# Úvod

* Jak se aplikace jmenuje a k čemu slouží.

**Z. 12 – Lutz,Python**

**Z.3-Flanagan,JS**

**Z.4-Elliott,JS**

**Z. 11 – Brown,JS**

**Z. 10 – Mardan,JS**

Z. 14 – Mueller,Sec

Z. 15 – Hoffman, Sec

**Z.5-Northwood,FS**

**Z.6-Mendez,Dev**

Z.9-Hunter,DEV

Z. 13 - SQL

Z. 7-Matthew Postgre

Z. 8-Juba Postgre

**Z.1-Coronel,MorrisDB**

**Z.2-Kroenke,DB**

# Vývoj webové aplikace

Autoři uvádějí že cílem navrhování architektury systému je navrhnout nejefektivnější systém pro řešení určitého problému s tím, aby stále zůstal otevřen případným budoucím změnám. V protikladu vůči sobě stojí monolitické systémy, ve kterých jedna aplikace dělá vše, a systém (?) mikroslužeb, ve kterých více menších aplikací? spolupracuje k cíli. I proto, že je změna monolitické architektury složitější, je dnes dávána přednost architektuře mikroslužeb, i přesto, že provázání jednotlivých komponent stojí více práce. Návrh architektury začíná identifikací jádrových funkcí, které má systém plnit, a akcemi, které by uživatel mohl činit, aby funkce využil. Na tomto základě jsou navrhnuty jednotlivé komponenty a způsob komunikace mezi nimi. Většina komunikace pak funguje na principu požadavek-odpověd (request-response) nebo asynchronních řad (asynchronous/queue). Kromě samotných funkcí systémů hraje roli i bezpečnost, efektivita (perfomrance) a odolnost. Nakonec musí být každý návrh realizovatelný za použití vhodných nástrojů jako jsou databáze.

Většina webových aplikací vychází z Mode-View-Controller architektury. ve kterém Controller slouží jako prostředník mezi modelem, ze kterého získává nebo do kterého vkládá data, a pohledem, kterému předává data k zobrazení. Controller zároveň slouží pro komunikaci s uživatele – zádávání požadavků.

Zdroj 5 – str. 77 poslední odstavec Popisuje HTTP GET a REST; jinde se hodně mluví o modulech, CRUD, SOLID, Asyncgronous-quee a request response

Současný vývoj nejen webových aplikací klade stále větší důraz na UX (z anglického user experience) neboli uživatelský prožitek.

Auto ve své knize ZDROJ 6 uvádí základní elementy webových serverů: hardware, operační systém a HTTP server. Operační systém a HTTP server, které tvoří softwarovou část webového serveru bývají ve většině případů doplněny o další dva elementy: databáze a skriptovací jazyk.

Dynamické webové stránky využívají buďto skriptování obsahu na straně serveru, kdy je webová stránka vytvořena nejdříve na serveru a poté poslána uživateli, nebo skriptování na straně klienta, kdy je stránka generována podle zaslaného kódu přímo na zařízení uživatele. Využívání jazyků pro skriptování na straně serveru (server-side languages) jako je PHP, přináší bezpečnostní výhody jako skrytý kód nebo výkonnostní výhody, neboť je zatížen server, nikoliv zařízení uživatele. Oproti tomu jazyky pro skriptování na straně klienta jako je JavaScript, využívají pro zpracovávání kódu webový prohlížeč a nepotřebují tedy explicitní webový server a výkonost je vynahrazena nižší potřebou komunikace a přenosem dat mezi zařízením a serverm. Jazyky pro skriptování na straně klienta jsou také užitečné pro vývoj webových aplikací, neboť mohou běžet na prohlížeči pouze lokálně bez potřeby internetu.

API, z anglického application programming interface neboli rozhraní pro programování aplikací slouží ke komunikaci s daty a funkcionalitami jiného aplikace. API poskytuje bezpečný přístup k datům jiné aplikace aniž by byl potřeba přístup k jejímu zdrojovému kódu nebo databázi např. mapám. To je zajištěno využívání předem určených možností. Uživatelům API poskytuje mantinely pro vytváření vlastních systémů, jak chtějí

Z9 – Ch.1, s. 74,75

Z.4 – s. 4,5,6,7,8 – Vývoj JS appky

Microservices – Z.14 – s. 159,160

CSS – Z. 3 s.413

HTML – Z5 – str.103

API – Z5 – str.227

**Zdroje N**

**Zdroj 10 – Basics – HTML, CSS, HTTP Requests and Responses, RESTful API**

the back end is another name for the server. It’s everything after the browser. It includes server platforms like PHP, Python, Java, Ruby, and of course Node.js, as well as databases and other technologies

# Programovací jazyk JavaScript

Aplikace, která byla v rámci bakalářské práce vytvořena, je až na jednu funkcionalitu napsána v jazyce JavaScript, je proto vhodné popsat tento jazyk z akademicko-teoretického hlediska.

JavaScript je vysokoúrovňový interpretovaný jazyk s dynamickou typovou kontrolou vhodný jak pro objektově orientované, tak pro funkcionální programování (i zde). Místo tradiční třídové dědičnosti používá JavaScript dědičnost prototypovou. I zde. V jazyce JavaScript ve skutečnosti třídy neexistují, objekty dědí z objektů. Prim hrají v jazyce JavaScript funkce. Funkce může být v jazyce JavaScript využita jak jako proměnná, tak jako objekt, a může být například předána jako argument do jiné funkce.

Jazyk JavaScript byl vytvořen v roce 1996. Jak zmiňuje AUTOR, syntaxe jazyka JavaScript vychází ze syntaxe jazyka Java, jedná se ale o plně samostatný jazyk k jehož používání není znalost jazyka Java nutná. AUTOR dokonce uvádí, že dnes má jazyk JavaScript s jazykem Java společného velmi málo, a jeho pojmenování byl v podstatě marketingový tah využívající zvyšující se popularitu jazyku Java v dané době. Jazyk JavaScript začínal, jak již název napovídá, jako skriptovací jazyk, nyní se ale jedná o robustní a efektivní univerzální (obecný, general-purpose) jazyk. Současný JavaScript je v různých verzích standardizován organizací ECMA (European Computer Manufacturers Association, neboli Evropská asociace výrobců počítačů) pod názvem ECMAScript. Jazyk JavaScript je jednoduchý k naučení a dle AUTORA je i nejvhodnějším vstupním jazykem do světa programování. Tento názor podporuje i fakt, že JavaScript nevyžaduje instalaci žádných komplexních vývojových prostředí.

AUTOR také označil JavaScript za nejdůležitější programovací jazyk. Společně se značkovacím jazykem HTML (z anglického Hypertext Markup Language), který specifikuje obsah webových stránek, a jazykem CSS (z anglického Cascading Style Sheets tj. Kaskádové styly), který specifikuje jejich prezentaci, tvoří JavaScript, specifikující jejich chování, základ vývoje webových stránek. Drtivá většina moderních webových stránek používá JavaScript a všechny moderní webové prohlížeče obsahují jeho interpret. V moderních prohlížečích je většina JavaScriptového kódu kompilována a vysoce optimalizovaná, a rychlost běhového prostředí se proto velmi blíží kódu napsanému v C nebo C++. Ve webovém prohlížeči používá JavaScript pro komunikaci s uživatelským prostředím DOM (Document Object Model neboli objektový model dokumentu) API.

JavaScript byl nejdříve používán pouze na straně klienta protože běžel v prohlížeči příliš pomalu. Vydání vykreslovacího jádra prohlížeče (web browser engine) V8 společností Google JavaScript zrychlilo a dnes je JavaScript využívám jak na straně klienta, tak na straně serveru, a je jím možné vyvíjet i desktopové aplikace, či například softwarovou podporu dronů nebo Internetu věcí (IoT, Internet of Things). Využívání jednoho jazyka jak pro programování frontendu, tak pro programování backendu, urychluje vývoj a zjednodušuje práci vývojářů. Tento vývoj od možnosti používat stejný jazyk jak na straně klienta, tak na straně serveru, pokračoval možností sdílení kódu mezi klientem a serverem, což představilo nový styl aplikace nazývaný izomorfní či univerzální, u které nezáleží zda běží na straně klienta či serveru. Dnes je JavaScript na straně klienta všudypřítomný a dominantní, ale díky Node.js je použitelný i na straně serveru.

JavaScript je asynchronní a využívá tedy pouze jedno vlákno, přičemž I/O operace probíhají na pozadí.

N ode.js is an open source, event-driven asynchronous I/O technology for building scalable and efficient web servers.

Node.js is built to support real-time applications by utilizing web sockets.

Express je minimalistický a flexibilní webový framework.

Node.js nabízí databázová rozhraní pro největší relační databázové systémy, včetně PostgreSQL, a dle AUTORA revitalizoval koncept NoSQL databází.

Moduly jsou znovupoužitelné softwarové komponenty.

Autor dokonce navrhuje čtenářům, aby si moduly představili jako malé a nezávislé aplikace, které jsou plně funkční a testovatelné.

Zároveň je dobrou praxí, aby moduly byly co nejmenší a nejjednodušší a plnily pouze jednu specifickou funkci. Není také doporučováno, aby spolu jednotlivé moduly přímo interagovali.

Jedna z největších výhod používání modulů tkví v jejich samostatnosti; změna v kódu neovlivní samotnou funkci modulu a jednotlivé moduly jsou znovupoužitelné i v jiných aplikacích.

**Zdroj N1**

# Programovací jazyk Python

Jak již bylo zmíněno v předchozí kapitole, jedna z funkcionalit naší webové aplikace není napsána v jazyce JavaScript. Tato funkcionalita, jež je podrobně popsána především v kapitole pdfGenerator.py je napsána v jazyce Python, je proto vhodné k tomuto jazyku zmínit pár slov.

Jazyk python je univerzální (obecný, general-purpose) programovací jazyk, jež je často také definován jako objektově orientovaný skriptovací jazyk (object-oriented scripting langauge). Univerzální v tomto popisu značí, že jej lze použít na mnoho různých činností od tvorby webových aplikací, přes tvorbu počítačových her až po robotiku. Tento fakt je v silném kontrastu k jeho označení za jazyk skriptovací. Jazyk python však není za skriptovací jazyk označován proto, že by patřil mezi tradiční skriptovací jazyky jako je Perl nebo Tcl, ale spíše proto, že se jedná o jednoduchý jazyk vhodný pro rychlé programování malých úkolů. Zároveň je jednoduše škálovatelný a i díky jeho silné podpoře objektově orientovaného programování, která zajišťuje např. opakované používání již napsaného kódu, vhodný i pro vývoj robustních aplikací. Jazyk python zároveň podporuje i procedurální programování.

Jazyk python byl navrhnut tak, aby byla jeho syntax jednoduchá, čitelná a úsporná, což z něj dělá jazyk příjemný pro vývojáře. Pro svou jednoduchost bývá python občas označován za spustitelný pseudokód. Díky těmto vlastnostem je python optimalizovaný pro rychlý vývoj, neboť zvyšují produktivitu vývojáře. Rychlost vývoje umocňuje i fakt, že je python jazyk interpretovaný, a není tedy třeba jej kompilovat, což je zároveň i jeho největší slabinou, neboť je pomalejší než kompilované jazyky jako jsou C a C++.

Kód ve skutečnosti kompilovaný je, ale pouze do mezi-formátu bajtkód, nikoliv přímo do strojového kódu, který právě zajišťuje nejen jeho interpretovanou povahu, ale také jeho multiplatformnost. Samotný python je napsán v ANSI C a kód je tedy obvykle ve stejné formě spustitelný na různých zařízeních (např. jak na operačním systému Linuxu, tak na operačním systému Windows)

Jazyk python je vyvíjen jako software s otevřeným zdrojovým kódem, využívá podporu robustní kolekce knihoven a je jednoduše kombinovatelný s jinými programovacími jazyky. Jazyk python dále nabízí například dynamickou typovou kontrolu a automatickou správu paměti. Tvůrcem jazyka je Guido van Rossum.

