

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, inž. inf.

Velenje, 2025

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

Mentor: Samo Železnik, inž. inf.

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 XII, 57 str., 4 pregl., 6 graf., 52 sl., 3 pril., 38 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, Microsoft Azure Vision in Google Places API. V raziskovalni nalogi preverim, čas dela, pokritost objav, natančnost in ceno v primerjavi s 26 dijaki. V natančnost spadajo pravilen ulični naslov dogodka, pravilno mesto dogodka in pravilen datum začetka dogodka.

KEY WORDS DOCUMENTATION

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

CX application / programing / artificial intelignce / 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 WITH ARTIFICIAL INTELLIGENCE

DT RESEARCH WORK

NO XII, 57 p., 4 tab., 6 graf, 52 fig., 3 ann., 38 ref.

LA SL

AL sl / en

AB The research paper focuses on the use of artificial intelligence for the 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, Microsoft Azure Vision and Google Places API. In the research, I examine work time, post coverage, accuracy, and cost compared to 26 students. Accuracy includes the correct street address of the event, the correct city of the event, and the correct event start date.

KAZALO VSEBINE

1. UVOD	1
2. HIPOTEZE	2
3. PREGLED STANJA TEHNIKE.....	3
3.1. PROBLEM KOKOŠI IN JAJCA V DIGITALNIH PLATFORMAH	3
3.1.1. Obsotoječe rešitve tega problema	3
3.2. UMETNA INTELIGENCA KOT REŠITEV PROBLEMA.....	4
3.2.1. Kako UI pomaga pri avtomatiziranem zbiranju podatkov.....	4
3.2.1.1. Glavne metode avtomatiziranega zbiranja podatkov	4
3.2.2. Pregled sorodnih rešitev	5
4. METODOLOGIJA IN TEHNIČNA IMPLEMENTACIJA.....	6
4.1. PREGLED UPORABLJENIH TEHNOLOGIJ	6
4.1.1. Node.js in njegovi moduli.....	6
4.1.2. Apify client za scraping	6
4.1.3. Google Vision API za analizo slik	7
4.1.4. Google PSE za preverjanje informacij na spletu	7
4.1.5. Google Places API za iskanje mest.....	8
4.1.6. OpenAI API za obdelavo besedila	8
4.1.7. PHP API in MySQL.....	9
4.1.8. Microsoft Azure Vision.....	9
5. RAZVOJ REŠITVE	10
5.1. PRIDOBIVANJE OBJAV IZ INSTAGRAMA.....	10
5.2. OBDELAVA IN PREVERJANJE PODATKOV	10
5.2.1. Pridobivanje besedila iz slike.....	10
5.2.2. Strukturiranje dogodka v JSON format	11
5.2.3. Pridobivanje mesta dogodka in pošiljanje v bazo.....	13
5.2.4. Preverjanje in dodajanje v bazo	13
5.3. PRAKTIČNI PRIMER POTEKA OBDELAVE	14
5.3.1. Uvod.....	14
5.3.2. Objava na Instagramu	15
5.3.3. Besedilo iz slike in napis.....	15
5.3.4. Pošiljanje v ChatGPT za strukturiranje.....	16
5.3.5. Iskanje po spletu in posodabljanje dogodka	16
5.3.6. Izpis končnega dogodka.....	17

5.3.7.	Končni izgled dogodka	17
5.4.	PREDNOSTI IN OMEJITVE SISTEMA	19
6.	ANALIZA REZULTATOV	23
6.1.	PRIMERJANJE FIZIČNEGA VNOSA Z VNOSOM PROGRAMA	23
6.1.1.	Natančnost.....	24
6.1.2.	Cena	25
6.2.	OCENITEV KOREKCIJE NAPAK APLIKACIJE.....	25
6.3.	IZRAČUN STROŠKOV	26
7.	RAZPRAVA.....	28
8.	ZAKLJUČEK	30
9.	POVZETEK.....	31
10.	ZAHVALA	32
11.	PRILOGE	33
11.1.	REZULTATI PRVEGA TESTIRANJA.....	33
11.1.1.	Zanesljivost	33
11.1.2.	Natančnost.....	33
11.1.3.	Cena.....	34
11.2.	NAVODILA DIJAKOM	35
11.3.	SCRAPER.JS	36
11.3.1.	Uvoz knjižnic in pridobivanje okoljskih spremenljivk	36
11.3.2.	runScraper().....	37
11.3.3.	processPosts() preverjanje ustreznosti objave.....	38
11.3.3.1.	SanitizeOwnerFullName().....	39
11.3.3.2.	IsEventProcessed()	39
11.3.4.	ProcessPosts() shranjevanje slike.....	39
11.3.4.1.	DownloadImage()	40
11.3.4.2.	UploadImageToServerCache()	41
11.3.5.	ProcessPosts() izpis besedila iz slike	42
11.3.5.1.	AnalyzeImageWithAzure().....	43
11.3.5.2.	ExtractFutureDates()	43
11.3.6.	ProcessPosts() pošiljanje podatkov za strukturiranje dogodka	45
11.3.6.1.	ExtractEventDetails()	45
11.3.6.2.	GetCityFromPlace() pridobivanje podatkov	47
11.3.6.3.	GetCityFromPlace() iskanje podatkov	48

11.3.6.4.	GetCityFromPlace() vstavljanje podatkov	49
11.3.7.	ProcessPosts() dodajanje slike v mapo Slike	49
11.3.7.1.	UploadImageToServerComplete().....	49
11.3.8.	ProcessPosts() dodajanje v bazo	50
11.3.8.1.	AddToDatabase().....	50
11.3.8.2.	ProcessPosts() drugi tipi	51
11.3.8.3.	ProcessPosts() konec	51
11.3.8.4.	MarkEventAsProcessed()	51
11.4.	ADD-EVENT.PHP	52
11.4.1.	Nastavitve API.....	52
11.4.2.	Pripravljanje podatkov za uporabo.....	52
11.4.3.	Preprečevanje podvojenih dogodkov in zagotavljanje kakovosti	53
11.4.3.1.	IsDuplicateEvent().....	54
11.4.4.	Dodajanje dogodka in ravnanje s ponavljajočimi se dogodki.....	55
11.4.5.	Dodajanje kategorij	56

