TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Webová aplikace pro monitorování a tracking autonomních zařízení

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Autor práce: **Bc. Simona Němečková**
Vedoucí práce: Ing. Igor Kopetschke

Liberec 2017

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Web Application for Monitoring and Tracking of Autonomous Devices

Master thesis

Study programme: N2612 – Electrical Engineering and Informatics
Study branch: 1802T007 – Information Technology

Author: **Bc. Simona Němečková**
Supervisor: Ing. Igor Kopetschke

Liberec 2017

Technická univerzita v Liberci
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2016/2017**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Simona Němečková**
Osobní číslo:	**M15000182**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Informační technologie**
Název tématu:	**Webová aplikace pro monitorování a tracking autonomních zařízení**
Zadávající katedra:	**Ústav nových technologií a aplikované informatiky**

Zásady pro vypracování:

1. Seznamte se s experimentálním zařízením typu auto a dron a jeho HW periferiemi.
2. Popište komunikaci s uvedenými periferiemi včetně způsobu získávání dat.
3. Navrhněte webovou aplikaci pro online monitoring zařízení v 3D prostředí včetně online přenosu dat z periferií.
4. Webovou aplikaci implementujte v jazyce Python.

Rozsah grafických prací: **dle potřeby**

Rozsah pracovní zprávy: **40 - 50 stran**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] RESTful Web Services. Sebastopol: O'Reilly Media, Inc, 2008. ISBN 9780596554606.
[2] SHREINER, Dave. OpenGL: průvodce programátora. Vyd. 1. Přeložil Jiří FADRNÝ. Brno: Computer Press, 2006. DTP & grafika. ISBN 80-251-1275-6.
[3] PILGRIM, Mark. Ponořme se do Python(u) 3: Dive into Python 3. Praha: Cz.Nic, c2010. CZ.NIC. ISBN 978-8-904248-2-1.

Vedoucí diplomové práce: **Ing. Igor Kopetschke**
Ústav nových technologií a aplikované informatiky

Datum zadání diplomové práce: **20. října 2016**

Termín odevzdání diplomové práce: **15. května 2017**

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

L.S.

prof. Dr. Ing. Jiří Maryška, CSc.
vedoucí ústavu

V Liberci dne 20. října 2016

Prohlášení

Byla jsem seznámena s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědoma povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracovala samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 15.5.2017

Podpis: 

Poděkování

Ráda bych poděkovala panu Ing. Igorovi Kopetschke za pomoc při psaní diplomové práce a za jeho cenné rady.

Abstrakt

Tato diplomová práce popisuje vývoj interaktivní webové aplikace pro monitorování autonomních zařízení.

Zpráva nejprve charakterizuje jednotlivá autonomní zařízení, mezi která se řadí auto a dron. Obsahuje stručný popis hardwarových periferií těchto zařízení. Dále se práce zabývá způsoby komunikace s těmito zařízeními a definuje způsob získávání dat z periferií pomocí HTTP požadavků.

Další část této diplomové práce se zabývá návrhem schématu relační databáze typu MariaDB, která slouží k uchovávání dat o jízdách (případně letech) jednotlivých aut a dronů. Specifikujeme formát HTTP požadavků, které bude autonomní zařízení využívat k odesílání dat o své jízdě (svém letu). Jako formát dat těchto požadavků byl zvolen JSON z důvodu jeho snadné rozšiřitelnosti a nízké paměťové náročnosti.

Tato databáze spolu s RESTful webovou službou pro získání mapových podkladů tvoří zdroj dat pro serverovou část webové aplikace. Na straně klienta pak tato aplikace umožňuje uživateli sledovat aktuální či poslední jízdu/let zařízení ve 3D prostředí a prohlédnout se několik posledních snímků z jejich kamer.

3D prostředí pro monitorování zařízení se skládá ze 3D mapy blízkého okolí posledního zaznamenaného bodu jízdy/letu tohoto zařízení. Tato mapa je tvořena výškovým profilem zakřiveným podle reliéfu země a modely budov v této oblasti. Pro teselaci (triangulaci) polygonů bylo nutno využít externí knihovny, protože vybraný framework pro práci s 3D grafikou touto funkcí nedisponuje.

Serverová část webové aplikace je implementována v programovacím jazyce Python s využitím frameworku Flask. Pro klientskou část aplikace byl použit jazyk JavaScript rozšířený o knihovnu jQuery. Práci s 3D grafikou zajišťuje BabylonJS – framework založený na JavaScriptovém API pro interaktivní 3D grafiku s názvem WebGL.

Klíčová slova

Webová aplikace, Python, Flask, JavaScript, BabylonJS, autonomní zařízení, 3D mapa

Abstract

This master thesis describes the development of an interactive web application for monitoring autonomous devices.

Firstly, the report characterizes each of the autonomous devices – a car and a drone. It contains a brief description of their hardware peripherals. Further on, the thesis addresses the means of communication with these devices and defines form of getting data from the peripherals using HTTP requests.

Next part of this master thesis describes the design of MariaDB relation database scheme which is used to store data about the drives (or flights) of each car and drone. We specify the format of the HTTP requests that the autonomous device will use to send data about its drive (flight). JSON was chosen as a format of the requests data because of its easy scalability and its low memory requirements.

This database along with RESTful web service for getting map backgrounds create data source for server site of the web application. On client side, this application allows user to watch current or last drive/flight of each device on a 3D map and also to look through a few last photos taken by the cameras of the devices.

The 3D setting for device monitoring consists of a 3D map with the surroundings of the last recorded drive/flight point. This map is made of height profile curved according to the ground relief and models of the buildings in this area. External library had to be used for polygon tessellation (triangulation) due to absence of such a function in the selected framework for 3D graphics.

The backend of the web application is implemented in Python programming language and Flask framework. For the frontend of the application, JavaScript language with jQuery library was used. 3D graphics is provided by BabylonJS – framework based on JavaScript API for interactive 3D graphics called WebGL.

Keywords

Web application, Python, Flask, JavaScript, BabylonJS, autonomous device, 3D map

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek a symbolů

- SHA1
  - Secure Hash Algorithm
  - Rozšířená hašovací funkce, která vyváří ze vstupních dat výstup (otisk) fixní délky
- DMR
  - Digitální model reliéfu
  - Digitální reprezentace reliéfu terénu
- DMP
  - Digitální model povrchu
  - Model terénu doplněný o všechny umělé a přírodní objekty
- WGS-84
  - World Geodetic System 1984
  - Světový souřadnicový systém
- S-JTSK
  - Systém jednotné trigonometrické sítě katastrální
  - Vychází z Křovákova zobrazení
  - Souřadnicový systém pro Českou republiku a Slovensko

1. Úvod

Cílem diplomové práce bylo na základě nabytých znalostí o autonomních zařízeních dostupných pro tento projekt navrhnout interaktivní webovou aplikaci pro jejich monitorování.

Mezi typy autonomních zařízení patří auto a dron. Oba mají možnost připojení k internetu a mohou také disponovat kamerou, jejíž snímky by se měly zobrazovat ve webové aplikaci. K tomu je třeba navrhnout a implementovat systém komunikace serveru s autonomními zařízeními. Kromě fotografií se poté budou na server odesílat i data o aktuální poloze a nadmořské výšce zařízení. Přenos by měl být realizován tak, aby bylo možné později rozšířit přenášená data o další parametry týkající se jízdy/letu zařízení.

Pro zabezpečení komunikace jsem navrhla systém ověření, který využívá neveřejný klíč, který bude znát každé autonomní zařízení a samozřejmě samotná webová aplikace. Kromě použití šifrovacího protokolu HTTPS tak bude komunikace zabezpečena i kontrolou SHA1 otisku (hash) odesílané zprávy spojené s tímto neveřejným klíčem.

Získaná data je nutné na serveru uložit. I když by obrázky bylo možné umisťovat na disk, zvolila jsem pro ně ukládání do databáze stejně jako pro ostatní data. Na serveru je k dispozici databáze typu MariaDB.

Data budou poté prezentována ve webové aplikaci. Jedna část aplikace bude zobrazovat náhled snímků z kamer jednotlivých zařízení, druhá část se bude věnovat vykreslení trasy jízdy/letu vybraného zařízení do 3D interaktivní mapy. Mapa by se měla skládat z modelu země, který je zakřiven podle reliéfu krajiny a pokryt texturou tvořenou ortofotem. Další komponentou by měly být modely budov vytvořené podle jejich půdorysu a výšky. V takto vymodelované mapě se bude nacházet křivka znázorňující trasu dané jízdy (daného letu) vybraného zařízení.

Je nutné vyřešit otázku způsobu vykreslování 3D grafiky v prohlížeči. Výpočet musí probíhat na straně klienta. Pro tyto účely slouží knihovna WebGL, která poskytuje API pro práci s grafikou v prohlížeči v jazyce JavaScript. Dále existují různé nadstavby nad touto knihovnou (frameworky), které by měly přinést vyšší míru abstrakce a tím práci s WebGL zjednodušit.

Datové podklady pro mapu poskytuje webová služba typu REST, která běží pod adresou https://drone.fm.tul.cz:18443/DroneComponent/service/. Jedním požadavkem lze z tohoto API získat data o zakřivení reliéfu země (DMR) a zároveň data týkající se jednotlivých entit jako například budovy, lesy, cesty a podobně (DMP). Zároveň lze tuto službu využít k převodu zeměpisných poloh z WGS-84 do zobrazení S-JTSK. [1]

2. Experimentální autonomní zařízení

2.1 Dron

Prvním typem experimentálních zařízení, na která je aplikace cílena, je dron. Konkrétně se jedná o dva modely: DJI Phantom 2 a DJI Tarot 690. Oba drony mohou disponovat kamerou Go Pro HERO 4, která má volitelné rozlišení, maximální však 4K. Přenos obrazu na ovládací jednotku operátora se přenáší pomocí jednotky FT956 na pásmu 5.8 GHz na LCD panel s přijímačem. Ovládání kamery je realizováno pomocí minipočítače jako například Arduino NANO nebo Raspberry Pi 2.

K řízení dronu se používá systém DJI Naza-M V2. Tento autopilotní systém obsahuje všechny potřebné komponenty (kompas, gyroskopy, akcelerometry, tlakové čidlo).

Svou polohu může zařízení zjistit díky navigačnímu USB modulu GT-730F, případně modulu NEO-6M nebo NEO-7M. Pro zpracování dat z těchto modulů slouží knihovna TinyGPS.

Mezi další periferie, kterými drony disponují, patří akcelerometr a gyroskop CJMCU-MPU, tlakové čidlo BMP180 a PWM čip PCA9685 pro řízení servomotorů a regulátorů. Sběr dat z těchto čidel probíhá po sběrnici typu I2C.

Zařízení je připojeno k internetové síti díky LTE SIM kartě. Jeho IP adresa se mění, proto nelze jeho totožnost určit podle tohoto parametru.

2.2 Auto

Druhým typem autonomního zařízení, které bude možné v aplikaci sledovat, je auto. Jedná se o zařízení na dálkové ovládání. Periferie, které mohou být na auto připojeny, se shodují s periferiemi dronů. Zařízení tedy bude také moci pořizovat snímky z kamer a zjišťovat svou aktuální polohu pomocí obdobných čipů, jaké mají drony. Připojení k internetové síti probíhá stejně jako u dronů.

2.3 Komunikace s autonomními zařízeními

Pro uskutečnění komunikace mezi autonomním zařízením a serverem, na kterém bude běžet webová aplikace, jsem navrhla soubor HTTP (resp. HTTPS) požadavků, které budou moci zařízení odesílat serveru. Pro používání aplikace bude tedy připojení k internetové síti podmínkou. Autonomní zařízení bude server kontaktovat samo, výhodou je nezávislost volání požadavků na programovacím jazyce.

Zařízení bude serveru oznamovat příznak počátku svého letu (svojí jízdy). Zpátky se mu v odpovědi vrátí číslo nově vytvořeného letu (jízdy) a toto číslo bude poté odesílat spolu s ostatními údaji, jako jsou aktuální poloha, snímek z kamery a další, v pravidelných intervalech během svého letu. Předpokládáme, že záznam o aktuální poloze nebude zařízení odesílat častěji než každou vteřinu.

