ŠOLSKI CENTER VELENJE
ELEKTRO IN RAČUNALNIŠKA ŠOLA
Trg mladosti 3, 3320 Velenje

MLADI RAZISKOVALCI ZA RAZVOJ SAŠA REGIJE

RAZISKOVALNA NALOGA

ALI LAHKO REŠIMO PROBLEM KOKOŠ IN JAJCE Z UMETNO INTELIGENCO

Tematsko področje: RAČUNALNIŠTVO

Avtor:
Sašo Tamše

Mentor:
Samo Železnik

Velenje, 2025

Raziskovalna naloga je bila opravljena na Elektro in računalniški šoli Velenje, 2025.

Mentor: Samo Železnik

Datum predstavitve: marec 2025

KLJUČNA DOKUMENTACIJSKA INFORMACIJA

ŠD Elektro in računalniška šola Velenje, šolsko leto 2024/2025

KG aplikacija / programiranje / umetna inteligenca / natančnost

AV TAMŠE, Sašo

SA ŽELEZNIK, Samo

KZ 3320 Velenje, Trg mladosti 3

ZA ŠC Velenje, Elektro in računalniška šola, 2025

LI 2025

IN **ALI LAHKO REŠIMO PROBLEM KOKOŠ IN JAJCE Z UMETNO INTELIGENCO**

TD Raziskovalna naloga

OP LXV, 65 str., 61 sl., 20 vir.

IJ SL

JI sl / en

AI Raziskovalna naloga se osredotoča na uporabo umetne inteligence pri avtomatiziranem prepoznavanju in dodajanju dogodkov na spletno platformo. V sklopu raziskave je bila razvita aplikacija, ki omogoča prepoznavo, preverjanje in dodajanje dogodkov organizacij iz Instagram profilov z uporabo različnih orodij, kot so Apify client, ChatGPT API, Google Vision API, Google PSE in Google Places API. Raziskava je pokazala, da je sistem lahko učinkovitejši, hitrejši in cenejši kot ročni vnos dogodkov v bazo podatkov. Sistem je dosegel natančnost približno 84%, medtem ko ročni vnos 82%. Pri tem je uporabil čas 22 minut in 10 sekund, kar je v primerjavi z ročnim vnosom kar 70 minut hitreje. Za takšno delo bi študentu plačali 11,74€, medtem ko program porabi žetone v vrednosti 2,05€.

KEY WORDS DOCUMENTATION

ND Elektro in računalniška šola Velenje, šolsko leto 2024/2025

CX appplication / programing / artificial inteligence / accuracy

AU TAMŠE, Sašo

AA ŽELEZNIK, Samo

PP 3320 Velenje, Trg mladosti 3

PB ŠC Velenje, Elektro in računalniška šola, 2025

PY 2025

TI **CAN WE SOLVE THE CHICKEN AND EGG PROBLEM USING ARTIFICIAL INTELIGENCE**

DT RESEARCH WORK

NO LXV, 65 p., 61 fig., 20 ref.

LA SL

AL sl / en

AB The research project focuses on the use of artificial intelligence for automated recognition and addition of events to an online platform. As part of the research, an application was developed that enables the identification, verification, and addition of events from organizations' Instagram profiles using various tools such as Apify Client, ChatGPT API, Google Vision API, Google PSE, and Google Places API. The study showed that the system can be more efficient, faster, and cheaper than manually entering events into a database. The system achieved an accuracy of approximately 84%, compared to 82% for manual entry. It completed the process in 22 minutes and 10 seconds, which is 70 minutes faster than manual input. A student would be paid €11.74 for this task, whereas the program consumes tokens worth only €2.05.

KAZALO

KAZALO SLIK

UPORABLJENE KRATICE

- **AI** – Artificial Intelligence (Umetna inteligenca)
- **API** – Application Programming Interface (Vmesnik za programiranje aplikacij)
- **FTP** – File Transfer Protocol (Protokol za prenos datotek)
- **POST** – HTTP metoda za zahtevo podatkov
- **GET** – HTTP metoda za pridobivanje podatkov
- **CORS** – Protokol skupne rabe
- **GPT** – Generative Pre-trained Transformer (Model umetne inteligence)
- **JSON** – JavaScript Object Notation (Format za izmenjavo podatkov)
- **NLP** – Natural Language Processing (Obdelava naravnega jezika)
- **OCR** – Optical Character Recognition (Optično prepoznavanje znakov)
- **PHP** – Hypertext Preprocessor (Skriptni jezik za splet)
- **SQL** – Structured Query Language (Jezik za upravljanje baz podatkov)
- **SELECT** – Vrne podatke iz tabele
- **INSERT** – Dodajanje zapisa v tabelo
- **URL** – Uniform Resource Locator (Spletni naslov)
- **PSE** – Programmable Search Engine (Spletni brskalnik z zmožnostjo nastavljanja konfiguracij)
- **UI** – Umetna inteligenca
- **ID** – Identifikator
- **Web scraping** – Spletno strganje
- **Uber** – Platforma za naročanje prevoza preko telefona
- **Netflix –** Platforma za gledanje filmov in serij
- **Facebook –** Socialno omrežje
- **Instagram –** Socialno omrežje
- **HTTP –** Metoda za prenos informacij po spletu (HyperText Transfer Protocol)
- **Event –** Dogodek

1. UVOD

Umetna inteligenca (UI) je v zadnjih letih doživela izjemen razvoj in postala ključna tehnologija v številnih panogah, vključno z avtomatizacijo procesov, analizo podatkov in prepoznavanjem vzorcev. Ena izmed velikih prednosti UI je njena sposobnost reševanja kompleksnih problemov, ki zahtevajo hitro obdelavo velike količine informacij. V tej raziskovalni nalogi se osredotočam na uporabo umetne inteligence pri avtomatiziranem prepoznavanju in dodajanju dogodkov na spletno platformo.

Digitalne platforme, ki temeljijo na vsebinah, kot so dogodkovni portali, pogosto naletijo na klasičen problem "kokoš in jajce". Gre za situacijo, kjer platforma potrebuje vsebino, da bi pritegnila uporabnike, hkrati pa brez uporabnikov težko pridobi zadostno količino relevantnih vsebin. To pomeni, da je začetna vzpostavitev platforme velik izziv, saj je brez kritične mase uporabnikov težko zagotoviti zadosten obseg vsebine, ki bi pritegnila nove obiskovalce.

V tej nalogi preučujem, kako lahko umetna inteligenca pomaga pri reševanju tega problema s samodejno analizo objav na družbenih omrežjih, prepoznavanjem vabil na dogodke in njihovim organiziranim vnosom na spletno platformo. Za to sem razvil sistem, ki uporablja spletno strganje (web scraping), analizo slik s pomočjo Google Vision API in naravno jezikovno obdelavo (NLP) za razumevanje besedil ter OpenAI-jeve modele za kategorizacijo in strukturiranje podatkov. Namen sistema je avtomatizirati proces zbiranja dogodkov in izboljšati dostopnost informacij.

1.1. NAMEN RAZISKAVE

Glavni namen raziskave je preučiti, kako lahko umetna inteligenca pripomore k reševanju problema kokoš in jajce na področju dogodkovnih platform. Cilj je razviti in testirati sistem, ki bo sposoben samodejno prepoznavati dogodke ter jih dodajati v podatkovno bazo brez potrebe po ročnem vnosu. S tem želim preveriti, ali lahko UI zmanjša potrebo po ročnem delu, poveča natančnost in hitrost prepoznavanja dogodkov ter dolgoročno zniža stroške vzdrževanja platforme.

2. HIPOTEZE

1. Izdelan program lahko hitreje in učinkoviteje prepozna dogodke iz družbenih omrežij kot ročni vnos uporabnikov.
2. Sistem umetne inteligence bo dosegel vsaj 80 % natančnost pri prepoznavanju ključnih informacij dogodkov (naslov, datum, kategorija, organizator).
3. Avtomatiziran sistem bo dolgoročno zmanjšal potrebo po ročnem vnosu dogodkov in s tem zmanjšal stroške vzdrževanja platforme.

3. PREGLED STANJA TEHNIKE

3.1. PROBLEM KOKOŠI IN JAJCA V DIGITALNIH PLATFORMAH

Problem kokoš in jajce se nanaša na situacije, kjer je uspeh ene komponente sistema odvisen od prisotnosti druge, vendar oboje zahteva sočasno vzpostavitev. Klasičen primer so platforme, kot so tržnice, kjer potrebujete prodajalce, da pritegnete kupce, in obratno. Ta problem je še posebej izrazit pri novih podjetjih in platformah, kjer je vzpostavitev začetne baze uporabnikov ključna za uspeh. To velja tudi za dogodkovne platforme, kjer so informacije o dogodkih ključne za privabljanje uporabnikov, pridobivanje teh informacij pa je pogosto časovno in finančno zahtevno.

3.1.1. Strategije reševanja problema v preteklosti

Različna podjetja so razvila strategije za premagovanje problema kokoš in jajce:

Finančne spodbude: Uber je na začetku ponujal finančne bonuse voznikom in popuste potnikom, da je ustvaril začetno bazo uporabnikov.
Ekskluzivne vsebine: Netflix in druge pretočne platforme so začele z lastnimi vsebinami, da bi pritegnile uporabnike.
Ciljno usmerjen začetek: Velike platforme, kot sta Facebook in Instagram, so se na začetku osredotočile na specifične demografske skupine, preden so se razširile globalno.

3.2. UMETNA INTELIGENCA KOT REŠITEV PROBLEMA

Umetna inteligenca (UI) je postala ključno orodje za optimizacijo in avtomatizacijo procesov v različnih panogah. Z napredkom v strojnem učenju, analizi podatkov in obdelavi naravnega jezika (NLP) omogoča avtomatizirano zbiranje, razvrščanje in analizo podatkov, kar lahko igra pomembno vlogo pri reševanju problema kokoši in jajca.

V kontekstu spletnih platform, ki se soočajo z začetno odsotnostjo vsebine in uporabnikov, lahko umetna inteligenca pomaga s samodejno identifikacijo in organizacijo podatkov, ki so ključni za privabljanje prvih uporabnikov.

3.2.1. Kako AI pomaga pri avtomatiziranem zbiranju podatkov

Eden ključnih načinov, kako UI pomaga pri reševanju problema kokoši in jajca, je avtomatizirano pridobivanje podatkov iz različnih virov. To omogoča, da platforma že v začetni fazi ponuja koristno vsebino in tako privabi uporabnike.

3.2.1.1. Glavne metode avtomatiziranega zbiranja podatkov s pomočjo umetne inteligence

1. Spletno strganje podatkov (Web Scraping)
   - UI lahko uporablja scraping algoritme, ki prepoznajo in pridobijo relevantne podatke iz spletnih virov.
   - Primer: Prepoznavanje dogodkov iz družbenih omrežij in njihova samodejna kategorizacija.
2. Obdelava besedila in naravno razumevanje jezika (NLP)
   - UI lahko analizira besedilo in prepozna ključne informacije, kot so datumi, kraji in opisi dogodkov.
   - Primer: Sistem samodejno prepozna vabilo na dogodek v objavi na Facebooku ali Instagramu ter ga doda na spletno platformo.
3. Strojno učenje za prepoznavanje vzorcev
   - Algoritmi lahko učinkovito razločujejo med relevantnimi in nerelevantnimi podatki, kar zmanjšuje količino nepotrebnih informacij.
   - Primer: AI razloči med resničnim vabilom na dogodek in običajno objavo uporabnika.
4. Optično prepoznavanje znakov (OCR)
   - UI lahko iz slik (npr. plakatov dogodkov) prepozna besedilo in ga pretvori v strojno berljivo obliko.
   - Primer: Uporaba OCR za ekstrakcijo podatkov iz slik z napovedniki koncertov ali konferenc.

Uporaba teh metod omogoča avtomatizirano in učinkovito ustvarjanje začetne baze podatkov, ki je ključna za uspeh platforme.

3.2.2. Pregled sorodnih rešitev

Umetna inteligenca se že uporablja v različnih storitvah in platformah, ki rešujejo problem kokoši in jajca s pomočjo avtomatiziranega zbiranja podatkov. Nekateri primeri vključujejo:

1. Google News in Google Discover
   - Uporabljata strojno učenje in NLP, da avtomatsko zbirata in kategorizirata novice iz različnih spletnih virov.
   - Sistem uporablja personalizacijo, ki omogoča, da so novice relevantne glede na interese uporabnika.
2. Eventbrite in Meetup
   - Ti dogodkovni portali uporabljajo AI priporočilne algoritme, ki analizirajo zanimanja uporabnikov in predlagajo dogodke.
   - Meetup omogoča samodejno predlaganje dogodkov na podlagi analize vedenja uporabnikov in zgodovine udeležbe.
3. Facebook Events in Instagram AI sistem
   - Facebook uporablja umetno inteligenco za prepoznavanje dogodkov iz objav in slik, pri čemer analizira ključne informacije, kot so datumi in lokacije.
   - Instagram lahko na podlagi NLP algoritmov prepozna vsebino objav in jo poveže z relevantnimi dogodki.

4. AI za analizo spletnih objav v marketinških orodjih (npr. Sprinklr, Brandwatch)
    - Uporablja NLP za prepoznavanje trendov in pomembnih informacij v družbenih omrežjih.
    - Ti sistemi so lahko osnova za avtomatizirano prepoznavanje dogodkov v spletnih objavah.

4. METODOLGIJA IN TEHNIČNA IMPLEMENTACIJA

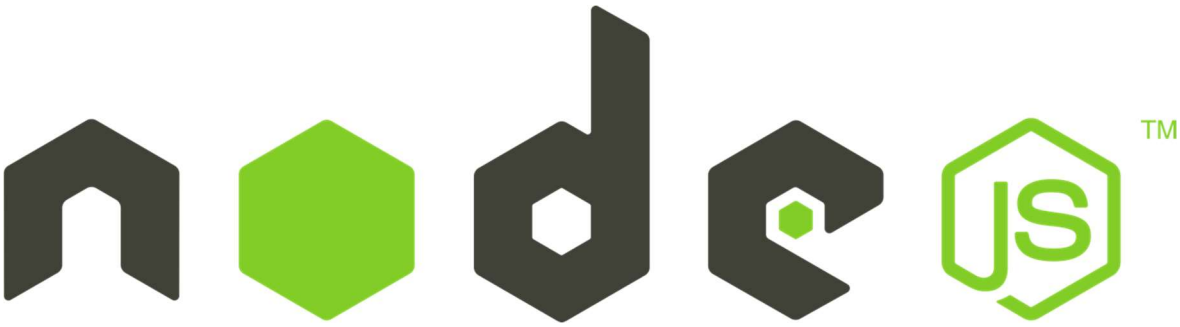
4.1. PREGLED UPORABLJENIH TEHNOLOGIJ

V sami kodi so uporabljene različne tehnologije, ki omogočajo avtomatizacijo procesov, analizo podatkov in učinkovito shranjevanje informacij o dogodkih. Spodaj so podrobno opisane ključne tehnologije, ki sestavljajo sistem.

4.1.1. Node.js in njegovi moduli (axios, fs, cron, basic-ftp, date-fns)

Programski jezik:

- **Node.js** je JavaScript okolje za izvajanje kode na strežniku, ki omogoča učinkovito obdelavo asinhronih operacij in delo z velikimi količinami podatkov.



Slika 1: Node.js (vir: curotec.com)

Knjižnice:

- **axios** je knjižnica za pošiljanje HTTP zahtev, ki omogoča enostavno komunikacijo s spletni API-ji.
- **fs (File System)** omogoča upravljanje datotek na strežniku, kot je branje, zapisovanje in premikanje podatkov.
- **cron** je knjižnica za avtomatizacijo opravil, kot so periodično izvajanje strganja podatkov in posodobitve baze.
- **basic-ftp** omogoča prenos datotek na oddaljeni strežnik prek FTP protokola, kar je ključno za shranjevanje slik dogodkov.
- **date-fns** je knjižnica za delo z datumi in časi, ki omogoča enostavno formatiranje in obdelavo časovnih podatkov.

4.1.2. Apify client za scraping

- **Apify Client** je orodje za spletno strganje (web scraping), ki omogoča avtomatizirano pridobivanje podatkov s spletnih strani in družbenih omrežij. Uporablja Instagram GraphQL za pridobivanje podatkov.
- **Omogoča** zajem in ekstrakcijo informacij, kot so besedila, slike in metapodatki iz objav na družbenih omrežjih.
- **Pomaga** pri zbiranju objav iz točno določenih Instagram profilov.

Slika 2: Apify.com (vir: dev.to)

4.1.3. Google Vision API za analizo slik

- **Google Vision API** je storitev za analizo in prepoznavanje besedil na slikah s pomočjo strojnega učenja.
- **Omogoča** avtomatsko prepoznati besedilo iz vizualnih virov brez potrebe po ročni analizi.
- **Pomaga** pri ekstrakciji besedila iz naslovnih slik scrapanih objav.