# Databázové systémy – 3 knihy

* Jak se dělí

Coronel, Morris a Rob ve svém shrnutí k první kapitole knihy NÁZEV připomínají, že znalost přesných, relevantních a včasných dat je klíčová pro správné rozhodování, které je klíčové k přežití. Data definují jako surová („raw“) fakta a informace jako výsledek zpracování těchto dat k objasnění jejich významu. Data jsou většinou ukládána v databázi a pro její implementaci a správu jejího obsahu potřebuje uživatel systém řízení báze dat (zkráceně DBMS z anglického Database Management System). Kroenke a Auer ve své knize Database Concepts přidávají do naznačeného modelu – uživatel, DBMS a databáze – ještě čtvrtou komponentu, databázovou aplikaci. Ta představuje spojnici, která zprostředkovává komunikaci mezi DBMS a uživatelem a čte a mění databázi na základě SQL dotazů odesílaných do DBMS. Databázovou aplikaci tvoří jeden nebo i více programů. Systém řízení báze dat je počítačový program, který slouží k vytváření, zpracovávání a řízení jednotlivých databází pomocí jazyka SQL. Databáze je pak definována jako kolekce provázaných tabulek a dalších struktur. Uživatel je ten, kdo databáze využívá a čte z nich, vkládá do nich a prochází v nich data.

Autoři dělí databázové systémy do tří modelů. V hierarchickém modelu jsou data kupena do postupně větších a větších celků a síťový model využíval ukazatele s odkazy na opakující se data. atd.

It’s not always helpful to define something as what it’s not, so we’ll add that these NoSQL databases might be more properly called “document databases” or “key/value pair databases.” They provide a conceptually simpler approach to data storage. There are many, but MongoDB is one of the frontrunners, and the one we will be using in this book.

Na trhu dnes převládají

<https://www.pdfdrive.com/beginning-database-design-e47252446.html>

Z1 – 43s., 409

Z2 – 11-18, 30

Typology - **Z5 – s.251-255**

**Typology - Z6 – str.211-216**

**DBMS – Z7 – s. 1-11**

**Z8 – 1-14**

# PostgreSQL – 2 knihy

* Co to je
* Proč bylo vybráno

PostgreSQL je

Z7 – 11-15

Z8 – 31-38, 38-43, 50-58

**Z9 – str 89-94**

**Z13 – str.1-7**

**Zdroj 8**

<https://www.pdfdrive.com/beginning-databases-with-postgresql-e28132477.html>

<https://welib.org/md5/f1c79c111fab143f26da8141efe916b2>

<https://www.pdfdrive.com/postgresql-up-and-running-a-practical-guide-to-the-advanced-open-source-database-e158212825.html>

<https://welib.org/md5/cc8aba6d90b184135d4225ab00722ff2>

<https://www.pdfdrive.com/postgresql-for-data-architects-discover-how-to-design-develop-and-maintain-your-database-application-effectively-with-postgresql-e168411699.html>

# Jazyk SQL – 2 knihy

* Co to je

V pozdějších edicích uvádějí AUTOŘI jazyk SQL jako pátý komponent databázového systému, neboť se jedná o mezinárodně uznávaný jazykový standard pro práci s databázemi.

Z13 – s. 7-12

Z8 – s.144-149

S2 – 47-49, 115, 181

Z1 – 221-223, 276, 363

# Základy bezpečnosti – 2 kniha

* Vypsat používané základy

Z.14 – str. 2-7, 14-17

Z.15 – str. 187-192

Libraries - Z.14 – str.117-125

HTTPS cert. – Z. 11 – s.207-2015

**Z6 – 249-257**

Z5 – 267-284?

# Architektura aplikace

Aplikace nese funkční název *heic-metadata-extractor*, což je pozůstatek z první fáze jejího vývoje. Je implementována v jazyce JavaScript a využívá jeden podpůrný skript napsaný v jazyce Python. Aplikace dále využívá objektově-relační databázový systém PostgreSQL.

V kořenovém adresáři aplikace se nachází celkem devět souborů a pět dalších adresářů (podsložek). Vstupním bodem aplikace je soubor *server.js*, který plní roli serveru. Hlavní funkcionality aplikace jsou dále zajištěny třemi soubory v jazyce JavaScript – *uploadData.js*, *convert.js* a *queries.js* – které jsou společně s podpůrným skriptem *pdfGenerator.py*, napsaným v jazyce Python, a souborem *schema.sql*, který slouží k případnému automatickému vytvoření databázové struktury, podrobně popsány v jednotlivých podkapitolách. Sedmým souborem je soubor *.env*, který slouží k uložení citlivých údajů v podobě systémových proměnných. Automaticky generovaný soubor *package.json* obsahuje základní informace o aplikaci a definuje seznam závislostí (knihoven), které aplikace potřebuje. Soubor umožňuje jednoduchou instalaci všech závislostí příkazem *npm install*. Robustnější, rovněž automaticky generovaný, soubor *package-lock.json* zajišťuje, aby se aplikace vždy instalovala se stejnými verzemi všech balíčků, čímž garantuje konzistentní prostředí.

Balíčky a knihovny uvedené v souboru *package.json* jsou při instalaci staženy do adresáře *node\_modules*, který se automaticky vytvoří při spuštění příkazu *npm install*. Jakýkoliv ruční zásah do tohoto adresáře je nežádoucí. Adresář *public* obsahuje šest HTML souborů, pět ikon ve formátu SVG a jeden obrázek ve formátu PNG, které společně tvoří webové uživatelské rozhraní aplikace. Tyto soubory jsou podrobně popsány v kapitole Adresář public. Adresář *uploads*, který se v případě potřeby vytvoří při nahrávání dat do aplikace automaticky, slouží k ukládání nahrávaných fotografií. Složka *fonts* obsahuje dva soubory ve formátu TrueType font (.ttf), které definují běžné a tučné písmo používané při generování PDF souborů z databáze aplikace. Pro tento účel je důležitý také poslední adresář *mapy*, do něhož může administrátor prostřednictvím webového rozhraní ukládat mapy s místy nálezů čtyřlístků. Vygenerovaný PDF soubor *records\_report.pdf* je následně uložen také do kořenového adresáře aplikace.

# Schéma databáze

Schéma databáze je definováno v souboru *schema.sql* a slouží k případnému automatickému vytvoření databázové struktury. Jak již bylo uvedeno v kapitole EEE, pro potřeby naší aplikace byla zvolena relační databáze PostgreSQL.

Po vytvoření samotné databáze a navázání spojení s ní jsou vytvořeny dvě tabulky. Tabulka s názvem *records* slouží k ukládání informací o nalezených čtyřlístcích, zatímco tabulka s názvem *locations* uchovává údaje o místech, kde byly čtyřlístky nalezeny.

Tabulka *locations* slouží výhradně k osobním účelům majitele stránek. Majitel si totiž vlastním způsobem eviduje lokality, v nichž čtyřlístky nachází. V tabulce proto nalezneme čtyři sloupce. Sloupec *id* je automaticky generovaný primární klíč a zároveň identifikátor dané lokace. Sloupec *name* představuje název lokace tak, jak ji pojmenoval majitel, a je definován datovým typem *VARCHAR* o rozsahu 255 znaků. Sloupec nesmí být u žádného záznamu prázdný a pro každý záznam musí být unikátní. Neprázdný musí být i sloupec *comment*, který je definován datovým typem *TEXT* a který slouží pro ukládání komentářů k lokacím. V tomto sloupci však, na rozdíl od sloupce *name*, nemusí být pro každý záznam unikátní hodnota. Poslední sloupec *anonymized* je datového typu *BOOLEAN* s výchozí hodnotou *false* a určuje, zda má být daná lokace anonymizována. Lokace je za anonymní označena nejčastěji tehdy, pokud by její znalost návštěvníkem stránek mohla zasahovat do soukromí majitele.

Tabulka *records* je tvořena devíti sloupci. Stejně jako v případě tabulky *locations* slouží jako primární klíč automaticky generovaný identifikátor záznamu uložený ve sloupci *id.* Ve sloupci *timestamp*, příhodného datového typu *TIMESTAMP*, je uložena časová stopa pořízení fotografie, jež je automaticky zjištěna při nahrávání snímku (viz kapitolu FFF). Sloupec *timestamp* nesmí být prázdný, stejně jako další tři sloupce *longitude, latitude* a *path*. Ve sloupci *longitude* je datovým typem *DECIMAL* s přesností devíti číslic, z toho šesti za desetinnou čárkou, uvedena zeměpisná délka místa, kde byla fotografie pořízena. Ve sloupci *latitude* je pak obdobným způsobem, avšak s pochopitelně nižší přesností osmi číslic, uložena zeměpisná šířka místa. I tyto dva sloupce jsou vyplněny automaticky na základě informací uvedených ve fotografii - majitel stránek totiž používá mobilní telefon, který do EXIF metadat snímků automaticky ukládá i GPS údaje. Do sloupce *path* je automaticky uložena cesta k uložené fotografii (viz kapitolu GGG), přičemž sloupec je definován datovým typem *TEXT.* Obdobným způsobem je do sloupce *pathMiniature* uložena cesta k administrátorem poskytnuté miniatuře fotografie. Sloupec *pathMiniature* však nemusí být při nahrávání záznamu, na rozdíl od sloupce *path*, vyplněn. Sloupec *location\_id* slouží jako cizí klíč odkazující na tabulku *locations* a musí být majitelem při nahrávání souboru specifikován v jeho názvu (viz kapitolu GGG). Do sloupce *address* je automaticky uložena adresa odvozená ze souřadnic - i tento proces je podrobněji popsán v kapitole GGG. Sloupec *address* je rovněž datového typu *TEXT*. Do posledního sloupce *status* je uložen stavový příznak záznamu. Příznak musí být, stejně jako identifikátor lokace, specifikován majitelem v názvu souboru a sloupec je omezen na akceptování pouze osmi povolených hodnot – 'V', 'D', 'Z', 'N', 'L', 'G', 'S' a 'J' – jejichž význam je objasněn v kapitole GGG.

Soubor *schema.sql* je ve své celistvosti uveden v příloze EEE.

# server.js

Vstupním bodem aplikace je soubor server.js. Na jeho začátku jsou tradičně importovány potřebné moduly; konkrétně se jedná o webový aplikační framework *Express*, modul *path* pro práci s cestami v systému souborů, modul *fs* umožňující čtení a zápis souborů, klienta *pg* pro připojení k PostgreSQL databázi a práci s ní a modul *child\_process* umožňující spuštění externích procesů. Dále je importován middleware *body-parser* pro zpracování těla HTTP požadavků a middleware *cors* pro povolení Cross-Origin Resource Sharing. Z knihoven jsou dále využity *jsonwebtoken* pro tvorbu a ověřování JSON Web Tokenů, *bcrypt*, která slouží k bezpečnému hašování hesel, a *express-rate-limit*, která omezuje maximální počet pokusů o přihlášení do administrátorské části webu za určitý časový interval. Nakonec jsou importovány moduly *dotenv* pro práci s konfiguračními soubory .env a vlastní moduly *./uploadData* pro práci s pomocnými funkcemi a *./queries* obsahující SQL dotazy. Zmíněné moduly jsou případně více popsány v kapitole XXX.

Samotný kód začíná vytvořením instance Express aplikace, definováním serverového portu a nastavením konfiguračního objektu pro připojení k PostgreSQL databázi. Následně jsou definovány cesty k veřejně dostupným stránkám, které jsou uloženy v adresáři *public* a podrobněji popsány v kapitole XXX, a definována je také cesta k adresáři *uploads*, jenž slouží pro ukládání nahraných fotografií. V kódu jsou dále volány funkce *uploadHEIC* a *uploadLocation* pro nahrávání dat, které jsou popsány v kapitole YYY.

Následuje větší část kódu sloužící k získávání dat z databáze. Nejdříve je definována cesta pro získání záznamů z tabulky *locations* (viz kapitola ZZZ) a poté jsou definovány dvě velmi podobné cesty pro získání záznamu z tabulky *records* (viz kapitola ZZZ): jedna vrací konkrétní záznam podle zadaného identifikátoru (ID) a druhá vrací všechny dostupné záznamy. V obou případech jsou z tabulek navráceny všechny informace o záznamech. Údaje týkající se polohy – tedy zeměpisná šířka a délka, adresa, název lokace a komentář k lokaci – jsou v případě její anonymizace upraveny přímo v rámci SQL dotazu (místo skutečných hodnot se předají souřadnice 0.0000 nebo text typu „Adresa anonymizována“). Výsledná data jsou následně zpracována pomocnými funkcemi popsanými v kapitole AAA a ve formátu JSON předána do veřejné části webové aplikace.

Dále je v kódu definována cesta pro spuštění externího skriptu v jazyce Python s názvem pdfGenerator.py, který blíže popisuji v kapitole BBB, cesta pro získání souřadnic konkrétního záznamu dle jeho identifikátoru a cesta pro ukládání snímku mapy ke konkrétnímu záznamu. Snímek mapy je ve formátu PNG uložen pomocí modulu *fs* do speciálního adresáři *mapy* a jeho název odpovídá identifikátoru záznamu.