Data v HTTP požadavcích budou mít formát JSON. Tento formát jsem vybrala, protože je kompaktní a snadno rozšiřitelný. Kdyby tedy časem přibyla další data, která by autonomní zařízení o svém letu mohlo zasílat serveru, snadno by se tento objekt o tato data obohatil.

Záznam o poloze zařízení nemusí obsahovat snímek z kamery. Pokud se ale fotografie pořídila a bude se odesílat serveru, soubor se snímkem se pro přenos zakóduje pomocí algoritmu Base64.

Pro zabezpečení komunikace jsem navrhla způsob ověření použitím neveřejných klíčů, které bude znát každé autonomní zařízení a samozřejmě webová aplikace. Každá zpráva serveru se s tímto neveřejným klíčem zakóduje pomocí hašovacího algoritmu SHA1 a na serveru se při příjmu požadavku zkontroluje správnost tohoto otisku. Tím se zajistí, že komunikaci nebude možné napodobit z jiného zdroje, pokud nebude znát neveřejný klíč.

3. Mapové podklady

Než bylo jasné, jaké mapové podklady bude webová aplikace využívat, dostala jsem za úkol prostudovat možnost použití map Melown. Tato služba poskytuje možnost 3D vizualizace map na vlastní webové stránce. Neplacená verze je určena pro veřejné weby, počet zobrazení je omezen na 3 000 za měsíc a úložiště má velikost 2 GB. Tato verze poskytuje všechny veřejně dostupné mapové podklady. Služba Melown se ale ukázala jako nevyhovující, protože její data pro 3D vykreslení nepokrývají celý povrch České republiky. [2]

Jako nejlepší řešení se ukázalo použití webové služby typu REST, která bude běžet na stejném serveru jako webová aplikace. Tato služba poskytuje mapové podklady stažené ze serverů ArcGIS a OpenStreetMap. Ortofoto dlaždice se získává ze systému ArcGIS, stejně jako DMR (digitální model reliéfu). Záznamy o jednotlivých entitách v krajině (budovy, cesty, lesy apod.) se stahují ze serveru OpenStreetMap. Informace o výšce budov ovšem v těchto datech chybí, proto se pro každou budovu zjišťuje její výška z DMP (digitální model povrchu) pomocí služby ArcGIS. Výška každé budovy se rovná nejvyšší naměřené hodnotě DMP v oblasti tvořené jejím půdorysem. [3]

Data získaná pomocí této služby se poté budou vykreslovat na straně klienta pomocí WebGL, knihovny pro práci s grafikou v prohlížeči.

4. Návrh databáze

Databázový systém, který je dostupný na serveru vyhrazeném pro aplikaci, se nazývá MariaDB. Jedná se o open source databázový server vytvořený vývojáři MySQL, databáze je relační. [4]

Pro účely webové aplikace je nutné, aby se do této databáze ukládaly tyto informace o jízdách/letech: poloha daného zařízení ve WGS-84 v daný čas a výška zařízení v metrech nad mořem. Pro tyto účely jsem navrhla databázovou entitu s názvem *location*, která má tyto atributy:

Tabulka 1: Seznam atributů entity location

Název atributu	**Datový typ**	**Výchozí hodnota**	**Klíč**	**Komentář**
Id_location	UNSIGNED INT	AUTO_INCREMENT	PRIMARY KEY	Identifikační číslo záznamu o aktuální poloze
Id_flight	UNSIGNED INT	Žádná hodnota	FOREIGN KEY (flight.Id_flight)	Identifikační číslo jízdy/letu
Stamp	DATETIME	Žádná hodnota	UNIQUE KEY (Id_flight, Stamp)	Datum a čas oznámení lokace
Height	DECIMAL (12, 6)	Žádná hodnota		Aktuální výška v metrech nad mořem
Longitude	DECIMAL (12, 6)	Žádná hodnota		Zeměpisná délka v WGS-84 zobrazení
Latitude	DECIMAL (12, 6)	Žádná hodnota		Zeměpisná šířka v WGS-84 zobrazení

Sloupce *Id_flight* a *Stamp* tvoří společně složený unikátní klíč, protože jedno zařízení nebude posílat údaj o své poloze častěji než jednou za vteřinu. Atribut *Id_flight* v tabulce *location* odkazuje na primární klíč entity *flight*. Ta slouží k uchovávání dat o jednotlivých jízdách či letech autonomních zařízení. V této

tabulce je definován složený unikátní klíč na atributech *Id_drone* a *Start*, protože jedno zařízení může v jednu chvíli započít svůj let pouze jednou. Entita *flight* neobsahuje žádný atribut, který by uchovával datum a čas ukončení letu, protože za ukončení letu můžeme považovat datum a čas posledního záznamu daného letu v tabulce *location*. Proto by atribut tohoto typu pouze duplikoval již známou hodnotu.

Tabulka 2: Seznam atributů entity flight

Název atributu	**Datový typ**	**Výchozí hodnota**	**Klíč**	**Komentář**
Id_flight	UNSIGNED INT	AUTO_INCREMENT	PRIMARY KEY	Identifikační číslo jízdy/letu
Id_drone	UNSIGNED INT	Žádná hodnota	FOREIGN KEY (drone.Id_drone)	Identifikační číslo auta/dronu
Start	DATETIME	Žádná hodnota	UNIQUE KEY (Id_drone, Start)	Datum a čas začátku letu

Sloupec s názvem *Id_drone* v tabulce *flight* odkazuje na primární klíč tabulky *drone*. Tato tabulka je určena k uchovávání dat o jednotlivých zařízeních a má následující atributy:

Tabulka 3: Seznam atributů entity drone

Název atributu	**Datový typ**	**Výchozí hodnota**	**Klíč**	**Komentář**
Id_drone	UNSIGNED INT	AUTO_INCREMENT	PRIMARY KEY	Identifikační číslo autonomního zařízení
Name	VARCHAR (50)	Prázdný řetězec	UNIQUE KEY (Name)	Jedinečný název zařízení

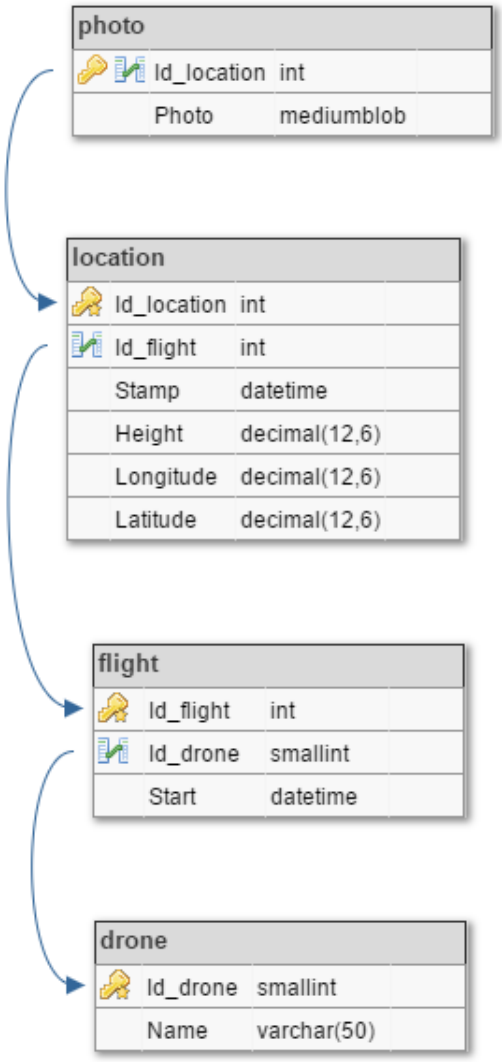
Poslední entita, která se v databázi nachází, nese název *photo*. V této entitě jsou umístěny snímky pořízené kamerami jednotlivých zařízení. Fotografie jsou uloženy v samostatné tabulce proto, že snímek není při ohlášení aktuální polohy povinným

prvkem, a proto ke každému záznamu o lokaci nemusí existovat fotografie. Primárním klíčem této entity je identifikační číslo záznamu o lokaci *Id_location* z tabulky *location*.

Tabulka 4: Seznam atributů entity photo

Název atri-butu	**Datový typ**	**Výchozí hodnota**	**Klíč**	**Komentář**
Id_location	UNSIGNED INT	Žádná hodnota	PRIMARY KEY	Identifikační číslo záznamu o aktuální poloze
Photo	MEDIUM BLOB	Žádná hodnota		Fotografie z kamery jako binární hodnota

Všechny entity používají storage engine InnoDB. Následující obrázek znázorňuje schéma celé databáze. Jednotlivé entity jsou znázorněny jako obdélníky, šipky mezi nimi naznačují propojení cizím klíčem. Primární klíče jsou označeny symbolem žlutého klíče, hvězdička znamená výchozí hodnotu auto increment. Zelený symbol označuje cizí klíč.



Obrázek 1: Schéma databáze

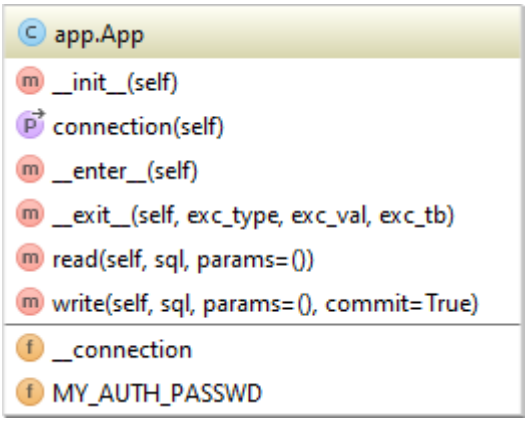
5. Backend aplikace

Serverová část webové aplikace jsem napsala v programovacím jazyce Python verze 2.6. Jako webový framework jsem vybrala Flask. Jeden z hlavních důvodů pro tuto volbu byla má předchozí zkušenost s tímto frameworkem. Mezi výhody tohoto nástroje patří jeho přehlednost a také kompaktnost ve srovnání s mírně známějším frameworkem Django. Flask je postaven na HTTP knihovně Werkzeug a používá šablonovací systém s názvem Jinja2. [5]

5.1 Struktura aplikace

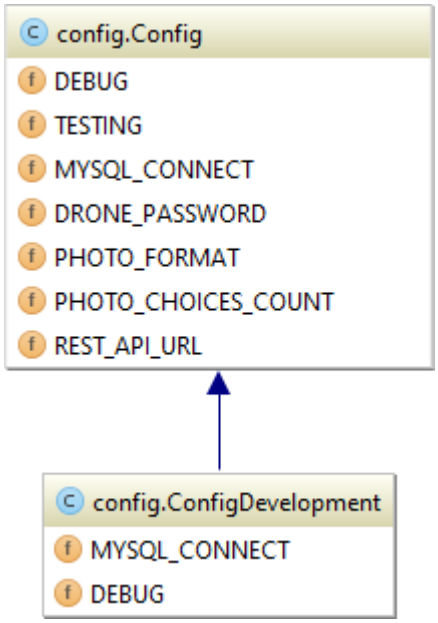
Vstupním bodem do webové aplikace je v ostrém režimu soubor *web.wsgi*, který se nachází v kořenovém adresáři projektu. V tomto souboru se importuje modul s názvem *web*, který obsahuje mimo jiné definici metod, které jsou namapovány na všechny URL adresy, které jsou pro aplikaci potřeba. Dále se v modulu *web* nachází definice metody, která obsluhuje chybu s kódem 404 (stránka nenalezena). Pro spuštění testovacího režimu je třeba spustit právě soubor *web.py*, který v takovém případě spustí webový server v režimu ladění.

Pro práci s databází slouží třída *App* umístěná v modulu *app*. Instance této třídy se vytváří pomocí příkazu *with* – při opuštění bloku *with* se uzavře spojení s databází. Při pokusu o první dotaz do databáze v obsluze požadavku je navázáno spojení s použitím knihovny MySQLdb [6]. Dále má třída *App* metody pro zápis do databáze a čtení z databáze. Tyto metody vyžadují jako vstupní parametr SQL dotaz ve formě řetězce a nepovinně data, která se do dotazu mají vložit, ve formě sekvence (např. *tuple* nebo *list*). Data musí být v takovém pořadí, v jakém se mají do dotazu vložit. Knihovna MySQLdb poskytuje ochranu proti útoku typu SQL injection, takže není nutné ji implementovat explicitně v aplikaci. [7]



Obrázek 2: Diagram třídy App

Konfigurace aplikace se nachází v modulu *config*. Pro hodnoty, které platí pro ostrou verzi aplikace, slouží třída *Config*. Třída *ConfigDevelopment* je pak potomkem této třídy a přepisuje hodnoty těch konstant, jejichž hodnota se v testovací verzi liší. Seznam konstant těchto tříd se nachází v tabulce pod obrázkem s diagramem.



