazyce C#). Ty by neměly obsahovat logiku aplikace, která patří do vrstvy ViewModel.
- **ViewModel** – Vrstva ViewModel propojuje Model a View a udržuje stav aplikace. Pomocí tzv. *bindingu* jsou na tuto vrstvu napojeny ovládací prvky uživatelského rozhraní. Aby se změny stavu aplikace ve View automaticky projevily, je potřeba, aby třídy ViewModelu implementovaly rozhraní `INotifyPropertyChanged`. Při změně je tak vyvolána událost `PropertyChanged`, na kterou View reaguje.

### 4.4.3 Rozšířený koncept

Na základě výše popsaného modelu byla navržena architektura konkrétně pro tuto aplikaci. Sdílený projekt (s pro obě platformy společným kódem) byl přehledně rozčleněn do dílčích, samostatně fungujících, celků. To bylo učiněno pro zpřehlednění, zejména pak ovšem pro seskupení částí aplikace, které řeší konkrétní problém, a k zajištění jejich vzájemné implementační nezávislosti. Tento rozšířený model je zobrazen na obrázku 4.5.

### 4.4.4 Popis vrstev aplikace

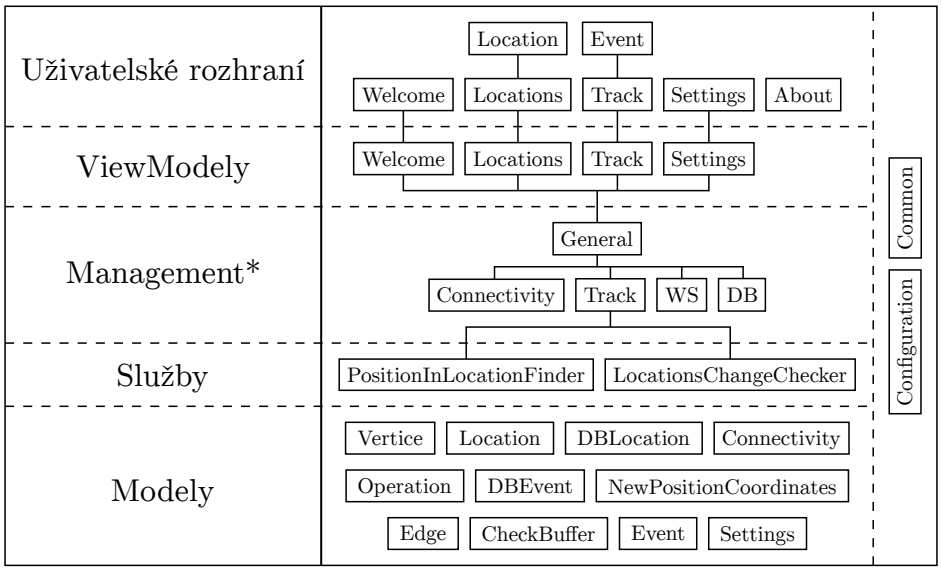
V této sekci se nachází popis vlastností a funkcionalit jednotlivých vrstev rozšířeného modelu mobilní aplikace včetně jejich vzájemných vazeb.

#### Uživatelské rozhraní

Tato vrstva obstarává pouze grafické uživatelské rozhraní. Definuje strukturu a vzhled jednotlivých stránek aplikace.

Pro každou tuto stránku existuje dvojice zdrojových souborů. Jednak XAML soubor, ve kterém je vždy definována grafická podoba stránky, a jednak soubor se zdrojovým kódem v jazyce C#, kde se nachází informace o tom, který z ViewModelů bude pro tuto stránku zdrojem dat. Zároveň jsou zde také definovány reakce na události, které mohou nastat při interakci GUI s uživatelem aplikace.

Podrobný popis všech sránek aplikace je k nalezení dále v sekci 4.4.5 *(Implementace uživatelského rozhraní)* na straně 35.




*V projektu mají všechny soubory této vrstvy na konci názvu slovo „Manager“ (tj. např. TrackManager).

Obrázek 4.5: Schéma aplikace.

### ViewModely

Vrstva ViewModelů v rámci této aplikace obsahuje třídy, které zabezpečují získávání a zpracovávání dat od centrálního řídícího bodu aplikace (tím je tzv. `GeneralManager`, který je popsán v následující sekci *Management*) za účelem předávání informací uživatelskému rozhraní (jednotlivým stránkám), které je následně prezentuje.

ViewModely jsou v aplikaci celkem čtyři, přičemž každý z nich obsluhuje „svou stránku“. Daná stránka uživatelského rozhraní přidružená k danému ViewModelu jednoznačně vyplývá z jeho samotného názvu.

### Management

Řízení chodu aplikace a jejího toku dat je zabezpečeno právě v této logické vrstvě. Nachází se zde nejdůležitější přístupový prvek celé aplikace, na který jsou napojeny čtyři (obecně ovšem libovolné množství) úzce specializované a navzájem nezávislé moduly.

Zmíněným centrálním prvkem celé mobilní aplikace je tzv. `GeneralManager`. Je to třída, která ve svých metodách volá metody napojených modulů nezávisle na způsobu jejich implementace. Zprostředkovává tak všem ViewModelům služby, které poskytují zmíněné moduly.

`GeneralManager` je naprogramován dle návrhového vzoru *Singleton*. Ten se obecně využívá pro třídy, u kterých je potřeba zajistit, aby měly v programu právě jednu instanci. Zároveň by neměl být tento objekt vytvořen dříve, nežli je poprvé přistupován. Cestou k tomuto řešení je vytvoření jediného privátního konstruktoru této třídy, jedné privátní instanční proměnné a v poslední řadě veřejné statické metody, která přístup k proměnné (včetně jejího prvotního instanciování) zajistí. [12]

V tomto seznamu jsou popsány všechny moduly, které `GeneralManager` využívá.

- **`TrackManager`** – Tato část systému zabezpečuje spojení s integrovaným GPS modulem v zařízení. Dosahuje toho velmi efektivně prostřednictvím NuGet balíčku *Xam.Plugin.Geolocator*, jehož popis se nachází v sekci 4.4.5 *(Kontinuální zjišťování polohy)* na straně 38. Tato třída je jediná z vrstvy *Managementu*, která využívá třídy z vrstvy *Služeb* – jak `PositionInLocationFinder`, tak i `LocationsChangeChecker`.

- **`WSManager`** – V tomto modulu probíhá kontakt se serverem, konkrétně s webovou službou. Obsahuje dvě části, z nichž jedna zajišťuje příjem dat, druhá naopak odesílání. Více informací ohledně práce s daty související s tímto přenosem dat je k nalezení v sekci 4.4.5 *(Příjem a ukládání dat)* na straně 37.

- **`DBManager`** – Pro spojení a následnou práci s interní databází v zařízení slouží v aplikaci tento modul. Zahrnuje metody, pomocí nichž je možné vkládat a mazat záznamy do, resp. z tabulek obsahujících následující data:

  - lokality stažené ze serveru,
  - jednotlivé body lokalit,
  - vygenerované události.

Jsou zde implementovány také metody pro konkrétní dotazy nad tabulkami. Další informace o problematice ukládání dat v zařízení jsou popsány v sekci 4.4.5 *(Příjem a ukládání dat)* na straně 37.

- `ConnectivityManager` – V tomto případě se jedná o oddělení jednoduché části programu, konkrétně zjištění, zda je zařízení připojeno k internetu a zda je povoleno zjišťování polohy.

### Služby

Služby, které aplikace – konkrétně tedy pouze `TrackManager` – využívá, jsou popsány v následujícím seznamu. Těmito službami jsou míněny interní aplikační zdrojové kódy (třídy), které jako oddělené moduly řeší konkrétní problematiku.

- `PositionInLocationFinder` – Tato třída slouží k ověřování, zda aktuální poloha zařízení náleží jedné konkrétní lokalitě. Nutné je říci, že tato lokalita musí splňovat všechna kritéria pro vznik lokality. Informace ohledně implementace této části systému jsou v sekci 4.4.5 *(Určování výskytu aktuální polohy v lokalitách)* na straně 39.
- `LocationsChangeChecker` – Tato třída řeší problematiku rozpoznávání změn oblastí (jedna z lokalit, či okolní prostor) při pohybu zařízení v reálném čase. Její účel (včetně implementace) je podrobněji vysvětlen v sekci 4.4.5 *(Řešení problému častých vstupů)* na straně 40.

### Modely

V této vrstvě se nachází třídy pro práci s daty. Jedná se například o popis striktního formátu v jakém se data přijímají ze serveru, ukládají v interním úložišti a naopak odesílají z aplikace.

### Správa aplikace

Pro bezproblémový chod aplikace bylo potřeba zavést speciální vrstvu přístupnou ze všech ostatních vrstev aplikace. Jejím obsahem jsou dva soubory (zdrojové kódy v jazyce C#), na které se obrací různé třídy ze všech vrstev aplikace. Jejich bližší popis je k nalezení níže.