Slika 3: Google Vision API (vir: medium.com)

4.1.4. Google PSE za preverjanje informacij na spletu

- **Google Programmable Search Engine (PSE)** omogoča prilagojeno iskanje informacij na spletu, ki se lahko uporablja za preverjanje verodostojnosti podatkov o dogodkih.
- **Omogoča** avtomatizirano iskanje in primerjavo podatkov, ki so pridobljeni iz družbenih omrežij, z obstoječimi informacijami na spletnih straneh organizatorjev dogodkov.
- **Pomaga** pri izboljšanju natančnosti podatkov in zmanjšanju napačnih zaznav dogodkov.

Slika 4: Google PSE (vir: herohunt.ai)

4.1.5. Google Places API za iskanje mest

- **Google Places API** je storitev, ki omogoča pridobivanje podrobnih informacij o lokacijah. Z njim lahko pridobite podatke, kot so ime kraja, naslov, geografske koordinate, ocene, fotografije in drugi pomembni podatki o lokaciji.
- **Omogoča** avtomatizirano iskanje in preverjanje lokacijskih podatkov. S tem lahko pridobimo natančne informacije o določenih lokacijah (njihov poln naslov).
- **Pomaga** pri točnosti izbire mesta, ki je ključno za prepoznavanje in beleženje dogodkov.

Slika 5: Google Places API (vir: spiralking.com)

4.1.6. OpenAI API za obdelavo besedila

- **OpenAI API** uporablja napredne modele umetne inteligence za analizo in obdelavo besedila.

- **Omogoča** prepoznavanje strukture dogodkov, razvrščanje vsebine v ustrezne kategorije, generiranje opisa ter izločanje ključnih informacij, kot so ime dogodka, organizator, datum in lokacija.
- **Pomaga** pri oblikovanju čistejših in uporabnikom prijaznih podatkov v JSON, ki jih nato sistem shrani v bazo.

Slika 6: OpenAI (vir: medium.com)

4.1.7. PHP API in MySQL

PHP API in MySQL je pristop, kjer spletna aplikacija implementira vmesnik (API) z uporabo jezika PHP, ki komunicira z relacijsko podatkovno bazo MySQL. V nadaljevanju je podroben opis delovanja takega sistema:

1. Prejemanje HTTP zahtev:
   - Klient/odjemalec (npr. spletna aplikacija, mobilna aplikacija ali drug strežnik) pošlje HTTP zahtevo (GET, POST, PUT, DELETE) na določen URL, ki je povezan z vašim PHP API-jem.
   - PHP skripta, ki se nahaja na strežniku, prejme to zahtevo in začne obdelavo.

2. Obdelava vhodnih podatkov in preverjanje:
   - Skripta najprej preveri, ali je bila zahtevana metoda (npr. POST) pravilna, in nato pridobi potrebne vhodne podatke.
   - Preden se podatki uporabijo, jih je potrebno očistiti in validirati (npr. odstraniti odvečne presledke, preveriti dolžino, tip podatkov, format) ter zagotoviti, da so varni za uporabo v SQL poizvedbah.
   - Ta korak vključuje tudi preverjanje avtorizacije, če API zahteva kakšno obliko dostopa (npr. preverjanje API ključev, žetonov ali uporabniških poverilnic).

3. Vzpostavitev povezave z MySQL:
   - PHP API se preko ustreznih knjižnic (kot sta mysqli ali PDO) poveže s strežnikom MySQL.
   - Povezava uporablja konfiguracijske podatke, kot so gostitelj (host), uporabniško ime (username), geslo (password) in ime baze podatkov, ki so običajno shranjeni v varnih konfiguracijskih datotekah ali okoljevarstvenih spremenljivkah.
   - Ta način shranjevanja podatkov (npr. z uporabo datoteke .env) zagotavlja varnost in fleksibilnost pri konfiguraciji.

4. Izvajanje SQL poizvedb:
   - Ko je povezava vzpostavljena, PHP API izvrši ustrezne SQL poizvedbe (na primer SELECT, INSERT, UPDATE ali DELETE) glede na naravo prejete zahteve.
   - Pri izvajanju poizvedb se običajno uporabljajo pripravljene izjave (prepared statements) in parametri, da se prepreči SQL injekcija in zagotovita varnost in integriteta podatkov.

5. Obdelava rezultatov:
   - Rezultati, pridobljeni iz MySQL poizvedb, se obdelajo v PHP-ju.
   - Na primer, pri SELECT poizvedbi se vrne niz podatkov (običajno v obliki asociativnega polja), ki ga je potrebno pretvoriti v ustrezen format.
   - Ta format je pogosto JSON, saj je JSON standard, ki se enostavno uporablja in prenaša med strežniki in odjemalci.

6. Generiranje in pošiljanje odgovora:
   - Po obdelavi rezultatov PHP API ustvari odgovor, ki običajno vsebuje kodo stanja (npr. HTTP status 200 za uspeh) in telo odgovora, kjer so podatki kodirani v JSON.
   - Ta odgovor se nato pošlje nazaj klientu, ki ga lahko uporabi za nadaljnje delo (npr. prikaz podatkov uporabniku, nadaljnja obdelava v aplikaciji itd.).

7. Obravnava napak in logiranje:
   - Med celotnim procesom so implementirani mehanizmi za obravnavo napak. Če pride do napake (npr. neuspešna povezava z bazo, napačna SQL poizvedba, nepravilni vhodni podatki), PHP API vrne ustrezno sporočilo o napaki in HTTP kodo (npr. 400 ali 500).
   - Napake in pomembni dogodki se pogosto logirajo v datoteke, kar omogoča lažje odkrivanje in odpravljanje težav.

8. Varnost in skalabilnost:
   - Zaradi narave podatkov, ki se prenašajo, je ključnega pomena, da API uporablja HTTPS za šifriranje komunikacije.
   - Prav tako se uporabljajo varnostni ukrepi, kot so omejevanje dostopa (CORS, API ključi), zaščita pred SQL injekcijo in validacija vhodnih podatkov.
   - Pri večjih aplikacijah se lahko API razširi, da podpira več endpointov, vsak s specifičnimi funkcionalnostmi, kar omogoča modularno in skalabilno arhitekturo.

4.2. OPIS RAZVITIH SKRIPT

Za svoj spletni portal sem razvil dva programa. Scraper.js je lociran lokalno in poskrbi za scrapanje objav, njihovo preverjanje ter pošilanje dogodkov v JSON obliki preko API na spletno aplikacijo. Add-event.php je lociran na strežniku in poskrbi za prevzem podatkov, preverjanje baze za enak dogodek in dodajanje v bazo.

4.2.1. scraper.js

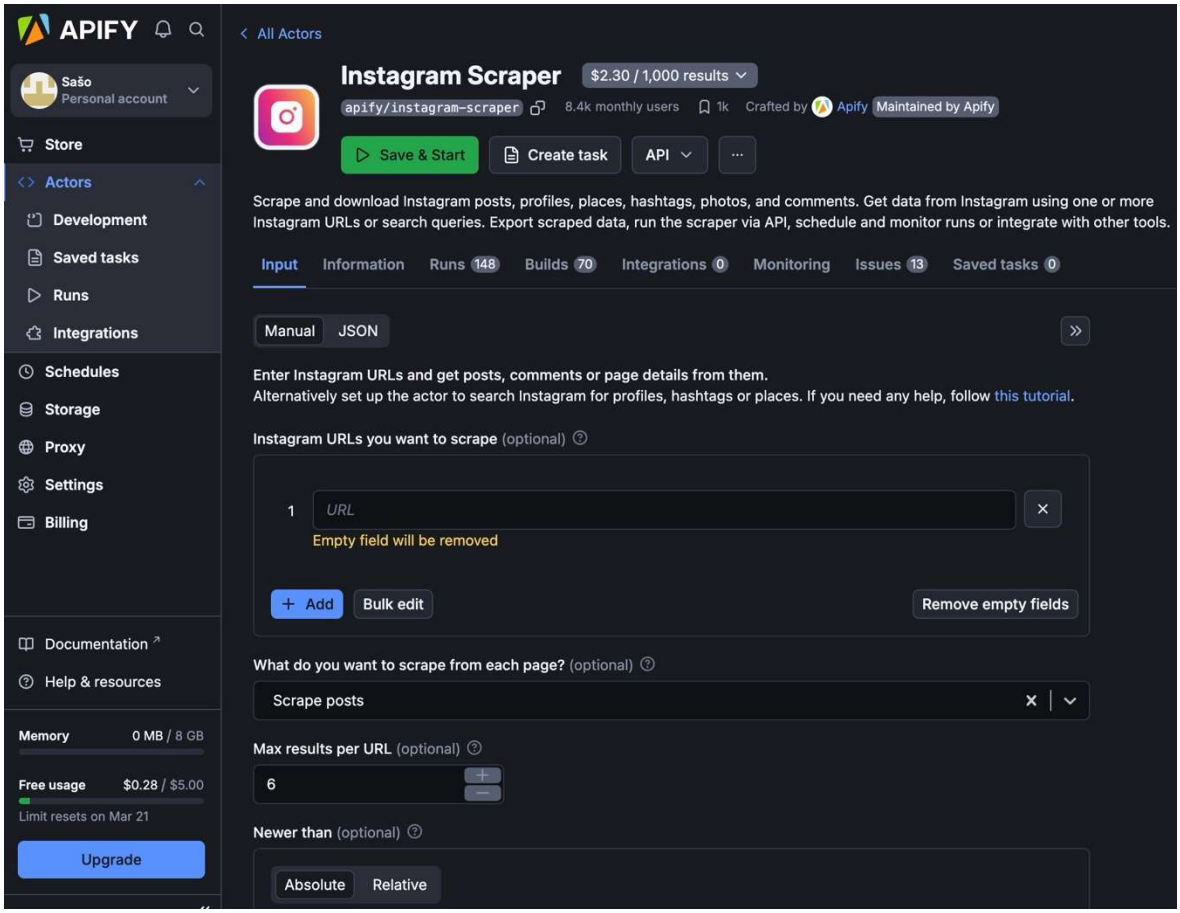
Scraper.js ima več funkcij.

4.2.1.1. Pridobivanje API ključev

Za uporabljanje orodij, kot so Apify client, Google Vision, ChatGPT, Google CSE in Google Places API potrebujemo njihove API ključe, ki delujejo, kot varnostna zaščita in identificiranje oseb, ki orodje uporablja. V nadaljevanju bom opisal, kako se ti ključi pridobijo in kaj vse je potrebno.

4.2.1.1.1. Apify client API

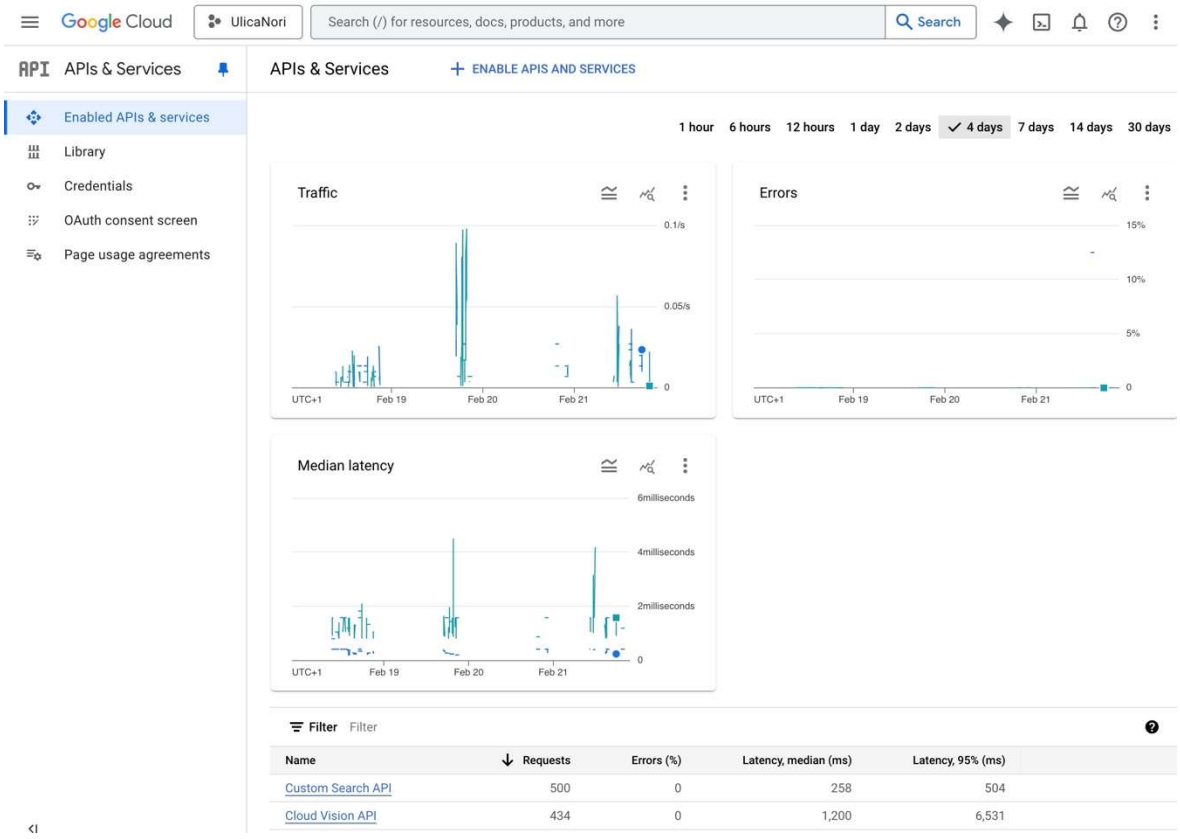
Za pridobitev ključa od Apify client-a je potrebna prijava na njihovi spletni strani in vpis bančne kartice. Cena uporabljanja orodja je 2.30$ za 1000 rezultatov, s tem da ima navaden uporabnik na mesec brezplačno porabo do 5.00$, kar pomen več kot 2000 rezultatov. Ko uporabnik izpolni pogoje izbere igralca. To je odvisno od tega, kaj hočemo pridobiti. Jaz sem izbral Instagram Scraper, kjer lahko iščemo objave na podlagi vpisanih profilov. Pod nastavitvami na spletni strani lahko vidimo naše API ključe.



Slika 7: Apify client UI (vir: lasten)

4.2.1.1.2. Google Vision API

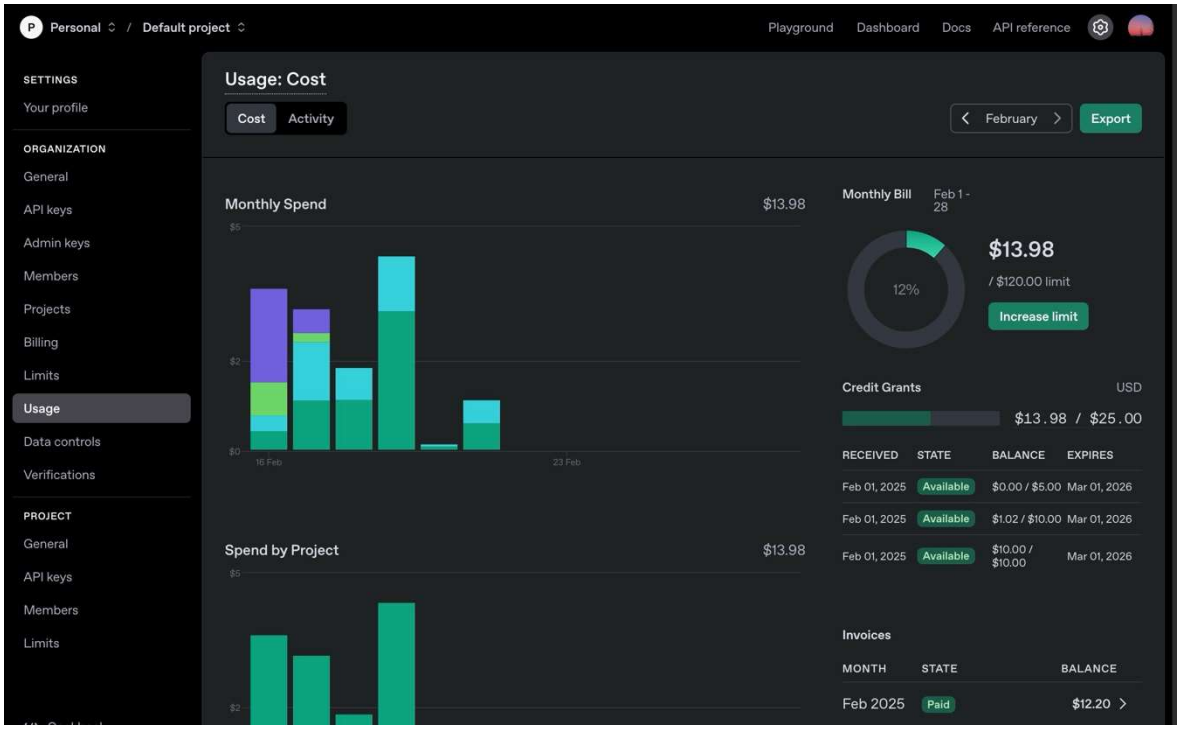
Pridobitev API ključa za Google Vision API je nekoliko bolj zapleten. Najprej se morate prijaviti v Google Cloud Platform in ustvariti nov projekt, v katerem boste uporabljali Google Vision API. Ko imate projekt, pojdite v razdelek API & Services → Library in poiščite Google Vision API. Ta API je potrebno omogočiti, saj bo brez aktivacije onemogočen. Pomembno je, da v postopku omogočanja API-ja vnesete podatke o bančni kartici. To je potrebno zaradi preverjanja identitete in preprečevanja morebitnih zlorab, četudi nameravate uporabljati brezplačno raven storitve. Ko je API omogočen, pojdite v razdelek API & Services → Credentials, kjer lahko ustvarite nov API ključ. Google Vision API ponuja brezplačen nivo, ki omogoča obdelavo 1000 zahtevkov na mesec. Cena se določi na podlagi vrste analize (OCR, zaznava obraza, označevanje slik).



