# Gymnázium, Praha 6, Arabská 14

Programování

# MATURITNÍ PRÁCE

**2024/25** **Josef Mitošinka**

**Gymnázium, Praha 6, Arabská 14**

Arabská 14, Praha 6, 160 00

# MATURITNÍ PRÁCE

Předmět: Programování

Téma: Aurora Explorer

Autor: Josef Mitošinka

Třída: 4.E

Školní rok: 2024/25

Vedoucí práce: Mgr. Jan Lána

Třídní učitel: Mgr. Blanka Hniličková

## Čestné prohlášení:

Prohlašuji, že jsem jediný autor tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené.

Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů udělujeme bezúplatně škole Gymnázium, Praha 6, Arabská 14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V Praze dne ..... .....

# Anotace

Aurora Explorer představuje webovou aplikaci, vytvořenou v rámci maturitní práce, která je zaměřena na sledování a sdílení informací o polární záři. Uživatel má možnost vidět aktuální i budoucí informace o polární záři a předpověď dalších faktorů, které její viditelnost ovlivňují. Jedna z hlavních možností je sdílet pořízené fotografie s jejich lokací, aby ostatní uživatelé měli možnost najít vhodné místo na pozorování a podívat se na polární záři v dané lokalitě.

# Abstract

Aurora Explorer is a web application developed as part of a seminary work, focused on watching and sharing information about the aurora borealis. Users can view the current and future information about the aurora as well as forecasts of other factors affecting its visibility. One of the main features is the option to share captured photos with their locations, allowing other users to find suitable observation spots and see the current appearance of the aurora in a given location.

# Zadání maturitní práce

Zadáním mé maturitní práce je vytvořit webovou aplikaci zaměřenou na sledování polární záře. Cílem projektu je poskytnout uživatelům všechny potřebné informace k pozorování polární záře. Aplikace bude obsahovat informace o intenzitě (Kp index), oblačnosti a aktuálním počasí, které ovlivňují viditelnost polární záře.

Uživatelé budou mít možnost registrace a přihlášení pomocí e-mailu. Dále bude aplikace obsahovat funkci notifikací, které uživatele upozorní na vysokou intenzitu polární záře prostřednictvím e-mailu.

Součástí webu bude také komunitní sekce, kde mohou uživatelé sdílet fotografie polární záře. Každá přidaná fotografie bude obsahovat informace o čase a místě pořízení, což uživatelům poskytne inspiraci a možnost objevovat vhodná místa k pozorování. Kromě toho bude aplikace obsahovat sekci s tipy na pozorování.

Data pro aplikaci budou čerpána zejména z NOAA (National Oceanic and Atmospheric Administration).

# Obsah

# 1 Úvod

Polární záře patří mezi nejúžasnější přírodní fenomény, které lidstvo pozoruje už od dávných dob. Tento světelný jev, který se objevuje v polárních oblastech na nebesích, je výsledkem interakce slunečního větru s magnetosférou Země. V průběhu historie byla polární záře často spojena s různými mýty a legendami, než byla vědecky vysvětlena.

Přestože je známo, jak polární záře vzniká a díky pokročilým technologiim je i možno její výskyt předpovědět, je stále poměrně obtížné ji zahlédnout na obloze. Mnoho faktorů, jako například oblačnost, může ovlivnit, zda bude možné tuto neuvěřitelnou světelnou podívanou spatřit.

Moje maturitní práce umožňuje uživatelům získat potřebné informace a předpovědi, aby měli co nejvyšší šanci polární záři spatřit. Kromě vědeckých dat mohou uživatelé také vidět aktuální fotografie ostatních uživatelů spolu s místy, kde byly pořízeny, čímž se ještě zvyší šance na úspěšné pozorování.

# 2 Použité technologie

Backend aplikace je postaven na výkonném JavaScriptovém runtime enviromentu Node.js a frameworku Express, který umožňuje snadnou tvorbu serverových aplikací. Node.js podporuje použití MVC architektury (Model-View-Controller), kterou jsem v mém projektu implementoval. Tato architektura přináší přehledné rozdělení logiky aplikace na tři hlavní vrstvy: modely, které reprezentují schéma jednotlivých modelů v databázi, kontrolery, které zpracovávají požadavky a upravují data v databázi, a šablony (view) ve formátu EJS, které generují dynamický HTML obsah zobrazovaný uživatelům. Primárně používaný programovací jazyk je JavaScript, který se používá jak na straně serveru, tak na klientské části.

Jako databázový systém jsem zvolil MongoDB. MongoDB je NoSQL databáze, která ukládá data ve formátu JSON-like dokumentů (BSON). Na rozdíl od relačních databází (jako MySQL nebo PostgreSQL), MongoDB nevyužívá tabulky, ale místo toho ukládá data jako flexibilní dokumenty. Díky tomu je MongoDB ideální pro moji aplikaci, kde se struktura dat může často měnit.

Frontend aplikace je tvořen šablonovacím enginem EJS (Embedded JavaScript). EJS mi umožňuje jednoduše generovat obsah založený na aktuálních datech z databáze a přizpůsobit zobrazení konkrétním uživatelům.

Pro stylování aplikace jsem využil CSS framework Tailwind CSS, který mi umožňuje rychle a efektivně implementovat styly do HTML kódu, čímž jednoduše mohu zajistit vzhled a responzivitu jednotlivých stránek.

## 2.1 Struktura aplikace

Aplikace je dělena do jednotlivých modulů, které nesou odpovědnost za jednotlivé části aplikace. Seznam modulů je popsán na další straně.

| Soubor | Popis |
|---|---|
| server.js | Spouští HTTPS server, definuje cesty, navazuje spojení s databází, nastavuje middleware a spravuje statické soubory a skripty. |
| dbConfig.js | Configuruje připojení databáze. |
| postController.js | Ovladač pro operace s daty příspěvků. |
| userController.js | Ovladač spravující operace s uživatelskými daty. |
| attachUser.js | Authentikuje uživatele a připojení uživatele k objektu req.user. |
| logEvents.js | Zapisuje provedené požadavky na server. |
| upload.js | Nahrává soubory na server. |
| Post.js | Model příspěvku. |
| User.js | Model uživatele. |
| aurorexRoutes.js | Zpracovává cesty operací na sociální síti. |
| userRoutes.js | Zpracovává cesty uživatelských operací. |
| routes.js | Zpracovává hlavní cesty aplikace. |
| kpForecast.js | Zajišťuje získání dat o KP indexu a vytvoření grafů. |
| weatherData.js | Zajišťuje získání dat o počasí a jejich zobrazení. |
| kpNotificationService.js | Odesílá emaily při výskytu vysokého KP indexu. |
| notificationService.js | Odesílá emaily při zveřejnění příspěvku v lokaci uživatele. |

# 3 Řešení problematiky

Pozorování polární záře je krásný a vzrušující zážitek, ovšem spatřit onu podívanou na obloze může být velice obtížné. Najít vhodné místo, odkud je dobrý výhled a zároveň nízké světelné znečištění z měst, není jednoduché. Když vezmeme v potaz ještě oblačnost a výskyt vysokého KP index (KP index určuje intenzitu aktivity polární záře), stává se z toho poměrně komplikovaný úkol. Proto jsem vytvořil aplikaci Aurora Explorer, která umožňuje uživatelům vidět aktualní i budoucí data o intenzitě polární záře a oblačnosti. Spolu s aktuálními fotografiemi, pořízenými ostatními uživateli, a mapou výskytu těchto příspěvků se Aurora Explorer stává jedinečnou webovou aplikací pro pozorovatele polárních září.