- `Common` – Obsahem prvního z těchto souborů je sada konstantních hodnot a výčtových typů, které jsou důležité pro správné fungovaní

převážné většiny tříd napříč celou aplikací. Tyto hodnoty by pro zachování funkcionality aplikace neměly být měněny. Dále soubor například obsahuje metodu, která zabezpečuje převod data na požadovaný textový formát.

- `Configuration` – Druhý soubor obsahuje naopak nastavitelné hodnoty, které aplikace využívá jako výchozí, pokud je uživatel nezmění na stránce `SettingsPage`. Jedná se v první řadě o adresu serveru a číslo portu, na kterém běží webová služba, dále pak o dvě hodnoty, kterými lze nastavit, jak často bude aplikace dotazovat GPS modul v zařízení a jaká minimální vzdálenost vyvolá událost, že poloha zařízení se změnila.

  Dalšími konstantami, které již není možné nastavit v aplikaci (a pro jejich zohlednění v aplikaci je nutné provést kompletní překlad), jsou dva řetězce, pomocí kterých lze volat webovou službu. Aplikace z nich tvoří URI (Uniform Resource Identifier), na které se odesílají HTTP požadavky za účelem stažení lokalit nebo odeslání událostí.

  Poslední položkou v tomto souboru je číselník (název a identifikátor) operací, ze kterých je na stránce `EventPage` (pouze, když typem události je výjezd) uživatel nucen jednu označit za provedenou. Tento číselník je staticky zapsaný v kódu aplikace a celý systém tedy pro zajištění konzistence musí splňovat předpoklad, že na straně serveru je v databázi tento číselník naprosto totožný.

### 4.4.5 Realizace

Jak bylo zmíněno v sekci 4.4.1 *(Xamarin)* na straně 28, vzhledem k rozdílnosti platforem je třeba k některým záležitostem přistoupit nativně – tedy určitou část kódu připravit zvlášť pro Android i iOS odděleně v příslušných projektech. Jednou z nich je inicializace samotného vstupního bodu celé aplikace. Tím je třída `App`, která je již obsahem sdíleného projektu.

#### Implementace uživatelského rozhraní

Každá stránka si přidružený ViewModel (za předpokladu, že jej má – více v obrázku 4.5 na straně 32) inicializuje sama ve svém konstruktoru. Při inicializaci vstupního bodu dojde k nastavení kořenové stránky aplikace na `WelcomePage`, přičemž je zavolán i její konstruktor. V tuto chvíli již tedy aplikace může běžet.

Jednotlivé stránky aplikací vytvořených pomocí frameworku *Xamarin.Forms* je možné různě hierarchicky uspořádat. Nejvhodnější pro tuto aplikaci je možnost vzájemně je vrstvit ve stylu zásobníku. Proto k tomuto zásobníku existují speciální metody pro přidávání a odebírání stránek. Využití těchto metod je zobrazeno v následující ukázce.

```
1   await Navigation.PushAsync(new SettingsPage());
2   await Navigation.PopAsync();
```

Aplikace je realizována s pomocí asynchronních metod. Tím je zabezpečeno, že uživatelské prostředí aplikace nepřestává reagovat ani při výpočtech či voláních vzdálených služeb. Tyto záležitosti jsou totiž vykonávány odděleně pomocí vláken, které jsou nezávislé na vláknech zabývajících se vykreslováním GUI a čtením požadavků uživatele.

Dle konceptu systému jsou všechny požadavky uživatele skrze ViewModely směřovány na `GeneralManager`, který je distribuuje dále do vyšších vrstev (obrázek 4.5 na straně 32).

V následujícím seznamu se nachází popis všech stránek nacházejících se ve vrstvě Uživatelského rozhraní. Popis ovládání aplikace včetně konkrétních ukázek se nachází v Uživatelské příručce v sekci C.2 na straně 59.

- **`WelcomePage`** – Stránka, která se zobrazuje ihned po spuštění aplikace. Zabezpečuje zobrazování stavu připojení k internetu a povolení ke zjišťování polohy v zařízení. Umožňuje přejít na stránku `LocationsPage`.
- **`SettingsPage`** – Obsahem této stránky je trojice zadávacích polí a dva posuvníky. První z polí vyžaduje řetězec identifikující uživatele aplikace, další dvě jsou určeny k zadávání hodnot potřebných pro připojení k serveru – adresy a portu. Uživatel má s pomocí posuvníků možnost měnit parametry dotazování GPS modulu na aktuální polohu (konkrétně se volí počet sekund mezi jednotlivými dotazy a nejmenší rozdíl vzdáleností, který se považuje za změnu polohy v metrech).
- **`LocationsPage`** – Na této stránce jsou v seznamu k nalezení stažené lokality (tj. uložené v interní databázi zařízení, sekce 4.4.5 – *Příjem a ukládání dat* – na straně 37). Zároveň je implementována možnost zaktualizování tohoto seznamu stažením lokalit prostřednictvím webové služby.

- **`LocationDetailPage`** – Zde se nachází jednoduchý detail lokality. Přejít sem je možné kliknutím na danou lokalitu v seznamu na stránce `LocationsPage`.

- **`TrackPage`** – Společně se vstupem této stránky na zásobník stránek se spouští samotné zjišťování polohy a ověřování, zda se zařízení nenachází v některé z lokalit (ukončuje se logicky s výstupem ze zásobníku). Tato stránka zobrazuje průběh této aktivity. Přehledně jsou zde automaticky zobrazovány souřadnice aktuální polohy (v souřadnicovém systému WGS-84) a název aktuální lokality (pokud se tedy v některé z nich zařízení v daný moment nachází). Zároveň se zde nachází seznam automaticky vygenerovaných (při každém vstupu a výstupu do a z lokality) událostí, které zatím nebyly odeslány na server a jsou tedy uloženy v interní databázi zařízení. Pokud tyto události splňují určité podmínky (jedná li se o výstup, uživatel má povinnost zadat operaci, kterou v dané lokalitě vykonal), je možné je prostřednictvím webové služby na server odeslat.

- **`EventPage`** – Kliknutím na libovolnou neodeslanou událost v seznamu na stránce `TrackPage` se zobrazí její podrobný detail, ve kterém je možné (v některém případě tedy i nutné) některé hodnoty zadat či pozměnit.

- **`AboutPage`** – Obsah této stránky představuje krátká informace o aplikaci.

### Příjem a ukládání dat

Pro komunikaci se serverem se v modulu `WSManager` vytváří tzv. `HttpClient` (s pomocí NuGet balíčku *Microsoft.Net.Http*), který vytváří HTTP požadavky dvojího typu. Prvním typem je GET požadavek, pomocí kterého je vyžádán seznam všech dostupných lokalit z databáze serveru. V případě, že vše proběhne správně, odpovědí webové služby je seznam lokalit v datovém formátu JSON. V tomto formátu je rovněž nutné odeslat události, tentokrát však prostřednictvím požadavku typu POST. Pro převody objektů do formátu JSON a nazpět je využíváno serializační, resp. deserializační metody. Obě tyto metody jsou poskytovány v NuGet balíčku *Newtonsoft.Json*.

V mobilních zařízeních je obecně potřeba dlouhodobě uchovávat data. Pro takové případy existuje v systémech Android i iOS relační databázový systém zvaný *SQLite*. S pomocí NuGet balíčku *sqlite-net-pcl* je aplikaci přístup k této databázi umožněn a data jsou zde s využitím modulu `DBManager`

ukládána obdobně, jako v databázi na straně serveru (sekce 4.3.1 *(Databázový model)* na straně 23). Výjimkou je zdejší absence tabulky `Operation`. Všechny operace, které je možné vybrat na stránce `EventPage` události výstupu z lokality, jsou v aplikaci zadány staticky ve zdrojovém souboru `Configuration`.

V mobilním zařízení dostávají nové události automaticky generovaný identifikátor `id`. Ten se pak (po odeslání) na straně serveru stává neplatným a dané události je přidělen identifikátor nový – jedinečný v rámci databáze serveru.

### Kontinuální zjišťování polohy

Velmi postatnou roli mezi využitými knihovnami má v této mobilní aplikaci také NuGet balíček – *Xam.Plugin.Geolocator*. Jde o rozšíření, které zajišťuje snadný přístup k datům z GPS modulu zařízení. Je uzpůsobené pro pravidelné získávání GPS souřadnic polohy dle předem definovaných pravidel. Nespornou výhodou této volby je nezávislost na cílové platformě – kód je stejný jak pro Android, tak i iOS.

V následující ukázce je zobrazen zdrojový kód (v jazyce C#) typického užití. Konkrétně se tedy jedná o spuštění automatického zjišťování polohy zařízení.

```
1   IGeolocator Geolocator = CrossGeolocator.Current;
2
3   async Task StartListening() {
4      await Geolocator.StartListeningAsync(
          TimeSpan.FromSeconds(INTERVAL_SEC),
          INTERVAL_MET);
5
6      Geolocator.PositionChanged += PosChanged;
7   }
8
9   void PosChanged(object sender, PositionEventArgs e) {
10     var newPosition = e.Position;
11
12     // obsluha události
13  }
```

Asynchronní metoda `StartListening` (v ukázce na řádku č. 3) zajišťuje spuštění procesu získávání dat z GPS modulu. Pomocí hodnot `INTERVAL_SEC` a `INTERVAL_MET` se nastavuje délka periody dotazování, resp. minimální vzdáleností, která se považuje za změnu polohy.

