# Vývoj webové aplikace

TEORIE

# Programovací jazyk JavaScript

TEORIE

# Programovací jazyk Python

TEORIE

# Databázové systémy

TEORIE

# PostgreSQL

TEORIE

# Jazyk SQL

TEORIE

# ImageMagick

TEORIE

# Architektura aplikace

Schéma

# 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, modul *fs* umožňující čtení a zápis souborů, klienta *pg* pro připojení k PostgreSQL databázi a práci s ní, modul *child\_proces* umožňující spuštění externích procesů, middlewary *body\_parser* pro zpracování těla HTTP požadavků a *cors* pro povolení Cross-Origin Resource Sharing. Dále je importována knihovna *jsonwebtoken* pro tvorbu a ověřování JSON Web Tokenů, modul *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. Dále jsou do konstant načteny citlivé údaje ze systémových proměnných a je také nastavena podpora pro zpracování požadavků prostřednictvím CORS a pro dekódování dat odesílaných prostřednictvím HTTP formulářů.

V poslední části kódu zajišťujeme bezpečný přistup do administrátorské části webového rozhraní. Nejprve je definována veřejně dostupná cesta k HTML stránce s přihlašovacím formulářem (viz kapitolu XXX). Zadá-li uživatel správné administrátorské heslo, které je uloženo v systémové proměnné, je mu vygenerován JSON Web Token sloužící k potvrzení jeho identity po dobu jedné hodiny. Zároveň je ověřený uživatel automaticky přesměrován na stránku s uživatelským rozhraním pro administrátorské činnosti. Cesta na tuto stránku je zabezpečena i přesto, že se nachází ve veřejně dostupném adresáři *public*.

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

Soubor *server.js* má i s komentáři jej vysvětlujícími celkem 474 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

Vstupním celý adresář public zpřístupněn jako statický, což umožňuje prohlížeči načítat HTML soubory, styly, skripty či obrázky.

Ve všech kapitolách jsou soubory popisovány „odspodu“, je tedy vždy nejdříve komentována část s podpůrným vnočeným JavaScriptovým kódem, poté je popsáná HTML struktura ve zbytku těla stránky, než je popsán design stránky zajištěný kaskádovými styly v hlavičce dokumentu doplněný o obrázek. Na všech stránkách je také použit externí bezpatkový font Raleway importovaný z Google Fonts.

Barevné ladění veřejných stránek je od tmavě zelené, přes světle zelenou až po žlutou.

Všechny veřejně dostupné stránky obsahují navigační menu reprezentované malým žlutým čtvercem v pravém horním rohu stránky s ikonou menu, který se po kliknutí na tento blok rozbalí do většího čtverce a zobrazí vždy tři odkazy na ostatní stránky. Po kliknutím na kterákoliv odkaze je uživatel automaticky přesměrován na danou stránku.

Z Chatu: „Soubor začíná deklarací typu dokumentu HTML5 a definicí základních metainformací, včetně kódování znaků (UTF-8) a zajištění responzivního zobrazení na různých zařízeních pomocí značky <meta name="viewport">. Dále je specifikován název stránky (<title>Database Records) a načteno bezpatkové písmo „Raleway“ ze služby Google Fonts pro sjednocení typografie.”