Uživatelé bez přihlášení mají možnost vidět předpověď KP indexu a procentuální oblačnost určenou podle jejich aktuální polohy. Také se mohou podívat do sekce pro tipy a rady, aby se mohli adekvátně na pozorování připravit. Po přihlášení se uživateli otevře možnost zhlédnout sekci Aurorex, která obsahuje fotografie pořízené a nahrané ostatními uživateli. Následně má možnost prohlédnout si příspěvky, které může okomentovat či označit, že se mu líbi. Kromě této sekce se zde nachází ještě živá mapa, na které je přehled všech lokací, které uživatelé označili ve svém příspěvku jako místo, odkud polární záři spatřili. Aby se nemohlo stát, že uživatel promešká příležitost vidět tento ohromující přírodní jev, může si ve svém profilu nastavit notifikace přes email v případě vysoké intenzity KP indexu. Uživatel má také možnost dostat notifikaci v případě, že v uživatelem zvolené libovolné lokaci je nahrán nový příspěvek.

# 4 Hlavní části webové aplikace

## 4.1 Forecast

Stránka /forecast je jednou z několika hlavních stránek, které má aplikace poskytuje. Zde má uživatel, i bez přihlášení, možnost vidět grafy s předpovědí KP indexu na aktuální i následující den spolu s dvěma mapami, které obsahují stávající a budoucí aktivitu polární záře na severní polokouli. Také je možnost prohlédnout si informace o počasí, jako jsou teplota, vlhkost vzduchu, větrnost a hlavně oblačnost, která je jedním z nejvíce ovlivňujících faktorů viditelnosti polární záře.

Data na počasí jsou čerpána z OpenWeatherMap pomocí jejich Weather API. Data o aktuálním počasí jsou zobrazena v elementu '`<div id="current-weather">`' a předpověď v elementu '`<div id="forecast-weather">`'. Aktualizaci dat v elementech zajišťuje script weatherData.js, který je inicializován ihned při načtení stránky. Pro správné fungování dat o počasí je potřeba povolit aplikaci informace o aktuální poloze, o které při načtení sama zažadá, jelikož data o počasí jsou hledána podle aktuální polohy uživatele. V případě nepovolení nebo nereagování na žádost data nebudou načtena.

Data pro grafy s hodnotami KP indexu jsou získávána z NOAA Space Weather Prediction Center. Obdržená data jsou následně zpracována ve scriptu kpForecast.js, ze kterého jsou poté načtena. Na samotné vytvoření grafu používám knihovnu Chart.js. Každý graf má vlastní element; graf pro dnešek je id=”kpChart1”a pro zítřek se jedná o ”kpChart2”.

```
1 <canvas id="kpChart1" class="w-full h-64"></canvas>
```

Listing 1: Element s aktuální tabulkou KP indexu

Fotografie s mapou výskytu polární záře na severní polokouli jsou také z NOAA Space Weather Prediction Center. Každá fotografie je uložena v elementu.

```
1 <img id="aurora-map-today" src="https://services.swpc.noaa.gov/images/
    animations/ovation/north/latest.jpg" class="absolute inset-0 w-full h-full
    object-cover">
```

Listing 2: Element s aktuální mapou polární záře

Fotografie jsou automaticky aktualizovány pomocí funkce '`refreshImages()`', která zajišťuje obnovení fotografií každých 10 minut. Obě mapy obsahují malou ’i’ ikonku, která odkazuje na zdroj daného obrázku.

## 4.2 Aurorex

/Aurorex je další esenciální stránka pro můj program. Nejprve se uživatel musí přihlásit, aby bylo možné zpřístupnit tuto stránku. /Aurorex je samotnou sociální sítí, kde mohou uživatelé nahrávát a zobrazovat příspěvky, commentovat je, či likovat. Obsahuje také postupné, nekonečné načítání již nahraných příspěvků.

V případě, že se uživatel snaží zpřístupnit stránku bez přihlašení, bude přesměrován na /login, tedy stránku, která slouží k přihlášení. Po přihlášení se uživateli zobrazí stránka, v jejíž horní části je možnost Create Post, kterou je uživatel přesměrován na /aurorex/post, kde si může nadefinovat svůj příspěvek a zveřejnit ho na síti. Vedle této možnosti se nachazí filtr.

Při nahrávání příspěvku uživatel musí zvolit místo na mapě nebo napsat adresu ručně do vyznačeného pole a vybrat fotografii, kterou chce publikovat. Následně má možnost napsat popisek a poté fotografii nahrát na server. Po nahrátí jsou data uložena do databáze a samotná fotografie, kterou do databáze nelze nahrát, je uložena middlewarem upload.js do složky /public/uploads pomocí javascriptové knihovny `multer` a cesta k dané fotografii je uložena do databáze [14] [34] [23].

Filtr slouží k možnosti seřazení příspěvků dle tří parametrů; dle času od nejnovějších, podle počtu liků a podle počtu commentářů, z čehož defaultně je nastaven filtr newest, tedy od nejnovějšího příspěvku.

Níže se uživateli načítají příspěvky pomocí infinite scrollu. Tedy když se uživatel posune na konec stránky, spustí se funkce ‘`loadMorePosts()`‘, která odešle požadavek na server s aktuální stránkou, maximálním počtem načtených příspěvků na jeden request, tedy 6 příspevků a aktuálně vybraným filtrem. Server vrátí další sadu příspěvků, které se následně zobrazí níže na stránce. Pokud již nejsou dostupné další příspěvky, načítání se zastaví. Při změně filtru se seznam příspěvků resetuje a načítá se znovu od začátku. Pro zabránění opakovaného načítání jsou zde implementovány 2 proměnné. Proměnná ‘`loading`‘ pomocí true nebo false určuje, jestli se příspěvky aktuálně načítají a proměnná ‘`hasMore`‘ kontroluje, zdali existují další příspěvky, které by bylo možné zobrazit.