Pro případ, že `Geolocator` zaregistruje změnu polohy zařízení, je připravena obslužná metoda (tzv. *handler*) `PosChanged` (v ukázce na řádku č. 9). Proměnná `newPosition` pak mimo jiné obsahuje aktuální GPS souřadnice (`newPosition.Latitude` a `newPosition.Longitude`), čas jejich pořízení (`newPosition.Timestamp`) či rychlost pohybu (`newPosition.Speed`).

Tento způsob získávání souřadnic aktuální polohy je implementován v modulu `TrackManager`.

### Určování výskytu aktuální polohy v lokalitách

Při každé změně aktuální polohy je nad množinou (validních) lokalit spuštěna cyklická kontrola, zda se daná aktuální poloha nenachází v některé z nich.

Tato kontrola nad jednou konkrétní lokalitou je zajišťována pomocí modulu `PositionInLocationFinder`. Zde je implementován mírně modifikovaný algoritmus pro ověřování výskytu bodu v polygonu. Vychází se z faktu, že navržený algoritmus (popsaný v sekci 3.3 na straně 13) je možné využít i v případě, že lokalita je tvořena více polygony (definice v sekci 4.2.1 *(Formát)* na straně 22). Nutné je ovšem přizpůsobit procházení jednotlivých stran lokality. Zde je využito atributu každého bodu lokality – `part_of_location`, který značí, do kterého z polygonů v lokalitě tento bod patří. Tak je možné zjistit rozhraní mezi jednotlivými polygony a docílit tak správného rozpoznání a zahrnutí všech stran lokality.

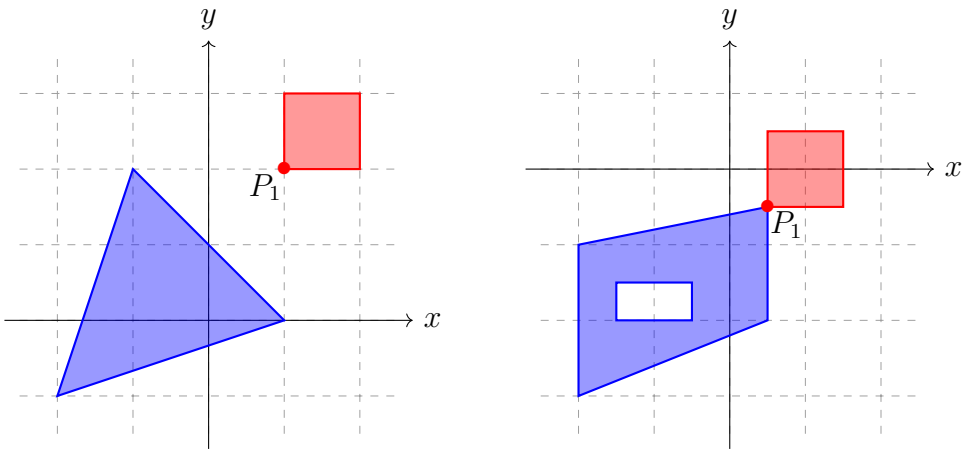
Do metody `IsInside` obsažené ve třídě `PositionInLocationFinder` vstupuje konkrétní lokalita a poloha zařízení, výstupem je pak tvrzení, zda se poloha v dané lokalitě vyskytuje, či ne.

Volba extrémního bodu (*1. krok algoritmu* – sekce 3.3 na straně 13) probíhá (rovněž ve třídě `PositionInLocationFinder`) dle následujícího postupu.

1. **Nalezení oblasti**

   Nejprve je nutné vymezit prostor, ve kterém bude extrémní bod $E$ ležet. Docíleno tak bude nalezením nejvyšších hodnot $x_{max}$ a $y_{max}$ jednotlivých souřadnic $x$ a $y$ mezi všemi vrcholy lokality. Tak vznikne bod $P_1[x_{max}, y_{max}]$, který společně s dalšími body $P_2[x_{max} + 1, y_{max}]$,

$P_3[x_{max}+1, y_{max}+1]$ a $P_4[x_{max}, y_{max}+1]$ vytvoří cílovou čtvercovou oblast. Tato záležitost je ilustrována na obrázku 4.6.




Obrázek 4.6: Nalezené oblasti (vyznačeny červenou plochou) pro volbu extrémního bodu u ukázkových lokalit (vyznačeny modrou barvou).

2. **Volba náhodného bodu**

   Pomocí metody `ChooseExtreme` je možné náhodně vygenerovat souřadnice potenciálního extrémního bodu $E$, který bude ležet ve zvolené oblasti.

3. **Kontrola splnění kritérií**

   Pro ověření, zda nově zvolený bod splňuje obě kritéria pro zvolení bodu extrémního, slouží metoda `TestExtremePossibility`. Dojde k cyklickému přezkoumání, zda tento bod společně s aktuální polohou a postupně všemi body lokality neleží v jedné přímce. Pakliže minimálně v jednom případě v přímce leží, je nutné volbu náhodného bodu a tuto kontrolu opakovat. V případě, že v přímce neleží, může být tento bod označen jako extrémní.

Cyklické procházení všech lokalit je předčasně ukončeno v případě, že je daná poloha v některé z lokalit nalezena. Výsledek, zda se aktuální poloha nachází mimo lokality či v některé z nich je následně předán službě `LocationsChangeChecker` (více v následující sekci).

### Určování vstupů a výstupů

Kromě zjišťování polohy je další úlohou modulu `TrackManager` určování, zda zařízení vstupuje či vystupuje do, resp. z některé z uložených lokalit.

V reálném světě však existuje za určitých podmínek možnost, že aktuální poloha zařízení nebude určena přesně. S vysokou pravděpodobností může tato situace nastat například v těsné blízkosti hustého lesa či budovy, s nižší pak i kdekoli jinde.

Již při návrhu aparátu pro rozpoznávání vstupů a výstupů zařízení do, resp. z lokalit bylo tedy nutné uvažovat následující modelovou situaci:

> *„Uživatel aplikace se pohybuje po cestě, která těsně přiléhá k hranici jedné z lokalit. Díky ne dokonale přesně určené poloze je možné, že ač je cesta zcela mimo lokalitu, aplikace zaznamená několik (teoreticky i velký počet) událostí – vstupů a výstupů do a z dané lokality.“*

K řešení tohoto problému bylo přistoupeno dvěma různými způsoby. Tím prvním – jednodušším – bylo implementování možnosti smazání vygenerované události na příslušné stránce `EventPage` (sekce 4.4.5 na straně 35).

Druhým způsobem bylo vytvoření mechanismu, který počet těchto nadbytečných událostí sníží. Zahrnut je ve třídě `LocationsChangeChecker`. V souboru `Configuration` je nastavena hodnota `COUNT_OF_CHECKS_ON_BORDERS` představující minimální počet v řadě jdoucích signálů z totožné oblasti (tou může být jakákoli z lokalit i okolní prostor mimo lokality).

Aplikace průběžně zaznamenává jedinečné identifikátory navštívených oblastí (pro prázdnou oblast využita konstanta `UNDEFINED`) do kruhové fronty o kapacitě $k$. Až pokud tedy aplikace zaznamená $k$ za sebou bezprostředně následujících souřadnic náležících právě jedné z oblastí (na každé pozici fronty je tedy shodný identifikátor), kdy $k$ = `COUNT_OF_CHECKS_ON_BORDERS`, a zároveň se bude jednat o oblast, která nebude shodná s aktuální, dojde ke změně aktuální oblasti a aplikace zároveň vygeneruje událost, že nastala změna (výstup, vstup nebo obě tyto aktivity zároveň).

### 4.4.6 Testování

Pro ověření správnosti navržených algoritmů bylo v průběhu vývoje realizováno několik druhů testů. Části systému byly podrobeny tzv. jednotkovému testování a u aplikace jako celku bylo ověřováno její chování v reálných i simulovaných situacích.

### Testování vybraných komponent

Ověření správnosti navržených a implementovaných algoritmů proběhlo s využitím jednotkových testů v odděleném projektu. Jedná se o typ funkčního testování, které se zaměřuje na jednotlivé komponenty systému (metody, třídy) bez ohledu na jejich vztah a návaznost k okolí. [16]

S pomocí těchto testů uvedených v následujícím seznamu bylo u vybraných komponent ověřováno, zda aplikace vyhodnotí všechny situace, které mohou kdykoli nastat, správně.

- **Testování validace lokalit**

  Záležitostí, kterou aplikace musí bezpodmínečně obsahovat, je ověření, zda každý z dílčích polygonů lokality obsahuje alespoň minimální počet bodů (definice v úvodu sekce 3.3 na straně 13).