Následuje větší část kódu sloužící k editaci záznamů v obou databázových tabulkách prostřednictvím webového rozhraní. Pro každou tabulku jsou vytvořeny dvě API cesty - jedna pro načtení konkrétního záznamu podle jeho identifikátoru a druhá pro jeho editaci. Stejně jako v předchozích částech kódu je i zde kontrolována správnost zadaných údajů a je zajištěna správa chyb.

V kódu jsou dále volány všechny funkce spouštějící SQL dotazy pro získání statistických údajů o aktuálním stavu databáze, které jsou detailně popsány v kapitole CCC.

V poslední části kódu zajišťujeme bezpečný přístup do administrátorské části webového rozhraní. Nejprve jsou do konstant načteny citlivé údaje ze systémových proměnných – *SECRET\_KEY* pro podepisování JSON Web Tokenů a *ADMIN\_PASSWORD\_HASH*, což je hash administrátorského hesla uloženého v bezpečné podobě. Dále je nastavena podpora Cross-Origin Resource Sharing (CORS) pro zpracování požadavků a body-parser pro dekódování dat odesílaných prostřednictvím HTTP formulářů. Pro zajištění ochrany proti útokům hrubou silou je nastaven maximální počet pokusů o přihlášení za daný čas – konkrétně 5 pokusů za 15 minut. Jako další je definována veřejně dostupná cesta k HTML stránce s přihlašovacím formulářem (viz kapitolu XXX). Poté je uživatelem zadané heslo porovnáno s hashovaným administrátorským heslem, jež je uloženo v systémové proměnné, a shodují-li se, je uživateli vygenerován JSON Web Token sloužící k potvrzení jeho identity s expirací 1 hodiny. Zároveň je ověřený uživatel automaticky přesměrován na stránku s uživatelským rozhraním pro administrátorské činnosti, která je bez platného autentizačního tokenu nepřístupná.

Zcela na konci kódu je zajištěno spuštění celé aplikace na definovaném serverovém portu.

Soubor *server.js* má i s komentáři jej vysvětlujícími celkem 485 řádků a je k dispozici v příloze DDD.

# uploadData.js

Soubor uploadData.js slouží, jak již název napovídá, ke zpracovávání nahrávaných fotografií a k jejich následnému uložení do databáze. Na jeho začátku jsou importovány následující moduly a knihovny: modul *fs* umožňující čtení a zápis souborů, modul *path* pro práci s cestami k souborům a adresářům, knihovna *multer* pro zpracování souborů v rámci HTTP požadavků, modul *pg* pro komunikaci s PostgreSQL databází, knihovna *exif-parser* pro extrakci metadat z nahrávaných fotografií, modul *node-fetch* pro odesílání HTTP požadavků a vlastní modul *convert* obsahující funkci *convertHeicToJpeg*, kterou blíže komentujeme v kapitole HHH.

Na začátku kódu definujeme cestu k adresáři *uploads*, do něhož jsou za pomoci knihovny *multer* ukládány konvertované fotografie. Pokud tento adresář neexistuje, je pomocí modulu *fs* automaticky vytvořen.

Dále je definována asynchronní funkce *uploadHEIC*, která nejprve rozlišuje originální fotografie od případných miniatur (viz kapitolu JJJ) na základě údajů získaných pomocí níže definované funkce *parseFileName*. Ta rozděluje název nahrávaného souboru na jeho funkční části a extrahuje z nich potřebné informace (viz kapitolu JJJ). V případě, že název souboru neodpovídá očekávanému formátu, funkce vyvolá chybu. Funkce *uploadHEIC* následně volá externí funkci *convertHeicToJpeg* ze souboru *convert.js*, která převádí formát fotografie z HEIC do JPEG (funkce je podrobněji popsána v kapitole HHH), a poté volá níže definovanou funkci *insertMetadata*. Ta z fotografie extrahuje metadata, upraví je pomocí níže definovaných funkcí *epochToIso8601* a *reverseGeocode* a všechna získaná data následně uloží do tabulky *records* v naší PostgreSQL databázi (viz kapitolu KKK). Funkce *epochToIso8601* jednoduše převádí z fotografie získané unixového časové razítko nejprve na milisekundy a poté do formátu ISO 8601, což je požadovaný standard pro ukládání časových údajů v datovém typu *TIMESTAMP*. Funkce *reverseGeocode* zase využívá reverzní geokódování (zpětné vyhledávání adresy) prostřednictvím veřejného API služby Nominatim, která je součástí projektu OpenStreetMap, a převádí GPS souřadnice na textovou adresu. Neobsahuje-li fotografie potřebná metadata, jsou do tabulky *records* k danému záznamu uloženy výchozí hodnoty: datum 1. 1. 2000 a souřadnice 0.0000 pro zeměpisnou délku i šířku.

Po definici asynchronní funkce *uploadLocation*, která - jak již název napovídá - slouží k ukládání lokací do databázové tabulky *locations* (viz kapitolu III), jsou až do konce souboru definovány pomocné funkce určené pro formátování dat z databáze pro jejich zobrazení na veřejné část webu. Jako první je definována funkce *formatTimestamp*, která nejdříve převádí datum a čas uložený v databází požadovaném formátu ISO 8601 do pro uživatele čitelnější podoby a ten následně rozděluje na funkční části, tedy sekundy, minuty, hodiny, dny, měsíce a roky. Ke dnům poté přidává slovní označení dne v týdnu (například pondělí) a číselné označení měsíců převádí na jejich slovní variantu. Funkce nakonec vrací finálně zformátovaný řetězec ve tvaru: „neděle 2. června 2024 v 11:54:06“. Následuje definice funkce *decimalToDMS*, která jednoduše převádí decimálně uvedené koordináty na formát DMS, tedy stupňů, minut a vteřin. Tato funkce je následně využita v dále definované funkci fo*rmatCoordinates*, která souřadnice doplňuje o zeměpisné směrovky. Výstupem funkce je naformátovaný textový řetězec následujícího formátu: „17° 39‘ 25.5“ V“. Jako poslední je definována funkce *translateStatus*, která převádí jednoznakový kód statusu uložený v databázi na jeho plnovýznamovou textovou reprezentaci. Jedná se o následující seznam: hodnota „V“ je převedena na „V pořádku“, hodnota „D“ je převedena na „Darován“, „Z“ je převedeno na „Ztracen“, „N“ je převedeno na „Nevyfocen“, „L“ je převedeno na „Bez zadané lokace“, „G“ je převedeno na „Byl mi darován“, „S“ je převedeno na „Status není určitelný“ a hodnota „J“ je převedena na „S jinou fotografií“. Pokud hodnota neodpovídá žádnému definovanému statusu, je na veřejnou část webu vykreslen text „Status neznámý“. Zcela na konci souboru jsou všechny funkce (a konstanty) exportovány jako součást modulu.

Soubor uploadData.js má i s komentáři jej vysvětlujícími celkem 293 řádků a je k dispozici v příloze HHH.

# convert.js

Soubor *convert.js* obsahuje pouze definici jedné funkce, nejdříve jsou ale importovány potřebné moduly, a to modul *fs* pro čtení a zápis souborů, modul *path* pro práci s cestami a modul *child\_proces*, který umožňuje spouštění externích nástrojů z prostředí Node.js.

Následně je v souboru definována funkce *convertHeicToJpeg*, která za pomoci open-source balíku softwarových nástrojů ImageMagick (viz kapitola JJJ) převádí fotografie z formátu HEIC do formátu JPEG. Funkce jako vstup přijímá cestu k původnímu HEIC souboru, který byl uložen do adresáře *uploads*, a na jejím základě vygeneruje novou cestu s příponou .*jpg*, kam bude uložen výstupní soubor. Převod je realizován spuštěním systémového příkazu *magick convert*, který vytvoří JPEG verzi fotografie. Po úspěšném dokončení převodu je původní HEIC soubor automaticky smazán, čímž se minimalizuje zbytečné zatížení uložiště. V případě jakékoli chyby při převodu nebo mazání je chyba zachycena a předána zpět volajícímu skriptu.

Funkce je implementována jako *Promise*, což umožňuje její použití pomocí klíčového slova *await* v rámci asynchronních funkcí. Při úspěšném dokončení vrací funkce cestu k nově vytvořenému JPEG souboru. Na konci souboru je funkce exportována v rámci modulu.

Použití této funkce předpokládá, že je v systému správně nainstalován nástroj ImageMagick a že je dostupný příkazem magick.

Celý soubor má i s komentáři pouhých 31 řádků a je uveden v příloze LLL.

# queries.js

Soubor *queries.js* obsahuje výhradně funkce, které pomocí SQL dotazů generují statistiky o aplikací používané databázi záznamů, a pro svou činnost vyžaduje pouze import třídy *Client* z modulu *pg*, která slouží pro připojení a interakci s PostgreSQL databází. Všechny definované funkce přijímají vždy dva parametry – instanci aplikace (objekt *app*) a konfigurační objekt *dbConfig*, jenž obsahuje údaje potřebné k navázání spojení s databází – a každá z nich má přiřazenou vlastní HTTP GET cestu. Výsledky všech funkcí jsou vraceny ve formátu JSON, a v případě výskytu jakékoli chyby při práci s databází je výjimka zachycena a odeslána klientovi ve formě chybového hlášení. Spojení s databází je pro zpracování každého požadavku vždy korektně ukončeno.

První definovanou funkcí je *top5Locations*, která vrací seznam pěti lokalit, kde bylo nalezeno nejvíce čtyřlístků. Funkce nejprve zjistí celkový počet záznamů v tabulce *records*, aby mohla v následném dotazu vypočítat relativní procentuální zastoupení jednotlivých lokalit (dle počtu nalezených čtyřlístků). Hlavní SQL dotaz využívá spojení (*JOIN*) mezi tabulkami *locations* a *records* (viz kapitolu MMM), přičemž z výsledků vyřazuje anonymizované lokace. Výsledky se totiž zobrazují na veřejné části webu a zveřejnění byť jen názvů anonymizovaných lokalit by potenciální mohlo narušit soukromí majitel stránek. Dotaz vrací název lokality, počet v ní nalezených čtyřlístků a jejich procentuální podíl na celkovém počtu záznamů. Výsledky jsou seřazeny sestupně podle četnosti výskytu, omezeny na pět nejfrekventovanějších.

Další funkcí je *firstLastYear*, která vrací záznam prvního a posledního čtyřlístku nalezeného v každém jednotlivém roce. Funkce provádí vícestupňový SQL dotaz využívající tzv. společné tabulkové výrazy (CTE). První z nich je *yearly\_records*, který extrahuje rok z časového razítka každého záznamu v tabulce *records*. Následují dva výrazy *first\_records* a *last\_records*, jež pomocí konstrukce *DISTINCT ON (year)* vybírají z každého roku první, respektive poslední záznam podle časového razítka. Výsledkem je tabulka, která pro každý rok obsahuje identifikátor a časové razítko prvního i posledního záznamu.

Následuje funkce *earliestLatestHour*, která vrací záznam čtyřlístku nalezeného v nejčasnější a nejpozdější denní dobu. Jinými slovy, cílem funkce je zjistit, který čtyřlístek byl nalezen nejdříve ráno a který nejpozději večer. Funkce provádí dva oddělené SQL dotazy, každý pro jeden z časových extrémů. Dotaz *earliestQuery* vyhledává čtyřlístek s nejdřívějším časem nálezu, zatímco dotaz *latestQuery* hledá záznam s nejpozdějším časem. V obou případech je časové razítko s celým datumem převedeno na čistý čas (bez dní, měsíců a roků), díky čemuž lze nezávisle na datu identifikovat minimální a maximální hodnotu časového údaje. Funkce vrací odpověď ve formátu JSON obsahující identifikátor a časové razítko nejčasnějšího a nejpozdějšího záznamu.

Další funkcí definovanou v souboru *queries.js* je *mostInDay*, která hledá den, během něhož bylo nalezeno nejvíce čtyřlístků. Jedná se o jednoduchý SQL dotaz, jenž seskupí všechny záznamy v tabulce *records* podle dne bez ohledu na čas, pro každý den sečte počet nálezů a dny poté sestupně seřadí. Navráceno je datum dne s nejvyšším počtem nálezů a tento počet.