Následně u každého příspěvku je potřeba přeměnit souřadnice uložené v databázi na uživatelsky čitelnou lokaci. To zajišťuje funkce ‘`geocoder()`‘. Ta nejprve najde všechny elementy ‘`location-display`‘ a vezme zeměpisnou šířku a délku uloženou v jejich datech. Pomocí třídy ‘`Geocoder()`‘ z Google Maps JavaScript API knihovy mám možnost přes funkci ‘`geocode()`‘ konvertovat souřadnice do adres, které následně zobrazím v elementu

‘location-text‘ [16] [5].

### 4.3 Live-map

Webová stránka /aurorex/live-map slouží k zobrazení mapy, na které jsou označené lokace aktuálních příspěvků nahraných uživateli. Stejně jako u /aurorex se uživatel musí nejprve přihlásit, aby mu byl umožněn přístup na stránku. Po příhlášení se uživateli zobrazí mapa s příspěvky nahranými za posledních 24 hodin. Příspěvky jsou na mapě zobrazeny ve formě markeru, který se po kliknutí zvětší a zobrazí se uživateli pole s fotkou příspěvku, jménem uživatele, co příspěvek nahrál, datumem nahrání a popiskem, který je k příspěvku při jeho vytváření vložen.

Při načtení stránky se zavolá funkce ‘initMap()‘, která pomocí třídy ‘Map()‘ z Google Maps JavaScript API knihovny inicializuje mapu. Funkce následně zavolá další funkci ‘loadPosts()‘, která odešle požadavek na server spolu se zvoleným časovým filtrem.

```
1 const response = await fetch(‘/aurorex/api/posts-by-time?range=${range}‘);
```

Listing 3: Cesta pro vyžádání příspěvků pro live-map

Následně router v aurorexRoutes na odeslaný požadavek zareaguje zavoláním funkce ze souboru postController ‘timeMapFilter‘. Ta spočítá čas ‘startDate‘, od kterého se mají příspěvky načítat, pomocí zadaného parametru $range a s využitím operátoru $gte (greater than or equal) implementovaném v MongoDB najde veškeré příspěvky v kolekci posts, které mají čas vytvoření větší než spočítaný ‘startDate‘. Následně vratí vytříděné příspěvky ve formě JSON a ty jsou zobrazeny na stránce ve formě markerů.

Google Maps JavaScript API knihovna poskytuje třídy ‘Marker()‘ a ‘InfoWindow()‘. Třída ‘Marker()‘ mi umožňuje vytvořit samotný marker a nadefinovat si jeho vzhled a lokaci na mapě a třída ‘InfoWindow()‘ jednoduše vytvoří po kliknutí na marker malé okénko, jehož obsah si mohu customizovat a vložit do něj požadované informace o příspěvku [18].

Filtrovat příslevky podle času lze podle tří kritérií; příspěvky za posledních 24 hodin (’24 hours’), za poslední týden (’past week’) a za poslední měsíc (’past month’).

### 4.4 Guide

Stránka /guide je určena primárně jako informační stránká, kde se nachazí informace, které jsou esenciální či mohou být užitečné při výpravě za pozorováním polární záře. Obsahuje informace o nejlepších lokalitách, optimálním čase, tipech na fotografování a nezbytné přípravě pro úspěšné sledování aurory borealis. /Guide pomáhá uživatelům lépe

se orientovat v podmínkách esenciálních k jejímu pozorování a poskytuje praktické rady, jak se na výpravu připravit a jak se vrátit s nejlepšími zaběry.

# 5 Řešení klíčových problémů

## 5.1 Autentikace uživatele

Autentikace je proces ověření identity uživatele. Jejím cílem je se ujistit, že uživatel, který se snaží přihlásit, je opravdu tím, za koho se vydává. Autentikace na mé webové aplikaci probíhá standartím způsobem pomocí u a hesla.

Registrace začíná na stránce /users/register, kde uživatel zadá své údaje, tj. uživatelské jméno, e-mail, heslo a heslo po druhé, pro oveření, že se uživatel při zadávání nepřepsal. Po předložení dat na server router v userRoutes.js zareaguje zavoláním funkce ‘`register`‘ z controlleru userController.js. Ta nejprve ověří, zdali zadaný e-mail není již v databázi uložen, a následně je využita knihovna `bcrypt`, která implementuje hashovací algoritmus pro bezpečné ukládání hesel. Pomocí bcrypt je heslo zahashováno a je k němu přidáno 10 salt rounds (k heslu se před hashovaním přidá 10 náhodných znaků), aby byla zajištěna jedinečnost hashe hesla.

```
1 const hashedPassword = await bcrypt.hash(password, 10);
```

Listing 4: Hashování hesla se solí

Následně jsou uživatelem zadaná data spolu s zahashovaným heslem uložena do databáze a uživatel je přesměrován na stránku /users/login, která slouží k přihlášení.

Na této stránce musí uživatel zadat e-mail, který použil při registraci, a heslo k němu vytvořené. Po předložení dat router v userRoutes.js zavolá funkci login z controlleru usersController.js, která nejprve ověří existenci e-mailu v databázi a následně pomocí funkce ‘`compare()`‘ z knihovny `bcrypt` porovná, zdali zadané heslo odpovídá zahashovanému heslu pro daný e-mail v databázi. V případě shody je využita funkce ‘`sign()`‘ z knihovny `jwt` (jsonwebtoken), která vytvoří JSON Web Token. Nejprve jsou do argumentu funkce vložena uživatelská data, následně je třeba token podepsat tajným klíčem, který je uložen v .env souboru a chrání integritu tokenu, a nakonec je nadefinováno časové rozmezí, po jehož uplynutí platnost tokenu vyprší a uživatel se musí znovu přihlásit. Následně je JWT token uložen do cookie s parametrem ‘`httpOnly: true`‘, což zamezuje přístup ke cookie v JavaScriptu na frontendu [20][25].

```
    //create JWT with user detail and sign it
    const accessToken = jwt.sign(
      { id: user._id, userName: user.userName, email: user.email },
      process.env.ACCESS_TOKEN_SECRET, //secret key for signing token
      { expiresIn: '1h' }
    );
    //set JWT as a cookie named authToken
    res.cookie('authToken', accessToken, { httpOnly: true });
```

Listing 5: Vytvoření JWT tokenu a jeho uložení do cookie

Následně je třeba připojit uživatele z cookie do objektu, aby bylo možné s nim pracovat. O to se stará middleware attachUser.js. Ten vezme uživatelská data uložená v cookie 'authToken' a nejprve pomocí funkce '`verify()`' z `jwt` knihovny ověří platnost tokenu. V případě, že token je platný, najde v databázi uživatele s id, které bylo uložené v cookie. Následně uživatelská data připojí k objektu req.user.

```
  try {
    const decoded = jwt.verify(token, process.env.ACCESS_TOKEN_SECRET); //
    verify the token using secret token
    const user = await User.findById(decoded.id);
    if (!user) {
      req.user = null;
    } else {
      //if user found, attach it to object user
      req.user = user;
    }
  }
```

Listing 6: Ověření JWT tokenu a připojení uživatele k objektu req.user

## 5.2 Získávání dat o KP indexu a vytváření grafů

Aby bylo možné na stránce /forecast zobrazovat grafy s předpovědí KP indexu, je třeba data nejprve získat z důvěrného zdroje a následně z nich vytvořit grafy. O to se stará script kpForecast.js. Data jsou získana ze spolehlivého zdroje, jedná se o SWPC od NOAA, tedy Space Weather Prediction Center, který je pod správu National Oceanic and Atmospheric Administration.