KAZALO SLIK

Slika 1: Node.js.....	6
Slika 2: Apify.com	7
Slika 3: Google Vision API	7
Slika 4: Google PSE	8
Slika 5: Google Places API	8
Slika 6: OpenAI	9
Slika 7: Microsoft Azure Vision	9
Slika 8: Diagram poteka strganja podatkov	10
Slika 9: Diagram potek pridobivanja besedila iz slike	11
Slika 10: Diagram poteka strukturiranja dogodka	12
Slika 11: Diagram poteka pridobivanja mesta dogodka	13
Slika 12: Diagram poteka preverjanja in dodajanja v bazo	14
Slika 13: Objava Štuk kluba	15
Slika 14: Pridobljeno besedilo	16
Slika 15: Strukturiran dogodek.....	16
Slika 16: Ponovno strukturiranje dogodka	17
Slika 17: Izpis dogodka z datumom v prihodnosti	17
Slika 18: Dogodek na www.ulicanori.si	18
Slika 19: Primer težko berljive slike dogodka	19
Slika 20: Primer odličnega napovednika	20
Slika 21: Primer zelo slabega napovednika	21
Slika 22: Uvoz knjižnic in pridobivanje okoljskih spremenljivk	37
Slika 23: RunScraper() funkcija	38
Slika 24: ProcessPosts() preverjanje ustreznosti	38
Slika 25: SanitizeOwnerFullName()	39
Slika 26: IsEventProcessed funkcija.....	39
Slika 27: ProcessPosts() shranjevanje slike	40
Slika 28: DownloadImage() funkcija	41
Slika 29: UploadImageToServerCache() funkcija.....	42
Slika 30: Sklic funkcije analyzeImageWithAzure()	42
Slika 31: AnalyzeImageWithAzure() funkcija	43
Slika 32: Variacije datumov v extractFutureDates() funkciji	44
Slika 33: Preostanek extractFutureDates() funkcije	45
Slika 34: ProcessPosts() pošiljanje podatkov za strukturiranje dogodka	45
Slika 35: Preverjanje JSON v extractEventDetails() funkciji	46
Slika 36: Preverjanje datuma v extractEventDetails() funkciji	47
Slika 37: Klicanje nove funkcije v extractEventDetails() funkciji.....	47
Slika 38: Pridobivanje podatkov v getCityFromPlace() funkciji	48
Slika 39: Iskanje podatkov v getCityFromPlace() funkciji	48
Slika 40: Vstavljanje podatkov v getCityFromPlace() funkciji.....	49
Slika 41: Dodajanje slike v mapo Slike v processPosts() funkciji	49
Slika 42: UploadImageToServerComplete() funkcija	50
Slika 43: Dodajanje v bazo v ProcessPosts() funkciji	50
Slika 44: AddToDatabase() funkcija.....	51
Slika 45: ProcessPosts() konec	51
Slika 46: MarkEventAsProcessed() funkcija.....	51
Slika 47: Nastavitve API v add-event.php.....	52

Slika 48: Pripravljanje podatkov za uporabo v add-event.php.....	53
Slika 49: Preprečevanje podvojenih dogodkov v add-event.php	54
Slika 50: IsDuplicateEvent() funkcija	55
Slika 51: Dodajanje in ravnanje z dogodki v add-event.php.....	56
Slika 52: Dodajanje kategorij v add-event.php	57

KAZALO GRAFOV

Graf 1: Ustrezni dogodki	24
Graf 2: Cena.....	25
Graf 3: Čas identifikacije objave	26
Graf 4: Zanesljivost 1. testiranje.....	33
Graf 5: Natančnost 1. testiranje	34
Graf 6: Cena 1. testiranje	35

KAZALO TABEL

Tabela 1: Povprečna natančnost.....	24
Tabela 2: Cena	25
Tabela 3: Natančnost 1. testiranje	33
Tabela 4: Cena 1. testiranje.....	34

SLOVAR

- **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 inteligenca)
- **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

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 kokoši in jajca. 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 Microsoft Azure Vision 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.

2. HIPOTEZE

- Izdelan program lahko učinkoviteje prepozna dogodke iz družbenih omrežij kot ročni vnos uporabnikov.
- Sistem umetne inteligence bo dosegel vsaj 80 % natančnost pri prepoznavanju uličnega naslova, mesta ter datuma dogodkov.
- 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.¹

3.1.1. Obsotoječe rešitve tega problema

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.²

Ekskluzivne vsebine: Netflix in druge pretočne platforme so začele z lastnimi vsebinami, da bi pritegnile uporabnike.³

Ciljno usmerjen začetek: Velike platforme, kot sta Facebook in Instagram, so se na začetku osredotočile na specifične demografske skupine, preden so se razširile globalno.⁴

Prikazovanje umetno ustvarjenih objav: Reddit je svojo podatkovno bazo napolnil z umetno ustvarjenimi objavami, da je pritegnil uporabnike.⁵

Zaposlitev delavcev: To rešitev ponuja Amazon mechanical turk, ki je platforma za množično izvajanje majhnih nalog, kjer lahko posamezniki in podjetja oddajo naloge delavcem po vsem svetu.⁶

Zelo sem bil presenečen, da za ta problem še ne obstaja avtomatizirana rešitev z uporabo umetne inteligence. In sem videl priložnost, da lahko tu razvijem to kot neko novost na trgu.

¹ <https://www.sharetribe.com/marketplace-glossary/chicken-and-egg-problem/> (8.1.2025)

² <https://www.uber.com/us/en/drive/uber-pro/> (8.1.2025)

³ <https://press.farm/netflixs-original-content-strategy-and-leadership/> (8.1.2025)

⁴ <https://www.theguardian.com/technology/2007/jul/25/media.newmedia> (8.1.2025)

⁵ <https://www.vice.com/en/article/how-reddit-got-huge-tons-of-fake-accounts-2/> (8.1.2025)

⁶ <https://www.mturk.com/> (8.1.2025)

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 UI pomaga pri avtomatiziranem zbiranju podatkov

Eden ključnih načinov, kako UI lahko 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

1. Spletno strganje podatkov (Web Scraping)
 - UI pomaga pri analizi velikih količin podatkov, prepoznavanju vzorcev in iskanju najbolj primernih rešitev za določene izzive. To vključuje analizo podatkov iz različnih virov, kot so družbena omrežja, spletne strani in drugi spletni viri, za pridobivanje vpogledov in informacij, ki so ključne za poslovne odločitve.⁷
2. Obdelava besedila in naravno razumevanje jezika (NLP)
 - UI lahko analizira besedilo in prepozna ključne informacije, kot so datumi, kraji in opisi dogodkov.⁷
 - 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
 - Strojno učenje omogoča algoritmom prepoznavanje vzorcev v podatkih, kar je ključno za razločevanje med relevantnimi in nerelevantnimi informacijami. To zmanjšuje količino nepotrebnih podatkov in izboljšuje učinkovitost sistemov.⁸
 - Na primer, v finančnem sektorju se algoritmi strojnega učenja uporabljajo za prepoznavanje goljufivih transakcij z učenjem vzorcev in anomalij v podatkih.
4. Optično prepoznavanje znakov (OCR)

⁷ https://youth.europa.eu/get-involved/your-rights-and-inclusion/artificial-intelligence-what-you-should-better-know_sl (9.1.2025)

⁸ <https://www.joker.si/kaj-je-strojno-ucenje-in-zakaj-je-tako-pomembno/> (9.1.2025)