# 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.

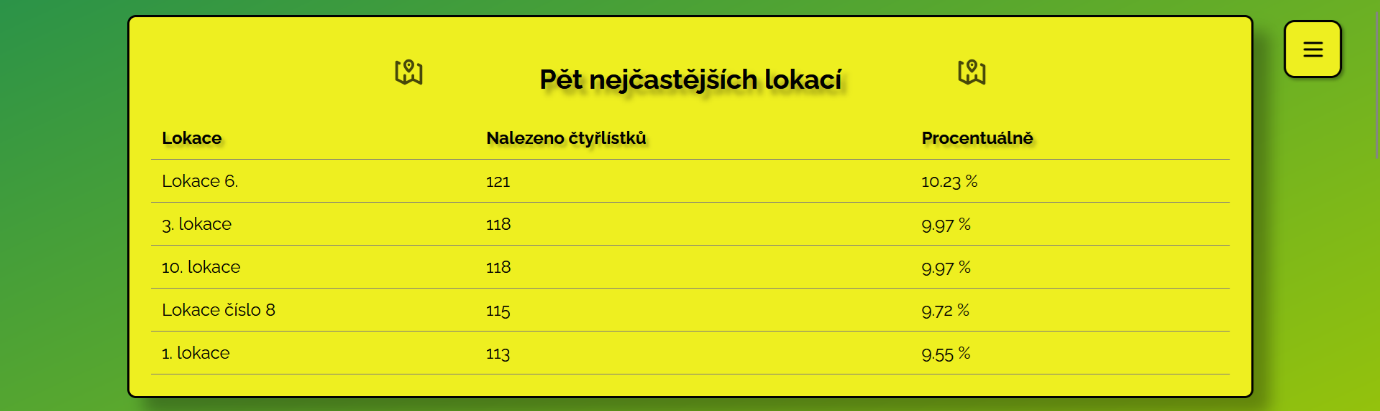
V horní části stránky se nachází již zaoblený a stínovaný blok se žlutým pozadím sloužící jako hlavní nadpis stránky (s jejím názvem tj. Seznam záznamů). Pod ním se nachází šest horizontálně seřazených prvků pro ovládání stránkování. Konkrétně se jedná o následující prvky jdoucí zleva doprava: tlačítko 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.

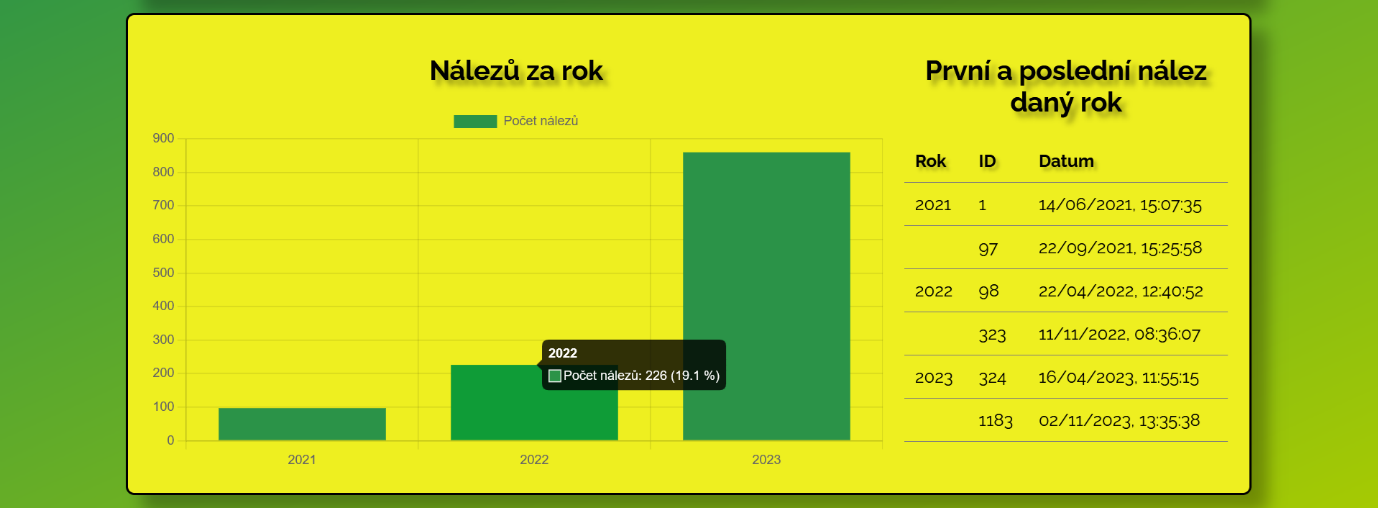
# statistics.html

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*.