Ve scriptu se nachází dvě hlavní funkce, '`fetchKpForecastToday()`' a '`fetchKpForecastTomorrow()`'. Obě funkce získávají data z třídenní předpovědi od SWPC ve formě textového souboru. Funkce nejprve pošlou požadavek na stránku a následně obdrží textový soubor [31].

```
const response = await fetch('https://services.swpc.noaa.gov/text/3-day-forecast.txt');
```

Listing 7: Získávání dat o KP indexu

Data ze souboru je nutné upravit a odstranit nepotřebné informace. Každá funkce data zpracovává jinak, neboť ‘`fetchKpForecastToday()`‘ vyhledává první sloupec v tabulce, která je součástí třídenní předpovědi, a ‘`fetchKpForecastTomorrow()`‘ druhý sloupec. U ‘`fetchKpForecastToday()`‘ probíhá vybírání dat následovně; soubor se prochází řádek po řádku než se narazí na větu, která je řádek před tabulkou. Následně jsou využity regulární výrazy, aby byly definovány řádky, které odpovídají požadavkům. V případě nalezení takové řádky, je první část řádky (získaný čas a velikost KP indexu jsou odděleny mezerou) uložena do proměnné time a druhá do proměnné kpIndex. Následně jsou data vložena do array ‘`timeKpMap1`‘. Z proměnné time je třeba odebrat poslední dvě písmena, jelikož obsahuje zkratku časové zóny na konci, tj. UT.

```
const response = await fetch('https://services.swpc.noaa.gov/text/3-day-forecast.txt');
const textData = await response.text();
const lines = textData.split('\n');
const timeKpMap1 = {};

for (const line of lines) {
  if (line.includes('NOAA Kp index breakdown')) continue;
  const match = line.match(/(\d{2}-\d{2}UT)\s+([\d.]+)/);
  if (match) {
    const time = match[1].trim();
    const kpIndex = Math.round(parseFloat(match[2].trim()));
    timeKpMap1[time.slice(0, -2)] = kpIndex;
  }
}
```

Listing 8: Třídění dat ze souboru forecast.txt a vkládání hodnot do timeKpMap1

Po vytvoření objektu, který obsahuje dané časy a velikosti KP indexu, je zavolána funkce ‘`renderChart1(timeKpMap1)`‘. Ta nejprve najde element grafu v souboru forecast.ejs. Následně nadefinuje hodnoty grafu, x-ová souřadnice je časové rozmezí a y-ová jsou hodnoty KP indexu, a barvy odpovídající daným velikostem sloupců KP indexu. Dále je nadefinován samotný graf ‘`myChart`‘, který je vytvořen pomocí knihovny `Chart.js`[1][22][15]. Podobný proces probíhá u funkce ‘`fetchKpForecastTomorrow()`‘. Zpracování dat při výběru

hodnot KP indexu z třídenní předpovědi probíhá odlišně, jelikož se získává z druhého sloupce tabulky.

Poté následuje rozesílání e-mailů. Array ‘`timeKpMap1`‘ je odeslána na serverový endpoint ‘`/update-kp-data`, kde je následně zpracován routerem v routes.js a e-maily jsou rozeslány.

Nakonec jsou obě funkce exportovány jako globální, aby bylo možné je využít v souboru forecast.ejs.

### 5.3 Získávání dat o počasí

Aby bylo možné na stránce /forecast zobrazovat data o počasí, je potřeba je někde získat a upravit je tak, aby bylo možné je zobrazit. O to se stará script weatherData.js. Data o počasí, jako oblačnost, teplota, vlhkost a rychlost větru jsou získávána ze spolehlivého zdroje OpenWeatherMap. OpenWeatherMap poskytuje jejich Weather API, které obsahuje všechny základní informace o počasí, které jsou pro moji stránku důležité.

Tento soubor se skládá ze 4 hlavních funkcí, z nichž první je ‘`fetchWeatherData()`‘. Do parametru této funkce se vkládá zeměpisná šířka, délka a API klíč, který je potřebný k získání dat z Weather API. Následně funkce pošle request na API a odpověď vratí ve formátu JSON.

```
1    const response = await fetch(
2      `https://api.openweathermap.org/data/2.5/forecast?lat=${latitude}&lon=${
     longitude}&appid=${apiKey}&units=metric`
3    );
```

Listing 9: OpenWeatherMap API požadavek

Další z funkcí je ‘`getCurrentWeather()`‘, která zajišťuje zpracování získaných dat. Z odpovědi z API vybere potřebné informace a zpracuje je tak, aby se s nimi jednodušeji pracovalo. Získaná a zpracovaná data jsou teplota, která je zaokrouhlena, oblačnost, vlhkost, ikonka počasí, rychlost větru, popis počasí a 4 předpovědi na dalších 12 hodin v 3-hodinových intervalech [29].

Následně se zpracovana data musí připravit i vzhledově, aby bylo možné je zobrazit na stránce /forecast. O to se stará funkce ‘`updateWeatherDisplay()`‘. Ta rozdělí data na dvě hlavní části, aktuální, která obsahuje veškeré informace o aktuálním počasí z ‘`getCurrentWeather()`‘ a předpověď, která obsahuje pouze oblačnost, ikonku počasí a teplotu.

Funkce, která všechny výše zmíněné funkce spojuje dohromady a inicializuje samotné

fungování předpovědi je ‘initWeatherTracking()‘. Ta nejprve najde potřebné HTML elementy na zobrazovaní dat nebo errorů a poté zkontroluje, že spojení mezi klientem a serverem je přes síťový protokol HTTPS, který je pro využívání API nutný [9]. Následně určí aktuální lokaci uživatele pomocí geolokační API v prohlížeči (je nutné udělit serveru povolení k lokaci, jinak předpověď počasí nebude inicializována) a se získanými souřadnicemi a privátním OpenWeatherMap API klíčem v parametru je zavolána funkce ‘fetchWeatherData()‘.

```
1    const position = await new Promise((resolve, reject) => {
2      navigator.geolocation.getCurrentPosition(resolve, reject);
3    });
```

Listing 10: Získání lokace z prohlížeče

Následně je zavolána funkce ‘getCurrentWeather()‘, do jejíhož parametru je vložena odpověď z předchozí funkce. Nakonec se pomocí ‘updateWeatherDisplay()‘ získaná a zpracovaná data zobrazí. Data se automaticky obnovují v časovém intervalu 1800000ms, tedy 30 minut.