Dále je definována čtveřice funkcí, které poskytují přehled o počtu nálezů v různých časových úsecích. Všechny tyto funkce pracují pouze s časovým razítkem záznamů v tabulce *records*, avšak každá s ním nakládá odlišným způsobem. První je definována funkce *byYear*, která z časového razítka každého nálezu nejdříve pomocí funkce *EXTRACT(YEAR FROM timestamp)* vyextrahuje pouze rok a následně spočítá počet nálezů pro každý jednotlivý rok. Výsledkem je přehled celkového počtu nálezů za jednotlivé rok. Následuje funkce *byMonth*, která obdobně z časového razítka nálezu získá pomocí funkce *EXTRACT (MONTH FROM timestamp)* údaj o měsíci a záznamy pak pro každý měsíc sečte nezávisle na roce či dni. Výsledkem je tedy souhrn, kolik čtyřlístků bylo nalezeno v jednotlivých měsících bez ohledu na rok, a funkce tedy slouží např. k určení, které měsíce bylo nalezeno nejvíce čtyřlístků. Funkce *byDay* seskupuje nálezy nikoliv podle pořadového čísla dne v měsíci, ale podle dne v týdnu. Je použit složitější výraz *(EXTRACT(DOW FROM timestamp) + 6) % 7 + 1*, který převádí dny týdne na číselné hodnoty v rozsahu 1-7 a zároveň zajišťuje převod z amerického systému číslování dnů, kde je za první den týdne považována neděle, na evropský systém, kde je za první den považováno pondělí. To je důležité pro správné vykreslení statistik na veřejné části webu (viz kapitolu NNN). Dotaz záznamy seskupí podle těchto čísel a vrací počet nálezů pro každý den v týdnu. Poslední ze čtveřice funkcí je funkce *byHour*, která pomocí výrazu *EXTRACT(HOUR FROM timestamp)* seskupuje záznamy podle hodiny nálezu a vrací počet záznamů pro každou celou hodinu dne, nezávisle na konkrétním dni, měsíci či roce – může například vrátit údaj, že mezi 14.00 a 15.00 bylo za celou dobu evidence nalezeno 99 čtyřlístků.

Poslední funkcí definovanou v souboru *queries.js* je *mostInHour*, jejímž cílem je určit, v kterou konkrétní hodinu konkrétního dne bylo nasbíráno nejvíce čtyřlístků. Funkce tedy nekonkuruje předchozí funkci *byHour*, která agreguje počty nálezů podle hodin napříč všemi dny, ale snaží se najít absolutně nejproduktivnější hodinový úsek v celé historii evidence. Podobně jako v předchozích funkcích je i zde z časového razítka extrahována hodina nálezu, tentokrát však v kombinaci s datem. Záznamy jsou následně seskupeny podle celých hodin jednotlivých dnů a je vybrán ten časový úsek, ve kterém bylo nalezeno nejvíce čtyřlístků.

Zcela na konci souboru jsou všechny funkce – *top5Locations*, *firstLastYear*, *earliestLatestHour*, *mostInDay*, *byYear*, *byMonth*, *byDay*, *byHour* a *mostInHour* – prostřednictvím exportu modulu zpřístupněny pro použití v jiných částech serverové aplikace. Soubor *queries.js* má celkem (včetně komentářů) 311 řádků a je přiložen jako příloha HHH.

# pdfGenerator.py

Majitel webové aplikace požadoval možnost generovat z databáze aplikace PDF soubor s vybranými záznamy. Pro implementaci této funkcionality byl vytvořen skript v jazyce Python, jehož použití se jevilo jako jednodušší než realizace stejné funkcionality v jazyce JavaScript.

Soubor *pdfGenerator.py* začíná stejně jako soubory psané v jazyce JavaScript importováním potřebných modulů, jichž je v tomto případě použito poměrně velké množství. Většina z nich pochází z knihovny *reportlab*, která slouží k tvorbě PDF dokumentů. Jedná se o moduly –*pdfbase.pdfmetrics* a *pdfbase.ttfonts*, které umožňují registraci a používání vlastních TrueType fontů, *pdfgen.canvas* umožňující vykreslování textu, tvarů a obrázků, *lib.pagesizes*, jenž poskytuje přednastavené rozměry stránek, v našem případě formát klasického dopisního papíru *letter*, a *lib.utils* usnadňující načítání a vkládání obrázků. Modul *psycopg2* je importován pro připojení k PostgreSQL databázi. Kromě toho jsou použity tři standardní knihovny jazyka Python: *sys* a *os* pro práci s cestami a parametry příkazové řádky a *datetime* pro práci s datem a časem.

V další části kódu je definována proměnná zajišťující konfiguraci pro připojení k aplikací používané PostgreSQL databázi. V této proměnné jsou uloženy přihlašovací údaje k databázovému serveru, jako je jeho adresa, název databáze, uživatelské jméno, heslo a číslo portu, na kterém server naslouchá. Dále jsou v různých proměnných specifikovány tři cesty – k adresáři obsahujícímu mapy s místy, kde byly nalezeny čtyřlístky (viz kapitolu OOO), a k adresáři obsahujícímu dva TrueType fonty: běžnou a tučnou variantu písma Arial Unicode, které jsou nezbytné pro správné vykreslení textu v českém jazyce. Tyto fonty jsou následně registrovány pomocí modulu *pdfmetrics* a zpřístupněny pod názvy Arial a Arial-Bold.

Dále je definována první funkce *fetch\_records\_in\_range*, která přijímá dva parametry – *min\_id* a *max\_id*, jež dohromady určují rozsah záznamů zahrnutých do generovaného PDF souboru. Společně s identifikátory záznamů jsou pomocí SQL dotazu získány také časové razítko, zeměpisné souřadnice a cesta k miniatuře fotografie z tabulky *records* a název lokace z tabulky *locations* pro každý záznam. V případě chyby je vypsána chybová hláška a funkce vrátí *None*.

Následně jsou v souboru definovány tři pomocné funkce pro formátování údajů získaných z databáze. Funkce *decimal\_to\_dms* převádí souřadnice uložené v desetinném formátu na formát DMS obsahující stupně, minuty a vteřiny. Funkce nejprve oddělí celočíselnou část jako počet stupňů, následně přepočítá desetinnou část na minuty a zbývající zlomek minut převede na vteřiny. Výstup této funkce je využit v rámci funkce druhé, *format\_coordinates*, která doplňuje převedené souřadnice o písmeno značící příslušnou světovou stranu. Výstupem funkce je textový řetězec ve formátu: „49° 13‘ 51.0“ S“. Poslední z trojice funkcí je *format\_timestamp*, která modifikuje v databázi uložené časové razítko, obohacuje ho o název den v týdnu a nahrazuje v něm číselné označení měsíce jeho slovním vyjádřením. Výsledný formátovaný řetězec má podobu: „středa 23. dubna 2025 16:26:40“. Obdobné funkce byly vytvořeny také v souboru *uploadData.js* a jsou popsány v kapitole PPP.

Následuje de facto předposlední funkce *draw\_image\_preserve\_aspect*, která do PDF souboru přidává ke každému z generovaných záznamů mapu místa nálezu, jež je v levém horním rohu překryta miniaturou fotografie nalezeného čtyřlístku. Funkce za parametry přijímá plátno PDF dokumentu (*c*), cestu k obrázku mapy (*mapPath*), souřadnice *x* a *y* určující umístění obrázku na stránce, požadovanou výšku a šířku obrázku, a volitelně také cestu k miniatuře fotografie. Pomocí nástroje *ImageReader* obsaženého v modulu *reportlab.lib.utils* jsou nejprve zjištěny původní rozměry obrázku s mapou a vypočítán poměr jeho stran, který je zachován i při následném zmenšení, aby nedošlo k jeho deformaci. Následně jsou vypočítány výsledné rozměry obrázku tak, aby co nejlépe vyplnil dostupný prostor, obrázek je zmenšen a umístěn na správné místo v PDF souboru. V případě, že byla nalezena cesta k miniatuře fotografie čtyřlístku, je zmenšena i ta na velikost odpovídající 23 % šířky a 36 % výšky mapy a umístěna do jejího levého horního rohu. Pokud není miniatura fotografie čtyřlístku nalezena, je do konzole vypsána chybová hláška, dokument je však i přesto vygenerován bez ní.

Poslední funkcí v souboru je *generate\_pdf*, která je zodpovědná za samotné generování výstupního PDF souboru. Za své vstupní parametry přijímá seznam záznamů a cestu k vygenerovanému souboru. Na začátku funkce jsou definovány rozměry jednotlivých prvků stránky, jako je výchozí pozice textu, rozestupy mezi jednotlivými záznamy nebo velikost obrázku mapy. Každý záznam je následně vykreslen do jednoho ze šesti dostupných bloků na stránce – každá stránka souboru je rozdělena na dva sloupce po třech řádcích. U každého záznamu jsou uvedeny následující informace: jeho pořadové číslo (identifikátor), mapa s miniaturou fotografie (pomocí funkce *draw\_image\_preserve\_aspect*), datum a čas nálezu ve zformátované podobě, zformátované zeměpisné souřadnice a název lokace. Soubor je nakonec uložen a do konzole je vypsán výsledek procesu. Ukázku výsledného PDF dokumentu můžeme nalézt v kapitole QQQ.

Soubor končí definicí funkce *main()*, která představuje výchozí bod pro spuštění celého skriptu. V ní je nejprve zkontrolován uživatelem zadaný rozsah záznamů pro generování a určen název výsledného souboru – *records\_report.pdf*. Následně je zavolána funkce *fetch\_records\_in\_range* a po jejím úspěšném dokončení je výsledek předán jako parametr funkci *generate\_pdf*. Při neúspěšném dokončení funkce *fetch\_records\_in\_range* je na konzoli vypsána chybová hláška. Zcela na konci souboru je pomocí konstrukce *if \_\_name\_\_ == "\_\_main\_\_* *": main()* zajištěno, že se funkce *main()* spustí pouze při přímém spuštění skriptu.

Soubor *pdfGenerator.py* má včetně komentářů 208 řádků a je k dispozici v příloze LLL.

# Adresář public

V adresáři *public* se nachází celkem dvanáct souborů – šest HTML souborů, pět souborů s příponou .svg a jeden soubor ve formátu PNG –, které dohromady tvoří uživatelské webové rozhraní.

Všechny soubory ve formátu SVG – *comment\_icon.svg*, *map\_icon.svg*, *menu\_icon.svg*, *photo\_icon.svg* a *secret\_icon.svg* – slouží jako ikony a jejich účelem je působit na návštěvníka stránek moderním, profesionálním a estetickým dojmem. Všechny ikony jsou volně dostupné a byly staženy ze stránek SVG Repo LLC. Soubor ve formátu PNG – *logo.png* – obsahuje obrázek loga stránek, který poskytl jejich majitel.

HTML soubory tvoří samotné uživatelské rozhraní stránek. Čtyři soubory – *main.html*, *records.html*, *list.html* a *statistics.html* – představují veřejnou část webu; zbývající dva soubory – *login.html* a *upload.html* – tvoří administrátorskou část webu. Všechny HTML soubory začínají jednotnou deklarací typu dokumentu HTML5 a nastavením kódování znaků na UTF-8. Jednotlivé HTML soubory jsou podrobně popsány v samostatných subkapitolách.

Soubory, které tvoří veřejnou část webu, jsou většinou popisovány „odspodu“: nejprve je komentována část s podpůrným vnořeným JavaScriptovým kódem, po ní následuje popis grafického zpracování stránky, které je primárně zajištěno kaskádovými styly v hlavičce dokumentu. Na všech veřejných stránkách je dále v hlavičce načten externí bezpatkový font Raleway, importovaný ze služby Google Fonts pro sjednocení typografie.

Barevné ladění veřejných stránek je zeleno-žluté. Gradient různých odstínů zelené barvy je většinou použit na pozadí stránek, zatímco žlutá barva je vyhrazena pro pozadí funkčních bloků. Všechny veřejně dostupné stránky obsahují navigační menu reprezentované malým žlutým čtvercem v pravém horním rohu stránky s ikonou menu. Tento blok se po kliknutí rozbalí do většího čtverce a zobrazí odkazy na tři zbývající stránky. Po kliknutí na kterýkoliv odkaz je uživatel automaticky přesměrován na příslušnou stránku.

# main.html

Soubor *main.html* poskytuje domovskou stránku veřejné části webové aplikace.

Část s vnořeným skriptem obsahuje pouze funkci *toggleMenu()*, která zajišťuje rozbalení a sbalení navigačního menu.

Navigační menu, včetně ikony a odkazů na ostatní stránky, je definováno v těle HTML dokumentu společně s úvodním textem. Ten vítá návštěvníka na stránkách Čtyřlístkotéky a předkládá informaci o jejich účelu, majiteli a autorovi. Text je doplněn logem stránek.