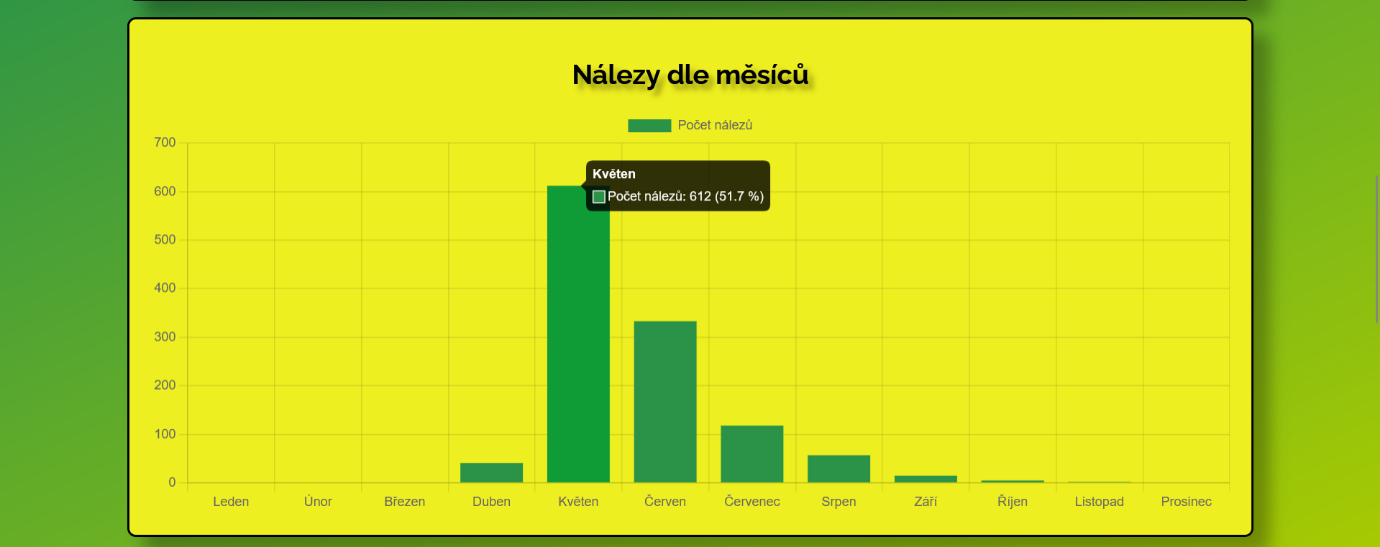
První z těchto funkcí je *fetchTopLocations()*, která vrací data o pěti lokacích, kde bylo nalezeno nejvíce čtyřlístků. Sestupně seřazené výsledky, jež obsahují název lokace, počet nalezených čtyřlístků a jejich procentuální podíl na celé sbírce, jsou dynamicky vykresleny do tabulky ve žlutém bloku. 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.



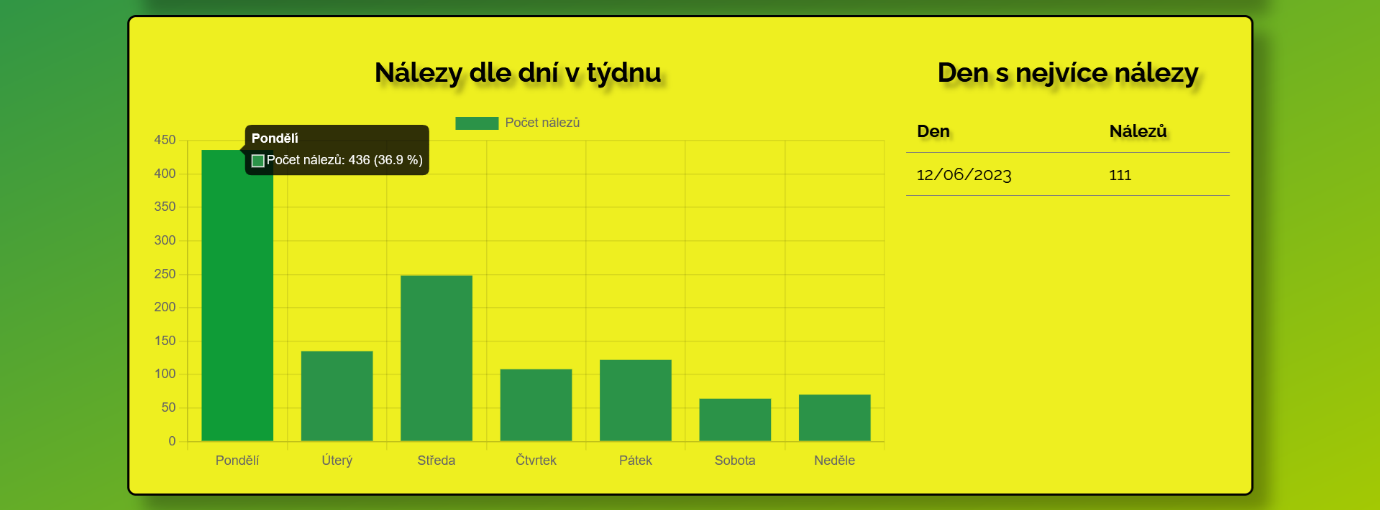
Druhý blok je vyplněn vizualizací dat získaných pomocí funkcí *fetchFindingsByYear()* a *fetchYearlyRecords()*. Větší část tohoto bloku, umístěná na levé straně, je věnována výstupu funkce *fetchFindingsByYear()*, která pod nadpisem „Nálezů za rok“ vykresluje sloupcový graf. Tento graf vizualizuje počet 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.