## 5.4 Zasílání e-mailů při vysokém KP indexu

V případě, že se vyskytne v předpovědi KP indexu na aktuální den časový interval s intenzitou KP vyšší nebo rovno 5, proběhne rozesílání e-mailů všem uživatelům, kteří maji v databázi u atributu notificationsForHighKp hodnotu true, tedy mají ve svém profilu zapnuté rozesílaní e-mailu při KP $>= 5$.

Tento proces zajišťuje služba kpNotificationService.js. Rozesílání e-mailu probíhá skrze knihovnu nodemailer, která umožňuje spojení s e-mailovým účtem a posílání e-mailů využitím připojeného účtu [27] [13].

Při vytváření instance třídy kpNotificationService se v konstruktoru inicializuje nodemailer transport s využitím služby gmail a nastaví se přihlašovací údaje k gmail účtu, které jsou načteny ze soukromého .env souboru. Zároveň se vytvoří mapa s časy odeslaných e-mailů jednotlivým uživatelům, což zabezpečí přehled, který uživatel již e-mail pro daný den dostal, aby se rozesílání týmž uživatelům neopakovalo.

Před rozesíláním e-mailu je třeba zjistit, kteří uživatelé mají notifikace zapnuté a o to se stará metoda ‘findUsersWithKpNotifications()‘.

Dále metoda ‘sendHighKpAlert()‘, která má jako parametr uživatele a KP data, která budou v e-mailu. Nejprve zkontroluje, zdali uživatel, který má obdržet e-mail, již neobdržel

jeden za posledních 24 hodin. V případě, že žadný neobdržel, průběh metody pokračuje. Následně v mapě KP dat, která obsahuje časový úsek a hodnotu KP indexu, najde úsek s nejvyšší hodnotou KP indexu a poté je nakonfigurovan samotný e-mail, tj. odesílatel, příjemce, předmět a obsah e-mailu. Data z mapy KP dat jsou vložena do e-mailu spolu se samostatným řádkem, kde je zmíněn, dříve zjištěný, časový úsek s nejvyšší hodnotu KP indexu. Po nadefinování e-mailu je e-mail poslán a v mapě časů poslaných e-mailů je upraven čas odeslání posledního e-mailu.

Poslední metoda '`checkAndNotifyUsers()`', jejíž parametrem je mapa časů a hodnot KP indexu, nejprve zkontroluje, zdali se v mapě nachazí hodnota KP indexu >= 5. Pokud ano, jsou z databáze vybráni všichni uživatelé, kteří mají zapnuté notifikace na vysoký KP index, tedy u atributu `notificationsForHighKp` mají hodnotu true. Následně je pro každého jednotlivého uživatele zavolána metoda '`sendHighKpAlert()`', která dostane jako parametr daného uživatele a mapu KP dat.

API endpoint POST /update-kp-data je zpracován routerem v routes.js. Script kpForecast.js na tento endpoint pošle mapu s KP daty, načež je zavolána metoda '`checkAndNotifyUsers()`'. V jejím parametru je vložena mapa KP dat, která byla vytvořena při vytváření grafu scriptem kpForecast.js.

```
router.post('/update-kp-data', async (req, res) => {
  try {
    const { kpData } = req.body;
    console.log('Recieved kp data: ', kpData);

    const kpNotificationService = require('../services/kpNotificationService');
    await kpNotificationService.checkAndNotifyUsers(kpData);

    res.json({ success: true });
  } catch (error) {
    console.error('Error handling kp data:', error);
  }
});
```

Listing 11: /update-kp-data endpoint v routes.js

## 5.5 Zasílání e-mailů při výskytu příspěvku v nastavené lokaci

Uživatel má ve svém profilu možnost označení libovolné oblasti. Na zobrazené mapě označí místo, jenž slouží jako střed kružnice, u které následně může měnit poloměr v

kilometrech. V případě, že je nahrán příspěvek, u kterého je označená lokace pořízení fotografie uvnitř zvolené oblasti, uživatel má možnost obdržet informační e-mail. E-mail obdrží uživatel v případě, že má zapnuté notifikace na zasílání e-mailů v označené oblasti. O tento proces se stará služba notificationService.js.

Pro rozesílání e-mailu je, stejně jako u notifikací na vysoký KP index, využita knihovna `nodemailer` [27] [13].

Při vytváření instance třídy NotificationService, podobně jako u KpNotificationService, se v konstruktoru inicializuje nodemailer transport s využitím služby gmail a nastaví se přihlašovací údaje k gmail účtu, které jsou načteny ze soukromého .env souboru.

Metoda '`calculateDistance()`', do jejíhož parametru se vloží zeměpisná délka a šířka dvou lokací a následně spočíta vzdálenost těchto dvou bodů. Pro vypočítání vzdálenosti se využívá Haversinuv vzorec, který umožňuje spočítání vzdálenosti dvou bodů na kouli (Zemi) [2].

```
1  calculateDistance(lat1, lon1, lat2, lon2) {
2    const R = 6371; //radius of earth
3    const dLat = (lat2 - lat1) * Math.PI / 180; //latitude difference to
     radians
4    const dLon = (lon2 - lon1) * Math.PI / 180; //longtitude difference to
     radians
5    const a = Math.sin(dLat / 2) * Math.sin(dLat / 2) + Math.cos(lat1 * Math.PI
      / 180) * Math.cos(lat2 * Math.PI / 180) * Math.sin(dLon / 2) * Math.sin(
     dLon / 2); //the Haversine formula
6    const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a)); // angular
     distance in radians
7    return R * c; //distance in kilometers
8  }
```

Listing 12: Metoda pro spočítání vzdálenosti dvou bodů na kouli

'`FindUsersInRange()`' metoda s parametrem zeměpisné šířky a délky vytvořeného příspěvku slouží k nalezení uživatelů, kteří mají zapnuté notifikace na příspěvky v oblasti, tedy v databázi u atributu notificationsEnabled mají hodnotu true a zároveň mají nastavenou svoji uživatelskou lokaci. Následně pro každého uživatele, který odpovídá těmto požadavkům, zavolá metodu '`calculateDistance()`' a zjistí, jestli byl příspěvek vytvořen v jejich lokaci či mimo.