Obrázek 3: Diagram tříd pro konfiguraci

Tabulka 5: Seznam konstant konfigurace na serveru

Název konstanty	Význam	Hodnota
DEBUG	Režim ladění / ostrý režim	True pro testovací verzi, False pro ostrou
TESTING	Režim testování	False
MYSQL_CONNECT	Informace pro navázání spojení s databází	Slovník (liší se pro ostrou a testovací verzi)
DRONE_PASSWORD	Neveřejný klíč pro ověření autonomních zařízení	Náhodně vygenerovaný řetězec znaků
PHOTO_FORMAT	Formát souborů se snímky z kamer dronu a auta	PNG
PHOTO_CHOICES_COUNT	Počet fotografií, které se uživateli zobrazují k náhledu	20
REST_API_URL	URL k REST API pro získávání mapových podkladů	https://drone.fm.tul.cz:18443/DroneComponent/service

5.2 API pro komunikaci s autonomním zařízením

Komunikace autonomních zařízení s aplikací probíhá pomocí HTTP požadavků zabezpečených SSL šifrováním komunikace. URI adresy těchto požadavků společně se vstupními parametry jsou definovány v modulu *web*.

5.2.1 Start jízdy/letu

První z těchto požadavků obsluhuje metoda namapovaná na URI */dronestart* a povoluje pouze volání HTTP metodou POST. Autonomní zařízení ji využije ve chvíli, kdy zahajuje svou jízdu (svůj let). Funkce očekává dva vstupní parametry požadavku. První se nachází pod klíčem *„data“* a obsahuje informace o zahájení svého letu ve formátu JSON. Druhým parametrem požadavku (pod klíčem *„hash“*) musí být SHA1 otisk

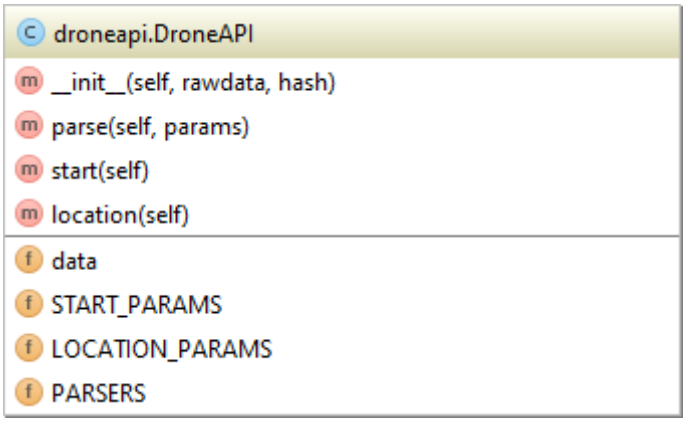
řetězce vytvořeného spojením JSON dat (prvního parametru), znaku svislá čára a neveřejného klíče pro ověření autonomního zařízení. Tento klíč se skládá z náhodně vygenerovaných znaků a jeho kopii má každé autonomní zařízení, které bude API využívat. Stejný otisk se následně vypočítá na serveru a jeho hodnota se porovnává s tím přijatým. Pokud tyto parametry chybí, odpověď na požadavek má chybový kód 405. V případě, že přijatá data nejsou validní (například je nelze přečíst, vypočítaný otisk se neshoduje s přijatým nebo toto zařízení svůj start v tento čas oznámilo již dříve), odpověď má chybový kód 403. Pokud se ale záznam o startu podaří uložit, odpověď na požadavek má kód 200 a v jejím těle se nachází identifikační číslo nově vytvořené jízdy (letu).

Data předaná v prvním parametru mají podobu slovníku (asociativního pole), jejich přesnou specifikaci popisuje následující tabulka. Všechny položky jsou povinné.

Tabulka 6: Data o startu jízdy/letu

Klíč	**Význam**	**Formát**
start	Datum a čas zahájení jízdy/letu	Řetězec ve formátu Y-m-d H:M:S
drone	Identifikační číslo dronu	Celé číslo

Funkce vytváří instanci třídy *DroneAPI* umístěné v modulu *droneapi*. V konstruktoru této třídy se provede zmiňované ověření otisku a také dekódování přijatých JSON dat na slovník. Po vytvoření objektu třídy *DroneAPI* se volá jeho metoda *start*. Ta převede hodnoty předaných položek z řetězců na příslušné datové typy, *pomocí instance třídy App* uloží záznam o startu do databázové tabulky *flight* a vrátí identifikační číslo nově vytvořeného letu (jízdy).



Obrázek 4: Diagram třídy DroneAPI

5.2.2 Oznámení o poloze zařízení

Druhý HTTP požadavek lze volat pod adresou */dronelocation* také metodou POST. Na tuto URI je v modulu *web* namapována metoda *dronelocation*. Autonomní zařízení ji použije k pravidelnému zasílání snímku z kamery a údajů o své aktuální poloze. Metoda opět očekává dva vstupní parametry požadavku – data ve formátu JSON a otisk dat pro ověření autonomního zařízení. Při absenci těchto parametrů je klientovi navrácena odpověď s chybovým kódem 405, nevalidní data jsou oznámena chybou číslo 403. Pokud uložení proběhne v pořádku, odpověď má status 200 a obsahuje identifikační číslo nově vytvořeného záznamu o poloze zařízení.

Data předaná v prvním parametru mají opět podobu slovníku. Všechny položky kromě snímku z kamery jsou povinné. V následující tabulce specifikujeme jednotlivé položky tohoto slovníku.

Tabulka 7: Data o aktuální poloze zařízení

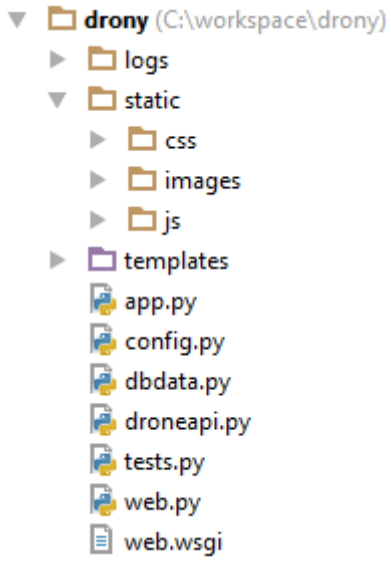
Klíč	Význam	Formát
flight	Identifikační číslo jízdy/letu	Celé číslo
stamp	Datum a čas oznámení polohy	Řetězec ve formátu Y-m-d H:M:S
height	Aktuální výška v metrech nad mořem	Desetinné číslo
longitude	Aktuální zeměpisná délka v WGS-84 zobrazení	Desetinné číslo
latitude	Aktuální zeměpisná šířka v WGS-84 zobrazení	Desetinné číslo
photo	Aktuální snímek z kamery (nepovinný)	Obrázek ve formátu PNG zakódovaný pomocí Base64

Funkce obsluhující tento požadavek vytvoří instanci třídy *DroneAPI* a poté volá její metodu *location*. Ta převede hodnoty předaných dat na příslušné datové typy. Záznam o poloze se pomocí instance třídy *App* uloží do databázové tabulky *location* a snímek z kamery (pokud existuje) do tabulky *photo*. Pokud ukládáme i fotografii, tyto dvě operace se vykonávají v transakci, aby bylo zajištěno, že se provedou obě anebo žádná z nich. Metoda vrací identifikační číslo nově vytvořeného záznamu o poloze zařízení.

5.3 Backend webové aplikace pro monitorování zařízení

V kořenovém adresáři aplikace se nachází složky *logs*, *static* a *templates*. Adresář *logs* slouží k ukládání záznamů o chybách, které při používání aplikace nastaly. Ve složce *static* najdeme podadresář *js*, který obsahuje všechny skripty pro klientskou stranu aplikace. Další podadresář má název *css* a definujeme v něm vzhled webové aplikace. Poslední složka uložená v adresáři *static* se nazývá *images* a slouží pro uchovávání statických obrázků zobrazovaných na výstupu

aplikace. Ve složce *templates* můžeme najít všechny HTML šablony, které aplikace využívá.



Obrázek 5: Adresářová struktura webové aplikace

5.3.1 Modul pro získání dat z databáze

Jedním ze souborů umístěných v kořenovém adresáři aplikace je modul *dbdata*. V tomto modelu je definována jediná třída s názvem *DBData*, která slouží jako prostředník mezi HTTP požadavky a daty v databázi. Všechny její metody jsou statické. První metoda této třídy nese název *image* a přebírá parametr *id*. Funkce nejprve kontroluje datový typ předaného parametru – pokud *id* není celé číslo, nastane výjimka typu *AssertionError*. Dále je proveden dotaz do databáze s použitím instance třídy *App* a metoda vrací fotografii uloženou v databázi pod číslem *id*. Pokud se v databázi nenachází fotografie s tímto identifikačním číslem, nastává chyba typu *AssertionError*.

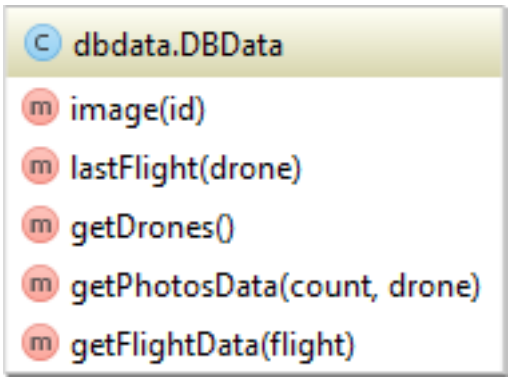
Další metoda se jmenuje *lastFlight* a jako parametr přebírá identifikační číslo dronu. Slouží k získání dat o posledním letu daného dronu (poslední jízdy auta). Po kontrole datového typu, která probíhá obdobně jako v metodě image, se z databáze zjistí informace o posledním letu, příp. jízdy cíleného zařízení (identifikační číslo letu/jízdy, datum a čas startu a jednotlivé nahlášené polohy zařízení) a tato data jsou navrácena.

Tento dotaz do databáze je zkonstruován tak, aby vrátil poslední let (jízdu), pro který už byla nahlášena alespoň jedna poloha.

Pro získání seznamu všech dronů a aut v databázi slouží metoda *getDrones*. Z návratové hodnoty můžeme získat identifikační čísla a názvy jednotlivých zařízení, která se v databázi nacházejí.

Další metoda této třídy se nazývá *getPhotosData* a lze ji použít k zjištění dat o pořízených fotografiích vybraného dronu. Její první parametr určuje, kolik nejnovějších snímků má funkce vrátit, a druhý parametr představuje identifikační číslo zařízení, jehož kamerové snímky se mají hledat. Po zkontrolování datových typů předaných parametrů se provede dotaz do databáze, jehož výsledek obsahuje identifikační čísla fotografií, datum a čas jejich pořízení a polohu autonomního zařízení v době, kdy byl snímek vyfocen. Tento výsledek se stává výstupem metody *getPhotosData*.

Poslední funkci třídy *DBData* lze použít k získání dat o určitém letu (jízdě) vybraného zařízení. Metoda se nazývá *getFlightData* a jako jediný parametr očekává identifikační číslo letu/jízdy v podobě celého čísla. Pokud má parametr jiný datový typ, metoda vyvolá výjimku typu *AssertionError*. Výstupem funkce je pole záznamů o poloze zařízení – jejich identifikační číslo, datum a čas a samotná poloha zařízení.



Obrázek 6: Diagram třídy pro získávání dat z databáze

5.3.2 HTTP požadavky

Třída *DBData* je importována v modulu *web* a využívá se zde v několika metodách mapovaných na URI adresy, které volá na své straně klient. První z nich je metoda namapovaná na HTTP požadavek typu GET, jehož URL končí řetězcem */image/* a identifikačním číslem fotografie. Odpověď na tento požadavek obsahuje soubor s fotografií, kterou získáme metodou *image* třídy *DBData*. Pokud tato metoda způsobí chybu typu *AssertionError*, odpověď na požadavek má chybový kód 404.