  Test `LocationValidationTest` tedy testuje správnost výsledků metody `IsValid` ve třídě `Location`, která se spouští vždy, pokud je přijata nová lokalita ze serveru. Je nutné zmínit, že tato metoda očekává seřezené body lokality dle jejich pořadí (dle hodnot atributu `order_in_location` třídy `Vertice`). Dalším předpokladem pro vstup lokalit do aplikace je skutečnost, že se nepřekrývají polygony jedné lokality, neprotínají se její strany a zároveň se nepřekrývají lokality navzájem.

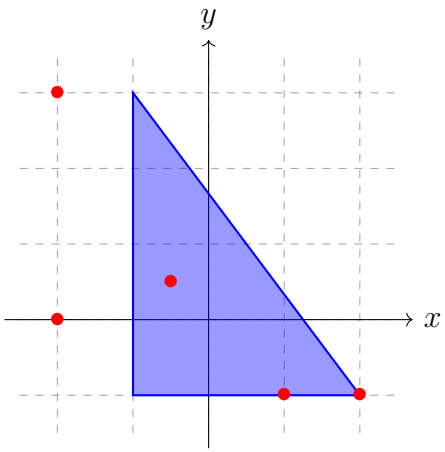
  Pro tento test byla připravena ověřovací data, která správnost algoritmu potvrdila. Metoda `IsValid` je tedy implementována správně, neboť mezi lokalitami je schopna bezpečně detekovat ty, jejichž seznam bodů není správný, a zabránit následně jejich vstupu do aplikace.

- **Testování třídy `PositionInLocationFinder`**

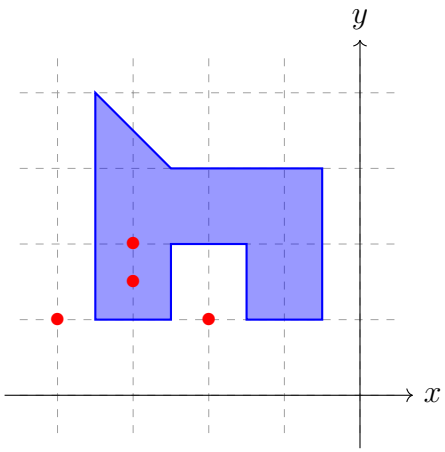
  Nejdůležitějším předmětem testování byla třída, která slouží k ověřování výskytu bodu v lokalitě. Tato třída obsahuje metodu `IsInside`, která očekává dva parametry: konkrétní lokalitu a bod (polohu zařízení). Její úlohou je zjistit, zda se daný bod v dané lokalitě nachází.

  V obrázku 4.7 na straně 43 jsou zobrazena testovací data pro test této komponenty. Jde o lokality (vyznačeny modře) a body (vyznačeny červeně), které ve vzájemné kombinaci pokrývají situace, které mohou v reálné situaci nastat.

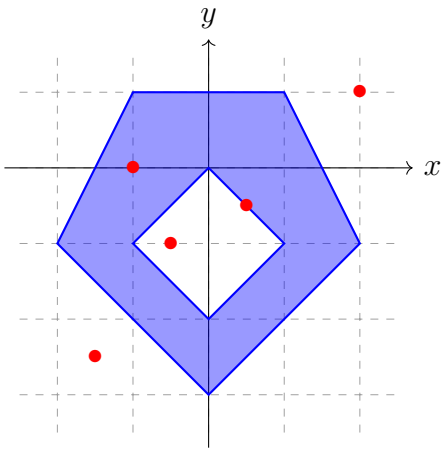
  Zachyceny byly situace, kdy bod leží uvnitř (i přímo na straně) i vně polygonů tvořících konkrétní lokality. Na základě výstupu testů provedených nad těmito daty lze konstatovat, že algoritmus funguje správně.



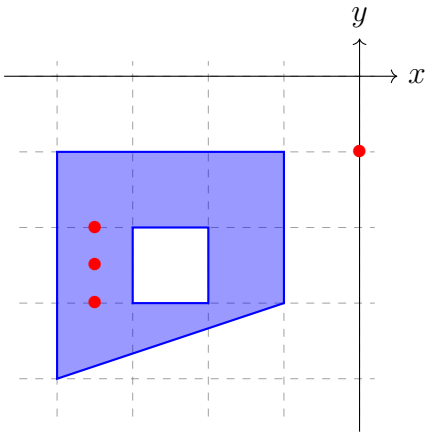

(a) Jednoduchá lokalita




(b) Jednoduchá lokalita




(c) Lokalita tvořená více polygony




(d) Lokalita tvořená více polygony

Obrázek 4.7: Testovací data pro jednotkový test komponenty určené k ověřování výskytu bodu v lokalitě.

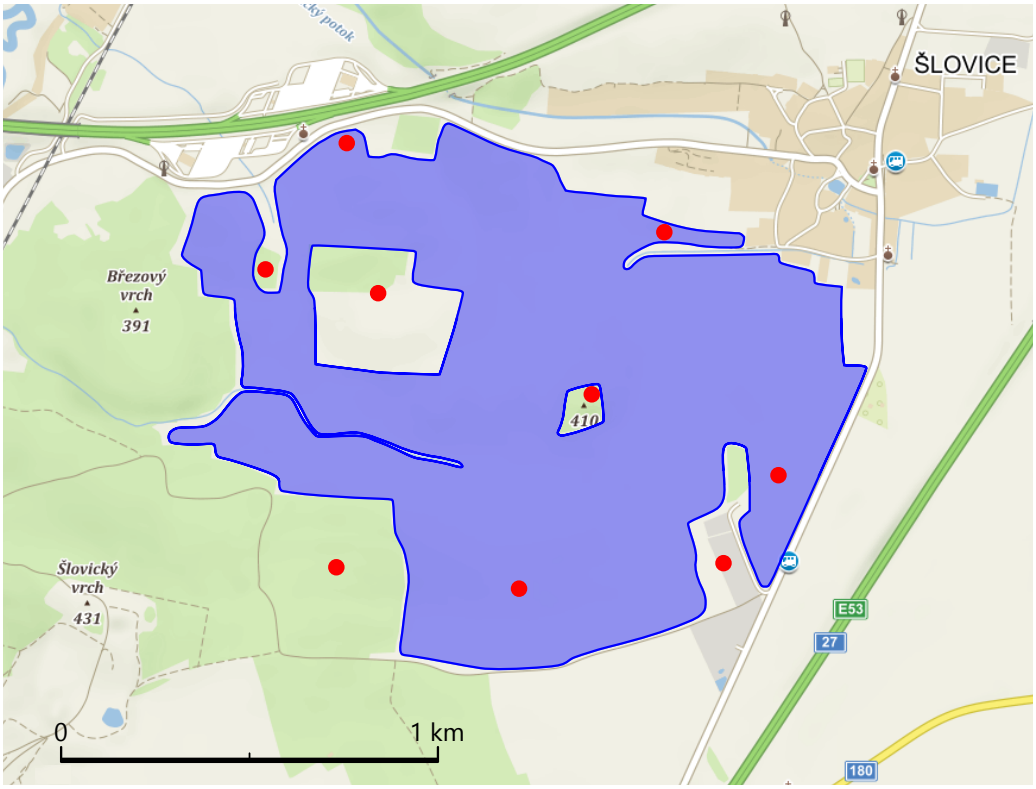
### Testování pomocí simulace polohy zařízení

Pro usnadnění vývoje této mobilní aplikace bylo využito možnosti simulovat polohu mobilního zařízení, na kterém vývoj probíhal. K simulaci existuje řada veřejně dostupných aplikací. Z těch, které jsou k dispozici v nabídce Obchodu Google Play[1] pro operační systém Android, byla vybrána bezplatná aplikace s názvem *Fake GPS*, jejíž autorem je společnost *ByteRev IT Solutions*.

[1] **Obchod Google Play** je služba sloužící k distribuci aplikací pro zařízení s operačním systémem Android.

Pomocí této aplikace je možné mobilní zařízení uvést do stavu, kdy každý dotaz na GPS modul (pro zjištění aktuální polohy zařízení) bude obsloužen právě touto aplikací. Zároveň je v aplikaci *Fake GPS* možné simulovat i náhodný pohyb zařízení. Díky těmto možnostem je tedy možné jednoduše a pohodlně simulovat vstupy a výstupy z lokalit a testovat tak chování vyvíjené aplikace.

Jednou z tímto způsobem testovaných lokalit byla reálná parcela vyskytující se v Plzeňském kraji. V obrázku 4.8 je tato lokalita vyznačena modrou plochou, červené body zde představují místa, na kterých konkrétní test proběhl.




Obrázek 4.8: Lokalita (vyznačena modrou plochou) zvolená pro testování pomocí simulace včetně testovaných bodů (vyznačeny červeně).

Aplikace správně rozpoznala případy, při kterých se v lokalitě v rámci simulace nacházela a kdy nikoli.