Poslední metoda '`sendAuroraAlert()`' má v parametru objekt obsahující uživatele, u kterého je příspěvek uvnitř uživatelské lokace, a objekt s daty o vytvořeném příspěvku. V

metodě se nejprve nadefinuje e-mail a následně je odeslán uživateli, který byl v objektu v parametru.

Tato třída NotificationService je využita ve funkci ‘`createPost()`‘ v controlleru postController.js, která zpracovává vytvoření příspěvku v databázi. Po vytvoření příspěvku je zavolána metoda ‘`findUsersInRage()`‘ a následně pro každého uživatele je zavolána metoda ‘`sendAuroraAlert()`‘, aby obdrželi e-mail o vytvoření nového příspěvku v jejich lokaci.

## 5.6 Nekonečné načítání příspěvků

Aby se nestalo, že načítání stránky /aurorex, která slouží k zobrazení příspěvků, potrvá příliš dlouho kvuli hromadnému zobrazování všech příspěvků naráz, implementoval jsem do projektu postupné, nekonečné načítání příspěvků.

Stránká má při načtení několik důležitých proměnných, `page`, která říká číslo aktuální stranky, `loading`, která určuje, zdali se aktuálně načítají příspěvky, aby bylo zabráněno několika načítáním zároveň, a `hasMore`, která kontroluje, jestli existují další příspěvky, které by se mohly načíst. Následně je potřeba někde uložit aktuálně zvolený filtr a k tomu slouží proměnná currentFilter, která má výchozí hodnotu nastavenou na ’newest’, tedy příspěvky jsou seřazeny od nejnovějšího.

Jakmile se stránka načte, server odešle API požadavek pro načtení první sady příspěvků. Požadavek zpracuje router v aurorexRoutes.js, který zavolá funkci ‘`getPosts()`‘ z controlleru postController.js.

```
1    const response = await fetch(‘/aurorex/api/posts?page=${page}&limit=6&
     filter=${currentFilter}‘);
```

Listing 13: API požadavek pro obdržení sady příspěvků

Aby mohla funkce správně fungovat, musel jsem využít MongoDB aggregation pipeline, která mi umožňuje provádět složitější operace s daty z databáze, tj. spojování kolekcí, filtrování či sčítání obsahů polí. Aggregation pipeline nejprve spojí data o příspěvku s daty o uživateli, následně ke každému příspěvku přidá 2 pole, první určující počet liků a druhé počet komentářu, aby bylo možné příspěvky řadit podle množství liků a komentářů. Následně aplikuje nastavený filtr, který přišel v API požadavku, přeskočí již načtené příspěvky a poté vrátí 6 příspěvků (počet příspěvků na jednu stránku). Následně server pošle sadů příspěvku na klientskou část [6].

Ve scriptu na stránce /aurorex je následně každý post obdržený z API požadavku

transformován do HTML elementu. Ten je poté přídán do kontejneru obsahující příspěvky.

Nekonečný scroll je aktivován pomocí scroll listeneru. Když se uživatel posune 100 pixelů před konec stránky, zavolá se funkce ‘`loadMorePosts()`‘, která opět pošle API request s novou sadou příspěvků a takto opakovaně, než jsou zobrazeny všechny příspěvky v databázi.

V případě změněny filteru během scrollovaní, funkce má nastavený event listener na změnu filteru. Když proběhne změněna filteru, funkce zavolá sama sebe s parametrem true, který zabezpečí, že se resetují načtené příspěvky a začnou se načítat od začátku s novým filterem.

# 6 Datové struktury

## 6.1 EJS

EJS (Embedded JavaScript) je šablonovací engine pro JavaScript, který se často používá při vývoji webových aplikací. Umožňuje vytvářet dynamický HTML kód pomocí JavaScriptu přímo v HTML a je navržen tak, aby zjednodušil proces generování dynamického obsahu na stránkách. Obsahuje kombinaci HTML a JavaScriptu, což usnadňuje vytváření dynamického obsahu, například pro stránku uživatele, na základě potřebných dat.

Jednotlivé EJS šablony jsou strukturovaně uspořádány podle jednotlivých sekcí webu v adresaři /views. Je zde také adresář /views/partials, který obsahuje header, tj. záhlaví stránky. Díky již zmíněné dynamičnosti mám možnost obsah daného souboru importovat do libovolného EJS souboru, čímž se elegantně vyvaruji psaní záhlaví na každou jednotlivou stránku.

Předávání dat ze serveru do EJS šablony probíhá pomocí metody `render()`, která umožňuje předat objekty, jejichž data jsou potřebná pro zobrazení. Tato metoda se volá v souborech v adresáři routes a kromě předání dat slouží k vykreslení a zkompilování vybrané stránky.

Při tvorbě EJS šablon využívám několik typů tagů, které mi umožňují vkládat JavaScriptový kód nebo využívat hodnoty proměnných přímo v HTML. Například pro vložení obsahu jednoho EJS souboru do druhého se použije tento tag.

```
1  <div class="fixed top-0 left-0 right-0 z-50 shadow-md">
2    <%- include('../partials/header') %>
3  </div>
```

Listing 14: EJS tag s include atributem

V kontextu MVC (Model View Controller) architektury fungují EJS šablony jako vrstva View, která přijímá data z kontrolerů a transformuje je do podoby, která se zobrazí uživateli.

## 6.2 Databáze

V mém projektu jsem zvolil NoSQL databázi MongoDB, která je objektově orientovaná, dynamická a dobře škálovatelná. Místo tabulek (jako u relačních databází MySQL apod.) používá kolekce a místo řádků a sloupců pole či dokumenty.

Pro práci s MongoDB v Node.js runtime enviromentu jsem zvolil knihovnu `Mongoose`.

Mongoose mi umožňuje definovat schémata pro jednotlivé kolekce v daných souborech a následně s nimi pracovat.

V hlavním souboru, kde se spouští samotný server, server.js je zavolána funkce ‘`connectDB()`‘, která je importována z config adresáře ze souboru dbConfig.js. Ten zajišťuje navázání spojení s databází pomocí `Mongoose` a MongoDB URI, tj. MongoDB connection string.