Slika 8: Google Cloud nadzorna plošča API-jev (vir: lasten)

4.2.1.1.3. OpenAI API

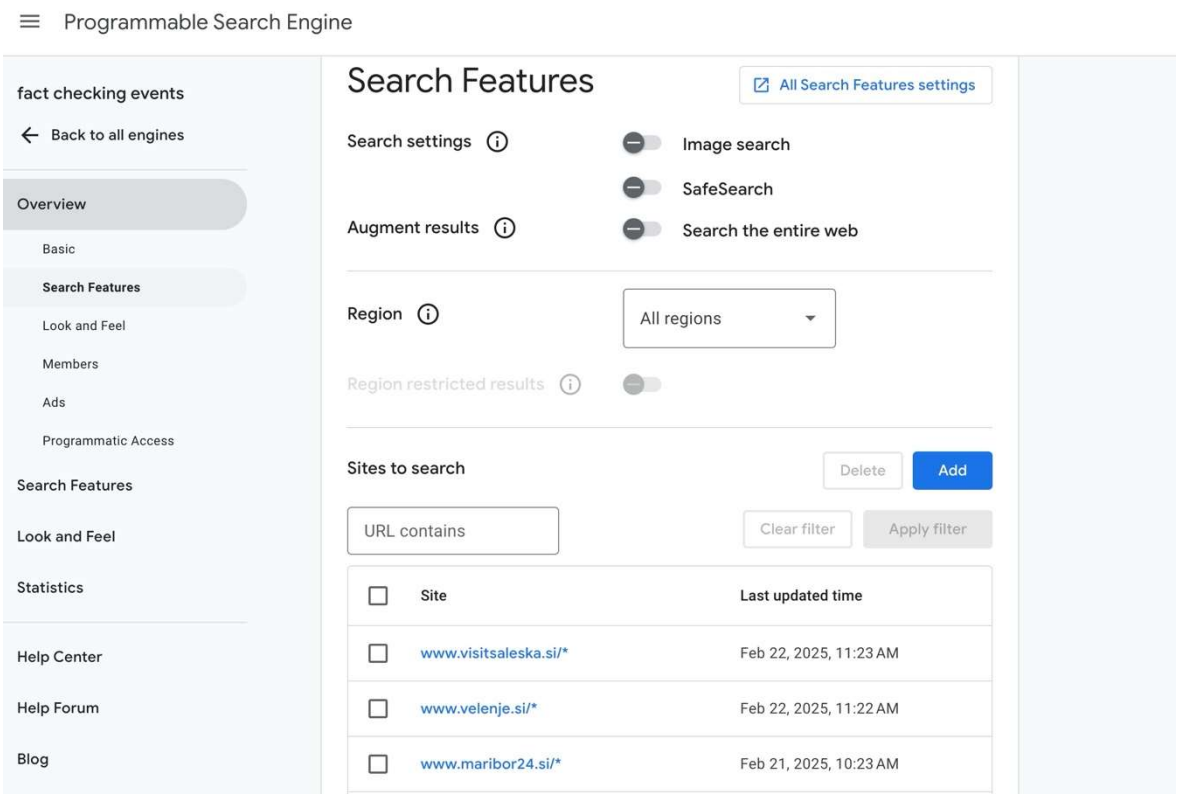
Pridobitev API ključa za OpenAI API je nekoliko bolj preprosta, vendar je storitev 100 % plačljiva. Najprej se morate registrirati na spletni strani OpenAI in ustvariti nov račun. Po uspešni registraciji boste morali potrditi svoj e-poštni naslov in vnesti tudi plačilne podatke, saj OpenAI API ne ponuja trajno brezplačnega nivoja uporabe. Ko je vaš račun aktiviran, se prijavite v vaš profil in pojdite v razdelek API Keys, kjer lahko ustvarite nov API ključ. Ta ključ boste nato uporabili za dostop do OpenAI API-ja, ki omogoča izvajanje različnih nalog, kot so generiranje besedila, analiza in prevajanje. OpenAI API je 100 % plačljiv, pri čemer se cena določi na podlagi števila zahtevkov, količine procesiranih podatkov in izbiro modela.



Slika 9: OpenAI API nadzorna plošča uporabe (vir: lasten)

4.2.1.1.4. Google PSE

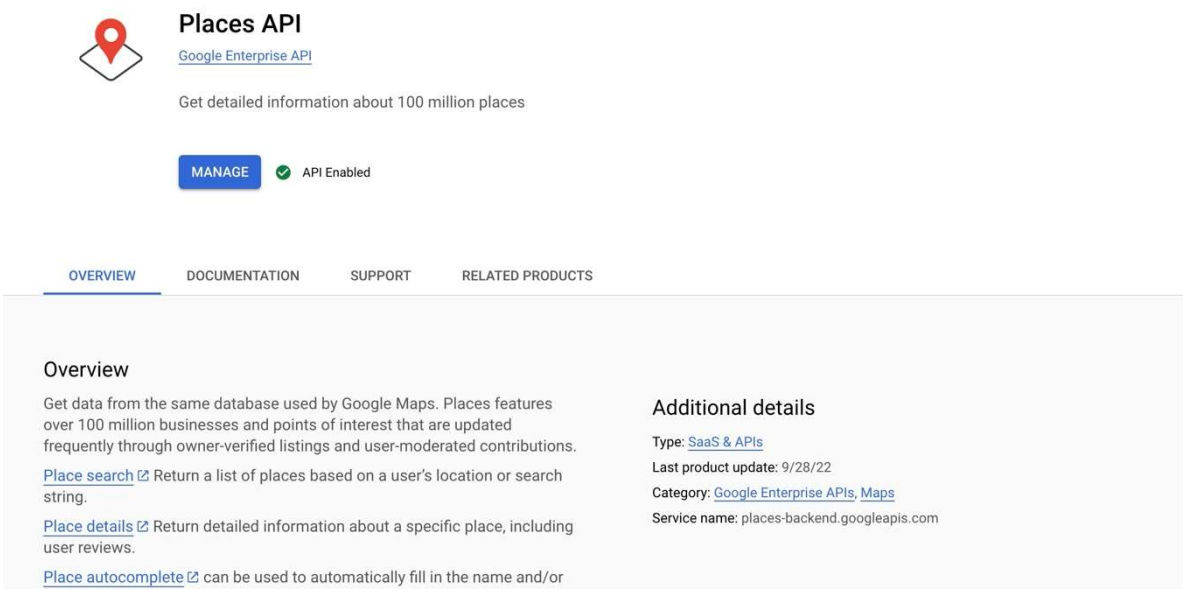
Za uporabo Google Programmable Search Engine-a je postopek podoben, kot pri Google Vision API-ju. Lahko uporabu isti API ključ. Vseeno moramo ustvariti svoj brskalnik. Iskalnik lahko prilagodimo. Išče lahko po celem spletu, po regijah ali samo po vnesenih spletnih straneh. Za to raziskovalno nalogo je bilo najbolje vnesti spletne strani organizacij, iz katerih sem scrapal dogodke. Za uporabo moramo vnesti bančno kartico, saj ga lahko brezplačno uporabljamo na dan do samo 100 zahtevkov. Cena je 5$ na 1000 zahtevkov, s tem da lahko na dan pošlješ samo 10000 zahtevkov.



Slika 10: Google PSE (vir: lasten)

4.2.1.1.5. Google Places API

Za uporabo Google Places API-ja je postopek podoben kot pri drugih Googlovih storitvah. Najprej se morate prijaviti v Google Cloud Platform in ustvariti nov projekt, v katerem omogočite Google Places API. Za dostop lahko uporabite isti API ključ, vendar morate zagotoviti, da je ta API v vašem projektu ustrezno omogočen. Google Places API omogoča pridobivanje podrobnih informacij o lokacijah – od imen krajev in naslovi do koordinat ter drugih podrobnosti, kot so ocene in slike. Podobno kot pri drugih Googlovih API-jih morate vnesti podatke o bančni kartici, ker brezplačni nivo omogoča le omejeno število zahtevkov na dan, dodatna uporaba pa se zaračunava (običajno je cena določena na 1000 zahtevkov). Pri uporabi lahko določite tudi omejitve, kot je maksimalno število zahtevkov na dan, s čimer zagotovite, da vaša aplikacija ne preseže dovoljenih kvot.



Slika 11: Google Places API (vir: console.cloud.google.com)

4.2.1.1.6. Levenshteinova razdalja

Levenshteinova razdalja je merilo, ki oceni, koliko osnovnih sprememb (operacij) je potrebnih, da pretvorimo en niz v drugega. Te osnovne operacije so:

- **Vstavljanje:** Dodajanje ene črke.
- **Brisanje:** Odstranjevanje ene črke.
- **Zamenjava:** Zamenjava ene črke z drugo.

Algoritem uporablja metodo dinamičnega programiranja, da poišče najkrajšo zaporedje teh operacij. Če je Levenshteinova razdalja majhna, sta niza si zelo podobna; če je visoka, pa se precej razlikujeta. Ta metoda je uporabna pri iskanju podobnih nizov, samodejnem popravljanju napak in drugih nalogah, kjer je treba oceniti stopnjo ujemanja med besedili.

4.2.1.2. Uvoz knjižnic in pridobivanje okoljskih spremenljivk

V kodo se uvozijo že prej imenovane knjižnice in moduli. dotenv.config() naloži spremenljivke iz datoteke .env v objekt process.env, kar omogoča uporabo varnih in konfigurabilnih vrednosti, kot so API ključi in FTP podatki. Nato se definirajo ključne spremenljivke, kot so scraperToken, openAiApiKey, visionApiKey in ftpConfig. V nadaljevanju ustvari novo instanco ApifyClient z uporabo pridobljenega tokena, kar omogoča izvajanje operacij preko Apify API-ja. V preostanku kode se definirajo konstantne poti in imena datotek, ki se uporabljajo za shranjevanje začasnih datotek (CACHE_DIR) ter za beleženje sporočil in obdelanih dogodkov (LOG_FILE in PROCESSED_EVENTS_LOG). Preden se začne s kakršnokoli obdelavo, koda preveri, ali mapa za predpomnilnik (CACHE_DIR) obstaja, če mapa ne obstaja, se ustvari skupaj z vsemi potrebnimi podmapami. Na koncu se definira funkcija logMessage, ki skrbi za beleženje sporočil. Ta se najprej izpišejo v konzolo, nato pa se skupaj s časovnim žigom dodajo v datoteko, kar omogoča sledljivost in lažje odpravljanje morebitnih težav med izvajanjem aplikacije.

```
import { ApifyClient } from 'apify-client';  220.7k (gzipped: 63.1k)
import dotenv from 'dotenv';  6.8k (gzipped: 3k)
import axios from 'axios';  62.3k (gzipped: 23.1k)
import fs from 'fs-extra';  30.4k (gzipped: 8.6k)
import path from 'path';  500 (gzipped: 316)
import ftp from 'basic-ftp';  28.4k (gzipped: 8.7k)
import cron from 'node-cron';  11.4k (gzipped: 4.3k)
import { isFuture, parseISO, parse, format } from "date-fns";  41.5k (gzipped: 9.8k)
import { sl } from 'date-fns/locale';  12.1k (gzipped: 3.1k)
import util from 'util';  500 (gzipped: 315)

dotenv.config();

const scraperToken = process.env.SCRAPER_TOKEN;
const openAiApiKey = process.env.CHATGPT_API_KEY;
const visionApiKey = process.env.VISION_API_KEY;

const ftpConfig = {
    host: process.env.FTP_HOST,
    user: process.env.FTP_USER,
    password: process.env.FTP_PASSWORD,
    port: process.env.FTP_PORT || 21,
    remotePathCache: process.env.FTP_REMOTE_PATH_CACHE || '/public_html/scraper/cache/',
    remotePathComplete: process.env.FTP_REMOTE_PATH_COMPLETE || '/public_html/Slike/',
};

const client = new ApifyClient({ token: scraperToken });

const CACHE_DIR = '../cache/';
const LOG_FILE = 'event.log';
const PROCESSED_EVENTS_LOG = 'processedEvents.log';

// Ensure cache directory exists
if (!fs.existsSync(CACHE_DIR)) {
    fs.mkdirSync(CACHE_DIR, { recursive: true });
}

// Logging function
function logMessage(message) {
    console.error(message);
    fs.appendFileSync(LOG_FILE, `[${new Date().toISOString()}] ${message}\n`);
}
```

Slika 12: Uvoz knjižnic in pridobivanje okoljskih spremenljivk (vir: lasten)

4.2.1.3. runScraper()

Najprej se izvrši funkcija runScraper(), ki pošlje URL profilov iz katerih želim strgat objave, kakšne podatke želim pridobiti (objave, komentarje, podatke o profilu ali reele), koliko zadnjih objav želim (v mojem primeru 6), kaj naj išče z pridobljenimi podatki (uporabnike, objave, hashtag-e ali kraje), koliko rezultatov naj bi bilo vrnjenih (25 profilov * 6 objav na profil = 150 objav) ter nazadnje ali naj doda nadrajene podatke (v mojem primeru podatke profila). Pridobljeni rezultati se nato shranjuje v spremenljivko items, ki se po zaključnem scrapanju pošlje v novo funkcijo processPosts(items).

```
async function runScraper() {
    logMessage('🚀 Starting scraper...');
    const input = {
        directUrls: [
            "https://www.instagram.com/maestroevents.eu/",
            "https://www.instagram.com/ssklub/",
            "https://www.instagram.com/cirkusklub/",
            "https://www.instagram.com/trust_maribor/",
            "https://www.instagram.com/studentski.zuri.ljubljana/",
            "https://www.instagram.com/emceplac/",
            "https://www.instagram.com/gledalisce.velenje/",
            "https://www.instagram.com/kamdanes.si/",
            "https://www.instagram.com/club.cvetlicarna/",
            "https://www.instagram.com/denea.si/",
            "https://www.instagram.com/galahala__/",
            "https://www.instagram.com/festival_ljubljana/",
            "https://www.instagram.com/mc_pekarna/",
            "https://www.instagram.com/stuk_klub/",
            "https://www.instagram.com/mojekarte.si/",
            "https://www.instagram.com/silent_disco_slovenija/",
            "https://www.instagram.com/koda.events/",
            "https://www.instagram.com/kinosiska/",
            "https://www.instagram.com/eagleevents01/",
            "https://www.instagram.com/veselica_radar/",
            "https://www.instagram.com/hushhushevents/",
            "https://www.instagram.com/bar_dolcevita/",
            "https://www.instagram.com/euterpeevents/",
            "https://www.instagram.com/sou_maribor/",
            "https://www.instagram.com/studentski_dogodki/"
        ],
        resultsType: "posts",
        resultsLimit: 6,
        searchType: "user",
        searchLimit: 150,
        addParentData: false
    };

    try {
        const run = await client.actor("shu8hvrXbJbY3Eb9W").call(input);
        const { items } = await client.dataset(run.defaultDatasetId).listItems();

        if (!items.length) {
            logMessage("⚠ No posts found.");
            return;
        }

        await processPosts(items);
    } catch (error) {
        logMessage(`❌ Scraper error: ${error.message}`);
    }
}
```

Slika 13: runScraper() funkcija (vir: lasten)