- Je tehnologija, ki omogoča prepoznavanje besedila v slikah, kot so skenirani dokumenti, fotografije ali plakati, in njegovo pretvorbo v strojno berljivo obliko.⁹
- 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.¹⁰
 - Sistem uporablja personalizacijo, ki omogoča, da so novice relevantne glede na interese uporabnika.¹¹
2. Eventbrite in Meetup
 - Ti dogodkovni portali uporabljajo AI priporočilne algoritme, ki analizirajo zanimanja uporabnikov in predlagajo dogodke.¹²
 - Meetup omogoča samodejno predlaganje dogodkov na podlagi analize vedenja uporabnikov in zgodovine udeležbe.¹³
3. UI za analizo spletnih objav v marketinških orodjih (npr. Sprinklr, Brandwatch)
 - Uporablja NLP za prepoznavanje trendov in pomembnih informacij v družbenih omrežjih.¹⁴
 - Ti sistemi so lahko osnova za avtomatizirano prepoznavanje dogodkov v spletnih objavah.¹⁴

⁹ <https://pressbooks.pub/umetnainteligenca/chapter/optical-character-recognition/> (9.1.2025)

¹⁰ <https://support.google.com/googlenews/answer/9010862?co=GENIE.Platform%3DAndroid&hl=en> (9.1.2025)

¹¹ <https://developers.google.com/search/docs/appearance/google-discover> (9.1.2025)

¹² <https://www.eventbrite.com/blog/ai-data-for-events/> (10.1.2025)

¹³ <https://arxiv.org/abs/2006.08893> (10.1.2025)

¹⁴ https://stratcomcoe.org/publications/download/Social-Media-Monitoring-Tools-DIGITAL-a23c5.pdf?utm_source=chatgpt.com (10.1.2025)

4. METODOLOGIJA 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

Programski jezik:

- **Node.js** je JavaScript okolje za izvajanje kode na strežniku, ki omogoča učinkovito obdelavo asinhronih operacij in delo z velikimi količinami podatkov.¹⁵



Slika 1: Node.js¹⁶

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 svoj algoritem za pridobivanje podatkov s spletne strani objave in jih strukturira v JSON format.¹⁷
- **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.

¹⁵ <https://nodejs.org/docs/latest/api/> (21.1.2025)

¹⁶ <https://www.curotec.com/wp-content/uploads/2023/09/curotec-nodejs.png> (21.1.2025)

¹⁷ <https://apify.com/> (21.1.2025)



Slika 2: Apify.com ¹⁸

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.¹⁹
- **Omogoča** avtomatsko prepoznati besedilo iz vizualnih virov brez potrebe po ročni analizi.
- **Pomaga** pri ekstrakciji besedila iz naslovnih slik scrapanih objav.
- **Se ne uporablja več kot del rešitve.** Nadomestil Microsoft Azure Vision.



Slika 3: Google Vision API ²⁰

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.²¹
- **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.
- **Se ne uporablja več kot del rešitve.** Nadomestil ChatGPT API s povezavo do spleta.

¹⁸<https://media2.dev.to/dynamic/image/width=1600,height=900,fit=cover,gravity=auto,format=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fuploads%2Farticles%2Fjs27asgdigl9m2ne74ke.png> (21.1.2025)

¹⁹<https://cloud.google.com/vision?hl=en> (21.1.2025)

²⁰https://miro.medium.com/v2/resize:fit:1400/1*tkAiRWvYmAi_RuRhRYgeiQ.jpeg (21.1.2025)

²¹<https://programmablesearchengine.google.com/about/> (21.1.2025)



Slika 4: Google PSE ²²

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.²³
- **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 ²⁴

4.1.6. OpenAI API za obdelavo besedila

- **OpenAI API** uporablja napredne modele umetne inteligence za analizo in obdelavo besedila.²⁵
- **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.

²²https://cdn.prod.website-files.com/5fc212183117036dc3c635d0/611f6155c187c5434d6880eb_Google%20Programmable%20Search%20Engine%20PSE%20CSE.png (21.1.2025)

²³<https://developers.google.com/maps/documentation/places/web-service/overview> (21.1.2025)

²⁴https://cdn.prod.website-files.com/62a9b4f36a23c435c9b9f3cc/634dcecb623e3c3538478dff_Google%20Places.png (21.1.2025)

²⁵<https://platform.openai.com/docs/overview> (21.1.2025)



Slika 6: OpenAI²⁶

4.1.7. PHP API in MySQL

- **PHP API in MySQL** predstavlja povezavo med spletno aplikacijo in podatkovno bazo MySQL. PHP skripte omogočajo komunikacijo z bazo podatkov prek API-ja (Application Programming Interface), s katerim lahko izvajamo operacije, kot so branje, vstavljanje, posodabljanje in brisanje podatkov.²⁷
- **Omogoča** povezavo s podatkovno bazo, upravljanje podatkov, avtentikacijo uporabnikov, obdelavo podatkov in služi kot vmesnik med podatkovno bazo in aplikacijo.
- **Pomaga** pri prejemanju strukturiranih dogodkov, njihovo preverjanje in zapis v bazo.

4.1.8. Microsoft Azure Vision

- **Microsoft Azure Computer Vision** je storitev za analizo slik z uporabo umetne inteligence.
- **Omogoča** prepoznavanje besedila (OCR), predmetov, obrazov, ozadij in scen na slikah.
- **Uporablja** se za samodejno analizo vizualnih vsebin brez ročnega dela.
- **Pomaga** pri ekstrakciji besedila iz slik scrapanih objav.



Slika 7: Microsoft Azure Vision²⁸

²⁶ https://miro.medium.com/v2/resize:fit:800/0*mE6ibxWepOrPQ2yT.png (21.1.2025)

²⁷ <https://www.zend.com/blog/building-php-api> (21.1.2025)

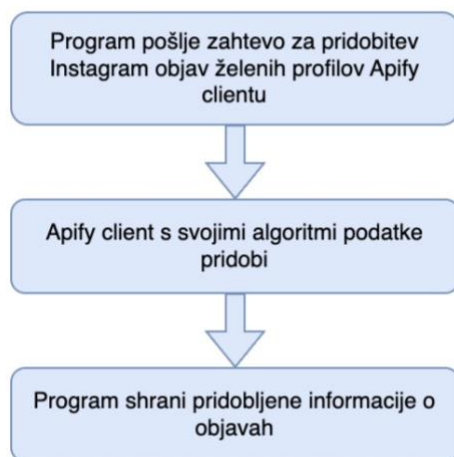
²⁸ <https://www.google.com/url?sa=i&url=https%3A%2F%2Flearn.microsoft.com%2Fen-us%2Ftraining%2Fpaths%2Fcreate-computer-vision-solutions-azure-ai%2F&psig=AOvVaw37c1ThNpSICDqimuhF80xG&ust=1744016565430000&source=images&cd=vfe&opi=89978449&ved=0CBQQjRxqFwoTCKDInZSGw4wDFQAAAAAdAAAAABAE> (6.4.2025)

5. RAZVOJ REŠITVE

V okviru raziskave sem razvil 2 programa. Namen prvega je pridobivanje objav iz Instagrama ter obdelavo in preverjanja podatkov. Namen drugega pa preverjanje baze za iste dogodke in dodajanje v bazo.