Samotné modely jsou uloženy v adresáři models, tedy modely User a Post. Model User obsahuje schéma uživatele, ve kterém jsou základní informace jako uživatelské jméno, email, zahashované heslo apod. Id MongoDB vytvoří automaticky, tedy není potřeba přidávat pole pro ID uživatele. Při vytvoření modelu, např. User, MongoDB automaticky vytvoří kolekci Users, do které se vytvořené dokumenty (jednotliví uživatelé) ukládají.

```
const userSchema = new mongoose.Schema({
    userName: { type: String, required: true },
    email: { type: String, required: true, unique: true },
    password: { type: String, required: true },
    created_at: { type: Date, default: Date.now },
    updated_at: { type: Date, required: false },
    profilePicture: { type: String, default: '/uploads/default-profile-picture.
    jpg' },
    location: { latitude: Number, longitude: Number },
    alertRadius: { type: Number, default: 50 },
    notificationsEnabled: { type: Boolean, default: true },
    notificationsForHighKp: {type: Boolean, default: true}
});
```

Listing 15: Schéma User

Díky modelům mám možnost jednoduše provádět dotazy na databázi. Například metoda `findById()`, která mi umožňuje najít v dané kolekci dokument s daným ID. Tato metoda je využita pro příklad při odstraňování vybraného komentáře, kde s její pomocí je nalezen příspěvek, u kterého je požadované smazaní komentáře. V tomto případě je do parametru metody vloženo ID příspěvku, které bylo obdrženo z požadavku odeslaného uživatelem.

```
    const post = await Post.findById(req.params.postId);
```

Listing 16: Metoda pro nalezení příspěvku pomocí ID

V kontextu MVC (Model View Controller) architektury fungují modely jako vrstva model, která slouží k nadefinování struktury dat, tedy schémat.

## 6.3 Controllery

V kontextu MVC architektury controllery v mé webové aplikaci slouží jako prostředníci mezi modely a views, tedy EJS šablonami. Zpracovávají požadavky uživatelů, upravují data v databázi prostřednictvím modelů a připravují data pro frontendovou část.

Funkce exportované z controllerů jsou volány v routerech, které na jednotlivé konkretní požadavky odeslané uživatelem zareagují zavoláním určité funkce dle požadavku.

V adresáři controllers se nachází dva controllery, postController.js a usersController.js. PostController.js slouží ke zpracování veškerých operací, které souvisí s příspěvky a usersController.js spravuje operace týkající se uživatelů.

V postControlleru je celkem 7 funkcí. První je '`timeMapFilter`', která je zavolána při změnění časového filteru na stránce /aurorex/live-map. Funkce v parametru range, uvedeného v URL adrese, obdrží časový rámec day, week nebo month a následně podle daného parametru spočítá čas, od kterého se mají příspěvky zobrazovat a vrátí vybrané příspěvky.

Funkce '`deleteComment`' slouží ke smazání komentáře z příspěvku. Funkce dle ID příspěvku najde daný příspěvek a pomocí ID komentáře požadovaný komentář. Nakonec oveří, že o smazaní se pokouší tvůrce příspěvku nebo tvůrce komentáře a poté komentář odstraní z databáze.

'`CreatePost`' vytváří nový příspěvek. Nejprve data obdržená při vytváření uloží do databáze a místo nahrané fotografie uloží cestu k fotografii. Následně zavolá metodu '`findUsersInRange()`', která je součástí sekce 5 Řešení klíčových problémů subsekce 5 Zasílání emailů při výskytu příspěvku v nastavené lokaci. Při volání této funkce v routeru aurorexRoutes.js je také zavolána metoda '`single()`', která je součástí objektu upload, jenž je importován z middlewaru upload.js, do jejíhož parametru je vložena nahrávaná fotografie a je uložena do adresáře /public/uploads.

'`DeletePost`' slouží ke smazání vybraného příspěvku, '`addComment`' k přidání komentáře k vybranému příspěvku a '`toggleLike`' k přidání nebo odstranění liku z příspěvku. Funkce '`getPosts`' je zdokumentována v sekci 5 Řešení klíčových problémů subsekce 6 Nekonečné načítání příspěvků.

Druhý controller usersController.js obsahuje 8 funkcí. Funkce '`register`' slouží k vytvoření nového uživatelského účtu. Nejprve zkontroluje, zdali není již uživatelem zadaný email registrovaný, v případě, že ne, tak vezme zadané údaje, zahashuje heslo a uloží je do databáze.

Funkce ‘`login`‘ slouží k přihlášení uživatele, tedy ověřuje přihlašovací údaje. V případě shody emailu a hesla vytvoří JWT token (JSON Web Token), do kterého vloží payload obsahující id uživatele, jeho email a uživatelské jméno, podepíše token privátním klíčem a nastaví expiraci na 1 hodinu.

‘`UpdateProfile`‘ slouží k aktualizaci uživatelských dat. Nejprve načte upravená data, zkontroluje zdali byl nahrán soubor, čímž pozná, jestli uživatel nahrál nový profilový obrázek. Poté přepíše potřebná data u uživatele a vytvoří nový JWT (JSON Web Token) s aktualizovanými údaji.

‘`GetProfile`‘ zobrazuje profil uživatele spolu s příspěvky vytvořenými daným uživatelem. Detekuje také, zdali se jedná o profil přihlášeného uživatele nebo cizí, pomocí porovnání id vyhledávaného uživatele s id uživatele z middlewaru attachUser.js. Objekty obou uživatelů poté předá spolu s objektem příspěvků vyhledávaného uživatele do renderované EJS šablony (v případě shody je objekt vyhledávaného uživatele null a tedy objekt s příspěvky vyhledávaného uživatele také).

Funkce ‘`changePassword`‘ zajišťuje změnu hesla, ‘`toggleNotifications`‘ se stará o zapnutí/vypnutí emailových notifikací v případě výskytu příspěvku v uživatelské lokaci a ‘`toggleKpNotifications`‘ o zapnutí/vypnutí emailových notifikací při vysoké intenzitě polární záře (KP index větší nebo rovno 5). Poslední funkce ‘`updateLocation`‘ zajišťuje uložení upravené uživatelské lokace.

# 7 Závěr

Cílem tohoto projektu bylo vytvořit webovou aplikaci, která ulehčí milovníkům polární záře onen nádherný jev spatřit. Tento úkol byl dle mého názoru splněn. Uživatelé mohou sledovat potřebná data jako oblačnost, KP index a další spolu s možností zhlédnutí fotografií, které pořídili a nahráli ostatní uživatelé. Zároveň pro zjednodušení hledání vhodných oblastí na pozorování jsem implementoval mapu, na které si uživatelé pomocí filtru mohou nastavit časový rámec, od jehož začátku se příspěvky zobrazují, a podívat se na lokace, odkud ostatní uživatelé nahráli příspěvky.

# Odkazy


[1] *Bar Chart*. Citováno 27. 1. 2025. Chartjs.org. 2025. URL: https://www.chartjs.org/docs/latest/charts/bar.html.

[2] *Calculate distance, bearing and more between Latitude/Longitude points*. Citováno 27. 1. 2025. Movable Type Scripts. 2025. URL: https://www.movable-type.co.uk/scripts/latlong.html.