### Reálný test v terénu

Aplikaci bylo dále potřeba celkově otestovat přímo v terénu. Připravena byla vhodná testovací data zahrnující jak příhodné, tak i potenciálně problémové prostory. Test byl proveden souběžně na několika mobilních zařízeních s různými verzemi operačního systému Android. Seznam těchto zařízení je obsahem tabulky 4.1.

| **Mobilní zařízení** | **Verze operačního systému** |
|---|---|
| Samsung Galaxy Trend Plus GT S7580 | Android 4.2.2 Jelly Bean (API 17) |
| Lenovo P70 | Android 4.4.4 KitKat (API 19) |
| Xiaomi Redmi 3 Pro | Android 5.1 Lollipop (API 22) |
| Huawei P9 Lite | Android 6.0 Marshmallow (API 23) |

Tabulka 4.1: Zařízení využitá při testování mobilní aplikace v terénu.

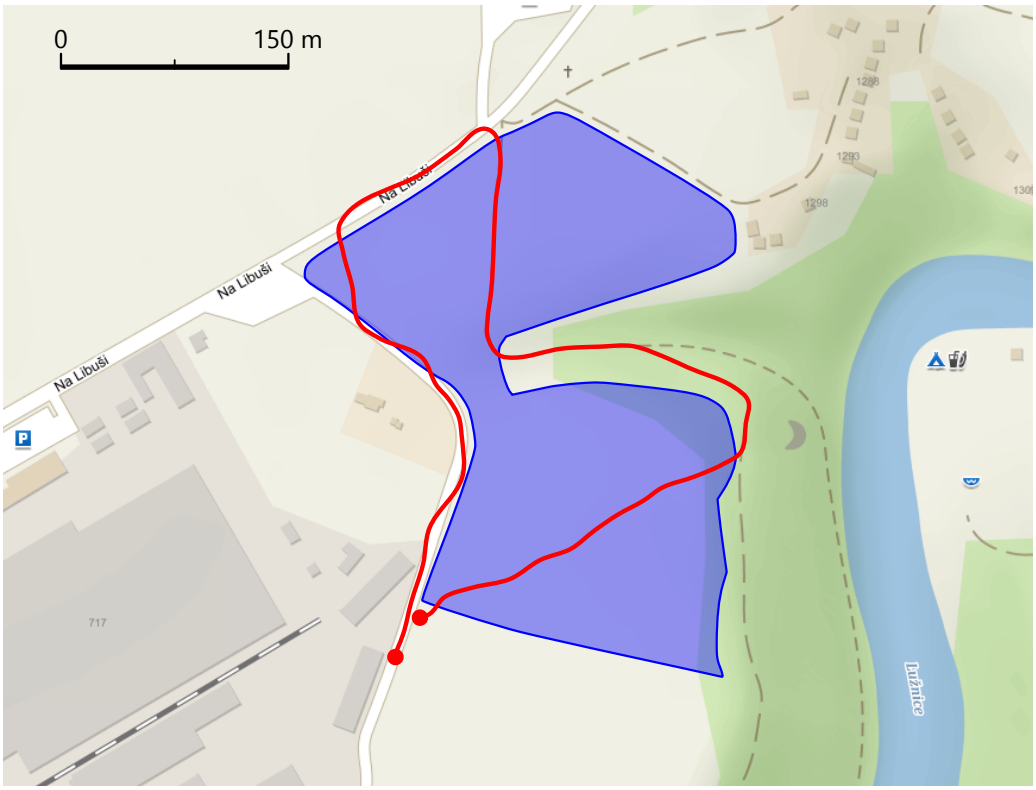
Pro souběžné testování správnosti chování aplikace (se shodným nastavením parametrů na `SettingsPage`) na těchto zařízeních byla zvolena lokalita nacházející se v Jihočeském kraji, která je zobrazena v obrázku 4.9. Při pohybu těchto zařízení po stanovené trase a průběžném porovnávání výstupů jednotlivých aplikací docházelo jen výjimečně k nesrovnalostem. Ty se projevovaly pouze v zařízení Lenovo. O tomto konkrétním přístroji však lze z dlouhodobých zkušeností s každenním používáním říci, že kvalita konkrétního integrovaného GPS modulu není uspokojující. Čas od času zde tedy dochází k situacím, kdy se určená poloha během jediné aktualizace skokově změní v řádech někdy až set metrů, aby se po několika okamžicích opět „vrátila“ do míst, kde se zařízení skutečně nachází. Při běhu aplikace v tomto zařízení tedy příležitostně docházelo ke generování událostí, které ve skutečnosti nenastaly.

Tento problém byl operativně zmírněn změnou v konfiguračním souboru `Configuration`, a to zvýšením konstanty `COUNT_OF_CHECKS_ON_BORDERS` na hodnotu vyžadující pět v řadě jdoucích signálů z totožné oblasti a i u tohoto zařízení tedy bylo posléze možné říci, že vykazuje spolehlivé výsledky.

Na základě provedení série testů byla určena ideální hodnota intervalu mezi kontrolami (v aplikaci nastavitelná na stránce `SettingsPage`) v rozsahu 3 až 5 sekund.

Funkčnost aplikace byla dále otestována i v zalesněné oblasti, kde v daném případě nedošlo k žádnému stavu neodpovídajícímu skutečnosti.

Mimo případ zobrazený v obrázku 4.9 bylo dále ověřeno, že v situaci, kdy zařízení v jeden moment z jedné z lokalit vystoupí a do další bezprostředně vstoupí, aplikace dále – rovněž správně – generuje dvě události namísto jedné (ta je generována při výstupu či vstupu z, resp. do lokality).




Obrázek 4.9: Lokalita zvolená pro testování v terénu včetně testovací trasy.

# 5 Zhodnocení výsledků

Cílovým operačním systémem této aplikace je primárně Android. Vzhledem k možnostem, které se v dnešní době v oblasti vývoje mobilních aplikací vyskytují, byl pro vývoj jednoznačnou volbou framework Xamarin. Ten totiž pokrývá právě dva nejrozšířenější operační systémy. Velkým přínosem tedy je, že bylo dosaženo mnohonásobně většího počtu zařízení, na nichž je možné aplikaci spustit.

Finální mobilní aplikace je z uživatelského pohledu přehledná a jednoduchá, nicméně existuje zde poměrně velké omezení – aplikace neběží na pozadí. Je tak po dobu spuštění hlavní sledovací aktivity potřeba udržet aktivní jak aplikaci, tak i displej zařízení. Tato skutečnost představuje podstatný – ne však neřešitelný – problém při použití aplikace v praxi.

## 5.1 Souhrn testování

Testování systému probíhalo v několika etapách, kdy byly původní jednotkové testy vybraných komponent následovány simulací pohybu zařízení a následně i testem v reálném prostředí.

Provedením jednotkových testů bylo ověřeno, že všechny testované situace zachycující běžné i mezní situace budou v aplikaci vždy vyhodnoceny správně. Využitím těchto komponent tak bude dosaženo vždy správných výsledků.

V rámci testování pomocí simulace polohy zařízení došlo ke kýženému zjištění, že – kromě již ověřených komponent – i všechny části aplikace fungují dle očekávání, neboť aplikace generovala patřičné události vždy, když docházelo k simulovaným změnám oblastí.

Následujícím testem bylo osobní ověření funkčnosti systému jako celku v terénu na několika různých zařízeních. I přesto, že došlo k mírným problémům (popsaných v sekci 4.4.6 – *Reálný test v terénu* – na straně 45) na straně jednoho ze zařízení, na kterém byla aplikace spuštěna, lze tento test prohlásit za úspěšný. Došlo totiž k mírné úpravě hodnot, dle kterých se určování polohy v aplikaci řídí, což vedlo ke zlepšení výstupů i v do té doby problémovém zařízení.

## 5.2 Potenciální rozšíření systému

Tento systém (jak serverová, tak i klientská část) byl již od počátku vývoje koncipován tak, aby jej bylo možné snadno rozšiřovat a upravovat. Vzhledem k tomuto faktu se nabízí několik možností, které by mohly systém rozšířit a pozvednout jej na vyšší úroveň, ale také vyřešit jeho určité nedostatky. Tyto možnosti jsou krátce představeny v následujícím seznamu.

- **Zabezpečení komunikace, identifikace uživatele**

  Komunikace v systému funguje bez jakéhokoli zabezpečení. Pro zajištění bezpečnosti v systému je možné zavést autentizaci uživatele přistupujícího k webové službě.

- **Běh aplikace na pozadí**

  Běh aplikace na pozadí (za účelem snížení spotřeby baterie) mobilního zařízení je jedním z nejpodstatnějších rozšíření, které aplikace do budoucna vyžaduje. V tomto případě lze také implementovat upozorňování uživatele pomocí notifikací.

- **Přenos operací pomocí webové služby**

  Nabízí se také rozšíření webové služby o stahování aktuálního seznamu operací pro pozdější volbu na stránce `EventPage`.

- **Zobrazení polohy v mapě**

  Další možností, jak práci lze v budoucnu posunout nad rámec zadání, je zobrazení aktuální polohy zařízení v reálném čase v mapě. Rovněž je možné umístit mapu s detailem stažené lokality na jednotlivé stránky `LocationDetailPage`.