5.1. PRIDOBIVANJE OBJAV IZ INSTAGRAMA

V programu se definira, iz katerih profilov želimo pridobiti objave, koliko jih želimo in koliko vrnjenih rezultatov pričakujemo. Te podatke pošlje preko Apify client API-ja. Apify client podatke pridobi, jih obdela in obišče zahtevane spletne strani objav, s katerih s svojim algoritmom izpiše vse podatke. Ko zbere vse podatke zelenih objav, jih vrne v aplikacijo, kjer se shranijo za nadaljno uporabo.



Slika 8: Diagram poteka strganja podatkov

5.2. OBDELAVA IN PREVERJANJE PODATKOV

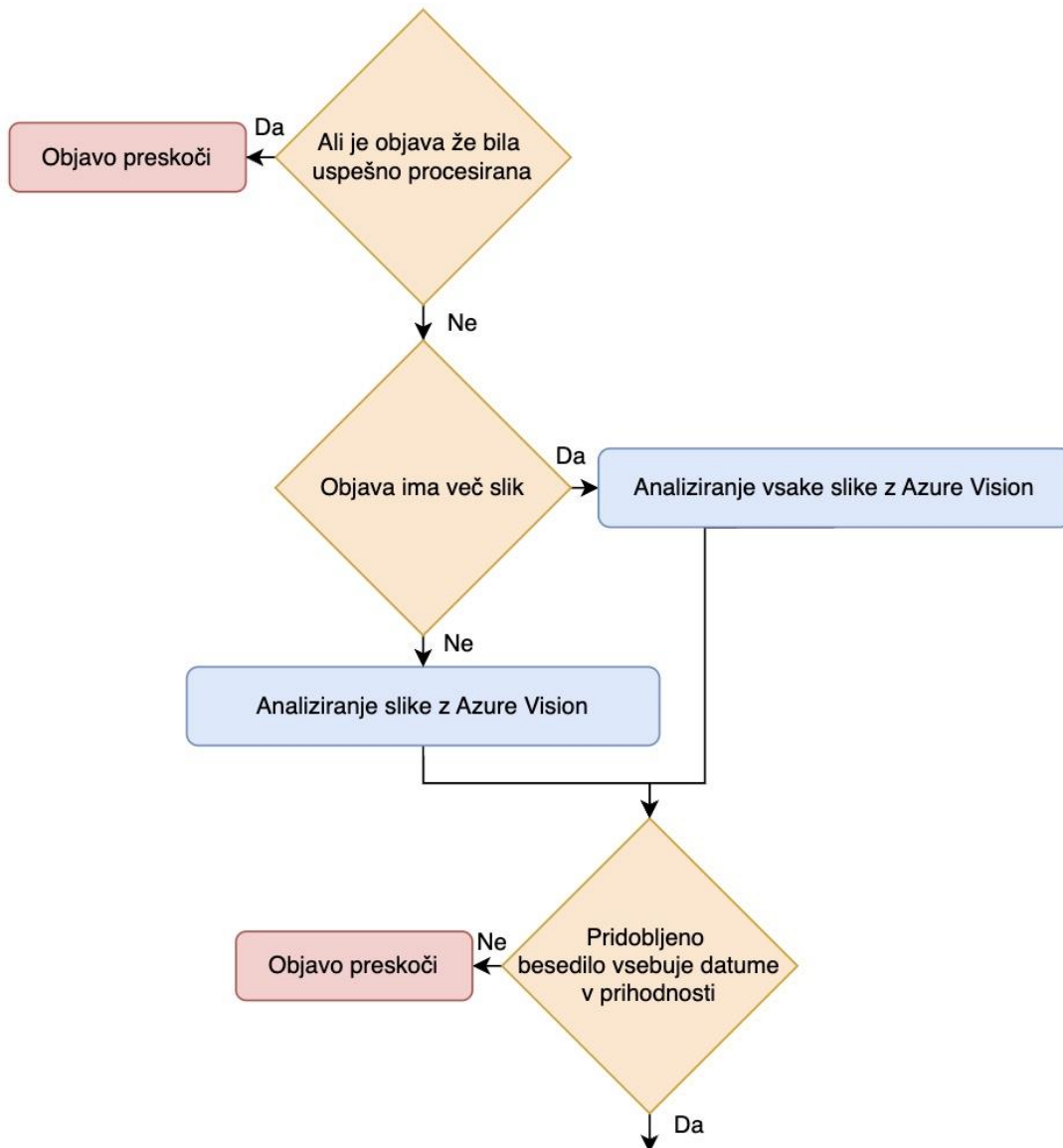
5.2.1. Pridobivanje besedila iz slike

V preteklosti sem za pridobivanje besedila uporabljal Google Vision API. Zamenjal ga je Microsoft Azure Vision, ki je pri procesiranju različnih pisav precej bolj uspešen.

V datoteki processedEvents.log so zabeležene povezave do že procesiranih objav. Ta se preveri in, če je dogodek že v datoteki objavo preskoči. Nato preveri, če je objava tipa Sidecar, kar pomeni, da je v objavi več slik. Vsako sliko pošlje na strežnik, kjer je dostopna na spletu, da lahko Azure Vision do nje dostopa. Če ni, naredi isto samo z naslovno sliko objave. Sliko analizira za besedilo, ki ga vrne v funkcijo in shrani.

Pridobljeno besedilo se pogleda za različne formate datumov v prihodnosti. V kolikor besedilo datumov v prihodnosti ne vsebuje, se objava preskoči.

Za namen optimizacije programa sem dodal datoteko noFutureEvents.log, kjer se shranijo slike, ki nimajo datuma v prihodnosti, da jih ob prihodnjem zagonu programa še enkrat ne procesira. S tem zmanjšam ceno zaznavanja besedila iz slike z OCR.