4.2.1.4. processPosts() preverjanje ustreznosti objave

Funkcija processPosts() pridobi niz podatkov vseh pridobljenih objav. V for zanki nato iz vsake objave posebej shrani ključne podatke, ki se bodo uporabljali pri procesu v posebej spremenljivke (displayUrl, caption, url, type). Prvo program preveri ali je objava definirana kot slika ali posnetek. Če je objava posnetek ga izloči, saj iščemo zgolj povabila na dogodek v obliki plakata, ki bi ga našli na ulicah. Potem pogleda, če je objava s tem URL-jem že bila procesirana.

```
async function processPosts(posts) {
    for (const post of posts) {
        const { displayUrl, caption, url, type } = post;
        logMessage(`🔹 Processing post: ${url}`);

        if (type !== "Image") {
            logMessage(`ℹ Skipping post: ${url} (not an image).`);
            continue;
        }

        // Check if this event has been processed already
        if (isEventProcessed(url)) {
            logMessage(`⚠ Event with URL ${url} has already been processed. Skipping.`);
            continue;
        }
```

Slika 14: processPosts() preverjanje ustreznosti (vir: lasten)

4.2.1.4.1. isEventProcessed()

Vrednost spremenljivke url pošlje v funkcijo isEventProcessed(), ki preveri processedEvents.log za isti url. Ta funkcija nato vrne true ali false vrednost. Če je true objavo izvrže.

```
function isEventProcessed(url) {
    try {
        if (!fs.existsSync(PROCESSED_EVENTS_LOG)) {
            fs.writeFileSync(PROCESSED_EVENTS_LOG, '');
            return false;
        }
        const data = fs.readFileSync(PROCESSED_EVENTS_LOG, 'utf8');
        const urls = data.split('\n').filter(line => line.trim() !== '');
        return urls.includes(url);
    } catch (error) {
        console.error(`❌ Error reading ${PROCESSED_EVENTS_LOG}:`, error);
        return false;
    }
}
```

Slika 15: isEventProcessed funkcija (vir: lasten)

4.2.1.5. processPosts() shranjevanje slike

Nato, če objava ustreza začetnim kriterijim sledi shranjevanje naslovne slike. Sliki se določi ime, ki temelji na trenutnem datumu in natančni uri. Ime slike ter povezava do te se nato pošlje v funkcijo downloadImage. Če objavi manjka uploadUrl se preskoči.

```
const fileName = `${Date.now()}.jpg`;
const uploadedUrl = await downloadImage(displayUrl, fileName);
if (!uploadedUrl) continue;
```

Slika 16: processPosts() shranjevanje slike (vir: lasten)

4.2.1.5.1. downloadImage()

Funkcija pridobi vrednosti in uporabi knjižnjico axios, ki pošlje GET zahtevek za nalaganje slike. responceType je nastavljen na stream, da bo odziv obravnavan kot podatkovni tok (to je pogosto uporabljeno za večje datoteke ali slike). Nato za sliko ustvari lokalno pot, kjer združi pot do mape CACHE_DIR in ime datoteke, ki se bo kasneje uporabila za lociranje slike in nalaganje slike na strežnik. Potem se odpre pisalni tok, ki bo sliko shranil na disk. Pipe() poskrbi, da se prenešeni podatki neposredno zapišejo v datoteko. Promise pa zagotavlja, da se shranjevanje zaključi pred nadaljevanjem. Ko je prenos pokliče funkcijo uploadImageToServerCache(localPath, fileName), ki sliko naloži na strežnik. Če je nalaganje uspešno vrne URL naložene slike v funkcijo processPosts(), v kolikor pa je neuspešno pa zavrne obljubo. Ob neuspešnem shranjevanju slike v datoteko, se ta posreduje naprej kot neuspešno obdelana obljuba. V primeru, da pride do kakršne koli napake pri prenosu slike, se napaka zabeleži in funkcija vrne null vrednost.

```
async function downloadImage(url, filename) {
    try {
        const response = await axios({
            url,
            responseType: 'stream',
        });

        const localPath = path.join(CACHE_DIR, filename);
        const writer = fs.createWriteStream(localPath);
        response.data.pipe(writer);

        return new Promise((resolve, reject) => {
            writer.on('finish', async () => {
                logMessage(`✅ Image downloaded: ${localPath}`);
                const uploadedUrl = await uploadImageToServerCache(localPath, filename);
                if (uploadedUrl) resolve(uploadedUrl);
                else reject("❌ FTP Upload failed");
            });
            writer.on('error', reject);
        });
    } catch (error) {
        logMessage(`❌ Error downloading image: ${error.message}`);
        return null;
    }
}
```

Slika 17: downloadImage() funkcija (vir: lasten)

4.2.1.5.2. uploadImageToServerCache()

Funkcija je odgovorna za nalaganje lokalno shranjene slike na spletni strežnik preko FTP-ja. Najprej se iz knjižnice ftp inicializira FTP odjemalec. client.ftp.verbose = true; omogoča podrobnejše zapisovanje dogajanja pri povezavi in prenosu datotek. Client.access({}) vspostavi povezavo z FTP strežnikom z uporabo konfiguracijskih nastavitev. remotePathCache vsebuje osnovno mapo na strežniku, kamor bodo shranjene slike. remoteFilename je ime datoteke, ki bo shranjena na strežniku, medtem ko pa je remotePathCache polna pot do datoteke na strežniku. client.uploadFrom(localFilePath,

remotePathCache) prenese sliko iz lokalne poti (localFilePath) na določeno mesto na FTP strežniku (remotePathCache). Funkcija nato vrne URL slike, ki je zdaj dostopna preko spleta. V kolikor pride do napake pri povezavi ali nalaganju se napaka zapiše in vrne null vrednost. Nazadnje se FTP povezava s strežnikom zapre.

```
async function uploadImageToServerCache(localFilePath, remoteFilename) {
    const client = new ftp.Client();
    client.ftp.verbose = true; // Enable logging

    try {
        await client.access({
            host: ftpConfig.host,
            user: ftpConfig.user,
            password: ftpConfig.password,
            port: ftpConfig.port,
            secure: false,
        });

        const remotePathCache = `${ftpConfig.remotePathCache}${remoteFilename}`;
        await client.uploadFrom(localFilePath, remotePathCache);
        logMessage(`✅ Uploaded to: ${remotePathCache}`);

        return `https://ulicanori.si/scraper/cache/${remoteFilename}`;
    } catch (error) {
        logMessage(`❌ FTP Upload Error: ${error.message}`);
        return null;
    } finally {
        client.close();
    }
}
```

Slika 18: uploadImageToServerCache() funkcija (vir: lasten)

4.2.1.6. processPosts() izpis besedila iz slike

Slika je bila uspešno dodana na spletni strežnik, kjer je dostopna na spletu. Pridobljena povezava ter napis objave se nato dodata v funkcijo analizeImageWithGoogleVision(). Katera vrne besedilo ter prihodnje datume.

```
const { text, futureDates } = await analyzeImageWithGoogleVision(uploadedUrl, caption);
```

Slika 19: Sklic funkcije analyzeImageWithGoogleVision() (vir: lasten)

4.2.1.6.1. analyzeImageWithGoogleImage()

Funkcija uporablja Google Vision API za prepoznavanje besedila na sliki (OCR). Najprej se pripravi zahteva za Google Vision API. googleVisionUrl je končna točka, kamor bo slika poslana, medtem ko pa requestBody vsebuje HTTP zahtevo. Nato axios pošlje zahtevo na googleVisionUrl. Odgovor se shrani v spremenljivko response za katero se nato pogleda, ali je prazna. Če je vrne prazno besedilo in prazn seznam prihodnjih datumov, ki v funkciji processPosts() pomeni, da se objava preskoči. V kolikor spremenljivka response ni prazna se odziv obdela. V spremenljivko fullText se doda

pridobljeno besedilo iz slike ter napis v sami objavi. Ostvari se nova spremenljivka futureDates v katero se shrani odgovor funkcije extractFutureDates, ki je odgovorna za pregled dobljenega besedila za kakršnekoli prihodnje datume. Celotno dobljeno besedilo se tudi izpiše za pregled kvalitete delovanja funkcije extractFutureDates(). Ob uspehu funkcija vrne celotno besedilo in zaznane prihodnje datume. V kolikor dobimo napako to izpiše in vrne prazne spremenljivke v funkcijo processPosts().

```
async function analyzeImageWithGoogleVision(imageUrl, caption) {
    try {
        const googleVisionUrl = `https://vision.googleapis.com/v1/images:annotate?key=${visionApiKey}`;
        const requestBody = {
            requests: [{ image: { source: { imageUri: imageUrl } }, features: [{ type: "TEXT_DETECTION" }] }],
        };

        const response = await axios.post(googleVisionUrl, requestBody);
        if (!response.data.responses || response.data.responses.length === 0) return { text: "", futureDates: [] };

        const annotations = response.data.responses[0].textAnnotations;
        let fullText = annotations?.[0]?.description?.toLowerCase() + caption.toLowerCase() || "";
        const futureDates = extractFutureDates(fullText);

        logMessage(`Posts text detected: ${fullText}`);

        return { text: fullText, futureDates };
    } catch (error) {
        logMessage(`❌ Google Vision API Error: ${error.message}`);
        return { text: "", futureDates: [] };
    }
}
```

Slika 20: analyzeImageWithGoogleVision() funkcija (vir: lasten)

4.2.1.6.2. extractFutureDates()

Ta funkcija je precej obširna zaradi velike količine različnih zapisov datumov, zato jo bom razdelil na več delov. Prva slika prikazuje spremenljivke, ki shranjuje različne načine zapisa datumov vse od slovenskih do angleških ter tudi različne številčne zapise.

```
function extractFutureDates(text) {
    const sloveneMonths = [
        "januar", "februar", "marec", "april", "maj", "junij",
        "julij", "avgust", "september", "oktober", "november", "december"
    ];
    const shortMonths = ["jan", "feb", "mar", "apr", "maj", "jun", "jul", "avg", "sep", "okt", "nov", "dec"];
    const declinedMonths = [
        "januarja", "februarja", "marca", "aprila", "maja", "junija",
        "julija", "avgusta", "septembra", "oktobra", "novembra", "decembra"
    ];
    const englishMonths = [
        "january", "february", "march", "april", "may", "june",
        "july", "august", "september", "october", "november", "december"
    ];
    const englishShortMonths = [
        "jan", "feb", "mar", "apr", "may", "jun",
        "jul", "aug", "sep", "oct", "nov", "dec"
    ];

    const datePatterns = [
        // Numeric dates with slash separator, e.g. "26/04" or "26/04/2025"
        /(\d{1,2})\/(\d{1,2})(?:\/(\d{4}))?/g,
        // 1. Numeric dates with spaces, e.g. "22. 11. 2025" or "22. 11."
        /(\d{1,2})\.\s*(\d{1,2})\.\s*(\d{4})?/g,
        // 2. Numeric dates without spaces, e.g. "12.11", "12.11.", "12.11.2025", "12.11.2025."
        /(\d{1,2})\.(\d{1,2})(?:\.(\d{4}))?\.?/g,
        // 3. Slovenian dates with month names (with a dot after day), e.g. "22. november 2025"
        /(\d{1,2})\.\s*(januar|februar|marec|april|maj|junij|julij|avgust|september|oktober|november|december|jan|f
        // 4. Slovenian dates with month names (without dot after day), e.g. "12 april" or "12 april 2025"
        /(\d{1,2})\s+(januar|februar|marec|april|maj|junij|julij|avgust|september|oktober|november|december|jan|feb
        // 5. English dates in "day month year" format with optional ordinal suffix, e.g. "22nd November 2025"
        /(\d{1,2})(?:st|nd|rd|th)?\s+(january|february|march|april|may|june|july|august|september|october|november|
        // 6. English dates in "month day, year" format, e.g. "November 22, 2025"
        /(january|february|march|april|may|june|july|august|september|october|november|december|jan|feb|mar|apr|may
    ];
```

Slika 21: Variacije datumov v extractFutureDates() funkciji (vir: lasten)

Naslednji del funkcije vsebuje logiko. Najprej se določi trenutno leto ter ustvari prazen seznam foundDates. Z zanko forEach pogleda vse vzorce datumov, medtem ko z zanko while poišče vse ujemajoče se datume. V naslednji if zanki preveri ali je mesec zapisan z številko ali besedo. Če je mesec beseda (match[1] ni številka), ga shrani v spremenljivko month, sicer ostane match[1]. V primeru, da leto ni dodano se privzeto nastavi trenutno leto. Če je mesec v besedilni obliki, ga pretvori v številčno obliko. Preveri slovenske in angleške oblike mesecev. Sledi oblikovanje datuma v obliko »d.M.yyyy« (12.4.2025). S funkcijo parse() iz knjižnice date-fns pretvori datum v objekt. Z funkcijo isFuture(parsedDate) preveri ali je datum v prihodnosti. Če je ga v obliki ISO 8601 (yyyy-MM-dd'T'HH:mm:ssXXX) in doda v seznam foundDates. Funkcija nato vrne seznam vseh prihodnjih datumov.

```
    const currentYear = new Date().getFullYear();
    const foundDates = [];

    datePatterns.forEach(pattern => {
        let match;
        while ((match = pattern.exec(text)) !== null) {
            let day, month, year;
            if (isNaN(match[1])) {
                month = match[1];
                day = match[2];
                year = match[3] || currentYear;
            } else {
                day = match[1];
                month = match[2];
                year = match[3] || currentYear;
            }

            if (isNaN(month)) {
                const lowerMonth = month.toLowerCase();
                if (sloveneMonths.includes(lowerMonth)) {
                    month = sloveneMonths.indexOf(lowerMonth) + 1;
                } else if (shortMonths.includes(lowerMonth)) {
                    month = shortMonths.indexOf(lowerMonth) + 1;
                } else if (declinedMonths.includes(lowerMonth)) {
                    month = declinedMonths.indexOf(lowerMonth) + 1;
                } else if (englishMonths.includes(lowerMonth)) {
                    month = englishMonths.indexOf(lowerMonth) + 1;
                } else if (englishShortMonths.includes(lowerMonth)) {
                    month = englishShortMonths.indexOf(lowerMonth) + 1;
                }
            }

            const dateString = `${day}.${month}.${year}`;
            const parsedDate = parse(dateString, "d.M.yyyy", new Date(), { locale: sl });

            if (isFuture(parsedDate)) {
                foundDates.push(format(parsedDate, "yyyy-MM-dd'T'HH:mm:ssXXX"));
            }
        }
    });

    return foundDates;
}
```

Slika 22: Preostanek extractFutureDates() funkcije (vir: lasten)

4.2.1.7. processPosts() preverjanje izpisa in pošiljanje podatkov za strukturiranje dogodka

Iz funkcije analizeImageWithGoogleVision() sem dobil besedilo iz slike združen z napisom iz objave in prihodnje datume. Program prvo preveri ali je seznam futureDates null, nedefiniran, false ali prazen. V koliko je prazen izpiše sporočilo in nadaljuje z naslednjo objavo, če ni pa izpiše vse najdene datume ter pošlje besedilo v novo funkcijo extractEventDetails, ki bo vrnila podatke v dogodku v obliki JSON.

```
if (!futureDates || futureDates.length === 0) {
    logMessage(`⏭ No future events detected in ${url}. Moving to next post.`);
    continue;
}

logMessage(`✅ Extracted future dates: ${JSON.stringify(futureDates)}`);

const eventDetails = await extractEventDetails(text);
```

Slika 23: processPosts() preverjanje izpisa in pošiljanje podatkov za strukturiranje dogodka (vir: lasten)

4.2.1.7.1. extractEventDetails()