- **Automatizace nahrávání dat na server**

  Dalším rozšířením serverové aplikace je potenciální automatizace nahrávání dat o lokalitách ze systému LPIS do databáze na serveru. Správu všech dat (nejen lokalit, ale i událostí) v databázi je tedy možné realizovat například prostřednictvím webového rozhraní. Při vstupu dat do systému je v tomto místě dále vhodné zavést ověření, zda lokalita splňuje vškeré předpoklady dle její definice.

- **Vylepšení řešení problému častých vstupů**

  Jedním z dalších způsobů, pomocí kterých by se dalo vylepšit řešení problému s častým opakováním se vstupů (a výstupů) do (a z) lokality při pohybu v těsném okolí její hranice, je vytvoření úlohy, jejíž úkolem by bylo zpětně procházet vygenerované (neodeslané) události a vyhledávat množiny událostí (např. podle velmi krátkého rozdílu časů daných událostí), které by bylo možné sloučit v událost jednu.

- **Ovládání aplikace hlasem**

  Pro jednodušší využívání aplikace řidičem vozidla by dále mohlo být například také implementováno hlasové ovládání.

# 6 Závěr

Na základě funkčních požadavků stanovených zadavatelem byl navržen a vypracován systém rozdělený na serverovou a klientskou část.

Serverová část (navržena jednoduše jen pro testovací účely) poskytuje webovou službu, s jejíž pomocí plní roli poskytovatele dat o lokalitách. Zároveň je rovněž konzumentem událostí pocházejících z klientských aplikací. Klientskou část pak představuje mobilní aplikace jak pro operační systém Android, tak i iOS, která kontinuálně snímá polohu zařízení a v reálném čase generuje události. Těmito událostmi jsou vstupy a výstupy do, resp. z určitých lokalit získaných prostřednictvím webové služby serveru. Potvrzené události jsou poté z klientských aplikací kdykoli uživateli odesílány na server. Obě zmíněné části dohromady tvoří provozuschopný celek splňující veškeré funkční požadavky, které byly na systém kladeny.

Systém je zároveň navržen tak, že je možné jej snadno měnit či rozšiřovat. A jak již bylo zmíněno v sekci 5.2 na straně 48, existuje řada záležitostí, kterými je možné tento systém posunout nad rámec zadání. Mezi nejžádanější patří zejména zabezpečení webové služby, automatizace nahrávání dat na server nebo běh mobilní aplikace na pozadí.

Na základě výsledků testování aplikace jako celku i jejích dílčích součástí lze ovšem o navrženém a implementovaném řešení prohlásit, že je plně funkční, splňuje všechny body zadání v plném rozsahu a je možné jej následně využít i v praxi.

# Literatura


[1] *Geoportál ČÚZK* [online]. [cit. 2019/04/02]. Dostupné z: https://geoportal.cuzk.cz/(S(ogtp2hecs5nb4o3raabbigv2))/default.aspx?mode=TextMeta&text=about_FAQ&side=about&menu=11.

[2] *HyperSQL Database – User Guide* [online]. [cit. 2019/04/07]. Dostupné z: http://hsqldb.org/doc/2.0/guide/guide.pdf.

[3] *Building RESTful Web Services with JAX-RS* [online]. [cit. 2019/04/06]. Dostupné z: https://docs.oracle.com/javaee/7/tutorial/jaxrs.htm.

[4] *Java Persistence API* [online]. [cit. 2019/04/06]. Dostupné z: https://docs.oracle.com/javaee/6/tutorial/doc/bnbpy.html.

[5] *Veřejný registr půdy LPIS* [online]. [cit. 2019/04/02]. Dostupné z: http://eagri.cz/public/web/mze/farmar/LPIS/.

[6] *Multimediální studijní materiály k předmětu KGM/MK2* [online]. [cit. 2018-04-17]. Dostupné z: http://old.gis.zcu.cz/studium/mk2/multimedialni_texty/index_soubory/hlavni_soubory/zaklady.html#rovina.

[7] *How to check if a given point lies inside or outside a polygon?* [online]. [cit. 2019/03/30]. Dostupné z: https://www.geeksforgeeks.org/how-to-check-if-a-given-point-lies-inside-a-polygon/.

[8] *Introduction to Xamarin* [online]. [cit. 2019/04/13]. Dostupné z: https://docs.microsoft.com/cs-cz/xamarin/cross-platform/get-started/introduction-to-mobile-development.

[9] *Úvod do Xamarin Forms* [online]. [cit. 2019/04/13]. Dostupné z: https://www.skeleton.cz/uvod-do-xamarin-forms.

[10] *Enterprise Application Patterns* [online]. [cit. 2019/04/13]. Dostupné z: https://docs.microsoft.com/cs-cz/xamarin/xamarin-forms/enterprise-application-patterns.

[11] Bartsch, H.-J. *Matematické vzorce*. V nakl. Academia 1, 2006. ISBN 80-200-1448-9.

[12] Bishop, J. *C# 3.0 design patterns*. O'Reilly Media, Inc., 2008. ISBN 978-0-596-52773-0.

[13] Šebesta, J. *Globální navigační systémy.* Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav radioelektroniky, 2012. ISBN 978-80-214-4500-0.

[14] Fatrdla, P. *Bakalářská práce.* Vysoké učení technické v Brně, Fakulta informačních technologií, 2010. Dostupné z: `https://dspace.vutbr.cz/xmlui/bitstream/handle/11012/54245/10727.pdf`.

[15] Grulová, P. *Bakalářská práce.* Univerzita Palackého v Olomouci, Přírodovědecká fakulta, 2012. Dostupné z: `https://geography.upol.cz/soubory/studium/bp/2012-rg/2012_Grulova.pdf`.

[16] Havelka, A. *JUnit 5: jednotkové testování na platformě Java.* Grada Publishing, a. s., 2018. ISBN 978-80-271-0733-9.

[17] Kubátová, R. *Bakalářská práce.* Západočeská univerzita, Fakulta aplikovaných věd, 2007. Dostupné z: `https://kgm.zcu.cz/studium/ZaverecnePrace/2007/Kubatova__System_JTSK_a_WGS-84_a_vzajemna_transformace__BP.pdf`.

[18] Novotná, M. *Modul: Geografie – teoretický a metodologický základ.* Západočeská univerzita, 2014. ISBN 978-80-261-0463-6.

[19] Rektorys, K. *Přehled užité matematiky.* Státní nakladatelství technické literatury, 1981.

# A Seznam zkratek

**API** – Application Programming Interface (rozhraní pro usnadnění programování aplikací)

**APK** – Android Application Package (balíčkový soubor užívaný k distribuci a instalaci mobilních aplikací do systému Android)

**BCL** – .NET Base Class Library (rozsáhlá kolekce tříd zahrnující základní datové struktury pro práci s textem nebo např. s datem)

**CRUD** – Create, Read, Update a Delete (základní operace nad záznamem v databázi)

**ER diagram** – Entitně Relační diagram (diagram pro popis datových modelů pomocí entit a relací, popř. atributů)

**GLONASS** – Globální Navigační Satelitní Systém (dříve sovětský, dnes ruský GNSS)

**GNSS** – Global Navigation Satellite System (systémy, které pomocí družic umožňují lokalizaci polohy přijímačů na zemském povrchu)

**GPS** – Global Positioning System (globální satelitní navigační systém Ministerstva obrany USA)

**GUI** – Graphical User Interface (grafické uživatelské rozhraní pro snadné ovládání zařízení)

**HSQLDB** – HyperSQL Databáze (relační databázový systém)

**HTTP** – Hypertext Transfer Protocol (protokol pro datovou komunikaci na webu)

**IERS** – International Earth Rotation and Reference Systems Service (služba poskytující data a standardy související s rotací Země a referenčními plochami)

**IRM** – IERS Reference Meridian (referenční – nultý – poledník pro souřadnicový systém WGS-84)

**JAX-RS** – Java API for RESTful Web Services (API usnadňující tvorbu aplikací využívajících architekturu REST)

**JPA** – Java Persistence API (rozhraní pro správu dat v aplikacích pomocí ORM)

**JSON** – JavaScript Object Notation („odlehčený“ formát pro uchovávání a přenos dat)

**LPIS** – Land Parcel Identification System (veřejný registr půdy Ministerstva zemědělství České republiky)

**MVVM** – Model-View-ViewModel (popis architektury aplikace, návrhový vzor)

**ORM** – Objektově Relační Mapování (způsob přemostění mezi objektově orientovaným a relačním paradigmatem)

**REST** – Representational State Transfer (způsob komunikace strojů v síti pomocí HTTP protokolu)

**SQL** – Structured Query Language (standardní dotazovací jazyk pro ukládání, manipulaci a načítání dat v databázích)

**S-JTSK** – Systém Jednotné Trigonometrické Sítě Katastrální (referenční souřadnicový systém na území České republiky)

**URI** – Uniform Resource Identifier (řetězec, který slouží k jednoznačné identifikaci zdroje dat v síti)

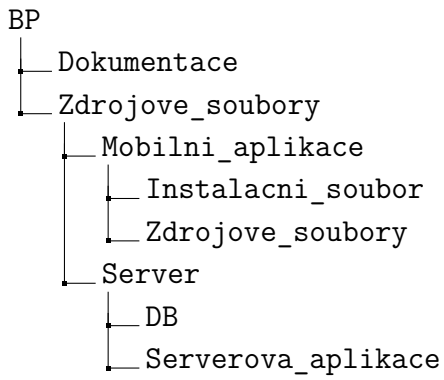
**WGS-84** – World Geodetic System 1984 (referenční souřadnicový systém satelitů GPS)

**XAML** – Extensible Application Markup Language (jazyk založený na XML, napomáhající k oddělení logiky od designu aplikace)

**XML** – Extensible Markup Language (rozšiřitelný značkovací jazyk pro uchovávání a přenos dat)

# B Instalační příručka

Veškeré soubory, které jsou součástí celého systému vyvinutého v rámci této bakalářské práce, jsou na přiloženém disku uložené v adresáři `BP`, jehož základní strukturu zobrazuje následující adresářový strom.