Stránku nelze „scrollovat“ – vertikálně ani horizontálně posouvat – neboť je navržena tak, aby se snažila vždy automaticky přizpůsobit výšce obrazovky zařízení. Výchozí pozadí stránky je tmavě zelené, ale díky animovanému přechodu (vertikálnímu gradientu) se postupně mění na světlejší odstíny a pak zpět. Tento efekt imituje plynulý pohyb a přidává tak jinak statické stránce mírnou dynamičnost a vizuální zajímavost. Efektu je dosaženo pomocí CSS animace *@keyframes gradientAnimation*. Ve spodní polovině stránky je prezentováno logo aplikace, nad kterým se nachází uvítací text s mírným stínováním. V pravém horním rohu se nachází malý žlutý čtverec s navigačním menu.

Soubor *main.html* má celkem 141 řádků a je k dispozici v příloze QQQ. Podobu domovské stránky můžeme vidět na obrázku AA1.

# records.html

Soubor *records.html* vytváří stránku *Záznamy*, na které si může návštěvník zobrazit údaje o konkrétním čtyřlístku.

Vnořený skript začíná importováním externí knihovny *Leaflet* vyvinuté Vladimirem Agafonkinem, která slouží k vykreslování interaktivních map (s využitím podkladových dat z projektu OpenStreetMap), a pokračuje definováním funkce *toggleMenu()* pro ovládání navigačního menu.

Dále je definována funkce *initializeMap*, jež zajišťuje inicializaci mapy v místě nálezu vyhledaného čtyřlístku. Jsou-li do funkce jako parametry předány anonymizované souřadnice s hodnotami rovnajícími se 0.0000 (decimální tvar), jsou tyto souřadnice automaticky nahrazeny souřadnicemi 49.2309 pro zeměpisnou šířku a 17.6571 pro zeměpisnou délku, na kterých se nachází sídlo Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně. V mapě je zobrazena šipka označující přesné místo nálezu, doplněná textem „Nalezen zde“. V případě anonymizovaného záznamu se popisek změní na „Nález anonymizován“. Funkce rovněž zajišťuje odstranění zobrazené mapy při načítání nového záznamu.

Nejdůležitější část skriptu tvoří funkce *retrieveData*, která slouží k asynchronnímu načtení dat z databáze na základě zadaného ID záznamu. Jedná se o poměrně složitou funkci, neboť zajišťuje nejen správné formátování a dynamické vykreslování údajů, ale také nahrazování výchozího loga a textu za každou novou fotografii čtyřlístku, která je, je-li na ni namířeno kurzorem, automaticky nahrazena její miniaturou. Funkce také pochopitelně ošetřuje vyhledávání záznamů, které byly anonymizovány, a v takových případech zobrazí pouze fotografie, status a čas nálezu. Ostatní údaje jsou nahrazeny oznámením o jejich anonymizaci a mapou ukazující Fakultu aplikované informatiky Univerzity Tomáše Bati ve Zlíně. Funkce rovněž ošetřuje případné chyby při získávání dat z databáze. Na konci skriptu je implementován posluchač událostí, který kontroluje platnost vstupního ID a v případě úspěšné validace spouští funkci *retrieveData*.

Stejně jako domovská stránka je i stránka Záznamy ve své výchozí podobě navržena tak, aby automaticky vyplnila celou výšku obrazovky zařízení a není na ní tedy možné vertikální ani horizontální posouvání („scrollování“). Také zde je zachován gradient pozadí, tentokrát však ve své statické podobě - tmavě zelená barva v horní části přechází směrem dolů ve světle zelenou.

Po vyčlenění malé části v horní části stránky je zbývající prostor rozdělen do tří pomyslných vertikálních sloupců. Levý a pravý sloupec zabírají přibližně stejný prostor, střední sloupec je proporčně o něco menší. Pravý sloupec je určen výhradně pro blok, do kterého je po vyhledání konkrétního záznamu načtena interaktivní mapa místa nálezu. Ve výchozím stavu je uprostřed bloku zobrazeno stylizované logo s textem „Mapa“.

Levý sloupec je z většiny vyhrazen pro blok s fotografií čtyřlístku, případně její miniaturou. Ve výchozí podobě je uprostřed bloku zobrazeno logo s textem „Fotografie“. Pod tímto blokem se nachází menší obdélníková sekce pro výpis statusu záznamu.

Střední sloupec je rozdělen na pět stejně velkých obdélníkových bloků a jeden větší blok v dolní části pro zobrazení komentáře k místu nálezu. Ve výchozí podobě obsahuje tento spodní blok uprostřed logo s textem „Komentář“. Zbývající bloky slouží postupně (shora dolů) k zobrazení identifikačního čísla (ID) záznamu, zobrazení formátovaného časového razítka, zobrazení formátovaných zeměpisných souřadnic, nejbližší adresy a názvu lokace.

Horní část stránky obsahuje dvě ovládací sekce – vpravo se nachází čtvereček s navigačním menu, zatímco vlevo je umístěn obdélníkový blok s formulářem pro vyhledání záznamu dle jeho identifikačního čísla. Všechny bloky mají žlutou barvu pozadí, jsou orámovány tmavou linkou, mají zaoblené rohy a stínování, které navozuje dojem prostorové hloubky.

Soubor má celkem 392 řádků a je přiložen v příloze RRR. Výchozí podoba stránky je k vidění na obrázku AA1; podoba stránky s načteným záznamem k dispozici na obrázku BB2.

# list.html

Soubor *list.html* slouží ke generování stránky Seznam záznamů, na níž je ve formě tabulky zobrazen přehled všech do databáze uložených čtyřlístků. Vnořený skript začíná definicí globálních proměnných, a to *currentPage* pro uchování čísla aktuálně zobrazené stránky seznamu (s počáteční hodnotou 1), konstanty *recordsPerPage*, která omezuje maximální počet záznamů zobrazovaných na jedné stránce na 50, a proměnné *records* pro pole obsahující všechny načtené záznamy. Dále je definována funkce *toggleMenu()*, sloužící k ovládání navigačního menu.

Následuje sada funkcí zajišťujících přepínání mezi stránkami seznamu. Funkce *goToFirstPage()* uživatele vždy jednoduše navrací na první stránku seznamu. Funkce *goToLastPage()* dělí celkový počet záznamů hodnotou *recordsPerPage*, čímž zjistí stránku, na které se nachází poslední nahraný záznam, a uživatele na ni přesměruje. Funkce *goToPreviousPage()* a *goToNextPage()* umožňují přepínání na předchozí, respektive následující stránku. Obě funkce zároveň kontrolují, zda uživatel nechce přejít na neexistující stránku – spodní hranice je omezena hodnotou 1, horní je určena stejným výpočtem jako ve funkci *goToLastPage()* – a tento přechod případně znemožňují.

Následuje dvojice funkcí, které dohromady umožňují přechod na libovolnou existující stránku seznamu. Funkce *validatePageInput()* kontroluje, zda je uživatelem zadané číslo stránky v rozsahu existujících stránek, a pakliže není, deaktivuje spuštění funkce *goToPage()*, která uživatele na požadovanou stránku přesměruje.

Všechny výše popsané navigační funkce spouštějí při validním vstupu funkce *renderTable()* a *updatePaginationControls()*. Funkce *renderTable()* dynamicky vykresluje požadovanou stránku seznamu včetně příslušných záznamů. Pro každý záznam je vytvořen jeden HTML řádek obsahující identifikační číslo (ID), fotografii miniatury, status, formátovanou časovou známku, zeměpisnou šířku, zeměpisnou délku, nejbližší adresu, název lokace a komentář k lokaci. Funkce *updatePaginationControls()* aktualizuje číslo aktuální stránky zobrazené v uživatelském rozhraní.

Na konci skriptu je implementován posluchač událostí, který odesílá požadavek na serverové API pomocí metody *fetch()*. Vrácená data jsou převedena do formátu JSON a následně uložena do proměnné *records*, přičemž jsou seřazena vzestupně podle identifikačního čísla (ID). Tabulka je poté vykreslena. V případě chyby během načítání dat je v konzoli prohlížeče vypsáno chybové hlášení.

Stejně jako v předchozích kapitolách, jež popisovaly domovskou stránku a stránku *Záznamy*, je i na stránce *Seznam záznamů* zachován barevný přechod pozadí (gradient). Jelikož je ale nutné stránku vertikálně posouvat („scrollovat“), je přechod mezi jednotlivými odstíny zelené ve vertikálním směru jemnější, gradient je ale na této stránce aplikován i ve směru horizontálním.

Stránka obsahuje tabulku se záznamy, přičemž je v její hlavičce popsán každý sloupec. Šířka jednotlivých sloupců je v kódu explicitně nastavena pomocí kaskádových stylů. Tabulka má žluté pozadí a její hlavička je zvýrazněna světle zelenou barvou. Tabulka je orámována černou linkou a nemá zaoblené rohy. Tělo tabulky je v HTML kódu ponecháno prázdné, neboť je jeho obsah dynamicky doplňován pomocí skriptu.

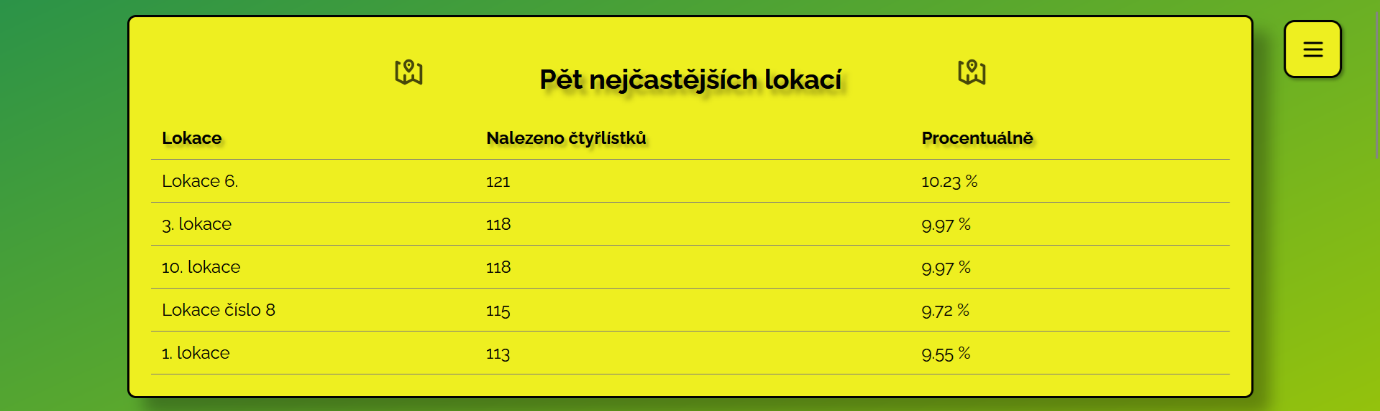
V horní části stránky se nachází již zaoblený a stínovaný blok se žlutým pozadím sloužící jako hlavní nadpis stránky (s jejím názvem tj. Seznam záznamů). Pod ním se nachází šest horizontálně seřazených prvků pro ovládání stránkování. Konkrétně se jedná o následující prvky jdoucí zleva doprava: tlačítko pro přechod na první stránku, tlačítko odkazující na předchozí stránku, formulář pro zadání čísla stránky, tlačítko pro potvrzení přechodu na konkrétní stránku, tlačítko pro přechod na další stránku a tlačítko odkazující na poslední stránku seznamu. Tlačítka vizuálně korespondují se stylem nadpisu a při najetí kurzorem mění barvu, čímž uživateli poskytují vizuální zpětnou vazbu. V pravém horním rohu se již tradičně nachází blok s navigačním menu.

Soubor *list.html* má 343 řádků a je uveden v příloze SSS. Ukázka jedné ze stránek testovací databáze je zachycena na obrázku CC3.