Metoda, která obsluhuje HTTP požadavek typu GET s URI složenou z řetězce */lastflight/* a identifikačního čísla autonomního zařízení, slouží k získání dat o posledním letu (jízdě) vybraného zařízení. Využívá k tomu metodu *DBData.lastFlight* a vrací JSON reprezentaci těchto dat. Převod do formátu JSON zajišťuje metoda *jsonify* frameworku Flask.

Adresa */dronephotos/* následovaná identifikačním číslem zařízení je mapována na funkci, která vrací vykreslenou šablonu *photochoices.html*. K vykreslení ji předává data zjištěná metodou *DBData.getPhotosData* (její volání je ošetřeno, při chybě typu *AssertionError* má odpověď na požadavek chybový kód 405). Šablona se vyplňuje daty pomocí metody *render_template*, kterou poskytuje framework Flask.

Analogicky funguje metoda obsluhující požadavek s URI */dronepath/* s identifikačním číslem letu/jízdy. Výstup funkce *DBData.getFlightData* se předává metodě pro vykreslení šablony *dronepath.html*.

Šablonu *index.html* vykresluje metoda *index*, která se volá při otevření hlavní stránky webové aplikace. Z databáze se nejprve pomocí metody *getDrones* třídy *DBData* získají data o všech autonomních zařízeních a ta se předávají funkci pro vykreslení dané šablony.

V neposlední řadě se v aplikaci nachází funkce *gpsToSjtsk* mapovaná na POST požadavek s URI */gpstosjtsk*. Tato funkce slouží k převodu souřadnic ze systému WGS-84 do S-JTSK. V těle HTTP požadavku musí být předán JSON objekt s polem souřadnic, které se mají převést. K samotnému převodu se využívá REST API metoda typu POST pod URI */multipleGpsToSJTSK*. REST API se volá pomocí knihovny *requests* a výsledek je navrácen opět ve formátu JSON.

Obdobně funguje metoda *getMap*, která obsluhuje požadavek typu GET s URI */getmap/x/y*, kde hodnoty *x* a *y* představují zeměpisnou délku a šířku v systému WGS-84. Tato metoda vrací mapové podklady okolí zadaného bodu ve formátu JSON. Data popisují 9 dlaždic o velikosti 128 krát 128 metrů, hledaný bod se nachází v dlaždici uprostřed. Funkce volá REST API metodu typu GET s adresou */getTilesAroundMap/x/y/1* (s dosazením hodnot *x* a *y*, poslední číslo je požadovaný počet dlaždic na každou stranu okolo dlaždice s daným bodem), jejíž výsledek je předán jako odpověď na původní požadavek.

6. Frontend webové aplikace

6.1 Struktura klientské strany aplikace

6.1.1 Použité knihovny

Pro moderní vzhled aplikace jsem se rozhodla použít open source framework Twitter Bootstrap verze 3.3.7. Díky této knihovně se web bude zobrazovat stejně ve všech prohlížečích na všech zařízeních. Mezi jeho přednosti patří také seznam vlastních snadno ovladatelných HTML prvků a komponent, které se pyšní kvalitně zpracovanou a rozsáhlou dokumentací. Soubory typu CSS z frameworku Twitter Bootstrap jsem umístila do složky *css* v adresáři *static*, soubory JavaScriptu se nachází ve složce *js* uložené také v adresáři *static*. [8]

Další knihovna, kterou jsem v klientské části aplikace použila, je jQuery ve verzi 3.2.0. Jedná se o funkčně bohatou knihovnu JavaScriptu, která výrazně zjednodušuje například práci s technologií Ajax, manipulaci s HTML dokumentem, obsluhování událostí a další. Poskytuje API, které funguje stejně v nejpoužívanějších prohlížečích. Prakticky se jedná o jeden soubor s příponou *js*, který jsem uložila do složky *js* v adresáři *static*. [9]

Pro vykreslování 3D scény na webu je nutné použít WebGL. Jedná se o API v JavaScriptu pro akcelerované vykreslování grafiky. Toto rozhraní je navrženo tak, aby bylo architektonicky identické s OpenGL ES 2.0. [10]

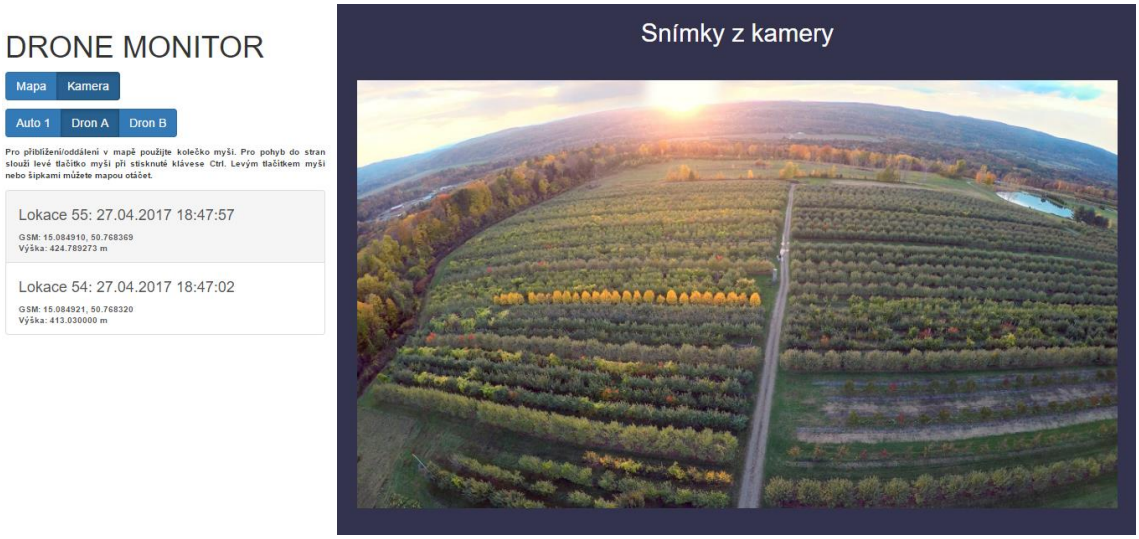
Práci s WebGL může výrazně usnadnit 3D framework. Při jeho výběru jsem se rozhodovala mezi Three.js a Babylon.js. Oba frameworky používají k vytvoření scény, kamery, objektů a vykreslení scény podobné metody WebGL. Three.js bylo vytvořeno s cílem vytvářet grafiku a animace posílené grafickou kartou, používá široký přístup k webové grafice, aniž by se soustředil na jednotlivé funkce pro animace. Naproti tomu Babylon.js zaujímá cílenější přístup – udržuje si sklon k vývoji webových her s funkcemi jako například detekce kolizí a antialiasing. Vzhledem k tomu, že oba frameworky se používají podobně a splňují požadavky na vývoj tohoto projektu, vybrala jsem si Babylon.js, protože jeho dokumentace a návody se mi zdají dobře zpracované a přehledné. [11]

6.1.2 Index

Webová stránka se dělí na dvě části. První, levá, část slouží převážně k ovládání aplikace. Nachází se zde název aplikace, tlačítka pro výběr mezi mapou a snímky z kamery a tlačítka pro výběr autonomního zařízení, které si uživatel přeje sledovat. V závislosti na vybraném režimu (mapa/kamera) se pak pod těmito tlačítky zobrazuje další obsah – v případě mapy se zde vypisuje krátký návod na její použití a dále seznam lokací vybraného zařízení. Pokud je vybrán režim kamery, vypíše se na toto místo seznam poloh vybraného zařízení, k jejichž oznámení byla přiložena fotografie.

Větší, pravá, část stránky je věnována samotné mapě a snímkům z kamery. V režimu sledování zařízení na mapě se zde vykresluje 3D mapa blízkého okolí posledního bodu poslední letu/jízdy vybraného zařízení. Pokud je zvolen režim kamery, po kliknutí na některou z nabízených lokací se na tomto místě zobrazí náhled fotografie, která byla přiložena k oznámení této polohy.

Obrázek 7: Webová stránka v režimu mapy



Obrázek 8: Webová stránka v režimu kamery

HTML kód tlačítek pro výběr zařízení se generuje na straně serveru pomocí šablonovacího systému Jinja2. Pro každé zařízení, které se nachází v databázi, vytváří jeden prvek *input* typu *radio*. Jako aktivní se zvolí první ze zařízení, která jsou seřazena abecedně podle názvu. Pokud zařízení nemá vyplněno název, zobrazuje se jako text tlačítka řetězec „Zařízení číslo“ následovaný jeho identifikačním číslem.

6.1.3 Skript index.js

Skripty, které se provádějí na straně klienta (s výjimkou skriptu pro samotné vykreslení 3D scény), jsem seskupila do souboru *index.js* ve složce *static/js*. Po načtení celého HTML dokumentu se do proměnné *canvas* uloží prvek typu *canvas*, do kterého se bude vykreslovat 3D mapa. Do proměnné *engine* se uloží objekt třídy *BABYLON.Engine* z frameworku Babylon.js – jako první parametr se předává proměnná *canvas*, druhý parametr má hodnotu *true* a udává, že *engine* bude podporovat antialiasing.

Ve skriptu *index.js* je dále definována funkce *htmlDronePath*, která slouží k vyplnění nebo vyprázdnění elementu s identifikátorem *dronepath* – zde se uživateli zobrazuje seznam několika posledních lokací posledního letu (popř. jízdy) vybraného zařízení. Očekává jeden parametr, který představuje identifikační číslo vybraného zařízení. V případě, že je hodnota tohoto parametru prázdná (nebo *null*), element s identifikátorem *dronepath* se vyprázdní. V opačném případě se provede asynchronní

požadavek na server na URI */dronepath/* následovanou předaným číslem zařízení. Server vyplní šablonu *dronepath.html* a vrátí výsledek této operace. Tělo odpovědi na tento požadavek se na straně klienta vloží do elementu s identifikátorem *dronepath* jako HTML kód. V levé části se tedy uživateli zobrazí seznam několika posledních záznamů o poloze vybraného zařízení.

Další funkce se jmenuje *htmlPhotoChoices* a slouží k naplnění elementu s identifikátorem *photochoices*. V tomto prvku se nachází tabulka několika posledních lokací vybraného zařízení, při jejichž oznámení byla pořízena fotografie. Funkce provádí asynchronní požadavek na URI */dronephotos/* následovanou identifikačním číslem vybraného zařízení. Výsledek tohoto požadavku je HTML kód, který se vloží do prvku s identifikátorem *photochoices*.

Událost pro výběr režimu kamery je obsloužena funkcí *camera*. Tato funkce nejprve skryje odstavec s instrukcemi pro ovládání mapy. Dále smaže text odstavce, ve kterém se objevují chybové hlášky. 3D mapa (*canvas*) je skryta a vykreslovací smyčka objektu *engine* se zastaví. Zobrazí se element pro zobrazování snímků z kamery. V poslední řadě se volají funkce *htmlDronePath* s parametrem *null* a *htmlPhotoChoices*.

Analogicky událost změny tlačítka pro režim mapy obsluhuje funkce *map*. Ta nejprve zobrazí odstavec s instrukcemi pro ovládání mapy a dále volá funkci *loadMap* pro vykreslení mapy. Schová element pro snímky z kamery a prvek pro výpis fotografií vyprázdní.

Událost, která nastane při výběru jiného autonomního zařízení, je obsloužena funkcí *droneChange*. Funkce zjistí, zda je vybrán režim mapy nebo kamery a podle toho buďto obnoví mapu pomocí funkce *loadMap* nebo obnoví prvky pro režim kamery.

Funkce *loadMap* je definována také v modulu *index.js*. Funkce očekává jako vstupní parametr identifikační číslo zařízení, které je aktuálně vybráno ke sledování. Funkce nejprve zobrazí uživateli informaci o tom, že mapa se načítá (načtení může trvat i několik minut). Aby uživatel během asynchronního načítání nezpůsobil klikáním na tlačítka pro výběr režimu a zařízení chybu, nelze na tato tlačítka v této době klikat. Následně se pomocí funkce *jQuery.getJSON* odešle asynchronní požadavek na server na URI */lastflight/* následované identifikačním číslem vybraného zařízení. Tento poža-

davek vrací pole lokací z posledního letu (popř. jízdy) daného zařízení ve formátu JSON.