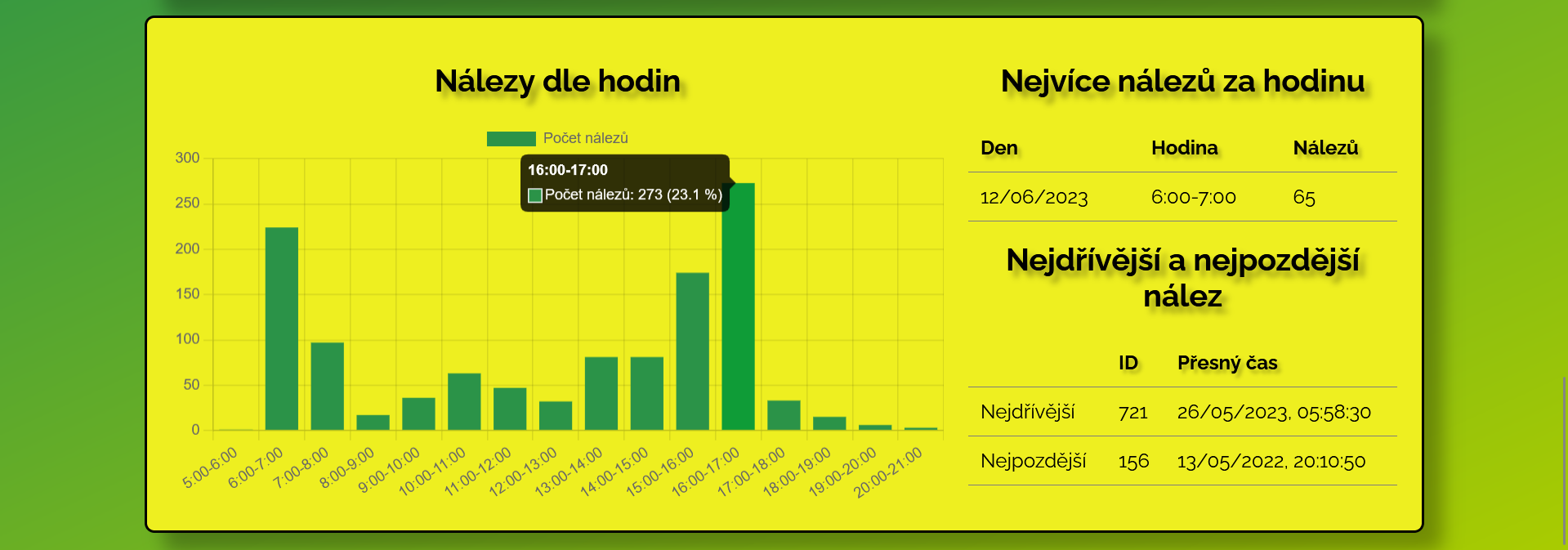
Třetí blok je nadepsán „Nálezy dle měsíců“ a jeho obsah je generován pomocí funkce *fetchFindingsByMonth()*. V rámci této funkce musela být inicializována dvě pole o délce dvanácti prvků. Do jednoho z nich jsou ukládány číselné hodnoty reprezentující měsíce převedené do indexového formátu (0-11), a to 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

Cod dělá?

Vnořený skript

Tělo HTML

Design

Soubor má XXX řádků a příloha . Podoba na obrázku AA1.

# upload.html

Cod dělá?

Vnořený skript

Tělo HTML

Design

Soubor má XXX řádků a příloha . Podoba na obrázku AA1.