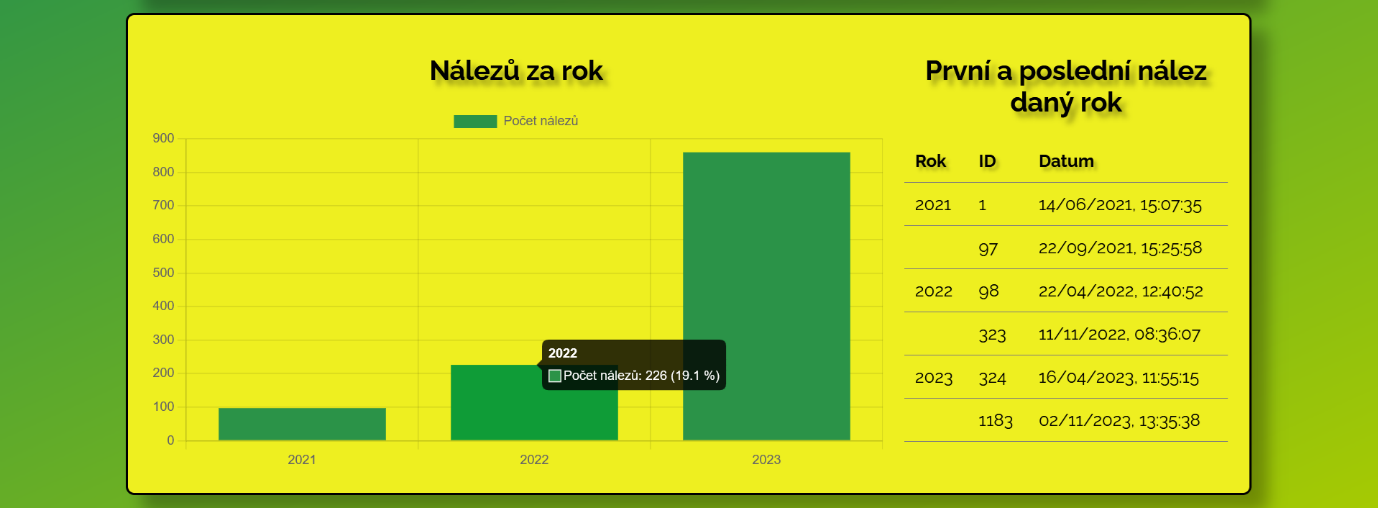
# statistics.html

Soubor *statistics.html* generuje stránku, která návštěvníkům zobrazuje statistiky o databázi čtyřlístků. Vnořený skript začíná importem externí knihovny *Chart.js*, která je vydaná pod licencí MIT a slouží pro generování grafů. Následuje definice funkce *toggleMenu()*, která slouží k ovládání navigačního menu. Zbytek skriptu je tvořen asynchronními funkcemi, jež prostřednictvím API požadavků volají funkce popsané v kapitole *queries.js*.

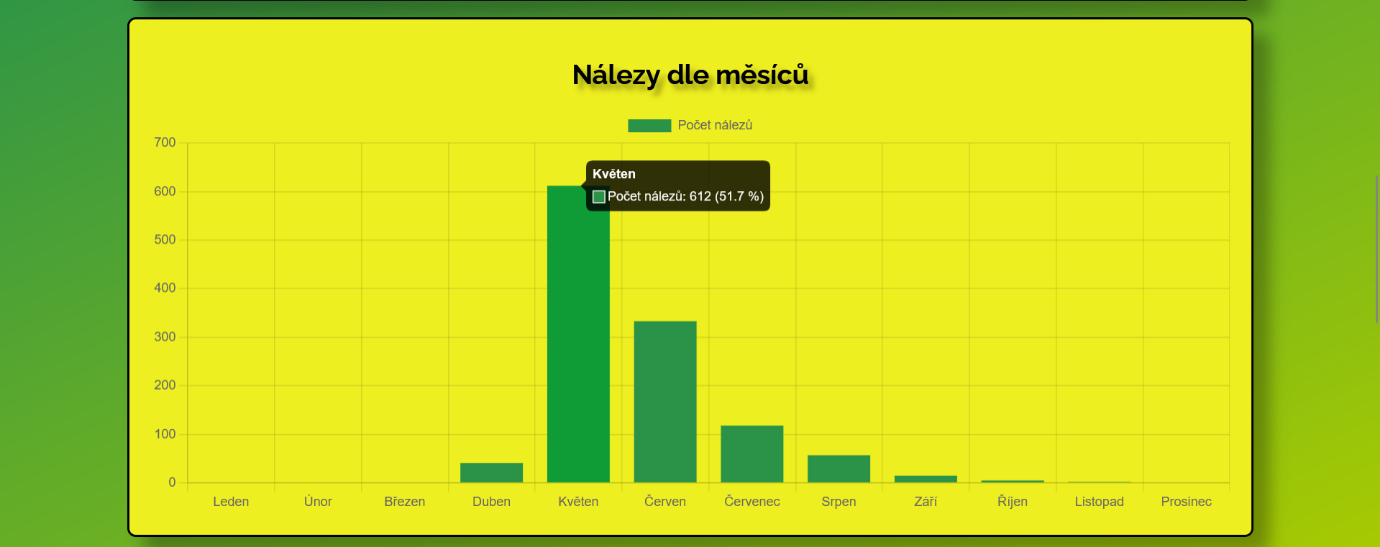
První z těchto funkcí je *fetchTopLocations()*, která vrací data o pěti lokacích, kde bylo nalezeno nejvíce čtyřlístků. Sestupně seřazené výsledky, jež obsahují název lokace, počet nalezených čtyřlístků a jejich procentuální podíl na celé sbírce, jsou dynamicky vykresleny do tabulky ve žlutém bloku. Blok obsahuje také nadpis „Pět nejčastějších lokací“, který je z obou stran ohraničen ikonou mapy. Ukázku bloku s testovacími daty může čtenář vidět na obrázku níže.



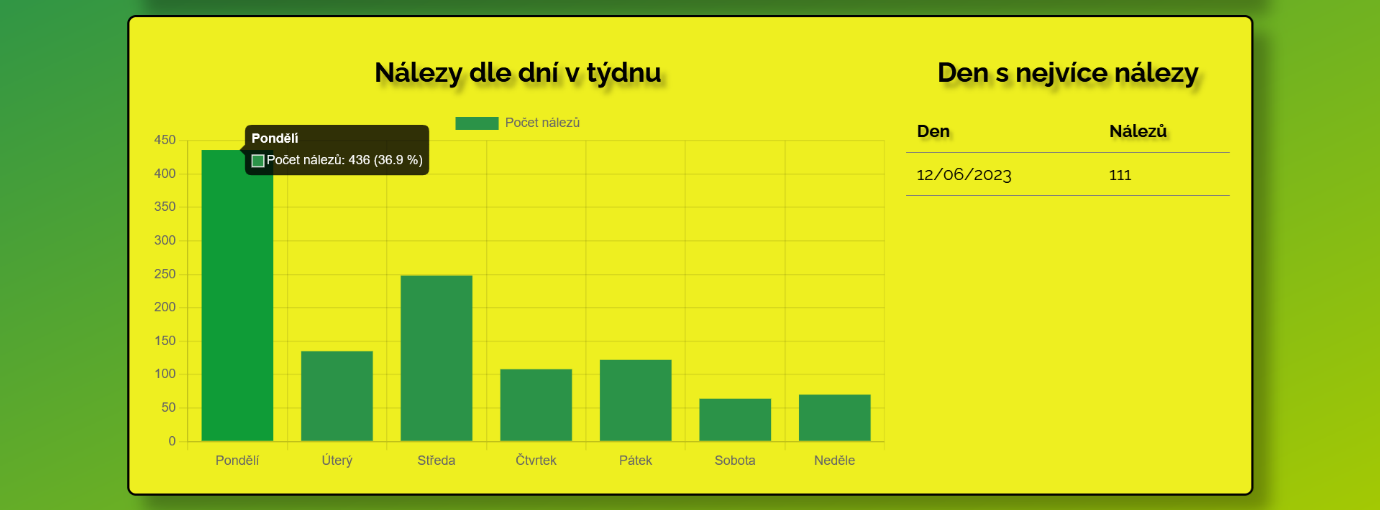
Druhý blok je vyplněn vizualizací dat získaných pomocí funkcí *fetchFindingsByYear()* a *fetchYearlyRecords()*. Větší část tohoto bloku, umístěná na levé straně, je věnována výstupu funkce *fetchFindingsByYear()*, která pod nadpisem „Nálezů za rok“ vykresluje sloupcový graf. Tento graf vizualizuje počet nalezených čtyřlístků v jednotlivých letech, přičemž každý zelený sloupec reprezentuje jeden konkrétní rok a jeho výška (osa y) odpovídá počtu zaznamenaných nálezů. Po najetí kurzorem na konkrétní sloupec se objeví přesný počet nálezů v příslušném roce a jejich procentuální podíl na celkovém počtu čtyřlístků. Menší pravá část bloku, která zabírá 30 % jeho celkové šířky, obsahuje tabulku s výstupem funkce *fetchYearlyRecords()*. Pod příslušným nadpisem je v tabulce zobrazen vždy první a poslední nález daný rok. Každý z těchto nálezů je reprezentován identifikačním číslem (ID) a přesným datem jeho nálezu. Pro ilustraci je blok s načtenými testovacími daty zachycen na obrázku uvedeném níže.



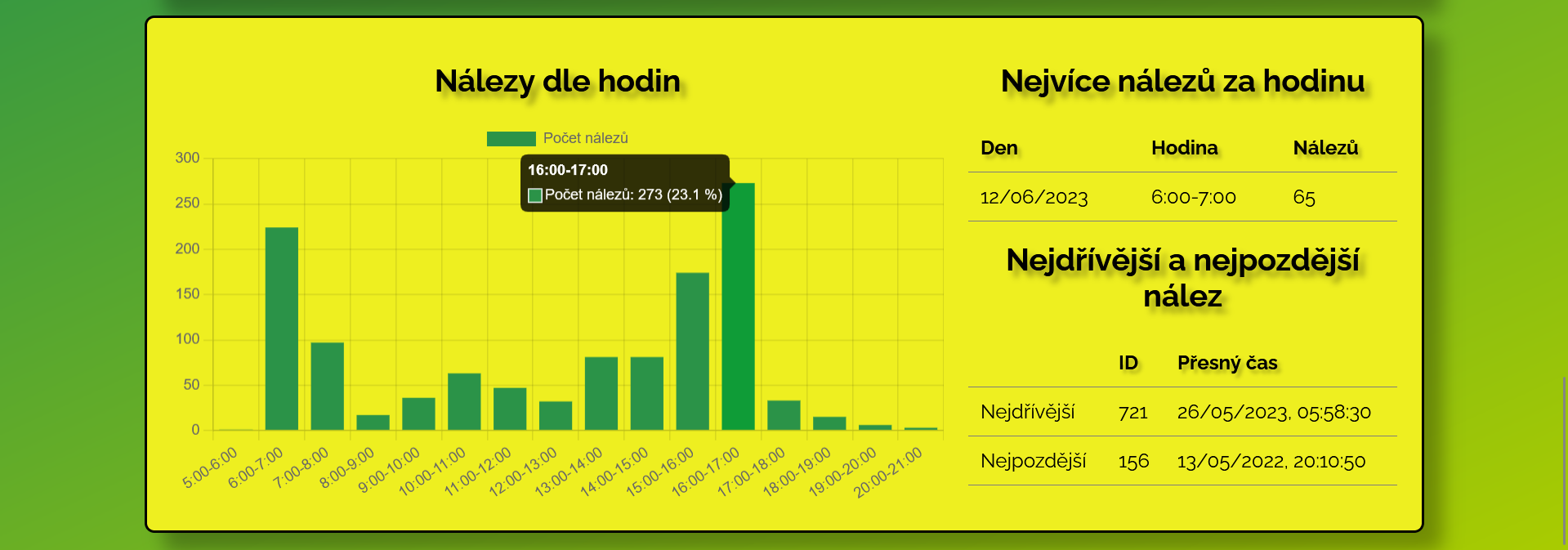
Třetí blok je nadepsán „Nálezy dle měsíců“ a jeho obsah je generován pomocí funkce *fetchFindingsByMonth()*. V rámci této funkce musela být inicializována dvě pole o délce dvanácti prvků. Do jednoho z nich jsou ukládány číselné hodnoty reprezentující měsíce převedené do indexového formátu (0-11), a to v souladu s konvencí používanou v jazyce JavaScript. Druhé pole uchovává počty nálezů v jednotlivých měsících. Výsledkem je sloupcový graf, ve kterém jsou jednotlivé měsíce reprezentovány zelenými sloupci, jejichž výška znovu odpovídá počtu nalezených čtyřlístků. Stejně jako u předchozích grafů se při najetí kurzorem na konkrétní sloupec zobrazí přesný počet nálezů za daný měsíc a jejich procentuální podíl na celkovém počtu čtyřlístků ve sbírce. Blok s testovacími daty je pro ukázku uveden na obrázku níže.