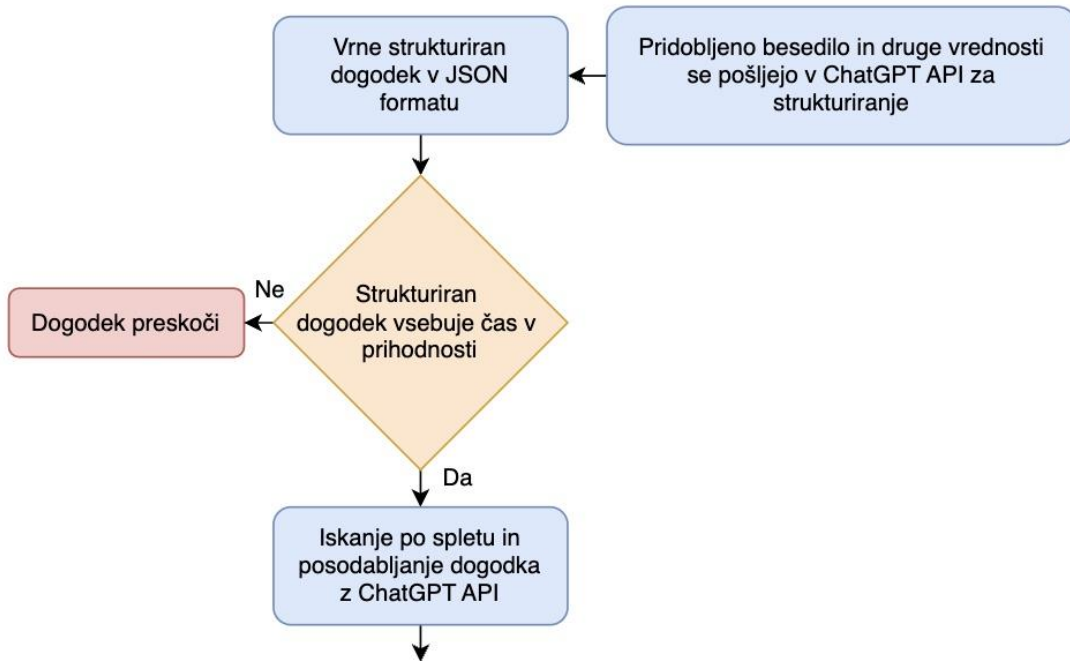
Slika 9: Diagram potek pridobivanja besedila iz slike

5.2.2. Strukturiranje dogodka v JSON format

Besedilo slike, opis objav (caption), skupno število različnih datumov iz slike, polno ime profila in lokacija vpisana na Instagramu se pošlje v ChatGPT API, ki strukturira dogodek

in ga vrne v JSON formatu. Začetek dogodka se nato ponovno preveri ali je v prihodnosti. To je nujno, saj se funkcija za preverjanje datumov v besedilu lahko zmoti. Če datum v prihodnosti ni prisoten, se dogodek preskoči. Če je dogodek še vedno aktualen se ponovno pošlje v ChatGPT API, ki dogodek išče po spletu in ga ustrezno posodobi.

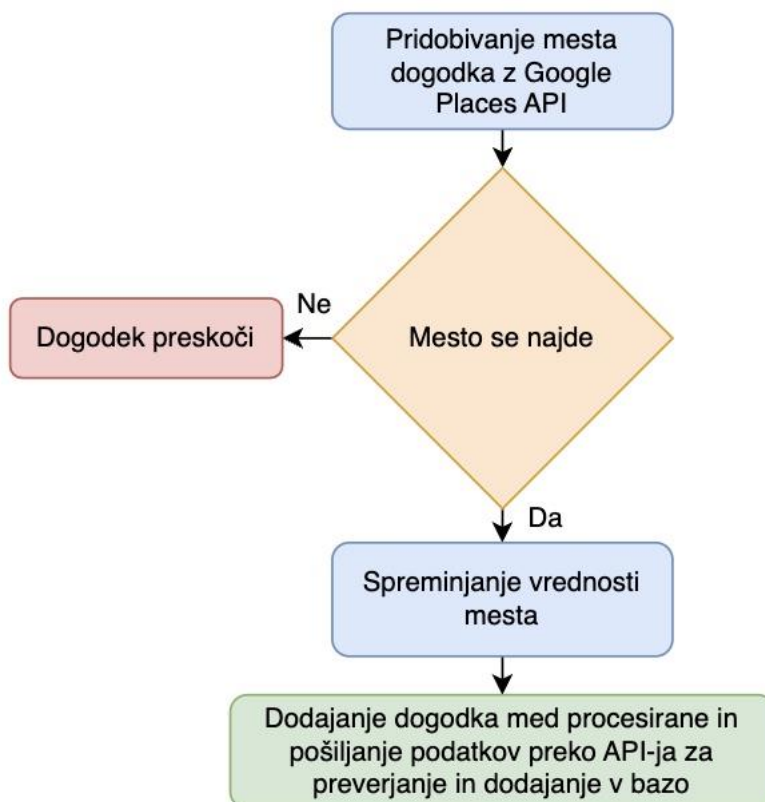
Ker v preteklosti ChatGPT API še ni imel dostopa do spleta sem za iskanje dogodka na spletu uporabljal Google PSE, ki je rezultate iskanja poslal v ChatGPT API.



Slika 10: Diagram poteka strukturiranja dogodka

5.2.3. Pridobivanje mesta dogodka in pošiljanje v bazo

Tukaj se ulični naslov dogodka pošlje v Google Places API, ki najde poln naslov in primerno mesto dogodka. Če mesto dogodka ni prazno se spremeni njegova vrednost v dogodku. Dogodek se nato doda med procesirane in pošlje na mojo bazo preko API-ja.