Funkcija uporablja ChatGPT API (GPT-4-Turbo) za izvleček dogodkov iz nestrukturiranega besedila, prepoznavanje ključnih informacij, strukturiranje dogodka v JSON ter ponovno preverjanje, ali so dogodki v prihodnosti. Funkcija pošlje POST zahtevo na ChatGPT API, ki vsebuje model, navodila za sistem, navodila uporabnika, temperaturo (kreativnost) in maksimalno velikost odgovora. ChatGPT API nato izvleče podatke o dogodku, kot so ime dogodka, opis, naslov, datum in čas začetka, datum in čas konca, mesto, iz nabora izbere 2 ali več kategoriji in doda organizatorja iz posredovanega besedila. Preverja tudi, če je ta objava zgolj promocija in nagradna igra (če dogodek vsebuje besede, kot so giveaway, like, share, se izbriše iz rezultatov).

```
async function extractEventDetails(text) {
    try {
        const response = await axios.post(
            "https://api.openai.com/v1/chat/completions",
            {
                model: "gpt-4-turbo",
                messages: [
                    {
                        role: "system",
                        content: `You are an AI assistant specialized in extracting events from unstructured text. Your primary goal is to parse the input text and
                        identify each event with 100% reliability in a structured JSON format. Please follow these rules:

                        1. **Input**: You will receive raw text describing one or multiple events (sometimes with partial or messy formatting).

                        2. **Output**: Return a **valid JSON array** of event objects. Each event object must have the following fields exactly:
                        - "title": (string) - The official or most relevant event name.
                        - "description": (string) - A short, marketable description of the event. If there's no explicit description, refine the text that appears next to the
                        event name. if there is any mention of having to apply to the event make sure to add that.
                        - "address": (string) - The address or location of the event. **This field cannot be null or empty, and must not contain any links or URL-like content.**
                        If the address is missing in the text:
                            - If multiple events are detected in the text, infer the address by cross-referencing the other events.
                            - If only one event is detected, search the provided text for organization or venue names and use that information to populate the address.
                        - "start": (string, ISO 8601 format) - The date/time the event starts (e.g. "2025-02-01T21:00:00"). If time is missing, set it to an approximate time
                        (e.g. "20:00:00"). if you are not sure about the year assume current. The current year is 2025.
                        - "end": (string, ISO 8601 format) - The date/time the event ends. If not stated, approximate (e.g. +4 hours from start).
                        - "city": (string) - The city in which the event takes place. **This field cannot be null or empty.** Try to guess the city from the context of the text.
                        If the city is missing:
                            - If multiple events are detected, infer the city from the other events' data.
                            - If only one event is detected, search the provided text for organization/venue names or location hints to determine the city.
                        - "categories": (array of strings) - Choose two or more categories from the following fixed list: ["Koncert", "Veselica", "Družina", "Klub", "Festival",
                        "Dobrodelno", "Kultura", "Šport", "Druženje"]. If no category is clearly applicable, pick the closest match. Keep the categories in Slovene language.
                        - "organization": (string) - The name of the organization hosting or organizing the event. This must be as accurate as possible. If the organization name is
                        not explicitly provided, infer it from the text (using any organization or venue names mentioned).

                        3. **Do not include any extra keys**. Only return these fields in each event object.

                        4. **Multiple events**: If multiple events are found in the text, return them as separate objects in the JSON array.

                        5. **Validity**: The final answer must be valid JSON with no additional commentary, text, or explanations. Do not wrap the JSON in markdown or code fences.

                        6. **Language**: Preserve the original language or context for names and descriptions. If the text is partially in Slovenian, keep it as such.

                        7. **Consistency**: If you find repeated info about date or name, choose the most consistent data. **Ensure that the 'address' and 'city' fields are never empty.**
                        If any event lacks these, use cross-event inference or, if only one event is detected, search the text for organization or venue names to fill in the missing data.

                        8. **Giveaway Dismissal**: Before outputting an event, check if its title or description contains giveaway or contest keywords (such as "giveaway", "follow",
                        "like", "share", "tag"). If these keywords are present and the content is primarily promotional (i.e., lacking clear event details like date, time, location),
                        dismiss the event entirely and do not include it in the final JSON array.

                        Follow these instructions carefully and output only the JSON array.`
                    },
                    {
                        role: "user",
                        content: `Text:\n${text}`
                    }
                ],
                temperature: 0.6,
                max_tokens: 2000
            },
            {
                headers: {
                    Authorization: `Bearer ${openAiApiKey}`,
                    "Content-Type": "application/json"
                }
            }
```

Slika 24: ChatGPT poziv v extractEventDetails() funkciji (vir: lasten)

V tem delu kode prejme besedilo iz API-ja in ga očisti (trim()). Nato preveri, ali se podatki pravnilo začnejo z [in končajo z] (JSON array). Če format ni pravilen, poskuša odstraniti morebitne odvečne znake. Nazadnje pa pretvori JSON besedilo v javascript objekt.

```
let extractedData = response.data.choices[0]?.message?.content.trim();
logMessage(`📦 Extracted Events: ${extractedData}`);

if (!extractedData.startsWith("[") || !extractedData.endsWith("]")) {
    logMessage("❌ Invalid JSON array format received. Attempting to fix...");
    extractedData = extractedData.substring(extractedData.indexOf("["), extractedData.lastIndexOf("]") + 1);
}

extractedData = extractedData
    .replace(/,\s*}/g, "}")
    .replace(/,\s*\]/g, "]")
    .replace(/(?<!\\)"/g, '"');

const events = JSON.parse(extractedData);
```

Slika 25: Preverjanje JSON v extractEventDetails() funkciji (vir: lasten)

Nato program preveri, če dogodek vsebuje začetni čas v prihodnosti. Ponovno ga preveri z že obravnavano funkcijo extractFutureDates(). Če dogodek ne vsebuje datuma v prihodnosti to sporoči in dogodek se zavrže.

```
const futureEvents = events.filter(event => {
    if (!event || !event.start) {
        logMessage(`❌ Missing start date for event: ${event?.title || 'Unnamed event'}`);
        return false;
    }
    // Format the ISO date to the expected format (e.g., "d. M. yyyy")
    const formattedDate = format(parseISO(event.start), "d. M. yyyy", { locale: sl });
    // Use extractFutureDates to check if the formatted date is in the future.
    return extractFutureDates(formattedDate).length > 0;
});

if (futureEvents.length === 0) {
    logMessage("⏭ No events with a valid future start date found.");
    return [];
}
```

Slika 26: Preverjanje datuma v extractEventDetails() funkciji (vir: lasten)

V kolikor dogodek še vedno vsebuje datum v prihodnosti se kliče nova funkcija verifyEventDetails(). Funkcija nato počaka na rezultat, ki ga vrne naza v metodo processPosts().

```
        return await verifyEventDetails(futureEvents);
    } catch (error) {
        logMessage(`❌ Error processing event details with ChatGPT API: ${error.message}`);
        return null;
    }
}
```

Slika 27: Klicanje nove funkcije v extractEventDetails() funkciji (vir: lasten)

4.2.1.7.2. verifyEventDetails() vrednosti v poizvedbo

V tem delu programa je najprej deklarirana prazna spremenljivka verfiedEvents, ki bo uporabljena za shranjevanje dogodkov, ki so bili potrjeni. Nato se deklarira spremenljivka monthMapping, ki shranjuje vse možne zapise meseca. Ta je uporabljena za pregledovanje pridoblenih rezultatov iz Googla, če vsebujejo kakršnekoli različice datumov. Potlej se v for zanki pregleda vsak dogodek posebej. Za vsak dogodek v spremenljivko eventTitle shrani ime dogodka, eventFullDate shrani slovenski zapis datuma (d. M. yyyy) in v eventOrganization shrani ime organizacije dogodka. Ti podatki se nato izpišejo za lažje preverjanje napak. Ustvari se spremenljivka searchQuery, ki vsebuje datum dogodka ter organizatorja dogodka, ki se nato pošlje v funkcijo searchGoogle(). Odgovor se nato shrani v spremenljivko googleResults.

```
async function verifyEventDetails(events) {
    const verifiedEvents = [];

    // Define a mapping for months with acceptable alternatives in both Slovenian and English.
    const monthMapping = {
      "1": ["1", "01", "januar", "januarja", "january", "jan"],
      "2": ["2", "02", "februar", "februarja", "february", "feb"],
      "3": ["3", "03", "marec", "marca", "march", "mar"],
      "4": ["4", "04", "april", "apr"],
      "5": ["5", "05", "maj", "may"],
      "6": ["6", "06", "junij", "junija", "june", "jun"],
      "7": ["7", "07", "julij", "julija", "july", "jul"],
      "8": ["8", "08", "avgust", "avgusta", "august", "aug"],
      "9": ["9", "09", "september", "septembra", "sep"],
      "10": ["10", "10", "oktober", "oktobra", "october", "oct"],
      "11": ["11", "11", "november", "novembra", "nov"],
      "12": ["12", "12", "december", "decembra", "dec"]
    };

    for (const event of events) {
      // Extract the first 5 words of the event title.
      const eventTitle = event.title;
      // Format the full date in Slovenian (e.g., "5. 5. 2025").
      const eventFullDate = format(parseISO(event.start), 'd. M. yyyy', { locale: sl });
      const eventOrganization = event.organization;

      logMessage(`🔎 Searching Google for event: ${eventTitle} ${eventFullDate} ${eventOrganization}`);

      // Construct enhanced search query with full date.
      const searchQuery = `${eventFullDate} ${eventOrganization}`;
      const googleResults = await searchGoogle(searchQuery);
```

Slika 28: spremenljivke in pošiljanje podatkov iz verifyEventDetails() v searchGoogle() (vir: lasten)

4.2.1.7.3. searchGoogle()

V tej funkciji se v spremenljivko searchApiKey shrani API ključ iz .env datoteke, ki sem ga pridobil iz Google Cloud strani. Spremenljivka cx predstavlja Search Engine ID, da ve kateri po meri brskalnik uporabiti. Nato se sestavi URL, ki vsebuje poizvedbo (query), ki smo jo omenili prej (eventFullDate in eventOrganization), doda se ključ (searchApiKey) ter ID iskalnika (cx). Axios nato pošlje HTTP zahtevo. Odgovor se shrani v spremenljivko response, ki se nato vrne v funkcijo verifyEventDetails().

```
async function searchGoogle(query) {
    try {
        const searchApiKey = process.env.GOOGLE_SEARCH_API_KEY;
        const cx = process.env.GOOGLE_SEARCH_CX;

        const searchUrl = `https://www.googleapis.com/customsearch/v1?q=${encodeURIComponent(query)}&key=${searchApiKey}&cx=${cx}`;

        const response = await axios.get(searchUrl);
        return response.data.items || [];
    } catch (error) {
        logMessage(`❌ Google Search API Error: ${error.message}`);
        return [];
    }
}
```

Slika 29: searchGoogle() funkcija

4.2.1.7.4. verifyEventDetails() filtriranje odgovorov

V tem if stavku se prvo pogleda, če so bili vrnjeni rezultati iskanja, v kolikor so bili v spremenljivko topResults shrani samo prve 3, da ohranimo čim večjo natančnost. Nato se

posebej iz začetnega časa dogodka shrani dan, mesec in leto. V spremenljivko monthAlternatives se shranijo vsi različni zapisi mesecev. Dan, meseci v različni obliki in leto se nato shranijo v spremenljivko dateRegex, ki se v nadaljne uporablja za preverjanje, če odgovori iz Googla vsebuje kakršnokoli obliko pravilnega datuma. Preverjanje sem moral naknadno dodati, saj sem opazil, da so se v Googlovih rezultatih pojavljali tudi odgovori z drugačnim datumom, kar je v nadaljevanju zmedlo ChatGPT.

```
if (googleResults.length > 0) {
  // Work with the top 3 Google results.
  const topResults = googleResults.slice(0, 3);

  // Extract day, month, and year from the event date.
  const eventDate = parseISO(event.start);
  const day = format(eventDate, 'd', { locale: sl });
  const year = format(eventDate, 'yyyy', { locale: sl });
  const monthNumeric = format(eventDate, 'M', { locale: sl });

  // Build a regex for the month alternatives.
  const monthAlternatives = monthMapping[monthNumeric].join("|");
  // The regex now matches:
  // - The day (with an optional trailing dot)
  // - Any combination of whitespace and/or dots as separator
  // - One of the allowed month representations (with an optional trailing dot)
  // - Optionally, any separator followed by the year (if present)
  const dateRegex = new RegExp(`\\b${day}\\.?[\\s\\.]*(?:${monthAlternatives})\\.?(?:[\\s\\.]+${year})?\\b`, "i");

  // Filter top results to include only snippets that mention the event date in any allowed format.
  const validResults = topResults.filter(result => {
    if (result.snippet) {
      const snippet = result.snippet.trim();
      return dateRegex.test(snippet);
    }
    return false;
  });
```

Slika 30: Filtriranje odgovorov v verifyEventDetails() funkciji (vir:lasten)

Pravilni rezultati se nato shranijo v combinedSnippets spremenljivko. Prej pa rezultatom odstranimo morebitne odvečne presledke in združi vse elemente novega polja v en sam niz. Nato se kliče funkcija getCityFromPlace(), ki iz naslova preveri mesto dogodka.

```
// Combine valid snippet texts.
const combinedSnippets = validResults
  .map(result => result.snippet.trim())
  .join("\n");

const eventWithCity = await getCityFromPlace(event);
```

Slika 31: Združevanje pravilnih odgovorov in klicanje funkcije getCityFromPlace() v verifyEventDetails() (vir: lasten)

4.2.1.7.5. getCityFromPlace() pridobivanje podatkov

Funkcija getCityFromPlace najprej preveri, če dogodek vsebuje naslov. Če naslov ni definiran, se vrne null vrednost dogodka, kar pomeni, da se dogodek nebo zapisal v bazo podatkov. Nato uporabi Google Places API: najprej s klicem Find Place, da na podlagi naslova pridobi identifikator kraja (place_id), in nato s klicem Place Details, da pridobi podrobnosti o naslovu (address_components in formatted_address). Na ta način pripravi podatke, s katerimi je mogoče določiti mesto za dani dogodek.

```
async function getCityFromPlace(event) {

    if (!event.address) {
        logMessage("⚠ Event does not have an address to verify city.");
        return null;
    }

    try {
        const apiKey = process.env.GOOGLE_MAPS_API_KEY;
        const address = event.address;
        // First, use Find Place to get the place_id
        const findPlaceUrl = `https://maps.googleapis.com/maps/api/place/findplacefromtext/json?input=${encodeURIComponent(address)}&inputtype=textquery&fields=place_id&key=${apiKey}`;
        const findResponse = await axios.get(findPlaceUrl);
        if (findResponse.data.status !== 'OK' ||
            !findResponse.data.candidates ||
            findResponse.data.candidates.length === 0) {
            logMessage("Find Place API error:", findResponse.data.error_message || findResponse.data.status);
            return null;
        }
        const placeId = findResponse.data.candidates[0].place_id;

        // Next, use Place Details to get the address components
        const detailsUrl = `https://maps.googleapis.com/maps/api/place/details/json?place_id=${placeId}&fields=address_components,formatted_address&key=${apiKey}`;
        const detailsResponse = await axios.get(detailsUrl);
        if (detailsResponse.data.status !== 'OK' || !detailsResponse.data.result) {
            logMessage("Place Details API error:", detailsResponse.data.error_message || detailsResponse.data.status);
            return null;
        }

        const addressComponents = detailsResponse.data.result.address_components;
```

Slika 32: pridobivanje podatkov v getCityFromPlace() funkciji (vir: lasten)

4.2.1.7.6. getCityFromPlace() iskanje podatkov

Ta koda preišče podatke o naslovu, shranjene v polju addressComponents. Najprej iterira skozi vse komponente in išče tisto, katere tip vključuje 'postal_town'. Če najde takšno komponento, nastavi spremenljivko foundCity na njeno dolgo ime in prekine zanko. Če pa ni najdena nobena komponenta z 'postal_town', nato iterira še enkrat in išče komponento, katere tip vključuje 'administrative_area_level_1'. Ko jo najde, nastavi foundCity na njeno dolgo ime in prekine zanko.

```
let foundCity = null;

for (const component of addressComponents) {
    if (component.types.includes('postal_town')) {
        foundCity = component.long_name;
        break;
    }
}

if (!foundCity) {
    for (const component of addressComponents) {
        if (component.types.includes('administrative_area_level_1')) {
            foundCity = component.long_name;
            break;
        }
    }
}
```

Slika 33: Iskanje podatkov v getCityFromPlace() funkciji (vir: lasten)

4.2.1.7.7. getCityFromPlace() vstavljanje podatkov

Če je spremenljivka foundCity definirana, se njena vrednost dodeli lastnosti city objekta event, obenem pa se zabeleži sporočilo o uspešni posodobitvi dogodka. Če foundCity ni najden, se zabeleži opozorilo, da za dani naslov ni bilo mogoče določiti mesta, in funkcija vrne null. Na koncu funkcija vrne posodobljen dogodek, medtem ko v primeru morebitne

izjeme z uporabo util.inspect zabeleži podrobne informacije o napaki in prav tako vrne null (dogodek se ne zapiše).

```
        if (foundCity) {
            event.city = foundCity;
            logMessage(`✅ Updated event city to: ${foundCity} based on address search.`);
        } else {
            logMessage(`⚠ Could not find city for: ${event.address}`);
            return null;
        }

        return event;
    } catch (error) {
        logMessage("Error in getCityFromPlace: " + util.inspect(error, { depth: null }));
        return null;
    }
}
```

Slika 34: Vstavljanje podatkov v getCityFromPlace() funkciji (vir: lasten)

4.2.1.7.8. Klicanje refineEventWithChatGPT() v verifyEventDetails()

V kolikor obstajajo preverjeni Googlovi rezultati se izvrši funkcija refineEventWithChatGPT(). V funkcijo se pošlje eventWithCity in preverjeni rezultati.

```
// If valid snippets exist, refine the event using ChatGPT with those snippets.
if (combinedSnippets) {
  logMessage(`🔎 Extracted and validated snippets: ${combinedSnippets}`);
  const refinedEvent = await refineEventWithChatGPT(eventWithCity, combinedSnippets);
```