## B.1 Server

V adresáři `BP/Zdrojove_soubory/Server` jsou veškeré soubory týkající se databázového systému a serverové aplikace. Ty jsou připraveny pro překlad a spuštění v operačním systému *Microsoft Windows*.

### B.1.1 Databáze

V adresáři `BP/Zdrojove_soubory/Server/DB` se nachází adresář `hsqldb`. Databázi spusťte z příkazové řádky provedením následujícího příkazu právě v tomto adresáři.

```
java -classpath lib/hsqldb.jar org.hsqldb.server.Server
-database.0 file:hsqldb/db -dbname.0 db1
```

### B.1.2 Serverová aplikace

Sestavování i samotné spouštění serverové aplikace je realizováno pomocí nástroje *Apache Maven*. Nejprve je tedy nutné jej nainstalovat. Návod je k dispozici například na webové adrese *https://maven.apache.org/install.html*.

#### Předklad

Pro překlad a sestavení serverové aplikace vykonejte následující příkaz v adresáři `BP/Zdrojove_soubory/Server/Serverova_aplikace/Server`.

```
mvn clean install
```

#### Spuštění

Pro samotné spuštění sestavené serverové aplikace je nutné, aby běžela databáze. Spusťte ji tedy nejprve dle návodu v sekci B.1.1.

Vzhledem k tomu, že projekt serveru je *tzv. Maven projekt*, jeho spuštění zabezpečuje nástroj *Apache Maven* při vykonání následujícího příkazu v adresáři `BP/Zdrojove_soubory/Server/Serverova_aplikace/Server`.

```
mvn exec:java -Dexec.mainClass="server.Main"
```

## B.2 Mobilní aplikace

Obsahem adresáře `BP/Zdrojove_soubory/Mobilni_aplikace` jsou veškeré soubory týkající se mobilní aplikace.

### B.2.1 Předklad

Překlad mobilní aplikace pro Android a její následnou archivaci do APK souboru je možné zajistit s pomocí vývojového prostředí *Microsoft Visual Studio* pro operační systém Microsoft Windows. To je k získání na webové adrese *https://visualstudio.microsoft.com/cs/downloads/*.

Překlad pro operační systém iOS je možný naopak jen s využitím vývojového prostředí *Visual Studio for Mac* v systému MacOS. To je možné stáhnout na webové adrese *https://visualstudio.microsoft.com/cs/vs/mac/*.

### B.2.2 Instalace

#### Android

V adresáři `BP/Zdrojove_soubory/Mobilni_aplikace/Instalacni_soubor` se nachází binární soubor s názvem `TractorTracker.apk`. Ten představuje instalační balíček ve formátu APK, který je připraven pro instalaci mobilní

aplikace do operačního systému Android s minimální verzí API 19 (Android 4.2.2, tzv. *Jelly Bean*). Při instalaci je vhodné postupovat následujícím způsobem.

1. Soubor `TractorTracker.apk` uložte do mobilního zařízení.
2. V nastavení zabezpečení daného mobilního zařízení povolte možnost instalace instalace aplikací z neznámých zdrojů (po nainstalování je kvůli bezpečnosti vhodné tuto možnost opět zakázat).
3. Ve správci souborů uložený soubor `TractorTracker.apk` vyhledejte a klepnutím jej nainstalujte.

Pozor! Při testování instalace aplikace na různá zařízení s odlišnými verzemi systému Android bylo zjištěno, že ne vždy si instalační program od uživatele vyžádá povolení pro přístup k funkcím telefonu, konkrétně pro zjišťování polohy a přístup k síti. V takovém případě je nezbytně nutné tyto přístupy povolit na základě manuálního zásahu v systémovém nastavení této konkrétní aplikace, neboť pro aplikaci jsou tyto záležitosti klíčové.

### iOS

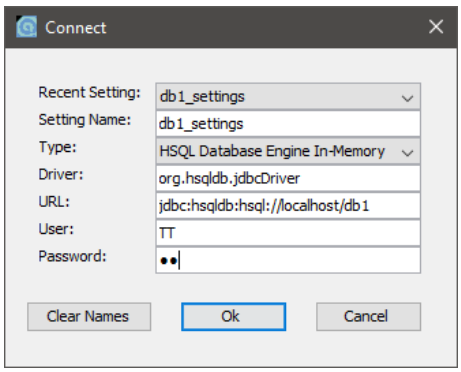
Překlad aplikace pro systém iOS realizován nebyl, nicméně projekt je pro tuto záležitost připraven.

# C Uživatelská příručka

## C.1 Server

Pro nahrávání dat do serverové databáze nejprve spusťte serverovou aplikaci dle pokynů v sekci B.1.2 na straně 55.

Dále spusťte dávkový soubor s názvem `runManagerSwing.bat`, který se nachází v adresáři `BP/Zdrojove_soubory/Server/DB/hsqldb/bin`. Zobrazí se grafické uživatelské rozhraní – konkrétně formulář pro připojení k databázi. Ten vyplňte tak, jak je zobrazeno na obrázku C.1, kde do pole *Password* zadejte řetězec „TT“.




Obrázek C.1: Požadovaný obsah formuláře pro připojení k databázi.

Po úspěšném připojení je možné vykonávat SQL příkazy či dávkově celé SQL skripty. Příkladový skript `create.sql` (obsahující vysvětlující komentáře), pomocí něhož je možné vytvořit tabulky a naplnit je daty, je uložen v adresáři `BP/Zdrojove_soubory/Server/DB` a načíst jej můžete vybráním možnosti *Open Script* v nabídce *File*. Následně je možné jej vykonat stiskem tlačítka *Execute SQL*.

Nutné je poznamenat, že souřadnice jednotlivých lokalit musí být uvedeny v souřadnicovém systému WGS-84, přičemž souřadnice `x` představuje zeměpisnou délku a `y` zeměpisnou šířku.

## C.2 Mobilní aplikace

### C.2.1 Použití

Pro spuštění mobilní aplikace je potřeba ji nainstalovat do zařízení dle pokynů v sekci B.2.2 na straně 56. Dále postupujte podle následujících bodů.

1. Spusťte aplikaci. Zobrazí se úvodní obrazovka (obrázek C.2a), která informuje o připojení zařízení k internetu a možnosti zjišťování polohy. V případě, že je v zařízení zjišťování polohy povoleno, tlačítko *NAČÍST AKTIVITY* je aktivní. Pro aktualizaci stavu informačních ikon je potřeba potáhnout prstem po displeji směrem dolů.

2. Klepněte v pravém horním rohu na ikonu s ozubeným kolem a otevřete tak stránku *Nastavení* (obrázek C.2b).

3. Zadejte identifikátor uživatele (celé kladné číslo) a adresu a port pro připojení k serveru.

4. Klepněte na tlačítko *NAČÍST LOKALITY*. Aplikace zobrazí lokality (obrázek C.3a), které jsou uloženy v zařízení. Aktualizaci seznamu lokalit (jejich stažení ze serveru) můžete provést takto:

   - je-li seznam prázdný – klepnete na tlačítko *AKTUALIZOVAT*,
   - obsahuje-li alespoň jeden záznam – potáhnete prstem seznamem směrem dolů.