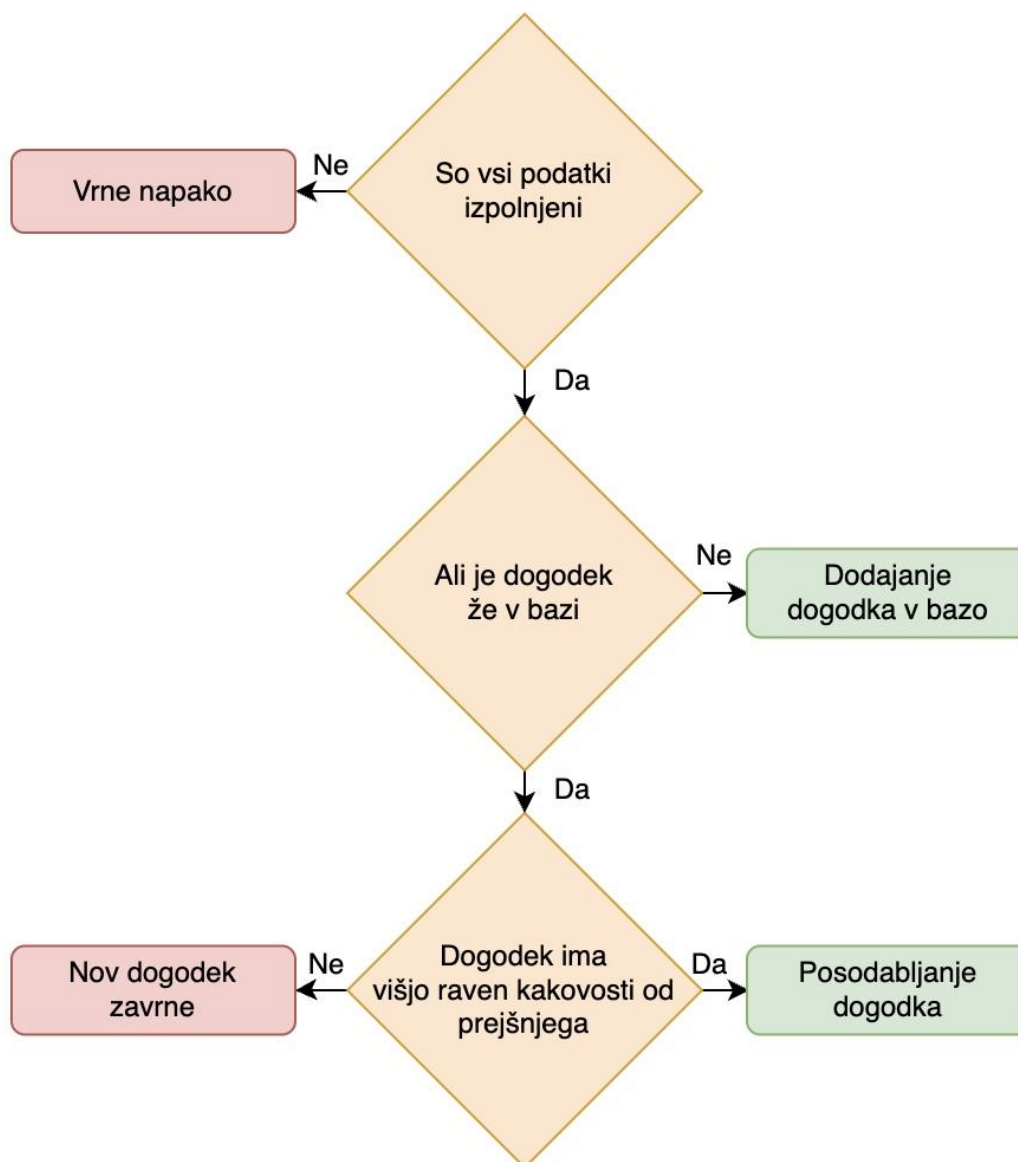


Slika 11: Diagram poteka pridobivanja mesta dogodka

5.2.4. Preverjanje in dodajanje v bazo

Podatki dogodka se prejmejo v API na mojem spletnem strežniku, kjer se preveri, če so izpolnjeni. V primeru da niso, vrne napako in dogodek se ne doda v bazo. Zatem pogleda, ali je dogodek z isto lokacijo, datumom in podobnimi izvajalci že v bazi. V kolikor ni, se doda v bazo. Če dogodek že obstaja se primerja njun tip (image, video, sidecar). Iz izkušenj sta za objavo najbolj primerna objave tipa slika in sidecar. Pogleda se tudi, če je vrednost more true, kar pomeni, da je v objavi našel več različnih dogodkov. Na koncu, če oba dogodka izpolnjujeta enaka merila, se pogleda kateri ima več izvajalcev. V primeru, da oba ponovno izpolnjujeta ista merila se doda dogodek z daljšim opisom. Na koncu se pogleda tudi, če je vrednost odpade true. Če je novega dogodka ne doda, kot tudi starega izbriše.

V prihodnosti bom vsak dogodek že v začetku točkovno kakovostno ocenil, kar bo omogočalo enostavnejše primerjanje med podvojenima dogodkoma.



Slika 12: Diagram poteka preverjanja in dodajanja v bazo

5.3. PRAKTIČNI PRIMER POTEKA OBDELAVE

5.3.1. Uvod

Za testiranje algoritma sem naredil tudi spletno stran www.ulicanori.si, ki sem jo uporabljal za testiranje in pridobivanje rezultatov. Potek kode je lažje razložiti na praktičnem primeru.

5.3.2. Objava na Instagramu

Primer objavljenega dogodka na Instagramu.



Slika 13: Objava Štuk kluba ²⁹

5.3.3. Besedilo iz slike in napis

Sliko pošlje v Microsoft Azure Vision in iz nje izpiše besedilo. Preveri se ali besedilo vsebuje prihodnje datume.

²⁹ <https://www.instagram.com/p/DIBUCB1o3yV/> (6.4.2025)

```
1 [2025-04-06T07:58:24.246Z] Posts text detected: ŠOUM LAMPIONČNI  
22. MAJ 2025  
MARIBOR. KAMPUS GOSPOSVETSKA FEHTARJI MI2 FARTY ANIMALS PIŽONI  
Se pa te veš!  
2 [2025-04-06T07:58:24.246Z] ✓ Extracted future dates:  
"2025-05-22T00:00:00+02:00"]2
```

Slika 14: Pridobljeno besedilo

5.3.4. Pošiljanje v ChatGPT za strukturiranje

ChatGPT API podatke obdela in pošlje dogodek strukturiran v JSON formatu.

```
1 [2025-04-06T07:58:27.157Z] 📄 Extracted Events: [  
2   {  
3     "title": "ŠOUM Lampiončki",  
4     "description":  
5     "Line-up je tu 🐣 Early bird karte lahko tut kupiš, datum pa tak že veš 🔥",  
6     "address": "Kampus Gosposvetska",  
7     "start": "2025-05-22T22:00:00",  
8     "end": "2025-05-23T02:00:00",  
9     "city": "Maribor",  
10    "categories": "Koncert",  
11    "full": "",  
12    "odpade": "false",  
13    "more": "false",  
14    "hashtags": "kupizaj,earlybird,sepateves,lampioncki2k25",  
15    "performers": "Fehhtarji, MI2, Farty Animals, Pižoni"  
16  }  
]
```

Slika 15: Strukturiran dogodek

5.3.5. Iskanje po spletu in posodabljanje dogodka

Dogodek se ponovno pošlje v ChatGPT API za iskanje novih informacij po spletu.
Dogodek posodobi.

```
1 [2025-04-06T07:58:30.596Z] ■ Fact-Checked & Optimized Event: {
2   "title": "ŠOUM Lampiončki 2025",
3   "address": "Kampus Gosposvetska, Maribor",
4   "description":
5     "ŠOUM Lampiončki 2025 prinašajo nepozabno glasbeno doživetje z nastopi pri
6     ljubljenih izvajalcev. Pridruži se nam na največjem študentskem dogodku v
7     Mariboru!"
8   ,
9   "start": "2025-05-22T22:00:00",
10  "end": "2025-05-23T02:00:00",
11  "city": "Maribor",
12  "categories": ["Koncert"],
13  "full": "",
14  "odpade": "false",
15  "more": "false",
16  "hashtags":
17    "#ŠOUM2025 #Lampiončki #MariborskiDogodek #FehhtarjiLive #MI2VŽivo #FartyAn
18    imalsKoncert #PižoniNaOdru #ŠtudentskiŽur #GlasbeniSpektakel #KoncertVMari
19    boru #NepozabenVečer #GlasbeniDogodek #ŠtudentskaZabava #Maribor2025 #Dogo
20    dekLeta #OutdoorKoncert #VečerZaMlajšeGeneracije #SlovenskiGlasbeniDogodek
21    #KampusGosposvetska #ŠtudentskiŽurMB"
22  ,
23  "performers": "Fehhtarji, MI2, Farty Animals, Pižoni"
24 }
```