Slika 35: Klicanje refineEventWithChatGPT() v verifyEventDetails() funkciji (vir: lasten)

4.2.1.7.9. refineEventWithChatGPT() poziv

Tako izgleda drugi poziv ChatGPT API-ju. Je zelo podroben, saj moramo ChatGPT-ju postavit stroga navodila. Njegova naloga je, da dopolni dogodek, če kaj manjka in išče primernejše podatke v besedilu pridobljenemu iz Googlovih rezultatov. V glavi se ponovno uporabi openAiApiKey, ki je bil definiran v prvem delu kode.

```
async function refineEventWithChatGPT(event, googleSnippets) {
    try {
        const response = await axios.post(
            "https://api.openai.com/v1/chat/completions",
            {
                model: "gpt-4-turbo",
                messages: [
                    {
                        role: "system",
                        content: `You are an AI assistant specializing in event fact-checking and optimization.
                        Your task is to compare the provided event details with any new relevant information extracted from the event's description and
                        Google search snippets, and update the following fields if better or missing information is available:
                        - **Title**: Examine the event title and description. If the title appears truncated or incomplete, update it by appending any missing
                        words while preserving its overall meaning.
                        - **Organization**: If the event's organization is missing or incomplete, update it with the most accurate name found in the description or search results.
                        - **Start time and End time**: Look for explicit time values (in the format HH:MM or HH:MM:SS) in the description that are clearly
                        associated with the event's date. Update only the time portion of the start and end fields (keeping the original date unchanged) if a more
                        precise time is found. If multiple time values are present, select the one most directly linked to the event's date.
                        - **Address**: If the event's address is missing or incomplete, search the description and Google search snippets for any venue or location
                        information (such as "Gala Hala", "Metelkova mesto", etc.) and update the address with the most specific, complete information available and must
                        not contain any links or URL-like content.
                        - **City**: Only update the 'city' field based on the address information if a clear, more accurate city is determined.
                        - **Description**: Update with a more detailed and marketable description if available, ensuring that the language matches the local language of
                        the event's city.
                        - **Categories**: If you find any categories thta match better with the event than change them. Choose from the following fixed list: ["Koncert",
                        "Veselica", "Družina", "Klub", "Festival", "Dobrodelno", "Kultura", "Šport", "Druženje"]. If no category is clearly applicable take a look at the
                        context of the event and pick the closest two matches. Keep the categories in Slovene language.

                        Return the event as a valid JSON object with the exact format below:
                        {
                        "title": "Event Name",
                        "address": "Event Address",
                        "description": "Marketable event description in the local language",
                        "start": "YYYY-MM-DDTHH:MM:SSZ",
                        "end": "YYYY-MM-DDTHH:MM:SSZ",
                        "city": "City Name",
                        "categories": ["category1", "category2"],
                        "organization": "Organizer Name"

                        }
                        The response must be a valid JSON object only, with no additional text or commentary.`
                    },
                    {
                        role: "user",
                        content: `Here is an event extracted from text:
                        ${JSON.stringify(event, null, 2)}

                        Google Search Results Snippets:
                        ${googleSnippets}

                        - Check the event's description for any new, more precise start or end time information. If a clear time value (e.g., "20:00" or "22:00") is
                        found that is associated with the event's date, update only the time portion while preserving the date.
                        - Update the organization if it is missing or if a more accurate organization name is found in the description or search snippets.
                        - Update the address if it is missing or if a better, more complete address is found in the description or search snippets and do not include URLs.
                        - Update the city if it is missing or if a more accurate city can be determined from the address or description.
                        - Update the description with a more detailed and marketable version, ensuring the language matches the local language of the event's city.
                        - Examine the event title; if it appears truncated or incomplete, update it by appending any missing words while preserving its overall meaning.
                        - Do not change the categories.
                        Return only a valid JSON object in the exact format specified above.`
                    }
                ],
                temperature: 0.3,
                max_tokens: 700
            },
            {
                headers: {
                    Authorization: `Bearer ${openAiApiKey}`,
                    "Content-Type": "application/json"
                }
            }
        );
```

Slika 36: Poziv ChatGPT API v refineEventWithChatGPT() funkciji (vir: lasten)

4.2.1.7.10. refineEventWithChatGPT() preverjanje formata

Ta del kode je odgovoren za shranjevanje odgovora v correctedData spremenljivko. Funkcija JSON.parse pretvori niz (string) correctedData v javascript objekt. Rezultat se shrani v correctedEvent spremenljivko. Nato preveri ali correctedEvent ni prazen, če vsebuje začetek dogodka in, ali je začetek dogodka v prihodnosti. Če ni dogodek preskoči. Drugače se vrne v funkcijo verifyEventDetails().

```
        let correctedData = response.data.choices[0]?.message?.content.trim();
        logMessage(`📝 Fact-Checked & Optimized Event: ${correctedData}`);

        try {
            const correctedEvent = JSON.parse(correctedData);

            // Ensure the event has a valid start date before returning
            if (!correctedEvent || !correctedEvent.start || !isFuture(parseISO(correctedEvent.start))) {
                logMessage(`⏭ Skipping past/invalid event: ${correctedEvent?.title || "Unnamed Event"}`);
                return null;
            }

            return correctedEvent;
        } catch (error) {
            logMessage(`❌ Error parsing optimized event data: ${error.message}`);
            return null;
        }
    } catch (error) {
        logMessage(`❌ Error refining event with ChatGPT: ${error.message}`);
        return null;
    }
}
```

Slika 37: Preverjaje formata v refineEventWithChatGPT() funkciji (vir: lasten)

4.2.1.7.11. verifyEventDetails() vračanje dogodka

Če funkcija refineEventWithChatGPT() uspešno vrne dogodek se ta doda v polje verifiedEvents. V kolikor izboljšanega dogodka ni se doda dogodek, ki je bil obdelan z vidika preverjanja mesta. V primeru, če Googlovi rezultati ne vsebujejo veljavnih informacij se doda v polje verifiedEvents dogodek, ki je bil obdelan za preverjanje mesta (eventWithCity). Če pa googlovih rezultatov sploh ni, se izpiše opozorilno sporočilo, kljub temu pa se preveri mesto z metodo getCityFromPlace() in tako se obdelan dogodek doda v verifiedEvents. Funkcija vrne končni seznam obdelanih dogodkov v funkcijo extarctEventDetails(), ki takoj vrne obdelan dogodek v processPosts() funkcijo.

```
          if (refinedEvent) {
            verifiedEvents.push(refinedEvent);
          } else {
            verifiedEvents.push(eventWithCity);
          }
        } else {
          // No valid snippets found, so push the event with the verified (or unmodified) city.
          verifiedEvents.push(eventWithCity);
        }
      } else {
        logMessage(`⚠ No web results found for event: ${event.title}, keeping extracted data.`);
        // Even if no web results are found, verify the city from the address.
        const eventWithCity = await getCityFromPlace(event);
        verifiedEvents.push(eventWithCity);
      }
    }

    return verifiedEvents;
}
```

Slika 38: Vračanje dogodkov v verifyEventDetails() funkciji (vir: lasten)

4.2.1.8. processPosts() dodajanje slike v mapo Slike

Po obdelavi dogodka se slika dogodka pošlje v drugo mapo z razlogom, da se bo mapa cache na strežniku pogosto praznila. Prvo definira pot do slike, ki je shranjena lokalno v

localFilePath spremenljivko, ki se nato pošlje v funkcijo uploadImageToServerComplete() skupaj z imenom slike. Pot do slike se shrani v spremenljivko completeUploadUrl, za nadaljno obravnavo pri dodajanju v bazo. Če dodajanje ni uspešno dogodek preskoči.

```
if (eventDetails && Array.isArray(eventDetails)) {
    // Calculate local file path from the cached image
    const localFilePath = path.join(CACHE_DIR, fileName);

    // Upload the image to the complete folder on the server
    const completeUploadUrl = await uploadImageToServerComplete(localFilePath, fileName);
    if (!completeUploadUrl) {
        logMessage(`❌ Failed to upload image to complete folder for ${url}. Skipping event.`);
        continue;
    }
```

Slika 39: Dodajanje slike v mapo Slike v processPosts() funkciji (vir: lasten)

4.2.1.8.1. uploadImageToServerComplete()

Funkcija deluje podobno, kot uploadImageToServerCache. Spremenjena je samo pot v katero sliko naloži (remotePathComplete).

```
async function uploadImageToServerComplete(localFilePath, remoteFilename) {
    const client = new ftp.Client();
    client.ftp.verbose = true; // Enable logging

    try {
        await client.access({
            host: ftpConfig.host,
            user: ftpConfig.user,
            password: ftpConfig.password,
            port: ftpConfig.port,
            secure: false, // Change to `true` if using FTPS
        });

        const remotePathComplete = `${ftpConfig.remotePathComplete}${remoteFilename}`;
        await client.uploadFrom(localFilePath, remotePathComplete);
        logMessage(`✅ Uploaded to: ${remotePathComplete}`);

        return `https://ulicanori.si/Slike/${remoteFilename}`;
    } catch (error) {
        logMessage(`❌ FTP Upload Error: ${error.message}`);
        return null;
    } finally {
        client.close();
    }
}
```

Slika 40: uploadImageToServerComplete() funkcija (vir: lasten)

4.2.1.9. processPosts() dodajanje v bazo

Končna naloga funkcije processPosts() je, da pokliče funkcijo addToDatabase() in funkciji posreduje obdelane podatke, kot so completeUploadUrl (pot so slike na

strežniku), url (povezava do Instagram objave) in podatke o dogodku. Po uspešnem dodajanju v bazo pokliče metodo markEventAsProcessed() in poreduje povezavo do instagram obaje.

```
            for (const event of eventDetails) {
                logMessage(`📅 Extracted future event: ${JSON.stringify(event)}`);
                // Pass completeUploadUrl (instead of uploadedUrl) to the database
                addToDatabase({ completeUploadUrl, url, ...event });
                // Mark the event as processed
                markEventAsProcessed(url);
            }
        }
    }
}
```

Slika 41: Dodajanje v bazo v ProcessPosts() funkciji (vir: lasten)

4.2.1.9.1. addToDatabase()

Metoda pošlje pridobljene podatke preko API-ja na spletni strežnik, kjer se v add-event.php kodi obdelajo.

```
async function addToDatabase(eventData) {
    try {
        const apiEndpoint = 'https://ulicanori.si/api/add-event.php';
        const response = await axios.post(apiEndpoint, eventData, {
            headers: { "Content-Type": "application/json" }
        });
        console.log("✅ Event added:", response.data);
    } catch (error) {
        console.log("❌ Error adding event:", error.response?.data || error.message);
    }
}
```

Slika 42: addToDatabase() funkcija (vir: lasten)

4.2.1.9.2. markEventAsProcessed()

Po obdelavi in dodajanje v bazo se povezava do instagram objave shrani v datoteki, ki omogoča, da se iste primerne objave ne preverjajo več. To dolgoročno zmanjša stroške.

```
function markEventAsProcessed(url) {
    try {
        fs.appendFileSync(PROCESSED_EVENTS_LOG, url + '\n');
        console.log(`✅ Successfully marked ${url} as processed.`);
    } catch (error) {
        console.error(`❌ Error writing to ${PROCESSED_EVENTS_LOG}:`, error);
    }
}
```

Slika 43: markEventAsProcessed() funkcija (vir: lasten)

4.2.2. add-event.php

Program add-event.php je odgovoren za sprejemanje podatkov preko API-ja, preverjanje, če dogodek že obstaja, normaliziranje naslova ter dodajanje dogodka v bazo.

4.2.2.1. Nastavitve API

Ta PHP koda nastavi API, ki sprejema le POST zahteve z JSON podatki. Najprej se nastavi Content-Type na JSON in omogoči CORS (dostop iz vseh virov) z dovoljenjem samo za POST metode. Nato se vključijo konfiguracijske nastavitve za povezavo z bazo iz datoteke UlicaNori_baza.php. Koda preveri, ali je prejet zahtevani tip POST, če ni, vrne napako in prekine izvajanje. Nato s funkcijo file_get_contents("php://input") prebere telo zahtevka in ga z json_decode pretvori v PHP asociativno polje. Če JSON ni veljaven, se vrne ustrezno napako. S tem se zagotovi, da strežnik obdeluje le pravilno oblikovane POST zahteve z veljavnim JSON podatkom.

```php
<?php
header("Content-Type: application/json");
header("Access-Control-Allow-Origin: *");
header("Access-Control-Allow-Methods: POST");
header("Access-Control-Allow-Headers: Content-Type");

require_once '../Baza/UlicaNori_baza.php';

if ($_SERVER['REQUEST_METHOD'] !== 'POST') {
    echo json_encode(["error" => "Invalid request method"]);
    exit;
}

$jsonData = file_get_contents("php://input");
$data = json_decode($jsonData, true);

if (!$data) {
    echo json_encode(["error" => "Invalid JSON"]);
    exit;
}
```

Slika 44: Nastavitve API v add-event.php (vir: lasten)

4.2.2.2. Pripravljanje podatkov za uporabo

Ta koda najprej preveri, ali vhodni podatki vsebujejo vsa zahtevana polja (npr. »title«, »description«, »address«, »start«, »end«, »city«, »categories« in »organization«). Če katero izmed teh polj manjka ali je prazno, se vrne JSON napaka in izvajanje se ustavi. Nato se vrednosti pridobijo iz vhodnih podatkov in ustrezno očistijo z metodo real_escape_string, s čimer se prepreči SQL injekcija. Datum začetka in konca se pretvorita v format Y-m-d H:i:s, za URL slike se preveri prisotnost ključa completeUploadUrl ter, če vsebuje podniz »Slike/«, se izreže del poti. Podobno se preveri tudi URL dogodka.

```php
$requiredFields = ["title", "description", "address", "start", "end", "city", "categories", "organization"];
foreach ($requiredFields as $field) {
    if (!isset($data[$field]) || empty($data[$field])) {
        echo json_encode(["error" => "Missing required field: $field"]);
        exit;
    }
}

$title          = $link->real_escape_string($data['title']);
$description    = $link->real_escape_string($data['description']);
$address        = $link->real_escape_string($data['address']);
$startFormatted = date('Y-m-d H:i:s', strtotime($data['start']));
$endFormatted   = date('Y-m-d H:i:s', strtotime($data['end']));
$city           = $link->real_escape_string($data['city']);
$organization   = $link->real_escape_string($data['organization']);

$image = isset($data['completeUploadUrl']) ? $link->real_escape_string($data['completeUploadUrl']) : "";

if (strpos($image, "Slike/") !== false) {
    $image = substr($image, strpos($image, "Slike/"));
}

$eventUrl = isset($data['url']) ? $link->real_escape_string($data['url']) : "";
```

Slika 45: Pripravljanje podatkov za uporabo v add-event.php (vir: lasten)

4.2.2.3. Preprečevanje podvojenih dogodkov

Ta koda preverja, ali je nov dogodek podvojen z obstoječim v bazi. Najprej se inicializira spremenljivka $duplicateEventId na null. Nato se izvede SQL poizvedba, ki pridobi ID, naslov, začetek dogodka, organizacijo, opis in ime mesta iz tabel dogodki in kraji. Če poizvedba vrne rezultate, se ustvari asociativno polje $newEvent, ki vsebuje podatke o novem dogodku (naslov, mesto, začetek, opis). Nato se zanka while uporablja za sprehod skozi obstoječe dogodke, za vsakega se ustvari asociativno polje $existingEvent z ustreznimi podatki. Funkcija isDuplicateEvent() se uporabi za primerjavo novega dogodka z obstoječimi. Če se ugotovi, da je dogodek podvojen, se najprej pogleda, če ima nov dogodek daljši opis. V kolkor je to res se spremenljivka $duplicateEventId nastavi na ID podvojenega dogodka in zanka se prekine.

```php
$duplicateEventId = null;
$checkDuplicatesQuery = "
    SELECT d.id, d.naslov, d.opis, d.zacetek_dog, d.organizacija, k.ime as city
    FROM dogodki d
    JOIN kraji k ON d.kraj_id = k.id
";
$result = $link->query($checkDuplicatesQuery);
if ($result && $result->num_rows > 0) {
    $newEvent = [
        'address'     => $address,
        'city'        => $city,
        'start'       => $startFormatted,
        'description' => $description
    ];
    while ($row = $result->fetch_assoc()) {
        $existingEvent = [
            'address'     => $row['naslov'],
            'city'        => $row['city'],
            'start'       => $row['zacetek_dog'],
            'description' => $row['opis']
        ];
        if (isDuplicateEvent($existingEvent, $newEvent)) {
            // If a duplicate event is found, only replace it if the new description is longer.
            if (strlen($newEvent['description']) <= strlen($existingEvent['description'])) {
                echo json_encode(["error" => "Duplicate event exists with equal or longer description"]);
                exit;
            } else {
                $duplicateEventId = $row['id'];
                break;
            }
        }
    }
}
```