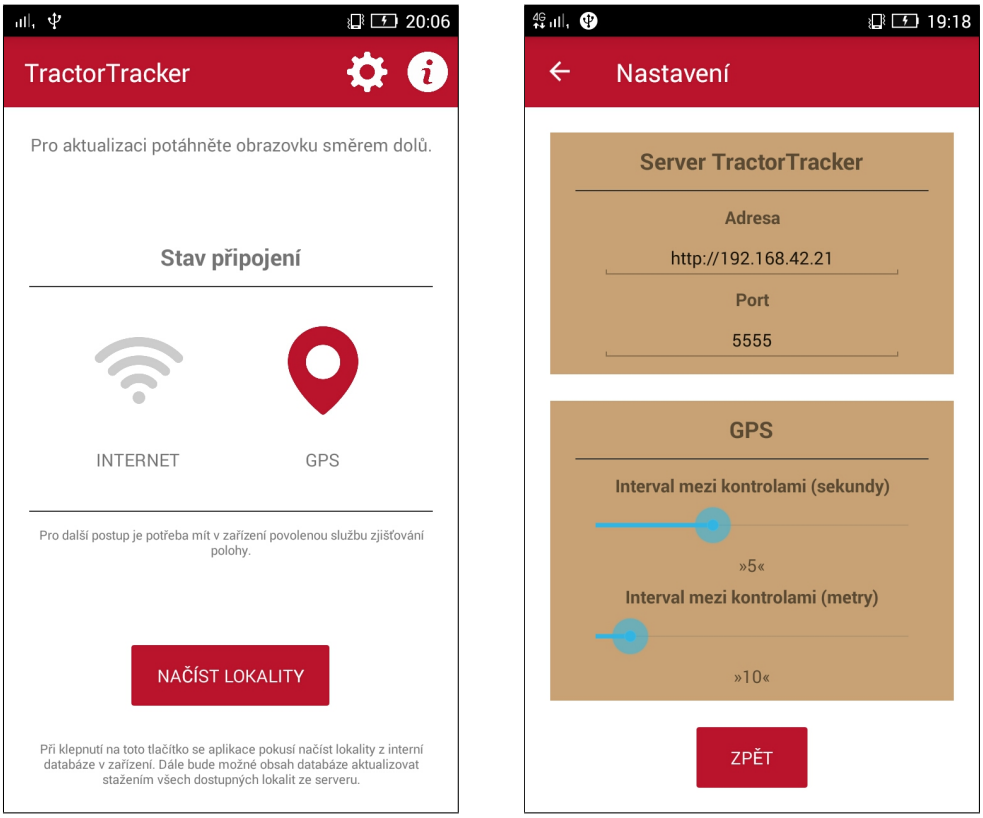
   Pro zobrazení detailu určité lokality klepněte na její záznam v seznamu.

5. Pokud seznam obsahuje alespoň jeden záznam, klepněte na tlačítko *POKRAČOVAT*, postupte tak na další stránku a zahajte tak sledování zařízení (obrázek C.3b). V horní části této stránky se nachází informace o aktuální poloze zařízení, ve spodní pak neodeslané vygenerované události.

6. Klepnutím na jednotlivé záznamy zobrazte detail událostí. Zde můžete upravit údaje. Jedná-li se o výstup z lokality (obrázek C.4b), vyberte jednu ze seznamu operací (detailní stránka vstupu do lokality je zobrazena v obrázku C.4a). Změny uložte klepnutím na tlačítko *ULOŽIT*. (Nevyžádaný záznam můžete dále také smazat.)

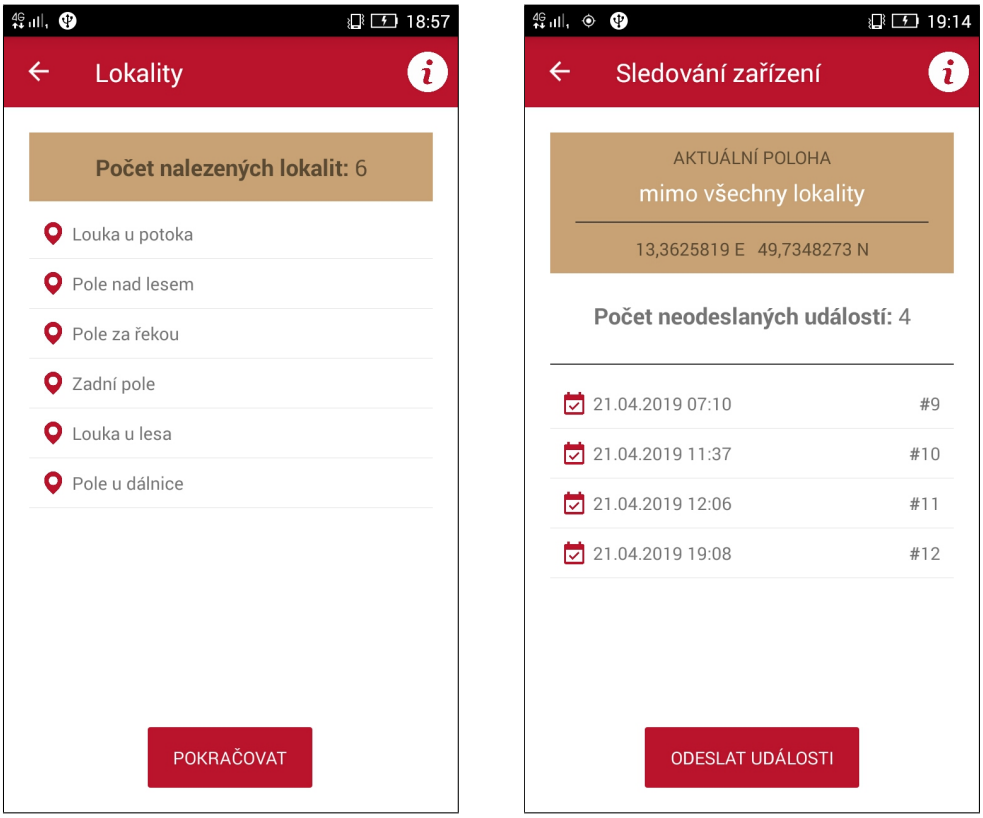
7. Ve spodní části stránky *Sledování zařízení* pak můžete události hromadně odeslat na server klepnutím na příslušné tlačítko.

### C.2.2 Obrazová příloha



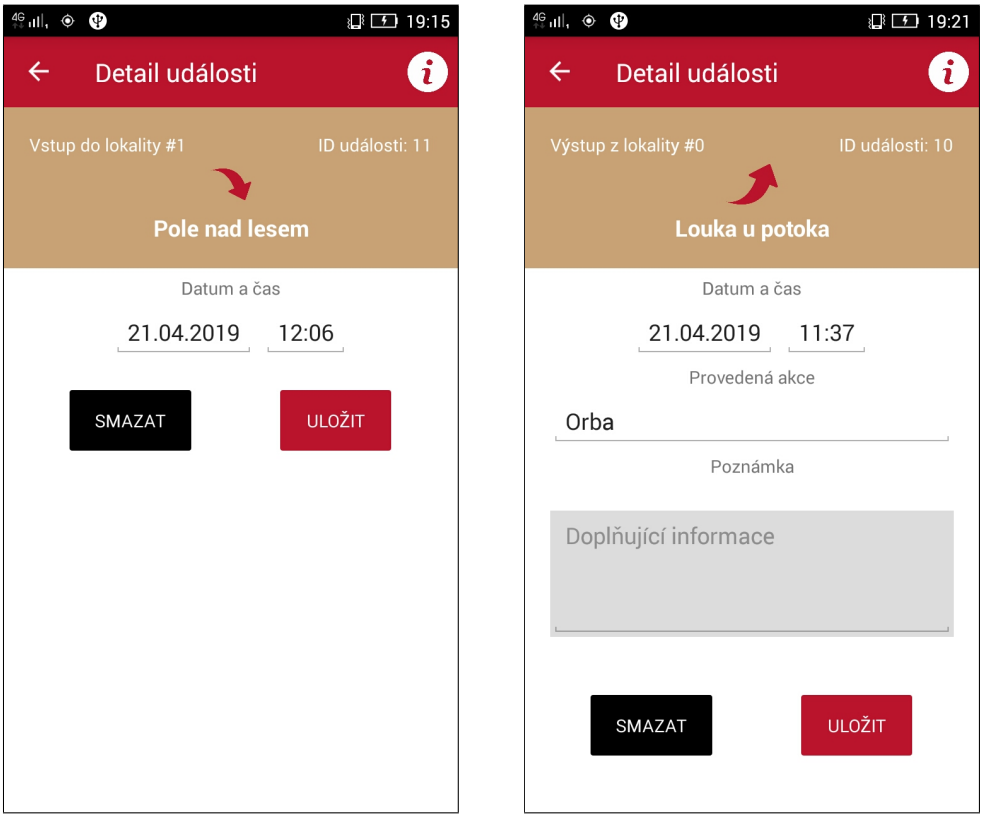

(a) `WelcomePage`. (b) `SettingsPage`.

Obrázek C.2: Ukázka stránek aplikace – konkrétně zobrazení úvodní stránky a stránky s nastavením.




(a) `LocationsPage`. (b) `TrackPage`.

Obrázek C.3: Ukázka stránek aplikace – konkrétně zobrazení stažených lokalit (vlevo) a neodeslaných událostí a aktuální polohy zařízení (vpravo).




(a) `EventPage` – vstup.

(b) `EventPage` – výstup.

Obrázek C.4: Ukázka stránek aplikace – konkrétně zobrazení detailů jednotlivých neodeslaných událostí.