Slika 16: Ponovno strukturiranje dogodka

5.3.6. Izpis končnega dogodka

Popolnoma strukturiranemu dogodku se preveri mesto in doda poln naslov, nato se preko API-ja pošlje na moj spletni strežnik za dodajanje.

```
1 [2025-04-06T07:58:31.099Z] ✓ Updated event city to: Maribor based on address search.
2 [2025-04-06T07:58:31.497Z] ✓ Uploaded to: /public_html/Slike/1743926299534.jpg
3 [2025-04-06T07:58:31.498Z] 📄 Extracted future event: {"title":"ŠOUM Lampiončki 2025","address":
4   "Kampus Gosposvetska, Maribor","description":
5     "ŠOUM Lampiončki 2025 prinašajo nepozabno glasbeno doživetje z nastopi priljubljenih izvajalcev. Pridruži
6     se nam na največjem študentskem dogodku v Mariboru!"
7   ,"start":"2025-05-22T22:00:00","end":"2025-05-23T02:00:00","city":"Maribor","categories":["Koncert"],
8   "full":"Gosposvetska Cesta 83, 2000 Maribor, Slovenia","odpade":"false","more":"false","hashtags":
9     "#ŠOUM2025 #Lampiončki #MariborskiDogodek #FehhtarjiLive #MI2VŽivo #FartyAnimalsKoncert #PižoniNaOdru #Štud
10    entskiŽur #GlasbeniSpektakel #KoncertVMariboru #NepozabenVečer #GlasbeniDogodek #ŠtudentskaZabava #Maribor
11    2025 #DogodekLeta #OutdoorKoncert #VečerZaMlajšeGeneracije #SlovenskiGlasbeniDogodek #KampusGosposvetska #
12    ŠtudentskiŽurMB"
13   ,"performers":"Fehhtarji, MI2, Farty Animals, Pižoni"
14 }
```

Slika 17: Izpis dogodka z datumom v prihodnosti

5.3.7. Končni izgled dogodka

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



LAMPIONČKI
22. MAJ 2025
Maribor, Šolski center Velenje

FEHTARJI
MI2
FARTY ANIMALS
PIŽONI

Se pa te veš!

0

ŠOUM Lampiončki 2025
Maribor
Koncert

Organizator:
Lampiončki

Opis:
ŠOUM Lampiončki 2025 prinašajo nepozabno glasbeno doživetje z nastopi priljubljenih izvajalcev. Pridruži se nam na največjem študentskem dogodku v Mariboru!

Datum:
22.05.2025 - 23.05.2025

Čas:
22:00 - 02:00

View larger map
Koroška
Univerza v Mariboru, Studentski...
Dijaški dom
Drava Maribor
Gosp...

Map data ©2025 Google Terms Report a map error

Slika 18: Dogodek na www.ulicanori.si

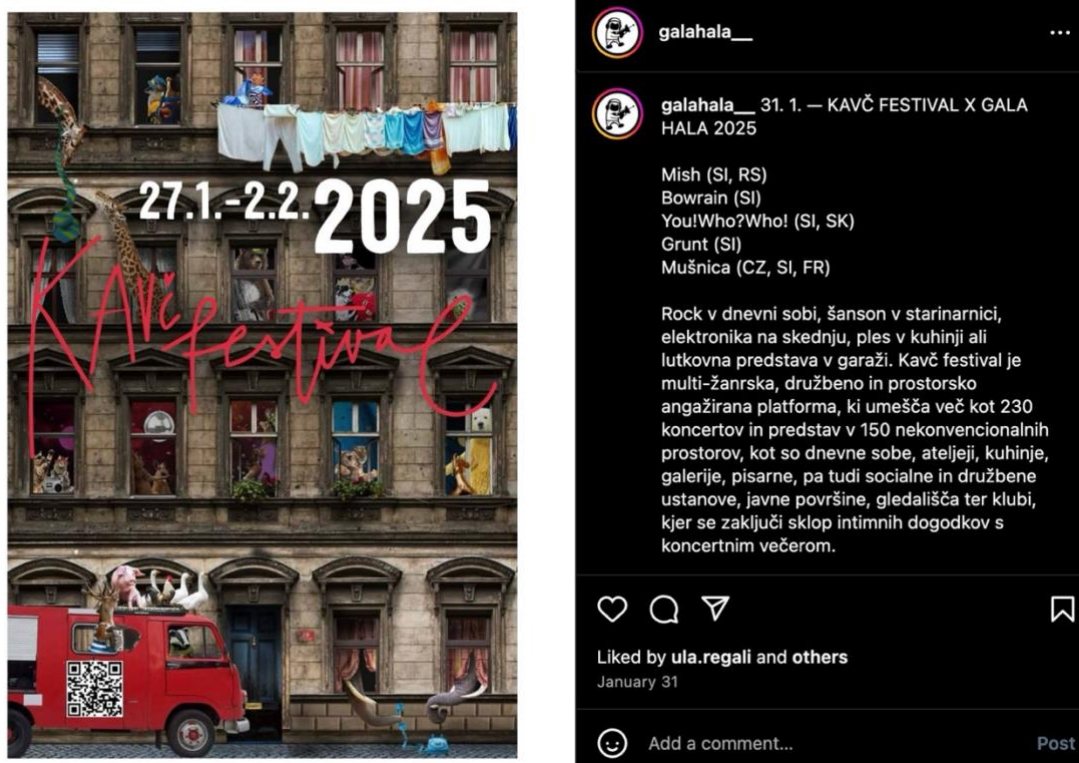
Ker mi besedilo za dogodek generira ChatGPT API je delo pri prilagajanju besedila zelo enostavno. V pozivu dodam zgolj stil v katerem želim, da je besedilo napisano.

Na podoben način se lahko prilagodi namen delovanja celotne aplikacije. Na primer to rešitev sem prilagodil za lokalno podjetje, ki vsak dan iz facebook objav pridobi malice različnih lokalov in jih objavi na njihovi spletni platformi. Lastniku sem s tem olajšal več kot 2 uri dela dnevno.

5.4. PREDNOSTI IN OMEJITVE SISTEMA

Sistem ima zaradi narave delovanja nekaj omejitev.

Prvi problem nastane pri zaznavanju besedila iz slike. Če je besedilo težko berljivo, ga Azure Vision ne more prebrati in dodati v strukturiranje dogodka. To sem rešil z dodatnim preverjanjem opisa Instagram objave (caption), ki mnogokrat vsebuje vse potrebno in se dogodek lahko uspešno doda. V primeru, da opisa ni, problem ostane, kar se pri testiranju sicer ni zgodilo, vendar to ne pomeni, da se v prihodnosti ne more.



