# 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.

Opravit odstavec podle chata a pokračovat na řádku 138

# pdfGenerator.py

Proč python a ne JS?

# Adresář public

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