[3] *Connect MongoDB with Node.js using mongoose — MongoDB + Express.js*. Citováno 27. 1. 2025. Youtube. 2025. URL: https://www.youtube.com/watch?v=30p9QfybWZg.

[4] *Error Handling in NodeJS (Complete Guide) — Node Tutorial*. Citováno 27. 1. 2025. Youtube. 2025. URL: https://www.youtube.com/watch?v=mGPj-pCGS2c.

[5] *Geocoding API Javascript Tutorial — Geoapify*. Citováno 27. 1. 2025. Geoapify. 2025. URL: https://www.geoapify.com/tutorial/how-to-implement-geocoding-javascript-tutorial/.

[6] *Getting total number of likes that user received by going through all his/her posts MongoDB*. Citováno 27. 1. 2025. Stack Overflow. 2025. URL: https://stackoverflow.com/questions/67041928/.

[7] *Home page background picture*. Citováno 27. 1. 2025. Pexels. 2025. URL: https://www.pexels.com/photo/time-lapse-photo-of-northern-lights-1933316/.

[8] *How to Build a Modal with JavaScript*. Citováno 27. 1. 2025. Free Code Camp. 2025. URL: https://www.freecodecamp.org/news/how-to-build-a-modal-with-javascript/.

[9] *How to Create HTTPS Server with Node.js ? - GeeksforGeeks*. Citováno 27. 1. 2025. Geeks For Geeks. 2025. URL: https://www.geeksforgeeks.org/how-to-create-https-server-with-node-js.

[10] *How to Create Responsive Modal Images using CSS + JavaScript?* Citováno 27. 1. 2025. Geeks For Geeks. 2025. URL: https://www.geeksforgeeks.org/how-to-create-responsive-modal-images-using-css-javascript/.

[11] *How to display error without alert box using JavaScript*. Citováno 27. 1. 2025. Geeks For Geeks. 2025. URL: https://www.geeksforgeeks.org/how-to-display-error-without-alert-box-using-javascript/.

[12] *How to make a responsive navbar with tailwind css*. Citováno 27. 1. 2025. Youtube. 2025. URL: https://www.youtube.com/watch?v=vYowvsUiChs&ab_channel=CodeAProgram.

[13] *How to Send Email Using Nodemailer in Node.js — Node.js Email Tutorial*. Citováno 27. 1. 2025. Youtube. 2025. URL: https://www.youtube.com/watch?v=fF-07yFTq5o.

[14] *How to Upload Files in Node.js Using Express and Multer*. Citováno 27. 1. 2025. Youtube. 2025. URL: https://www.youtube.com/watch?v=i8yxx6V9UdM.

[15] *How to Use Chart.js?* Citováno 27. 1. 2025. W3schools. 2025. URL: https://www.w3schools.com/ai/ai_chartjs.asp.

[16] *How will we convert latitude and longitude to the address in web site?* Citováno 27. 1. 2025. Stack Overflow. 2025. URL: https://stackoverflow.com/questions/41329323/how-will-we-convert-latitude-and-longitude-to-the-address-in-web-site.

[17] *HTML Favicon*. Citováno 27. 1. 2025. Geeks For Geeks. 2025. URL: https://www.geeksforgeeks.org/html-favicon/.

[18] *Change Google Maps Infowindow Close Icon*. Citováno 27. 1. 2025. Stack Overflow. 2025. URL: https://stackoverflow.com/questions/1614586/.

[19] *Intro to MongoDB and Mongoose*. Citováno 27. 1. 2025. Youtube. 2025. URL: https://www.youtube.com/watch?v=-PdjUx9JZ2E.

[20] *JWT Authentication Tutorial - Node.js*. Citováno 27. 1. 2025. Youtube. 2025. URL: https://www.youtube.com/watch?v=mbsmsi7l3r4.

[21] Mikko Lagersted. *How to Photograph the Northern Lights (Aurora Borealis)?* Citováno 27. 1. 2025. 2025. URL: https://mikkolagerstedt.com/blog//how-to-photograph-the-northern-lights-aurora-borealis.

[22] *Legend — Chart.js*. Citováno 27. 1. 2025. Chart.js. 2025. URL: https://www.chartjs.org/docs/latest/configuration/legend.html.

[23] *Multer*. Citováno 27. 1. 2025. npmjs. 2025. URL: https://www.npmjs.com/package/multer.

[24] *MVC Pattern Explained Easy — MVC Tutorial (Example in NodeJS)*. Citováno 27. 1. 2025. Youtube. 2025. URL: https://www.youtube.com/watch?v=bQuBlR0T5cc.

[25] *Node JS Register and Login API using JWT + MongoDB*. Citováno 27. 1. 2025. Youtube. 2025. URL: https://youtu.be/ZEg03f1o_vQ?si=05RuG_nl9fgnslp5.

[26] *Node.js Tutorials for Beginners-Whole series*. Citováno 27. 1. 2025. Youtube. 2025. URL: https://www.youtube.com/playlist?list=PL0Zuz27SZ-6PFkIxaJ6Xx_X46avTM1aYw.

[27] *Nodemailer: Tutorial with Code Snippets [2025]*. Citováno 27. 1. 2025. Mailtrap. 2025. URL: https://mailtrap.io/blog/sending-emails-with-nodemailer/.

[28] *Northern Lights — When and where to see the Aurora Borealis*. Citováno 27. 1. 2025. Discover the world. 2025. URL: https://www.discover-the-world.com/northern-lights/.

[29] *OpenWeatherMap API*. Citováno 27. 1. 2025. Open Weather. 2025. URL: https://openweathermap.org/current.

[30] *Polární záře*. Citováno 27. 1. 2025. Wikipedia. 2025. URL: https://cs.wikipedia.org/wiki/Polarni_zare.

[31] *Read JSON File into HTML with JavaScript Fetch API*. Citováno 27. 1. 2025. Youtube. 2025. URL: https://www.youtube.com/watch?v=Oage6H4GX2o.

[32] *Remove mouse wheel scroll, use draggable only*. Citováno 27. 1. 2025. Green Sock. 2025. URL: https://gsap.com/community/forums/topic/33265-remove-mouse-wheel-scroll-use-draggable-only/.

[33] *The BEST Way to Create Responsive Design with Tailwind CSS (2023)*. Citováno 27. 1. 2025. Youtube. 2025. URL: https://www.youtube.com/watch?v=PuovsjZN11Y&ab_channel=Lukas%7CWebDevelopment%26Design.

[34] *Uploading files with Node.js + Express using Multer*. Citováno 27. 1. 2025. Youtube. 2025. URL: https://www.youtube.com/watch?v=3DrZDvimkCE.

[35] *Where and when to see the northern lights in 2025*. Citováno 27. 1. 2025. Space. 2025. URL: https://www.space.com/32601-where-to-see-northern-lights.html.

# Listings