Předposlední, čtvrtý blok je stejně jako blok druhý rozdělen na dvě části v poměru sedm ku třem. Jeho větší část zabírá graf, ve kterém jsou zelenými sloupci znázorněny počty nalezených čtyřlístků podle jednotlivých dnů v týdnu. Také tento graf je opatřen vlastním nadpisem a při najetí kurzorem na konkrétní sloupec se i zde zobrazí přesný počet nálezů daný den spolu s jejich procentuální podílem na celkovém počtu. Data jsou pro graf zpracována pomocí funkce *fetchFindingsByDay()*, která převede číselné značení dnů v týdnu z rozsahu 1–7 na indexový formát 0–6, jenž odpovídá konvenci v jazyce JavaScript. Výsledky jsou následně ukládány do pole o sedmi prvcích, kde každý prvek reprezentuje jeden den v týdnu a uchovává počet nálezů pro daný den. Funkce zároveň počítá procentuální podíl nálezů pro každý den vůči celkovému počtu. Menší pravá část bloku je pak vyplněna jednoduchou jednořádkovou tabulkou s nadpisem, která prezentuje den, kdy bylo nalezeno nejvíce čtyřlístků. Uvedeno je datum a celkový počet nálezů tento den. Tyto informace jsou získány pomocí funkce *fetchDayWithMost()*, která ze získaných dat extrahuje datum a převádí jej pomocí metody *toLocaleDateString()* do lépe čitelného formátu. Celý čtvrtý blok je znázorněn na obrázku níže za využití testovacích dat.



Poslední blok je rozdělen na tři části. Horizontálně je rozdělen v poměru 65:35 a větší levou část vyplňuje nadepsaný sloupcový graf, který vizualizuje četnost nálezů dle jednotlivých hodin. Graf je generován funkcí *fetchFindingsByHour()*, jež seskupuje časové údaje dle hodin a zároveň počítá procentuální podíl nálezů v každém hodinovém intervalu vůči jejich celkovému počtu. Každý sloupec reprezentuje jeden hodinový interval, má i zde zelenou barvu a jeho výška odpovídá počtu nálezů, jehož přesný počet i procentuální podíl získáme po najetí kurzorem na konkrétní sloupec. V menší pravé straně bloku se nachází dvě nadepsané tabulky. Data do vrchní z nich jsou vybrána na základě funkce *fetchHourWithMost()*, která vrací absolutně nejproduktivnější hodinový interval ze všech zaznamenaných dnů. V tabulce je uvedeno konkrétní datum, časový úsek a počet nálezů. Spodní tabulka, naplněná prostřednictvím funkce *fetchEarliestLatest(),* zobrazuje identifikátor (ID) a přesný čas nejdříve a nejpozději zaznamenaného nálezu.



Zcela na konci skriptu je implementován naslouchač události *DOMContentLoaded*, který po načtení stránky automaticky spustí všechny popsané funkce.

Jelikož byl design stránky *Statistiky* částečně popsán při funkčním popisu jednotlivých bloků, zbývá už jen shrnout její celkové rozvržení. Na stránce je zachován barevný přechod pozadí (gradient) obdobným způsobem jako na stránce *Seznam záznamů*, tedy jak ve vertikálním, tak i horizontálním směru. Všechny bloky na stránce mají žluté pozadí, jsou ohraničeny černou linkou, mají zaoblené rohy a jsou doplněny jemným stínem. I na této stránce se v pravém horním rohu nachází minimalizované navigační menu.

Celý soubor *statistics.html* má celkem 630 řádků a je uveden v příloze TTT.

# login.html

Přihlašovací stránka do administrátorské části webu je spravována souborem *login.html* a neobsahuje žádné styly. Na zcela bílé stránce se nachází pouze nadepsaný přihlašovací formulář s jedním vstupním polem typu *password*, které zajišťuje nahrazení psaného textu tečkami, a jedním tlačítkem. Pomocí atributu *required* je zajištěno, že formulář nemůže být odeslán prázdný.

Vnořený skript tvoří jeden posluchač událostí, který zajišťuje autentizaci a autorizaci uživatele. Po zadání hesla a kliknutí na tlačítko je formulář odeslán. Z jeho vstupního pole je načtena hodnota hesla a odeslána ve formátu JSON na server. Pokud se odeslané heslo shoduje s administrátorským heslem uloženým na serveru, je ze serveru navrácena pozitivní odpověď společně s autentizačním tokenem a uživatel je automaticky přesměrován na administrátorskou stránku. Pokud se zadaný text s heslem neshoduje, je uživatel informován o chybě. Autentizace je tedy prováděna na straně klienta prostřednictvím asynchronního HTTP požadavku. Bezpečnost je posilněna umístěním přihlašovací stránky na „nepředvídatelný“ koncový bod s URL adresou BBx8olop, což ztěžuje automatizované útoky, které by jinak cílily na běžně používané koncové body jako je například */login*.

Soubor má pouhých 41 řádků a je k dispozici v příloze RRR. Na obrázku DD4 je pro ilustraci zobrazen snímek přihlašovací stránky.



# upload.html

Posledním souborem v adresáři public, který nebyl doposud popsán, je *upload.html*. Tento soubor vytváří administrátorské uživatelské rozhraní a stránka, kterou generuje, neobsahuje, stejně jako stránka generovaná souborem *login.html*, mnoho vizuálních prvků. Stránka má bílé pozadí a je rozdělena do tří sloupců. Než ale budou jednotlivé sloupce podrobně popsány, popíšeme začátek vnořeného skriptu.

Skript začíná importem dvou externích JavaScriptových knihoven a to knihovnou *Leaflet.js*, která je podepsána Vladimirem Agafonkinem a slouží pro práci s interaktivními mapami, a jejím rozšířením, knihovnou *leaflet-image.js*, jež umožňuje převod vykreslené mapy na obrazový formát vhodný k uložení. Obě knihovny jsou importovány ze serveru unpkg.com.

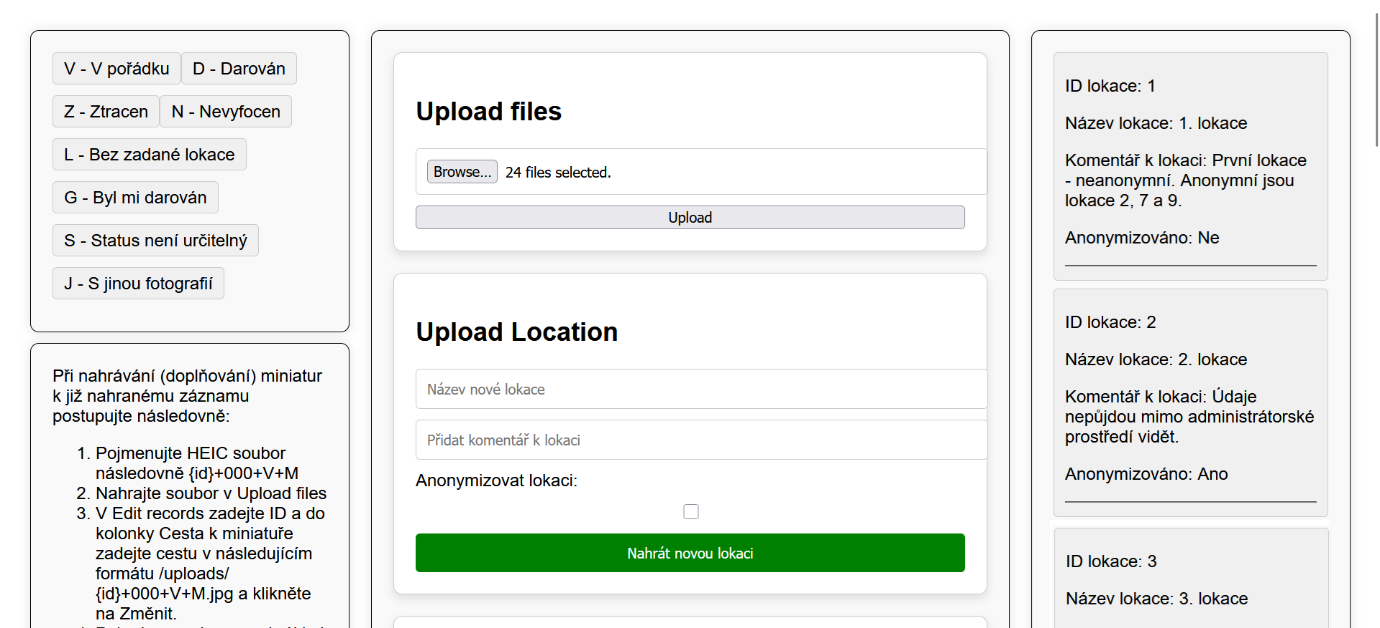
Bezprostředně po načtení knihoven je definována a spuštěna asynchronní funkce *fetchUploadPage()*, která načítá obsah stránky, ovšem pouze uživateli, jenž disponuje platným JSON Web Tokenem. Neověřeného uživatele nebo uživatele s expirovaným tokenem funkce automaticky přesměruje na přihlašovací stránku.

Levý ze tří sloupců je tvořen dvěma vertikálně umístěnými bloky. Vrchní blok, který je ve výchozím stavu sbalen a lze jej rozbalit kliknutím, obsahuje seznam zkratek statusů s jejich významy. Blok slouží uživateli k správnému pojmenovávání nahrávaných souborů, neboť status je do databáze ukládán na základě názvu nahrávaného souboru (viz kapitolu TTT). Rozbalování a sbalování bloku zajišťuje funkce *toggleStatusList()*, která je spouštěna posluchačem událostí reagujícím na kliknutí. Tato funkce zároveň spouští asynchronní funkci *fetchAndDisplayStatuses()*, jež získává a formátuje seznam statusů prostřednictvím HTTP požadavku na server. Funkce *fetchAndDisplayStatuses()* je ovšem spuštěna také ihned po načtení stránky, společně s níže popsanou funkcí *fetchAndDisplayLocations()*, prostřednictvím jiného posluchače událostí. Spodní blok je zcela statický a obsahuje pouze instrukce pro správné zpětné nahrávání (doplňování) miniatur k již nahraným záznamům. Tyto instrukce jsou formátovány jako očíslovaný seznam. Levý sloupec je, společně s několika dalšími prvky administrátorského rozhraní, zachycen na obrázku OBR níže.

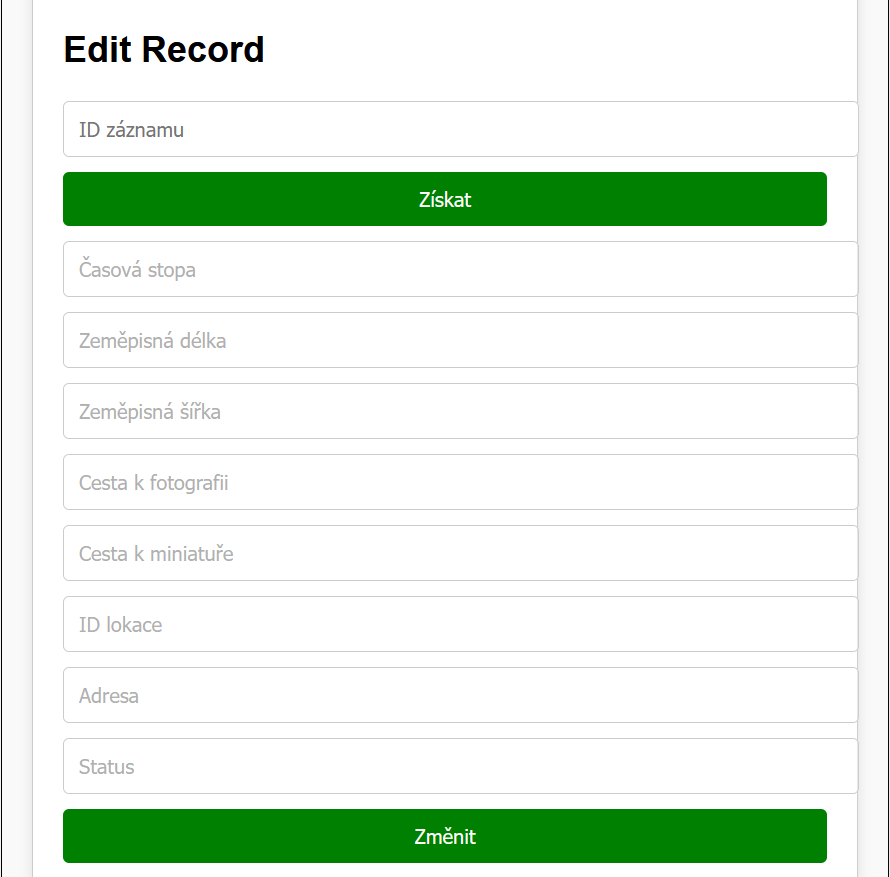
Pravý sloupec tvoří blok se seznamem všech v databázi existujících lokací, neboť i ta musí být uživatelem specifikována v názvu nahrávaného souboru (viz kapitolu TTT). Seznam uživateli tuto práci usnadňuje, neboť zobrazuje všechny informace z databázové tabulky *locations* (viz kapitolu UUU). Blok funguje na stejném principu jako blok zobrazující statusy – je tedy rozbalován a sbalován funkcí *toggleLocationList()*, kterou spouští posluchač událostí reagující na kliknutí. Tato funkce zároveň spouští asynchronní funkci *fetchAndDisplayLocations()*, jež načítá a formátuje data z databáze prostřednictvím HTTP požadavku na server. Část pravého sloupce zobrazujícího data z testovací databáze je společně s několika dalšími prvky administrátorského prostředí zachycena na obrázku OBR níže.