Po úspěšném zpracování tohoto požadavku se zjišťuje délka pole v jeho odpovědi. Pokud se v poli lokací nenachází žádný záznam, uživateli se zobrazí informace o tom, že v databázi se nenachází žádný let (popř. jízda) vybraného zařízení nebo toto zařízení právě odstartovalo a nenahlásilo ještě svou pozici. Pomocí funkce *engine.StopRenderLoop* se zastaví vykreslování 3D scény (pokud bylo spuštěno) a prvek *canvas* je skryt, stejně jako hláška o načítání. Tlačítka pro výběr režimu a zařízení jsou opět odkryta. Tabulka s jednotlivými lokacemi se vyprázdní pomocí funkce *htmlDronePath*, jíž se jako parametr předává hodnota *null*.

Pokud pole s lokacemi není prázdné, volá se funkce *htmlDronePath* a jako parametr se jí předává číslo letu (jízdy). Tím se vyplní tabulka se záznamy o polohách vybraného zařízení. Pomocí funkce *jQuery.ajax* se poté odešle další asynchronní požadavek typu POST na server na URI */gpstosjtsk*. V těle tohoto požadavku se nachází získané polohy zařízení jako pole souřadnic systému WGS-84 ve formátu JSON. V odpovědi na tento požadavek očekáváme tyto souřadnice převedené do systému S-JTSK. Nadále již pracuje aplikace téměř výhradně se systémem S-JTSK. Na základě získaných dat se naplní proměnná *drone_path*. Jeho hodnota se nastaví na pole, kde každá položka představuje jednu lokaci zařízení a naplněna polem tří hodnot: zeměpisná délka v S-JTSK, zeměpisná šířka v S-JTSK a výška zařízení v metrech nad mořem.

Následně se odesílá poslední asynchronní požadavek na server, a to na URI složenou z řetězce */getmap/* a souřadnic posledního bodu trasy vybraného zařízení v systému WGS-84. Tímto způsobem získáme mapové podklady blízkého okolí daného bodu. Po úspěšném provedení tohoto požadavku se volá metoda *threeDScene*, která slouží pro samotné vytvoření a vykreslení 3D scény. Jako parametry přebírá odpověď na poslední požadavek (mapové podklady) a proměnné *engine*, *canvas* a *drone_path*.

6.2 3D mapa

6.2.1 Mapové podklady

Mapové podklady, které jsou přijaty ze serveru ve formátu JSON, obsahují dvě položky: pole devíti dlaždic a pole entit. Dlaždice mají tvar čtverce, jejich velikost v metrech udává položka *size*. Každá dlaždice je identifikována souřadnicemi svého levého horního rohu. Mezi další položky každé dlaždice patří minimální a maximální nadmořská výška v dané oblasti a ortofoto zakódované pomocí Base64. Položka *dataDMR* obsahuje dvourozměrné pole hodnot DMR. Podle těchto dat se bude zakřivovat reliéf dlaždice. Krok měření, který obsahuje položka s názvem step, udává počet metrů, po kterém bylo prováděno měření DMR (zpravidla roven dvěma – mezi jednotlivými hodnotami je tedy skok dva metry). Pokud bychom tedy vydělili velikost dlaždice krokem, měli bychom dostat počet polí v poli *dataDMR* a zároveň délku každého tohoto vnitřního pole.

Entity, které se v mapových podkladech nachází, nejsou rozděleny do jednotlivých dlaždic, jsou uloženy všechny v jednom poli. Každý entita je identifikována jedinečným číslem. Dalším parametrem každé entity je její typ. Ten může nabývat hodnot „*building*“, „*highway*“, případně další. Aplikace vykresluje pouze entity typu „*building*“, tedy budovy. Dále má každá entita uvedenou svou výšku v metrech nad mořem (nadmořská výška nejvyššího bodu DMP v oblasti budovy) a také minimální výšku (nejnižší nadmořská výška DMR v oblasti budovy). Poslední položka *SJTSK* obsahuje pole S-JTSK souřadnic jednotlivých bodů půdorysu dané budovy. První a poslední bod je stejný, čímž se polygon uzavírá.

6.2.2 Souřadnice

3D mapa pracuje se souřadnicemi v systému S-JTSK. Tyto souřadnice je nutné namapovat na souřadnice 3D scény. Po načtení mapových podkladů se do proměnné *xzero* uloží souřadnice *x* levého horního rohu první dlaždice, do proměnné *yzero* souřadnice *y* levého horního rohu první dlaždice a do proměnné *zzero* se vloží minimální DMR

hodnota naměřená v první dlaždici. Tyto hodnoty se nadále budou používat jako bod [0, 0, 0].

6.2.3 Scéna, kamera, světlo

Kód pro tvorbu a vykreslení 3D mapy jsem uložila do souboru *threedscene.js*. V souboru se nachází jedna rozsáhlá funkce *threeDScene*. V této funkci se nejprve vytvoří scéna jako objekt třídy *BABYLON.Scene*, která se uloží do proměnné *scene*, konstruktoru je předána proměnná *engine*. Poté se volá funkce *engine.runRenderLoop*, která očekává jako parametr funkci pro vykreslení (renderování) scény. Tuto funkci jsem pojmenovala *renderLoop* a do jejího těla jsem napsala volání funkce *scene.render*. Díky tomuto postupu se spustí smyčka vykreslování scény.

Po spuštění tohoto skriptu vidíme v prohlížeči pouze jednobarevnou plochu. Abychom mohli vidět objekty, které později do scény přidáme, je nutné vytvořit objekt kamery. Babylon.js framework nabízí dva základní typy kamer: *FreeCamera* a *ArcRotateCamera*. *FreeCamera* je postavena na konceptu FPS (first person shooter). Naopak *ArcRotateCamera* se otáčí okolo jednoho daného cílového bodu. Pro zobrazení mapy se lépe hodí *ArcRotateCamera*. Otáčení se ovládá pomocí levého tlačítka myši (případně šipkami), přiblížení a oddálení pomocí kolečka myši. Pro pohyb do stran (posunutí cílového bodu, okolo kterého se kamera otáčí) slouží levé tlačítko myši při stisknuté klávese Ctrl.

Objekt *camera* se tvoří instancí třídy *BABYLON.ArcRotateCamera*. První parametr má hodnotu *„ArcRotateCamera“* a udává identifikační řetězec objektu ve scéně. Další dva parametry udávají výchozí otočení podle osy *y* a *x* (v aplikaci nastaveno na nuly). Další parametr udává výchozí hodnotu přiblížení (v aplikaci nastaveno na 10). Následující parametr představuje cílový bod, okolo kterého se má kamera otáčet. Body ve 3D scéně se vytváří instancí třídy *BABYLON.Vector3*. Tento bod je v aplikaci nastaven na bod [0, 0, 0]. Posledním parametrem pro vytvoření kamery je objekt scény. Aby s mapou nešlo otáčet tak, aby bylo možné vidět spodní část země, nastaví se omezení otáčení podle osy x a osy y. Přiblížení/oddálení kamery je omezeno na hodnotu 1 až 500.

Po nastavení těchto omezení se nastaví cílový bod kamery na bod, který bude ve středu devíti dlaždic. Souřadnice *x* a *y* tohoto bodu se vypočítají jako podíl velikosti první dlaždice a jejího kroku vynásobený číslem -1.5, čímž se cíl posune do poloviny druhé dlaždice. Souřadnice *z* nabývá hodnoty nula. Nastaví se přiblížení kamery na hodnotu 200 a kamera se připojí ke scéně použitím funkce *camera.attachControl*, jíž se jako parametr předává proměnná *canvas*. Následně se kamera otočí tak, aby její výchozí úhel odpovídal pohledu na mapu mírně shora.

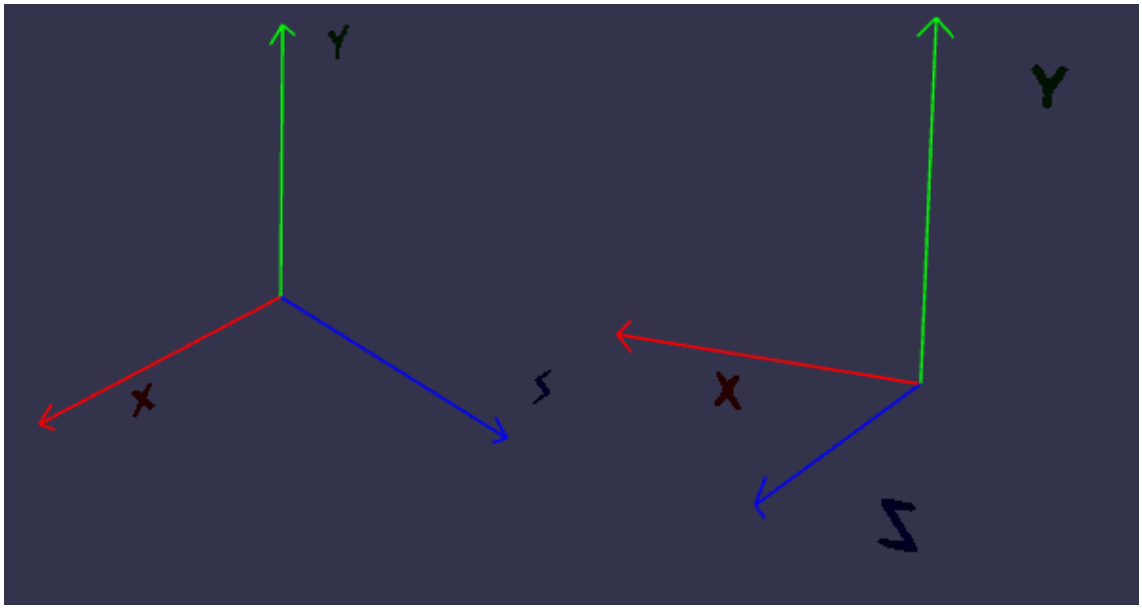
Aby byly ve scéně vidět její objekty, je také nutné vytvořit světlo. Realistické světlo se vytváří instancí třídy *BABYLON.HemisphericLight*. Jako parametr jí předáváme identifikační řetězec v rámci scény, bod v 3D prostoru, odkud bude svítit do všech směrů, a objekt scény.

6.2.4 Osy x, y a z

Pro lepší vizualizaci a orientaci v prostoru 3D scény při ladění aplikace jsem si vytvořila náhled souřadnicových os *x*, *y* a *z*. K tomu jsem definovala funkci *showAxis*, která přebírá jako jediný parametr velikost os. Každá osa se vytváří voláním funkce *BABYLON.Mesh.CreateLines*. Této funkci se předává jedinečný název objektu v rámci scény, pole bodů, ze kterých má být šipka složena, a objekt scény. Například šipka znázorňující směr osy *x* má první bod v místě [0, 0, 0], další bod je [*size*, 0, 0], následuje [0.95 * *size*, 0.05 * *size*, 0], poté se opakuje bod [*size*, 0, 0] a poslední bod má hodnotu [0.95 * *size*, -0.05 * *size*, 0], kde *size* je předaná velikost os.

Jednotlivé osy se obarvují nastavením jejich vlastnosti *color* na objekt třídy *BABYLON.Color3*, jehož konstruktoru byly předány hodnoty RGB v rozsahu od nuly do jedné.

Pro vyobrazení popisku osy jsem použila funkci, kterou jsem pojmenovala *makeTextPlane*. Jako vstupní parametry očekává text, který se má zobrazit, barvu textu a jeho velikost. Instancí třídy *BABYLON.DynamicTexture* se vytvoří textura, nastaví se jí alfa kanál (průhlednost) a pomocí funkce této třídy *drawText* se na ni vykreslí požadovaný text danou barvou. Dále se vytvoří objekt *BABYLON.Mesh.CreatePlane* o velikosti dané třetím parametrem a ten se pokryje zhotovenou texturou. Návratová hodnota funkce *makeTextPlane* je objekt třídy *BABYLON.Mesh.CreatePlane*.