Slika 46: Preprečevanje podvojenih dogodkov v add-event.php (vir: lasten)

4.2.2.3.1. isDuplicateEvent()

Funkcija isDuplicateEvent() primerja dva dogodka in ugotavlja, ali gre za podvojen dogodek. Najprej za vsak dogodek normalizira naslov s funkcijo normalizeAddress ter prilagodi mesto tako, da odstrani odvečne presledke in vse črke spremeni v male. Nato pretvori datum začetka dogodka v standardni format “Y-m-d”, s čimer se primerja samo dan dogodka, ne pa tudi točnega časa. Za primerjavo naslovov uporabi dve metodi: najprej preveri, ali je eden od normaliziranih naslovov podniz drugega, nato pa s pomočjo Levenshteinove razdalje oceni, ali sta si naslova dovolj podobna (če je razdalja manjša od 20 % dolžine daljšega naslova). Če se naslov, mesto in datum začetka ujemajo, funkcija vrne true, kar pomeni, da sta dogodka podvojena, sicer vrne false.

```php
function isDuplicateEvent($existingEvent, $newEvent) {
    $normAddress1 = normalizeAddress($existingEvent['address']);
    $normAddress2 = normalizeAddress($newEvent['address']);

    $normCity1 = strtolower(trim($existingEvent['city']));
    $normCity2 = strtolower(trim($newEvent['city']));

    $existingDate = date("Y-m-d", strtotime($existingEvent['start']));
    $newDate = date("Y-m-d", strtotime($newEvent['start']));

    // Check if one normalized address is a substring of the other
    $substringMatch = (strpos($normAddress1, $normAddress2) !== false || strpos($normAddress2, $normAddress1) !== false);

    // Additionally, perform fuzzy matching using the Levenshtein distance
    $levDistance = levenshtein($normAddress1, $normAddress2);
    $maxLen = max(strlen($normAddress1), strlen($normAddress2));
    $fuzzyMatch = ($maxLen == 0) ? true : ($levDistance / $maxLen < 0.2);

    // Combine both methods: if either check indicates a match, consider the addresses the same.
    $sameAddress = $substringMatch || $fuzzyMatch;
    $sameCity = ($normCity1 === $normCity2);
    $sameDate = ($existingDate === $newDate);

    return $sameAddress && $sameCity && $sameDate;
}
```

Slika 47: isDuplicateEvent() funkcija (vir: lasten)

4.2.2.3.2. normalizeAddress()

Funkcija normalizeAddress() očisti in enotno oblikuje vhodni naslov. Najprej ga pretvori v male črke, nato odstrani vse znake, ki niso črke, številke ali presledki. Nato odstrani besedo »klub« in na koncu odstrani odvečne presledke. Rezultat je očiščen, standardiziran naslov.

```php
function normalizeAddress($address) {
    $address = strtolower($address);

    $address = preg_replace('/[^\p{L}\p{N}\s]/u', '', $address);

    $address = str_replace('klub', '', $address);

    $address = preg_replace('/\s+/', ' ', trim($address));
    return $address;
}
```

Slika 48: normalizeAddress() funkcija (vir: lasten)

4.2.2.4. Preverjanje kraja

Ta koda preveri, ali v tabeli »kraji« obstaja zapis, kjer se ime ujema s spremenljivko $city. Najprej se z ustreznim SQL poizvedbo pridobi ID kraja, ki ustreza danemu imenu, in se omeji na en rezultat. Nato se izvede poizvedba preko $link->query(). Če rezultat obstaja in vsebuje vsaj eno vrstico, se z fetch_assoc() pridobi ID in shrani v spremenljivko $kraj_id. Če zapisov ni, se $kraj_id nastavi na 0.

```php
$queryKraj = "SELECT id FROM kraji WHERE ime = '$city' LIMIT 1";
$resultKraj = $link->query($queryKraj);
if ($resultKraj && $resultKraj->num_rows > 0) {
    $rowKraj = $resultKraj->fetch_assoc();
    $kraj_id = $rowKraj['id'];
} else {
    $kraj_id = 0;
}
```

Slika 49: Preverjanje kraja v add-event.php (vir: lasten)

4.2.2.5. Dodajanje dogodka in ravnanje z ponavljajočimi se dogodki

Ta koda najprej vstavi nov dogodek v tabelo dogodki z uporabo SQL ukaza INSERT, kjer se vnesejo podatki, kot so ime, naslov, opis, slika, datum začetka, datum konca, ID kraja, organizacija in URL dogodka. Če se poizvedba uspešno izvede, se s pomočjo $link->insert_id pridobi ID novega dogodka in shrani v spremenljivko $newEventId. Nato, če je zaznan podvojeni dogodek (kar pomeni, da je spremenljivka $duplicateEventId nastavljena), se iz tabele vsecki pridobijo vsi ID-ji uporabnikov, ki so povezani s podvojenim dogodkom, in se shranijo v polje $userIds. Po tem se odstranijo vsi zapisi iz tabele vsecki in tabele dogodek_kategorija, ki se nanašajo na podvojeni dogodek, nato se podvojen dogodek izbriše tudi iz tabele dogodki. Na koncu se za vsakega uporabnika, ki je bil povezan s podvojenim dogodkom, ustvari nov zapis v tabeli vsecki, kjer se uporabniški ID poveže z ID-jem novega dogodka, s čimer se ohranijo vse povezave med uporabniki in dogodkom.

```
$insertEventQuery = "INSERT INTO dogodki (ime, naslov, opis, slika, zacetek_dog, konec_dog, kraj_id, organizacija, url)
    VALUES ('$title', '$address', '$description', '$image', '$startFormatted', '$endFormatted', '$kraj_id', '$organization', '$eventUrl')";

if ($link->query($insertEventQuery)) {
    $newEventId = $link->insert_id;

    if ($duplicateEventId) {

        $userIds = [];
        $queryUserIds = "SELECT uporabnik_id FROM vsecki WHERE dogodek_id = '$duplicateEventId'";
        $resultUserIds = $link->query($queryUserIds);
        if ($resultUserIds && $resultUserIds->num_rows > 0) {
            while ($row = $resultUserIds->fetch_assoc()) {
                $userIds[] = $row['uporabnik_id'];
            }
        }

        $link->query("DELETE FROM vsecki WHERE dogodek_id = '$duplicateEventId'");

        $link->query("DELETE FROM dogodek_kategorija WHERE dogodek_id = '$duplicateEventId'");
        $link->query("DELETE FROM dogodki WHERE id = '$duplicateEventId'");

        foreach ($userIds as $userId) {
            $link->query("INSERT INTO vsecki (uporabnik_id, dogodek_id) VALUES ('$userId', '$newEventId')");
        }
    }
```

Slika 50: Dodajanje in ravnanje z dogodki v add-event.php (vir: lasten)

4.2.2.6. Dodajanje kategorij

Ta koda iterira skozi vse kategorije, ki so navedene v vhodnih podatkih. Za vsako kategorijo najprej očisti ime z metodo real_escape_string, nato pa izvede SQL poizvedbo, da preveri, ali v tabeli kategorije obstaja zapis z ustreznim imenom. Če se ustrezna kategorija najde, se njen ID shrani v spremenljivko, ki se nato uporabi za vstavljanje novega zapisa v tabelo dogodek_kategorija. Ta zapis poveže nov dogodek (identificiran z $newEventId) s kategorijo. Po uspešni obdelavi vseh kategorij se vrne JSON sporočilo o uspehu (“Event added successfully”), v primeru pa morebitne napake se vrne JSON sporočilo z opisom napake. Na koncu se zapre povezava z bazo.

```
    foreach ($data['categories'] as $category) {
        $cat = $link->real_escape_string($category);
        $catQuery = "SELECT id FROM kategorije WHERE ime = '$cat' LIMIT 1";
        $catResult = $link->query($catQuery);
        if ($catResult && $catResult->num_rows > 0) {
            $catRow = $catResult->fetch_assoc();
            $kategorija_id = $catRow['id'];
            $insertMappingQuery = "INSERT INTO dogodek_kategorija (dogodek_id, kategorija_id) VALUES ('$newEventId', '$kategorija_id')";
            $link->query($insertMappingQuery);
        }
    }

    echo json_encode(["success" => "Event added successfully"]);
} else {
    echo json_encode(["error" => "Database error: " . $link->error]);
}

$link->close();
?>
```

Slika 51: Dodajanje kategorij v add-event.php (vir: lasten)

4.3. PROCESIRANJE PODATKOV

Veliko podatkov se izpiše v events.log, kjer jih lahko preverjam in zagotovim, da se vsi nepravilni podatki popravijo.

4.3.1. Objava na instagramu

Tako izgleda neka objava na Instagramu.

Slika 52: Objava ŠŠ Kluba (vir: https://www.instagram.com/ssklub/)

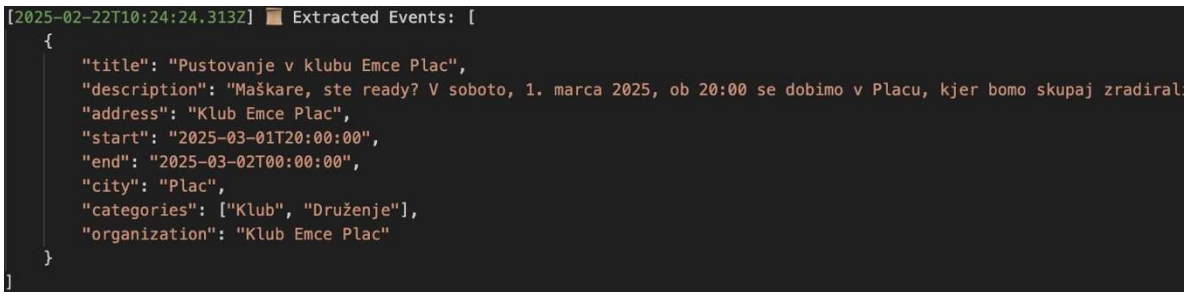
4.3.2. Besedilo iz slike in napis

Sliko pošlje v Google Vision API in iz nje izpiše besedilo. Doda se tudi napis objave (caption). Preveri se ali besedilo vsebuje prihodnje datume. Če jih se pošlje v ChatGPT API za obravnavo.

Slika 53: Pridobljeno besedilo in napis (vir: lasten)

4.3.3. Pošiljanje v ChatGPT za strukturiranje

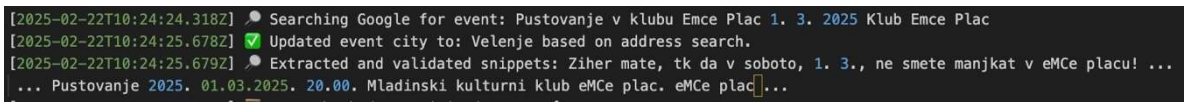
ChatGPT API podatke obdela in pošlje dogodek strukturiran v JSON.



Slika 54: Strukturiran dogodek (vir: lasten)

4.3.4. Iskanje dogodka na spletu in preverjanje mesta

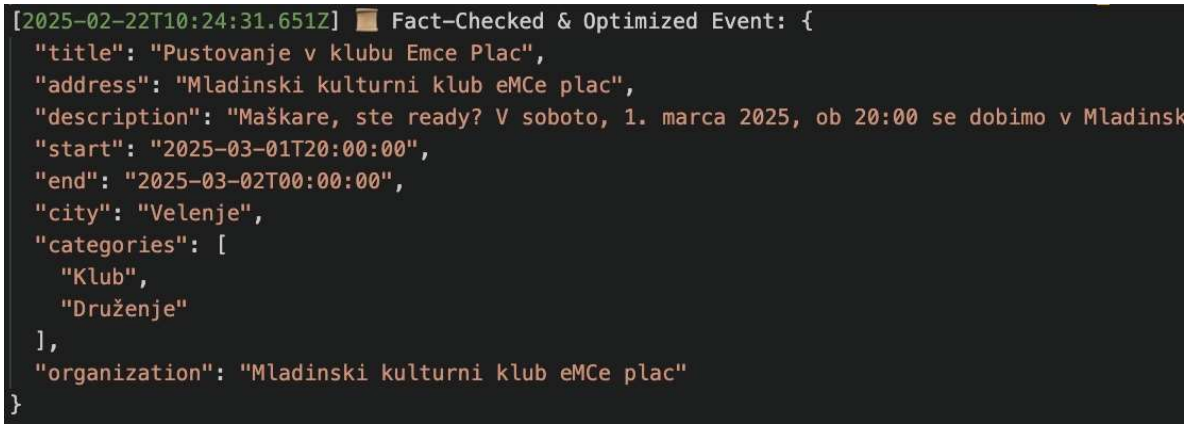
Datum dogodka in organizacija sta uporabljena za iskanje preko spleta. Odgovori, ki vsebujejo pravilen datum in, jih bo ChatGPT v prihodnje uporabil za dodatno izpolnitev podatkov se izpišejo. Izpiše se tudi novo pridobljeno mesto iz naslova.



Slika 55: Preverjanje mesta in izpis najdenih spletnih virov (vir: lasten)

4.3.5. Ponovno izpolnjevanje z rezultati iskanja

Pridobleni viri iz spleta se nato ponovno pošljejo v ChatGPT API skupaj z dogodkom, ki vsebuje pravilno mesto. Dogodek ponovno struktuira.



Slika 56: Ponovno strukturiranje dogodka (vir: lasten)

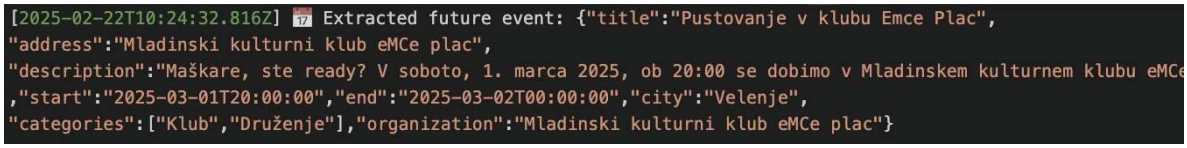
4.3.6. Dodajanje slike v pravilno mapo

Izpiše potrditev o dodajanju slike v pravilno mapo.

Slika 57: Potrditev o dodajanju slike (vir: lasten)

4.3.7. Izpis dogodka in vpis v bazo

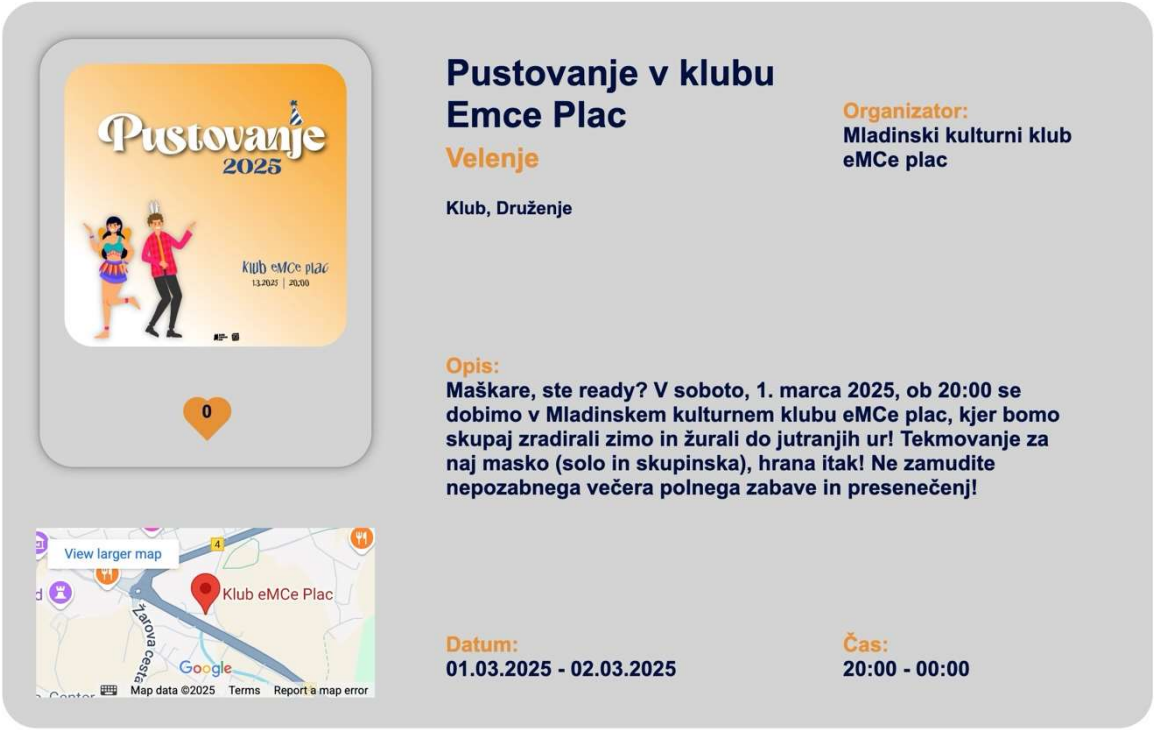
Izpišejo in dodajo se samo dogodki, ki vsebujejo datum v prihodnosti.