Největší prostřední sloupec se skládá ze šesti bloků, které zajišťují klíčové funkční prvky stránky. První z nich obsahuje formulář pro nahrávání fotografií na server. Formulář je ovládán dvěma tlačítky. Prvním je tlačítko označené anglickým výrazem „Browse…“, které slouží k výběru fotografií z lokálního adresáře. Vybrané fotografie jsou v cyklu vkládány do objektu typu *FormData* a akceptovány jsou pouze soubory s příponou .heic. Jakmile jsou fotografie do objektu vloženy, je jejich celkový počet uveden v krátkém anglickém textu vedle tlačítka. Fotografie se nahrají stiskem druhého tlačítka „Upload“, které vytvoří asynchronní HTTP požadavek na server. Správné kódování pro odesílání binárních dat zajišťuje atribut *enctype="multipart/form-data"*. Celá funkcionalita je realizována naslouchačem událostí. Blok „Upload Files“ pro nahrávání fotografií je společně s několika dalšími prvky administrátorského rozhraní zachycen na obrázku OBR níže.

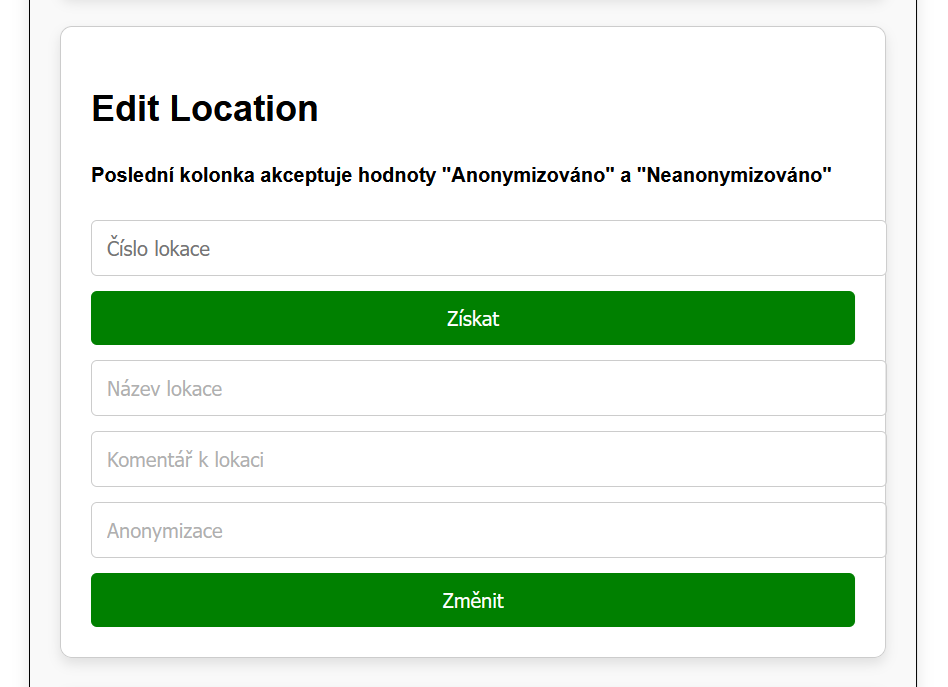
Druhý blok slouží k ukládání nových lokací do databáze. Uživatel do formuláře zadává název nové lokace, komentář k ní a pomocí zaškrtávacího políčka určuje, zda má být lokace anonymizována. Poté kliknutím na tlačítko „Nahrát novou lokaci“ lokaci uloží prostřednictvím asynchronního HTTP požadavku na server. O funkcionalitu se, stejně jako v předchozím případě, stará naslouchač událostí. Společně s několika dalšími prvky administrátorského rozhraní je blok „Upload Location“ zachycen na obrázku OBR níže.



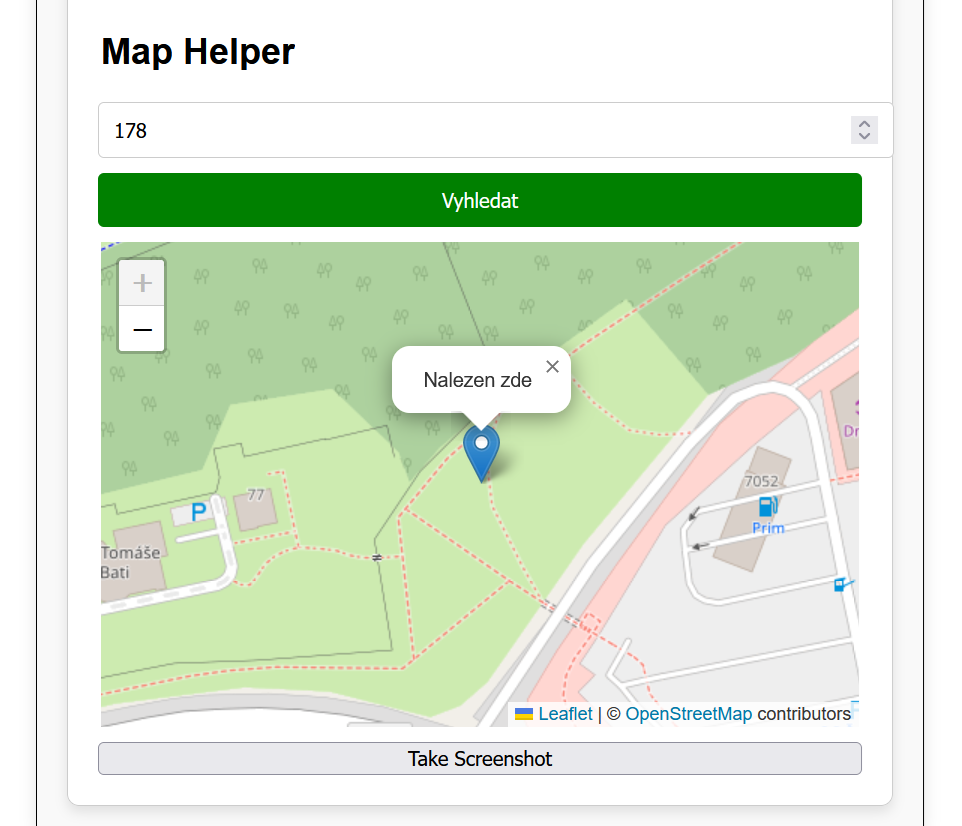
Třetí blok „Edit Record“ slouží k editaci již nahraných záznamů z webového rozhraní. Pod nadpisem se nachází vstupní pole, do kterého uživatel zadá identifikační číslo (ID) záznamu, který chce upravit, a po stisknutí tlačítka „Získat“ dojde k odeslání asynchronního požadavku *GET* na server, který do kolonek pod ním načte všechny ostatní údaje z databáze. Po provedení požadovaných změn může uživatel aktualizované údaje nahrát do databáze stiskem tlačítka „Změnit“, které spouští asynchronní požadavek *PUT* na server. Funkcionalitu zajišťuje robustní naslouchač událostí, který mimo jiné zpřístupňuje kolonky (kromě vstupního pole pro identifikační číslo) až po úspěšném načtení konkrétního záznamu. Naslouchač také kontroluje, zda uživatel nezadává do formuláře prázdné hodnoty a pokud ano, nahrání změn znemožní. Blok „Edit Record“ je pro ilustraci zobrazen na obrázku ORB níže.



Jako další je uveden blok pro editaci lokací. Blok „Edit Location“ funguje na stejném principu jako blok „Edit Record“. Uživatel nejprve zadá identifikační číslo (ID) lokace, podle kterého po kliknutí na tlačítko „Získat“ načte požadovaný záznam z tabulky *locations* v databázi. Hodnota pro anonymizaci je převedena na jednu z textových hodnot „Anonymizováno“ nebo „Neanonymizováno“, které je nutné použít i při případné editaci této kolonky. Po načtení záznamu se uživateli zpřístupní vstupní pole pro editaci záznamu, přičemž změněné, ne však prázdné, hodnoty může uložit do databáze stisknutím tlačítka „Změnit“. Funkcionalitu znovu zajišťuje robustní naslouchač událostí. Blok je zobrazen na obrázku BOR níže.



Předposlední blok prostředního sloupce má nadpis „Map Helper“ a uživatel do něj zadává identifikační číslo záznamu, pro který chce vyhledat a případně uložit mapu místa nálezu. Tato mapa je následně použita při generování PDF souboru s vybranými záznamy. V bloku se nacházejí dvě tlačítka oddělená prostorem pro zobrazení mapy. Nad horním tlačítkem je umístěno vstupní pole pro zadání identifikačního čísla záznamu, jehož místo nálezu bude po stisknutí tlačítka „Vyhledat“ zobrazeno. Spodní tlačítko „Take screenshot“ slouží k uložení obrázku s mapou ve formátu PNG do adresáře *mapy*. Každé tlačítko je obsluhováno samostatným naslouchačem událostí. Naslouchač pro tlačítko „Vyhledat“ spouští funkci *retrieveCoordinates*, která ze záznamu získaného asynchronním požadavkem na server extrahuje souřadnice a předává je funkci *initializeMap*. Ta podle těchto souřadnic vyhledá místo na stránkách OpenStreetMap a zobrazuje mapu s šipkou ukazující na přesné místo. Blok „Map Helper“ je zobrazen na obrázku BRB níže.



Poslední blok prostředního sloupce slouží ke generování PDF souboru na základě specifikovaného rozsahu identifikačních čísel záznamů. Uživatel do vstupních polí zadá identifikační čísla (ID) dvou záznamů, která určují spodní a vrchní hranici seznamu, dle nějž bude PDF souboru vygenerován. Po stisknutí tlačítka „Generate PDF“ je spuštěn naslouchač událostí, který pomocí asynchronního HTTP požadavku typu *POST* odesílá na server JSON soubor se těmito hodnotami. Server požadavek zpracuje a zajistí vytvoření požadovaného dokumentu. Grafickou podobu bloku „Generate PDF“ ukazuje obrázek ROB, zatímco podobu výsledného PDF dokumentu s testovacími daty nalezneme na obrázku OBB.



Soubor *upload.html*, který je k dispozici jako příloha UUU, obsahuje celkem 629 řádků.

# Nasazení aplikace

Aplikace byla nasazena na virtuální server, jehož pronájem byl zajištěn majitelem aplikace, doktorem Petrem Inkognitem, prostřednictvím společnosti SecurityNet.cz s.r.o. na stránkách hukot.net. Pro počáteční nasazení byl zakoupen virtuální server s 20 GB SSD úložištěm, 2 GB RAM a jedním procesorovým jádrem. Jako operační systém byla zvolena otevřená linuxová distribuce Ubuntu.

Server byl nakonfigurován vzdáleným přístupem z osobního počítače pomocí protokolu SSH. Nejprve byl na server nainstalován potřebný software, zejména Node.js, PostgreSQL, Python a ImageMagick. Dále byl nainstalován PM2, což je software s otevřeným zdrojovým kódem pro správu procesů Node.js, který zajišťuje nepřetržitý běh aplikace.

Zdrojový kód aplikace byl stažen z platformy GitHub, kde byl během vývoje zálohován. Poté byly pomocí příkazu npm install nainstalovány všechny závislosti uvedené v souboru package.json. Následně byly nainstalovány také knihovny jazyka Python, které aplikace využívá v podpůrném skriptu. Za pomoci souboru schema.sql, jehož obsah je podrobně popsán v kapitole schema.sql, byla vytvořena PostgreSQL databáze. Konfigurační soubor .env byl na serveru vytvořen ručně, neboť obsahuje citlivé údaje.

Po úspěšném spuštění aplikace byl nainstalován a nakonfigurován webový server s reverzní proxy Nginx, který má otevřený zdrojový kód. Bezpečnost serveru byla zajištěna nejen SSL (Secure Sockets Layer) certifikátem vystaveným pomocí nástroje Certbot, ale také nástavbovými bezpečnostními HTTP hlavičkami. Server byl nakonec propojen se zakoupenou doménou ctyrlistkoteka.cz. Funkčnost aplikace byla ověřena v produkčním prostředí a poté předána majiteli stránek.