Obrázek 9: Osy *x*, *y* a *z* z různých úhlů

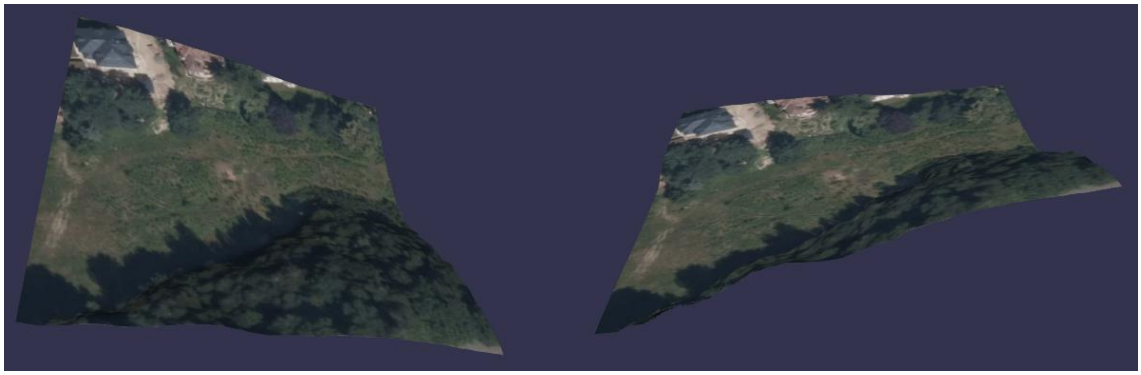
6.2.5 Země vymodelovaná podle DMR

Další část 3D mapy představuje země s povrchem ortofota zakřivená podle DMR. Pro její tvorbu jsem vybrala parametrický objekt *ribbon*, kterým lze vymodelovat téměř jakýkoli tvar. *Ribbon* ve své podstatě reprezentuje povrch mezi dvěma nebo více cestami. [12]

Nejprve si vytvoříme materiál (texturu), kterým se země pokryje, instancí třídy *BABYLON.StandardMaterial* (konstruktoru se předává jedinečný identifikátor v podobě řetězce a objekt scény). Obrázek tohoto materiálu nastavíme pomocí funkce *BABYLON.Texture.CreateFromBase64String*, jíž předáme ortofoto zakódované pomocí Base64. Materiálu ještě nastavíme barvu odrazu světla od povrchu na bílou.

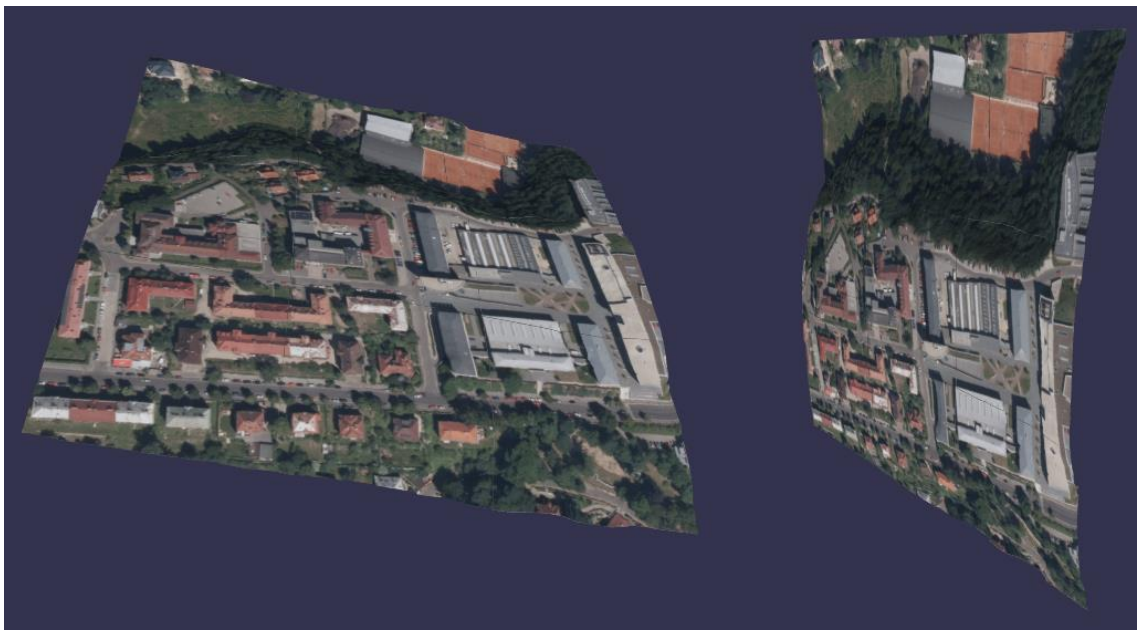
Dále potřebujeme dvourozměrné pole trojrozměrných bodů, ze kterých se bude skládat výsledný *ribbon*. Toto pole vytvoříme z mapových podkladů tak, že pro každý záznam z DMR vytvoříme trojrozměrný bod. Při výpočtu souřadnic *x* a *y* tohoto bodu vždy musíme rozdíl zeměpisné šířky/délky a proměnné *xzero*/*yzero* vydělit krokem měření, výsledek přičítáme k inkrementovaným proměnným cyklů (jedná se o dva vnořené cykly přes velikost dat DMR). Souřadnice *z* je rovna rozdílu hodnoty DMR a proměnné *zzero* vydělenému krokem měření.

Na základě těchto dat vytvoříme objekt třídy *BABYLON.Mesh.CreateRibbon*. Nastavení objektu zabraňuje jeho změně za běhu a zobrazuje jeho texturu po obou stranách, aby nebylo možné z ani jedné strany vidět skrz něj.



Obrázek 10: Země zakřivená podle DMR z různých úhlů (1 dlaždice)

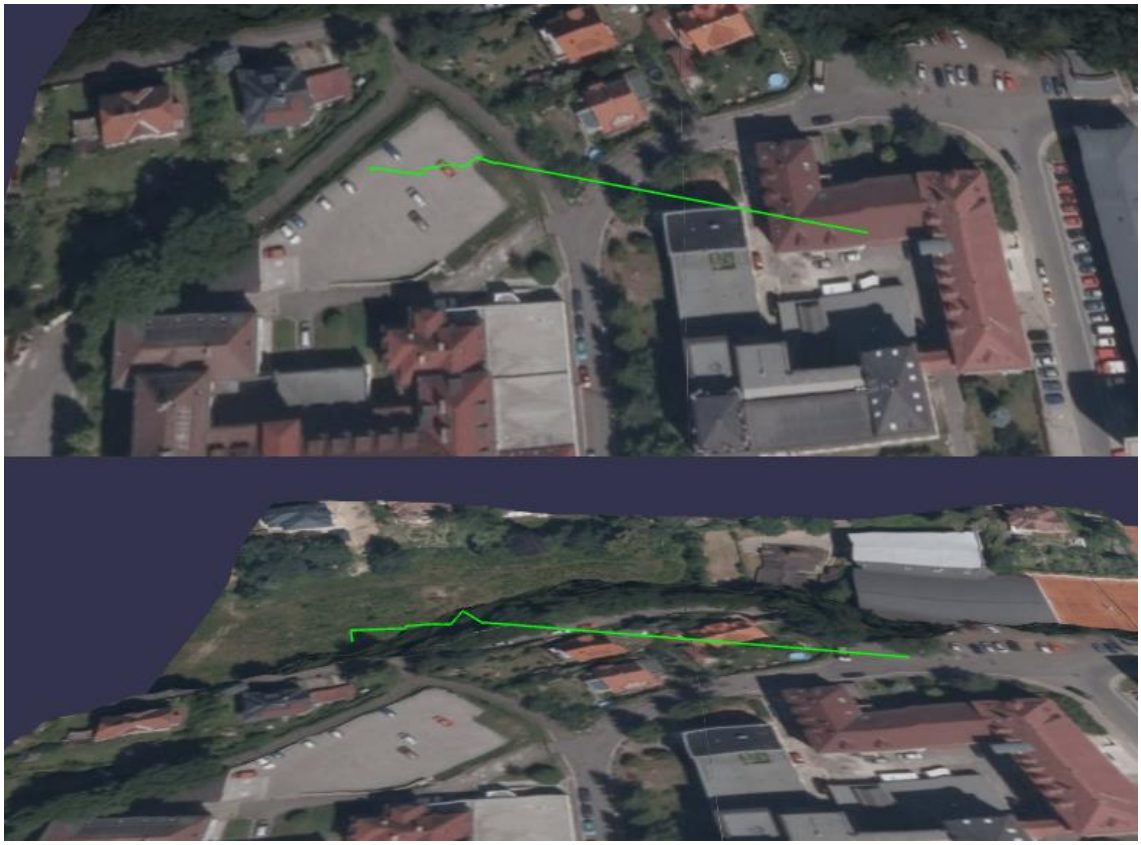
Abychom takto vykreslili více dlaždic než jednu, celou proceduru vložíme do cyklu, který iteruje přes počet dlaždic v mapových podkladech.



Obrázek 11: Země zakřivená podle DMR z různých úhlů (9 dlaždic)

6.2.6 Trasa letu/jízdy zařízení

Pro vykreslení trasy letu (popřípadě jízdy) autonomního zařízení použijeme opět funkci *BABYLON.MeshBuilder.CreateLines*, kterou jsme použili pro znázornění os. Tato funkce přebírá jako parametr pole bodů, kterými má křivka procházet. Toto pole si pojmenujeme *points* a naplníme jej v cyklu iterujícím přes délku pole *path*, které je naplněno body trasy v systému S-JTSK. Každá položka v poli *points* je trojrozměrný bod. Jednotlivé souřadnice mají hodnotu rozdílu odpovídající souřadnice v poli *path* a proměnné *xzero*/*yzero*/*zzero* vyděleného krokem měření.



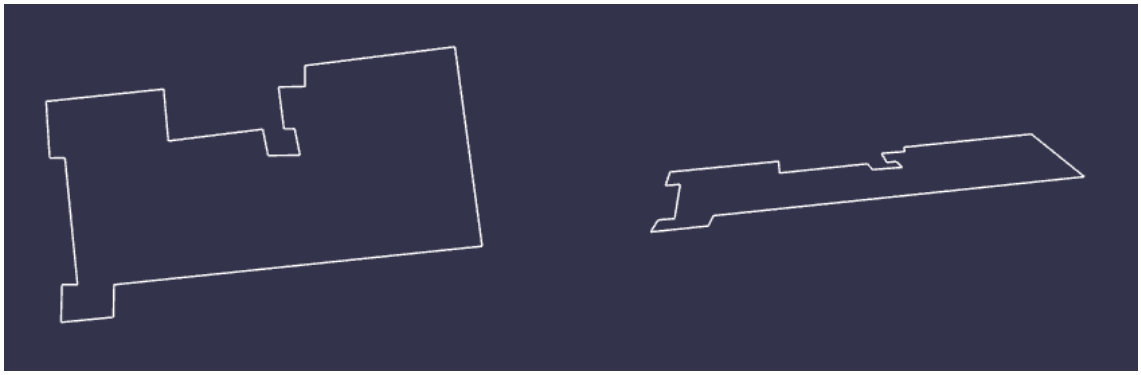
Obrázek 12: Trasa jízdy/letu zařízení zobrazená z různých úhlů

Po vytvoření křivky znázorňující trasu jízdy/letu zařízení přidáme ještě k prvnímu bodu trasy text „START“, aby bylo možné poznat, který konec křivky je počátek letu/jízdy. K tomu lze využít výše zmíněnou funkci *makeTextPlane*. Nastavíme barvu textu, rotaci a umístění pomocí vlastností *color*, *rotation* a *position*.