Slika 58: Izpis dogodka z datumom v prihodnosti (vir: lasten)

4.3.8. Končni izgled dogodka

Dogodek je nato dostopen na spletni strani www.ulicanori.si.



Slika 59: Dogodek na www.ulicanori.si (vir: lasten)

5. REZULTATI IN ANALIZA

5.1. NATANČNOST SISTEMA PRI PREPOZNAVANJU DOGODKA

V aplikaciji je definiranih 25 profilov iz družbenih omrežij. Sistem gleda zadnjih 6 objav vsakega profila, kar je 150 objav. Od tega je relevantnih dogodkov, kar pomeni dogodkov v prihodnosti vseh skupaj 56 (dne 22.02), če štejemo tudi podroben pregled napisa pod oz. ob objavljeni sliki ali videoposnetku.

Pri naslednjem delu mi je pomagal sošolec, ki je za vse napisane profile pogledal zadnjih 6 objav. Iz njih je izpisal iste podatke, kot jih je moral program. Našel je 47 dogodkov, za katerih izpis je porabil 1uro in 37 min. Informacije, ki niso bile razvidne je moral iskati po spletu. Urna postavka študenta je trenutno bruto 7,34€, kar znaša približno 11,74€. Njegova natačnost je bila 82%, saj je eno objavo podvojil. Kar pomeni, da je pravih dogodkov našel le 46. Spregledal je 10 skritih dogodkov, 8 od teh so bili napovedniki za nadaljne dogodke v napisu pod sliko (caption). Ti dogodki bodo v prihodnosti vrjetno predstavljeni posebej. 2 dogodka sta bila videoposnetka, ki jih je vrjetno preskočil, ker nista izgledala, kot vabilo na dogodek.

Program je našel 48 dogodkov. Zanje je porabil 22 minut in 10 sekund. Cena procesiranja 150 objav je znesla 2,15$, kar je pretvorjeno v evre 2,05€. Njegova natančnost je prav tako dosegla skoraj 84%, saj je en dogodek podvojil, zaradi različnih formatov lokacije. En je uporabljal ime kluba, medtem ko drug ulični naslov kluba. Izpustil je 9 aktualnih dogodkov na objavi, ki je vsebovala 18 dogodkov od tega 10 aktualnih.

Preden sodite sposobnosti prijatelja sem jaz preverjal, koliko dogodkov naj bi bilo. Ne bom se lagal po podrobnem iskanju sem jih našel 50. Šele, ko sem prvič program zagnal sem videl, koliko dogodkov sem izpustil. Program je našel vseh 6 skritih, ki jih jaz nisem.

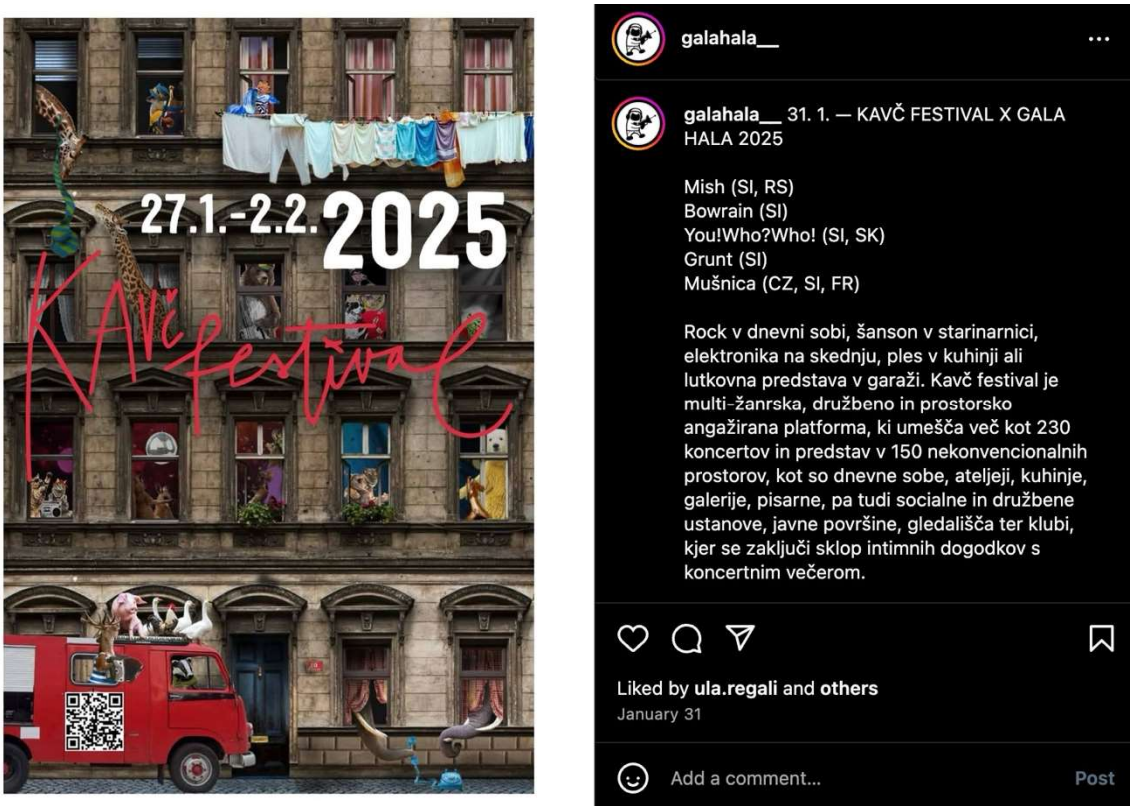
5.2. PREDNOSTI IN OMEJITVE SISTEMA

Prednost sistema je predvsem cena. Če imamo zaposlenega študenta, ki mora na dan preveriti več sto profilov za nove objave, pri tem predvidevamo, da bo njegov delovni čas 1 uro na dan (odvisno od števila novih dogodkov) lahko zanj na mesec odštejemo približno kar 220€. Za vsakodnevno delovanje aplikacije, ki bo pregledovala samo zadnji 2 objavi dnevno, saj organizacije redko objavijo več dogodkov hkrati, bi zanj odšteli približno 0,68€ za pregled 25 Instagram profilov, kar na mesec znaša 20,40€. Če ne upoštevamo mesečne in dnevne brezplačne ravni storitev. Da ne govorimo o tem, da mora nek študent stalno preverjati, če je dogodek že vnešen v bazo, kar mu lahko vzame še več časa.

Prednost sistema je tudi ta, da svoje napake skozi čas, ko organizacija doda nove podatke o dogodku. Potencialne napake sam odpravi v kolikor so datum dogodka, ulični naslov dogodka in kraj dogodka pravilni.

Sistem ima veliko omejitev. Čeprav se to težko razbere iz rezulatov raziskovanja. Prvi problem, ki lahko nastane je z samim strganjem objav iz Instagrama. Temu lahko uzamejo pravico uporabe njihovega API za pridobivanje objav, kar lahko onesposobi program za kakšen dan ali mogoče dva.

Naslednji problem nastane pri zaznavanju besedila iz slike. Če je besedilo težko berljivo, ga ne more prebrati in dodati v strukturiranje dogodka. To sem rešil z dodajanju napisa iz Instagrama (caption), ki lahko vsebuje vse potrebno in se dogodek vseeno uspešno doda. Če pa to še vedno ni dovolj, v kolikor se dogodek v procesu zavrže lahko naslednji dan, ko se program ponovno zažene objava ponovno predela.



Slika 60: Primer težko berljive slike dogodka (vir: www.instagram.com/galahala__)

Naslednji problem, ki nastane je raznolikost objav. Kot vsak človek ima tudi vsaka objava svoj stil na katerega se težko prilagodimo. Nekatere imajo zelo podrobne opise, na katere se lahko zanašamo, spet druge imajo v opisu samo kakšen stavek ali pa nič. Zato je največja omejitev ChatGPT API-ja doslednost. Zmožnost, da vsak dogodek napiše v popolnoma enaki strukturi. Kot sem že omenil, isti dogodek lahko ima 2 stila. En je slabo opisan in bo v njegov naslov vnesel ime kluba, drug je zelo dobro opisan zato uporabi naslov kluba. Ne glede na to kako jasno mu poveš, kaj naj uporabi. Stvar se lahko zelo hitro samo poslabša, saj moramo gledati širše in ne le na točno tisto problematično objavo.

Ena večjih omejitev je prepoznavanje večjega števila dogodkov iz ene objave. Zelo je odvisno od tega kakšen dan ima ChatGPT in koliko dogodkov je na sliki. Enkrat izpiše vseh 18 dogodkov iz slike drugič pa samo 8, brez da bi spreminjal kakršenkoli del kode. Ko so pa na sliki 3-je ali 4-je dogodki pa gre vse kot po maslu. Za tak primer se lahko nanašam zgolj na organizacijo, da bo pred dogodkom objavila oglas pogrešanega dogodka, ki se bo nato predelal in dodal v bazo.

Slika 61: Objava, kjer slika vsebuje 18 dogodkov (vir: www.instagram.com/galahala__)

Omejitve predstavlja tudi vpis točnih mest, saj ChatGPT API nima dostopa do interneta in ni zmožen samo pogledati spleta za naslov organizacije ali za mesto, kjer je lociran. V ta namen sem uporabil Google Places API, ki je zelo dobra rešitev, ampak hkrati v Sloveniji obstaja veliko istih naslovov, kar pomeni da v kolikor mesto ni dodano v naslov kluba (address) lahko Google Places API vrne čisto drugo mesto, čeprav je lahko ChatGPT mesto pravilno izbral iz konteksta. Če do tega pride in Google PSE slučajno najde potrjen odgovor iz spleta, ki vsebuje tudi mesto se lahko ta napaka odpravi. Ta rešitev sicer ni 100%, ampak bo za to razisovalno nalogo zadostovala.

Omejitve nastanejo tudi pri preverjanju, če je nek dogodek že v bazi, ker se na večino podatkov, kot so ime in opis dogodka težko zanesemo, saj se lahko stalno spreminjajo. Iz izkušenj sem opazil, da so najbolj točni datum začetka dogodka, mesto dogodka in naslov dogodka. Zato sem uporabil te.

5.3. ANALIZA STROŠKOV

Najdražje orodje, ki ga uporabljam je ChatGPT API, ki za popolno strukturiranje in preverjanje 50 dogodkov porabi žetonov v vrednosti 1,72$, kar je pretvorjeno 1,64€. Za strganje objav iz Instagrama odštejemo približno 0,32$, kar je pretvorjeno 0,31€ na 150 rezultatov. Če ne štejemo mesečne brezplačne ravni soritve, ki je 5$. Storitve Google Cloud so precej dostopne posameznikom, zato bi zanje za predelavo 150 objav moral plačati okoli 0,11$, vendar, ker brezplačnih kreditov še nisem uporabil, so bile vse storitve brezplačne.

6. RAZPRAVA

1. Izdelan program lahko hitreje in učinkoviteje prepozna dogodke iz družbenih omrežij kot ročni vnos uporabnikov.

Kot ste opazili je odgovor odvisen od vrste objav, ki jih preverjam in učinkovitost uporabljenih orodij, ki niso vedno povsem zanesljiva. Če pa gledamo zgolj moje rezultate, ki so bili bolj učinkoviti in so do zaključka prišli hitreje, kot prijatelj, lahko povzamemo hipotezo kot **potrjeno**.

2. Sistem umetne inteligence bo dosegel vsaj 80 % natančnost pri prepoznavanju ključnih informacij dogodkov (naslov, datum, kategorija, organizator).

Hipotezo lahko **potrdimo**, saj je sistem dosegel nivo natančnosti 84%.

3. Avtomatiziran sistem bo dolgoročno zmanjšal potrebo po ročnem vnosu dogodkov in s tem zmanjšal stroške vzdrževanja platforme.

Potreba po ročnem vnosu se bo zagotovo zmanjšala, saj je število napak sorazmerno napakam študenta. Zagotavljala bo tudi manjše dolgoročne stroške vzdrževanja platforme, zato lahko hipotezo **potrdimo**.

7. ZAKLJUČEK

V to raziskovalno nalogo sem se podal z bolj malo znanja o obstoju različnih orodij in okvirnem znanju o jeziku node.js. Pritegnil me je projekt, ki sem ga naredil v 3. letniku programa Tehnik računalništva. Spletna stran, ki omogoča organizacijam dodajanje svojih dogodkov. Moto spletne strani je bil in še vedno je »Vsi dogodki na enem mestu«. Od zaključka projekta nisem nehal razmišljati o načinih kako rešiti začetni problem (problem kokoši in jajca). Iskal sem načine, kako so to rešila druga podjetja. Logična se mi je zdela samo ena in sicer strganje podatkov iz spleta in družbenih omrežij. Začetna misel je bila »To bo enostavno potrebujem samo program za pridobivanje informacij o dogodkih in drugo bo naredila umetna inteligenca«. Spoznal sem, da se na umetno inteligeno, vsaj zaenkrat še ne moremo dovolj zanesti, brez uporabe drugih orodij, ki te podatke preverjajo.

Nadaljne delo na tem področju bi lahko vključevalo preverjanje ali so na voljo kakšna druga bolj zanesljiva orodja za prepoznavo besedila iz slik in preverjanje ali obstaja kakšno drugo podjetje katere model umetne inteligence bi bil bolj primeren. Pomembno je tudi poudariti, da se vsak mesec umetna inteligenca bolj in bolj razvije. Vedno več je tudi novih orodij. Zato je ključno, da smo vedno seznanjeni z novimi tehnologijami in sistemi.

8. POVZETEK

Umetna inteligenca potrebuje še veliko časa, da bo imela »človeško« razmišljanje. Se pa lahko z uporabo različnih drugih orodij precej približamo nekemu idealu, ki ga pričakujemo od človeka.

Problem kokoš in jajce temelji na zahtevnosti začetne populacije baze podatkov.

Zanimalo me je ali lahko svoj problem rešim zgolj z strganju podatkov iz Instagrama in uporabo umetne inteligence.

Izdelal in testiral sem program, ki uporablja strganje objav iz Instagrama in orodja, kot so Google Vision API, ChatGPT API, Google Places API in Google CSE za izpis, sturkturiranje in preverjanje podatkov o dogodkih, ki jih nato pošlje z uporabo API-ja na spletno stran, kjer se dodajo v bazo.

Potrdil sem, da je možno ustvariti program, ki presega ročni vnos podatkov, ki temelji na uporabi umetne inteligence.

Mislim, da mi je rešitev za problem kokoši in jajca precej dobro uspela. Program je dovolj natančen pri preverjanju, strukturiranju in beleženju dogodkov v bazo do te mere, da potrebuje res zgolj minimalne ročne popravke.

9. ZAHVALA

Zahvaljujem se mojemu mentorju Samotu Železniku za začetne ideje, mnenja in vodstvu skozi projekt. Zahvaljujem se tudi sošolcu Eneju Polaku, ki mi je pomagal pri primerjavi vnosa programa in fizičnemu vnosu ter preverjanju 150-tih objav in njegovo podporo skozi nalogo.

10. VIRI IN LITERATURA

- https://www.consilium.europa.eu/sl/policies/ai-explained/
- https://www.growthmentor.com/glossary/chicken-and-egg-problem/
- https://www.nfx.com/post/19-marketplace-tactics-for-overcoming-the-chicken-or-egg-problem
- https://www.ibm.com/think/topics/data-automation
- https://www.imperva.com/learn/application-security/web-scraping-attack/
- https://www.deeplearning.ai/resources/natural-language-processing/
- https://www.ibm.com/think/topics/optical-character-recognition
- https://medium.com/ondemand/how-uber-solved-its-chicken-and-egg-problem-and-you-can-too-fab1be824984
- https://newsinitiative.withgoogle.com/
- https://www.facebook.com/help/572885262883136/
- https://www.sprinklr.com/
- https://www.brandwatch.com/
- https://nodejs.org/docs/latest/api/
- https://docs.apify.com/api/client/js/
- https://cloud.google.com/vision/docs
- https://developers.google.com/custom-search/docs/overview
- https://developers.google.com/maps/documentation/places/web-service/overview
- https://platform.openai.com/docs/api-reference/introduction
- https://dev.mysql.com/doc/
- https://www.php.net/manual/en/