Obrázek 13: Trasa jízdy/letu se zobrazením startu z různých úhlů

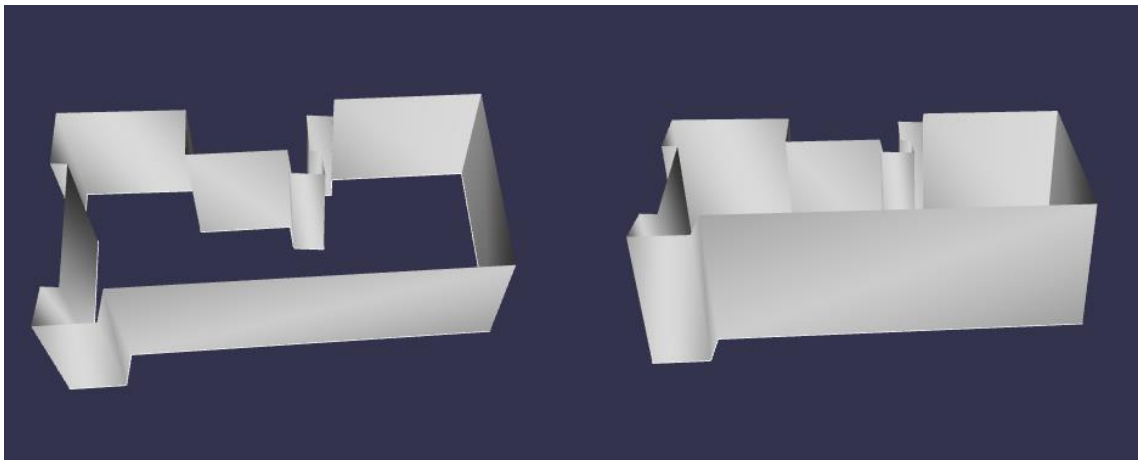
6.2.7 Modely budov

Proces vymodelování tvaru budovy jsem započala vykreslením polygonu tvořeného jejím půdorysem. K tomu jsem si vytvořila pole *shape*, které jsem naplnila trojrozměrnými body půdorysu převedenými podobně jako například body trasy jízdy/letu. Souřadnice z všech bodů v poli *shape* má hodnotu minimální naměřené výšky DMR v oblasti budovy. Polygon jsem ve scéně vykreslila pomocí funkce *BABYLON.Mesh.CreateLines*.



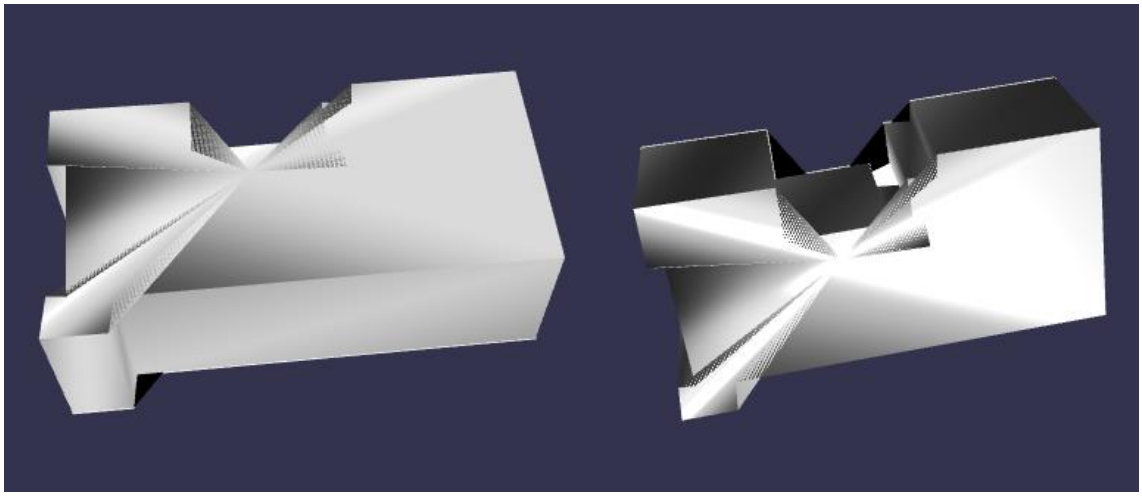
Obrázek 14: Půdorys budovy z různých úhlů

Dalším krokem bylo vytvořit stěny budovy. K tomuto účelu se hodí funkce *BABYLON.Mesh.ExtrudeShape*. Tato funkce přebírá jako jeden z parametrů pole bodů polygonu (pole shape), další z parametrů je pole bodů, které reprezentují osu „vytlačení“. Toto pole jsem si pojmenovala *path* a naplnila jsem ho body [0, 0, 0] a [0, 0, *height*], kde *height* je výška budovy (rozdíl nadmořské výšky střechy budovy a minimální hodnoty DMR v oblasti budovy). Poslední parametr, který udává, z jakých stran bude stěna viditelná, jsem nastavila na konstantu *BABYLON.Mesh.DOUBLESIDE*, aby se stěna zobrazila zevnitř i zvenčí.



Obrázek 15: Stěny budovy z různých úhlů

Objekt *ExtrudedShape* sice umožňuje vykreslit podlahu a střechu budovy nastavením parametru *cap* na hodnotu *BABYLON.Mesh.CAP_ALL*, ale pokud se v polygonu nachází nějaký konkávní úhel, objekt se ve scéně nezobrazí podle předpokladů. To je způsobeno tím, že tato funkce neprovádí teselaci daného polygonu. Teselace je proces, díky kterému lze pokrýt povrch plochých tvarů tak, aby nevznikly žádné přesahy ani mezery [13]. Toho lze dosáhnout tzv. triangulací, tedy rozdělením tvaru na trojúhelníky. Parametr *cap* tedy nastavíme na konstantu *BABYLON.Mesh.NO_CAP*, aby se tímto způsobem podlaha a střecha nevykreslovaly.



Obrázek 16: Model budovy vytvořený bez teselace z různých úhlů

Protože framework Babylon.js nedisponuje algoritmem pro triangulaci konkávních polygonů, hledala jsem externí knihovnu v JavaScriptu, která by tuto úlohu provedla. Knihovna *poly2tri* přesně odpovídá tomuto požadavku. [14]

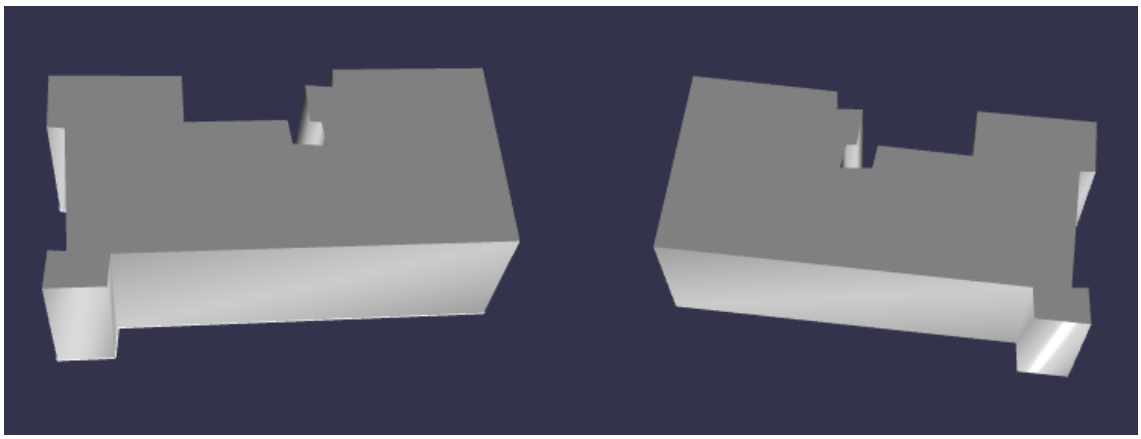
Body v poli, ze kterého se budou polygony skládat, musí být jedinečné (resp. kombinace souřadnic *x* a *y* musí být v poli jedinečná). Proto jsem v souboru *mylib.js* ve složce *static/js* implementovala funkci *removeDuplicates*, která očekává jeden parametr – pole dvourozměrných nebo trojrozměrných bodů, se kterým má pracovat. Tato funkce vytvoří prázdné pole *arr*. Dále prochází předané pole a pro každý bod kontroluje, zda se jeho kombinace souřadnic *x* a *y* již nenachází v poli *arr*. Pokud ne, tento bod vloží do pole *arr*. Nakonec vrátí pole *arr*.

Další funkce, která je implementována v modulu *mylib.js*, se nazývá *tessellate*. Slouží k vytvoření pole indexů bodů, které budou po trojicích tvořit trojúhelníky, ze kterých se daný polygon bude skládat. Jako parametr přebírá pole jedinečných dvourozměrných nebo trojrozměrných bodů *vList*. V této funkci si nejprve vytvoříme pole *contours*. Toto pole naplníme tak, aby každá jeho položka byla objekt odpovídající jedné položce v poli *vList* – vlastnost *x* tohoto objektu má hodnotu souřadnice *x* bodu z pole *vList*, vlastnost *y* je rovna souřadnici *y* tohoto bodu a vlastnost *indice* je pořadové číslo (index) tohoto bodu. Dále si vytvoříme prázdné pole *indices*, které bude sloužit nakonec jako návratová hodnota. Nyní vytvoříme objekt třídy *poly2tri.SweepContext*, konstruktoru předáme proměnnou *contours*. Objekt si uložíme do proměnné *swctx* a následně zavoláme jeho funkci *triangulate*. Tím se polygon rozdělí na trojúhelníky. Jednotlivé trojúhelníky zís-

káme pomocí metody *swctx.getTriangles* a index každého bodu každého trojúhelníku postupně vložíme do pole *indices*, které nakonec vrátíme.

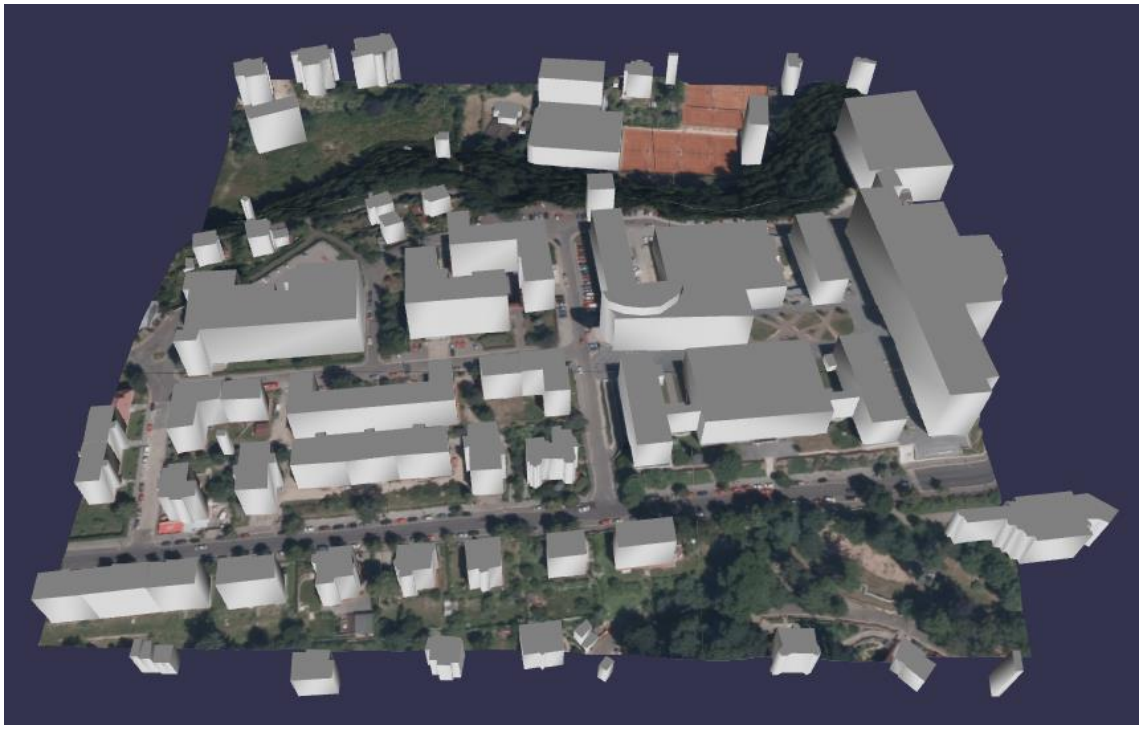
Pro vykreslení teselovaných polygonů si nejprve připravíme data. Pomocí zmiňované funkce *removeDuplicates* učiníme body v poli *shape* unikátní. Vytvoříme si pole indexů *indices* zavoláním funkce *tessellate* s parametrem *shape*. Dále vytvoříme pole pozic *positions*. Toto pole naplníme tak, že pro každý bod z pole *shape* do něj vložíme jeho souřadnici *x*, souřadnici *y* a nulu.

Abychom mohli vykreslit dva polygony (jeden pro podlahu a druhý pro střechu budovy), potřebujeme dvě instance třídy *BABYLON.VertexData*. Tato data budeme poté aplikovat na samotné polygony. Nastavíme jim vlastnosti *indices* a *positions* na vytvořená pole (u *vertexDataBack* musíme pole *indices* obrátit metodou *indices.reverse*). Nakonec vytvoříme dva objekty třídy *BABYLON.Mesh* (samotné polygony – jeden pro podlahu a druhý pro střechu) a objekty typu *VertexData* na ně aplikujeme. Polygony přesuneme do příslušné výšky a nastavíme jejich texturu na standardní materiál.



Obrázek 17: Model budovy vytvořený s teselací z různých úhlů

Pokud tento proces opakujeme pro každou budovu, která se nachází v mapových podkladech, získáme pohled na všechny budovy se správně vykreslenými střechami i podlahami. Vykreslování podlah není nutné, takže pokud se objeví problémy s výkonem, je možné vykreslovat pouze střechy budov. Nakonec můžeme smazat vykreslování obrysu polygonu pomocí *BABYLON.Mesh.CreateLines*, které jsme implementovali pro účely ladění.



Obrázek 18: Teselované modely budov na zemi zakřivené podle DMR

6.3 Snímky z kamery

Po kliknutí na tlačítko pro výběr kamery se uživateli v levé části webové stránky zobrazí nabídka lokací, při jejichž oznámení byl kamerou pořízen snímek a odeslán spolu s daty o poloze. Tato nabídka má formát odkazů. Kliknutí na tyto odkazy vyvolá provedení skriptu, který volá funkci *viewPhoto* a předává jí název souboru s vybranou fotografií složený z čísla fotografie (resp. čísla lokace) a přípony obrázku.

Funkce *viewPhoto* je uložena v souboru *mylib.js*, který se nachází v adresáři *static/js*. Nejprve schová odstavec „Vyberte lokaci ze seznamu…“ nastavením jeho vlastnosti *visibility* na hodnotu *hidden*. Dále nastaví atribut *src* obrázku, který se nachází v prostoru pro náhled snímku z kamery, na řetězec */image/* spojený s předaným názvem souboru s obrázkem. Stejnou hodnotu nastaví i atributu *href* odkazu, ve kterém se tento element *img* nachází. Na serveru se obrázek přečte z databáze a jeho instance se vrátí v odpovědi tak, jako by se jednalo o soubor přímo uložený na serveru v adresáři *image*. Tím je způsobeno, že se v pravé části stránky zobrazí náhled vybraného obrázku.

Pokud uživatel na náhled klikne, zobrazí se mu obrázek v plné velikosti v novém panelu.

Při výběru jiného autonomního zařízení se nabídka lokací s fotografií obnoví. Pokud se v databázi nenachází žádné snímky z kamery vybraného zařízení, uživatel uvidí na tomto místě oznámení o tom, že zařízení zatím žádné fotografie nepořídilo.

7. Testování aplikace

7.1 Testování API pro komunikaci s autonomním zařízením

Vytvořené API pro komunikaci s autonomními zařízeními jsem testovala pomocí knihovny Pythonu *unittest*. Třída *TestDroneRequests*, která se nachází v modulu *tests* v kořenovém adresáři aplikace, je potomkem třídy *TestCase* z knihovny *unittest*. Její metody představují jednotlivé testovací případy.

První část testů se věnuje požadavku, který slouží k oznámení o startu jízdy/letu autonomního zařízení. Pomocí knihovny *requests* odesílají HTTP požadavek a kontrolují jeho výsledek (status nebo tělo odpovědi). Nejprve se testuje modelový příklad volání tohoto požadavku, který by měl skončit úspěšně. Dále se ověřuje, že požadavek skončí chybou číslo 405, pokud chybí některý jeho parametr. Obdobně by odpověď na požadavek měla mít status 405 v případě, že chybí některý z povinných parametrů v samotných datech (například chybí identifikační číslo zařízení). Další test se týká nesprávně vygenerovaného otisku dat a neveřejného klíče pomocí hašovacího algoritmu SHA1 – v takové situaci by měla nastat chyba s kódem 403. V neposlední řadě se testuje volání požadavku s nevalidními hodnotami (například datum a čas startu jízdy/letu neobsahuje jednotku vteřin, zařízení s daným identifikačním číslem se nevyskytuje v databázi). Ověřuje se také, že nelze zapsat start zařízení v případě, že záznam o odstartování tohoto zařízení v tuto dobu se již v databázi jednou nachází. V poslední řadě je zajištěno, že pokud nelze dekódovat předaná data, která mají být ve formátu JSON, kód statutu odpovědi nabývá hodnoty 403.

Druhá část se zabývá HTTP požadavkem pro oznámení polohy autonomního zařízení. Skript nejprve vyzkouší dvojí volání tohoto požadavku, která by měla skončit úspěšně. Data prvního požadavku obsahují i snímek z kamery, data druhého požadavku jej neobsahují. Při chybějících parametrech požadavku a při chybějících povinných položkách v datech typu JSON očekáváme odpovědi se statutem 405 a 403. Stejně jako předchozí požadavek je i tento otestován pro případ, že neodpovídá otisk hašovací funkcí. Dále se testuje příjem nevalidních dat (například aktuální výška zařízení nemá formát čísla) v takovém případě očekáváme chybu s kódem 403. V posledním testu

odesílá skript chybně zakódovaná data, která by měla mít formát JSON a kód statutu odpovědi porovnává s číslem 403.

Tento skript byl spuštěn několikrát v průběhu programování API pro komunikaci s autonomními zařízeními a jeho výsledky zároveň posloužily jako testovací data pro ladění samotné webové aplikace.

7.2 Testování webové aplikace

Pro ladění webové aplikace jsem si vytvořila testovací verzi aplikace na jiném serveru, která za cenu stability umožňovala mimo jiné získat přehled nad tím, co přesně se na serveru děje (například jaké URL jsou volány). Pro tyto účely jsem si na testovacím serveru zřídila i databázi s testovacími daty. Po úspěšném naprogramování a vyladění celé aplikace jsem zdrojové kódy přesunula na server určený pro ostrý provoz a nakonfigurovala její spouštění. Ostrá verze aplikace se nachází na adrese http://drone.fm.tul.cz/.

Webová aplikace, i když v procesu testování, posloužila i jako pomůcka pro ladění serverové komponenty, která poskytuje mapové podklady. Díky 3D vizualizaci mapy bylo možné detekovat problémy a chyby v generování podkladů účinněji a pohodlněji než čtením objektů JSON nebo XML, protože tyto formáty jsou určeny pro strojové zpracování a pro člověka jsou těžko čitelné.

8. Závěr

Hlavním cílem práce bylo navrhnout a implementovat webovou aplikaci pro monitorování a tracking autonomních zařízení typu auto a dron včetně zobrazení snímků z jejich kamer. K tomu bylo nutné navrhnout způsob komunikace s těmito zařízeními. Pro přenos dat jsem připravila aplikační rozhraní ve formě HTTP(S) požadavků, které bude zařízení odesílat serveru. Pro data tohoto přenosu jsem zvolila formát JSON, protože je kompaktní a snadno rozšiřitelný. Fotografie se pro přenos kódují do Base64.

Komunikace s autonomními zařízeními je kromě šifrovacího protokolu SSL zabezpečena také použitím hašovacího algoritmu SHA1. Pomocí tohoto algoritmu se vypočítá otisk (hash) přenášené zprávy spojené s neveřejným klíčem, který zná každé zařízení a webová aplikace, a odešle se spolu se zprávou. Na straně serveru se pak kontroluje správnost tohoto otisku. Získaná data se na serveru ukládají do relační databáze.

Trasy jízd/letů jednotlivých zařízení lze poté ve webové aplikaci zobrazit v interaktivní 3D mapě, která znázorňuje blízké okolí posledního bodu trasy. Mapové podklady se získávají z externí webové služby. Druhou částí webové aplikace je zobrazení snímků z kamer jednotlivých zařízení. Na straně serveru jsem pro implementaci použila programovací jazyk Python, na straně klienta pak běží JavaScript spolu s knihovnou jQuery a o 3D grafiku se v prohlížeči stará framework Babylon.js.

Při vykreslování modelů budov v mapě jsem narazila na problém absence algoritmu pro teselaci ve vybraném frameworku pro 3D grafiku. Použila jsem tedy jinou knihovnu JavaScriptu, která provádí teselaci pomocí triangulace polygonů. Tím jsem problém vyřešila, modely budov nyní odpovídají skutečnosti.

Výsledná aplikace splňuje požadavky, které na ni byly kladeny. Přesto může být různými způsoby rozšířena. Při implementaci API pro komunikaci se zařízeními jsem kladla důraz na snadnou rozšiřitelnost kódu, aby bylo možné k přenášeným parametrům přidat další položky. V samotné webové aplikaci je možné přidat uživateli možnost zobrazit i starší lety/jízdy zařízení stejně jako starší fotografie. Dále se nabízí otázka přenosu videa z kamer zařízení a jeho přehrávání v aplikaci.

Aby byl web uživatelsky přívětivější, do mapy by se mohly zakreslovat například názvy ulic a jiné zeměpisné údaje, případně by se kromě budov mohly v mapě modelovat i jiné entity.

POUŽITÁ LITERATURA A ZDROJE

[1] Kozánek, Jan. *Serverová komponenta pro aktualizaci modelu autonomního navigačního systému*. Liberec, 2017. Diplomová práce. Technická univerzita v Liberci, Fakulta mechatroniky, informatiky a mezioborových studií. Vedoucí práce Igor Kopetschke.

[2] Melown – The 3D map developer's library. *Melown* [online]. 2016 [cit. 2017-05-03]. Dostupné z: https://www.melown.com

[3] Kopetschke, I., P. Kretschmer, J. Novák. *Data Preprocessing for Autonomous Navigation Systems*. International Conference on Military Technologies, Brno, 2017, ISBN 978-1-5386-1988-9.

[4] MariaDB.org – Ensuring continuity and open collaboration. *MariaDB.org* [online]. 2017 [cit. 2017-05-03]. Dostupné z: https://mariadb.org/

[5] Janků, Dominik. Ideální webový framework pro Python? *Root.cz* [online]. 2015 [cit. 2017-05-03]. Dostupné z: https://blog.root.cz/djanku/idealni-webovy-framework-pro-python/

[6] Dustman, Andy. MySQLdb User's Guide. *SourceForge* [online]. [cit. 2017-05-03]. Dostupné z: http://mysql-python.sourceforge.net/MySQLdb.html

[7] Shahriar, Tamim. Prevent sql injection in python using cursor.execute correctly. *Life is short – you need Python!* [online]. 2010 [cit. 2017-05-03]. Dostupné z: http://love-python.blogspot.cz/2010/08/prevent-sql-injection-in-python-using.html

[8] Bootstrap – The world's most popular mobile-first and responsive front-end framework. *Bootstrap* [online]. 2011 [cit. 2017-05-03]. Dostupné z: http://getbootstrap.com/

[9] jQuery – write less, do more. *jQuery* [online]. 2017 [cit. 2017-05-03]. Dostupné z: https://jquery.com/

[10] Žára, Ondřej. Vytváříme Hello World pro WebGL. *Zdroják – o tvorbě webových stránek a aplikací* [online]. 2013 [cit. 2017-05-03]. Dostupné z: https://www.zdrojak.cz/clanky/vytvarime-hello-world-pro-webgl/

[11] Hewitson, Joe. Three.js and Babylon.js: a Comparison of WebGL Frameworks. *SitePoint – Learn HTML, CSS, JavaScript, PHP, Ruby & Responsive Design* [online]. 2013 [cit. 2017-05-03]. Dostupné z: https://www.sitepoint.com/three-js-babylon-js-comparison-webgl-frameworks/

[12] Parametric Shapes. *BabylonJS Documentation* [online]. [cit. 2017-05-03]. Dostupné z: http://doc.babylonjs.com/tutorials/parametric_shapes

[13] Tesselation. *Math is Fun – Maths Resources* [online]. 2014 [cit. 2017-05-03]. Dostupné z: https://www.mathisfun.com/geometry/tessellation.html

[14] poly2tri.js. *GitHub – The world's leading software development platform* [online]. 2017 [cit. 2017-05-03]. Dostupné z: https://github.com/r3mi/poly2tri.js