Slika 19: Primer težko berljive slike dogodka ³⁰

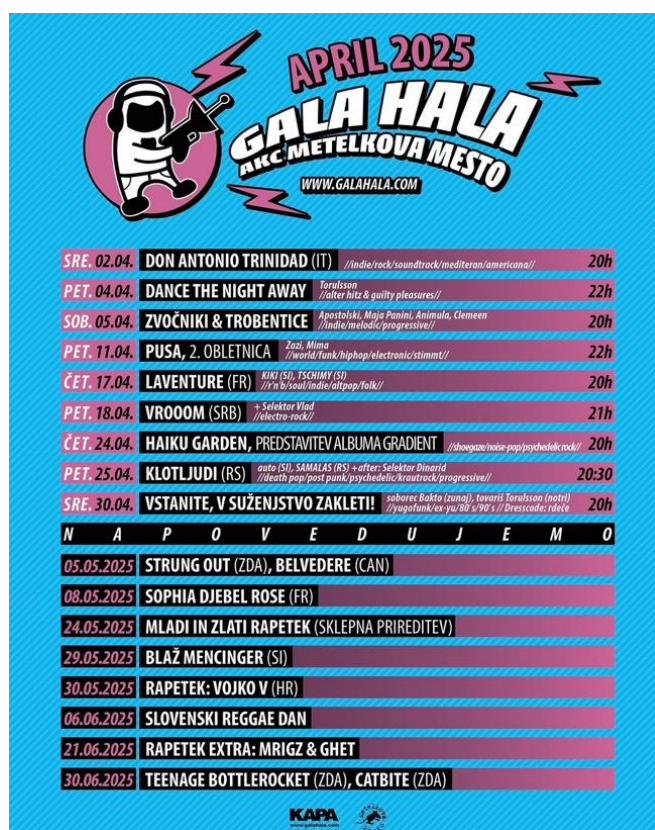
Naslednjo težavo lahko pripišemo človeškemu faktorju, ki je zelo relevanten pri reševanju problema kokoš in jajce skozi mojo metodo. Kot vsak človek ima tudi vsaka objava svoj stil, na katerega se težko prilagodimo. Nekatere imajo zelo podrobne opise, na katere se lahko zanašamo, spet druge imajo v opisu samo kakšen stavek ali pa nič. Zato je edina 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 za primer v njegov naslov vnesel ime kluba, drug je zelo dobro opisan zato uporabi ulični naslov kluba. Ta problem sem v preteklosti rešil z dodajanjem iskanja po

³⁰ https://www.instagram.com/galahala__ (18.2.2025)

spletu z Google PSE in ponovnim preverjanjem dogodka iz strani ChatGPT API-ja. Zdaj pa delo opravi ChatGPT API.

Človeškemu faktorju lahko pripišemo tudi oblikovanje napovednikov, ki so lahko precej zahtevni za OCR. Zato bom v prihodnosti namesto, da objavim vsak dogodek iz napovednika posebej naredil novo kategorijo samo za te. To bo precej zmanjšalo napake in problem poenostavilo. Naslednja možnost je tudi narediti OCR po meri, ki bo omogočal uspešnejši izpis besedila iz slabo oblikovanih napovednikov.

Primer odličnega napovednika:



Slika 20: Primer odličnega napovednika ³¹

³¹ <https://www.instagram.com/p/DHyC5gjoEyc/> (6.4.2025)

Primer zelo slabega napovednika:



Slika 21: Primer zelo slabega napovednika ³²

Naslednji izziv je nastal tudi pri preverjanju podvojenih dogodkov v bazi. Pojavilo se je vprašanje »Na katere podatke se lahko dovolj zanesem, da bodo dovolj standarizirani, da se na strani baze lahko preverjajo?«. Odločil sem se, da bom primerjal začetni datum dogodka brez časa, saj čas ni nujno, da je vedno točen, poln naslov dogodka, ki je standariziran iz strani Google Places API-ja in nastopajoče. Nastopajoče sem dodal, saj so me izkušnje naučile, da sta lahko 2 dogodka v enem dnevu na istem naslovu.

Naslednja ovira lahko nastane pri točnosti naslova in mesta, saj nekateri klubi oziroma lokali s podobnim imenom obstajajo v večih mestih. To napako sem rešil tako, da z scraper-jem pridobivam tudi parent data, kar pomeni podatke o samem profilu, kjer je zapisan točen naslov in kraj.

³² https://www.instagram.com/p/DIBm_ZLNp8Z/ (6.4.2025)

Po drugi strani sistem svoje napake skozi čas odpravi sam, v kolikor je dogodek ponovno objavljen z več informacijami. Prednost je tudi uporaba zunanjih API-jev in v kolikor razvijalci izboljšajo le-te, se izboljša tudi delovanje moje aplikacije. Takšen primer predstavlja ChatGPT API. Kjer se je ta med razvijanjem večkrat posodobil in razrešila problem.

6. ANALIZA REZULTATOV

V sklopu moje raziskovalne naloge sem izvedel dva testiranja. Pri prvem je sodelovalo 5 sošolcev, ki sem jih zadolžil, da iz 25 Instagram profilov izpišejo vse dogodke, ki jih vidijo na prvih 6-tih objavah. Delo so opravili samo 3-je, kar ni bilo dovolj za konkretne rezultate. Kljub obetavnim rezultatom (11.1) označujem to testiranje za pomanjkljivo zaradi premale velikosti vzorca. Pri tem je pripomoglo tudi preveliko število objav, ki so jih dijaki morali pregledati in pomanjkanje motivacije za dokončanje dela.

V nadaljnjem testiranju sem zagotovil manjše število profilov in dijakom ponudil nagrado za opravljeno delo. Za namen drugega testiranja sem izbral 10 najbolj aktivnih Instagram profilov, saj je fizično pregledovanje 25 profilov enostavno prezahtevno in časovno potratno za dijake.

V času testiranja 4. 4. 2024 je bilo na teh 10-tih profilih objavljenih 37 dogodkov, ki se bodo zgodili v prihodnosti (relevantni dogodki).

6.1. PRIMERJANJE FIZIČNEGA VNOSA Z VNOSOM PROGRAMA

26-tim dijakom 2. letnika programa tehnik računalništva sem dal nalogo, da iz vnaprej podanih 10 Instagram profilov izpišejo vse dogodke, ki jih vidijo v prvih 6-tih objavah. Točna navodila se nahajajo v prilogi 11.2.

Za primerjavo rezultatov sem preračunal p-vrednost (stopnja značilnosti), ki jo uporabljamo pri preverjanju hipotez. V veliki večini primerov pri kvantitativnih raziskavah ne moremo preveriti podatkov vseh enot populacije. Zaradi tega običajno iz populacije izberemo reprezentativni vzorec enot oziroma testirancev, na katerih opravimo raziskavo, kjer pridobimo podatke. Na podlagi testirancev iz vzorca želimo preverjati hipoteze na nivoju populacije.³³

Najpogosteje uporabljamo mejo stopnje značilnosti v višini 0,05. Če je p-vrednost manjša ali enaka 0,05, potem lahko z veliko gotovostjo (95 %) posplošimo rezultate iz vzorca na celotno populacijo. V primeru, da je vrednost večja od 0,05, pomeni, da določenega rezultata ne moremo posplošiti na nivo celotne populacije.³³

³³ <https://www.statistik.si/p-vrednost/> (19.2.2025)

6.1.1. Natančnost

V raziskavi sem primerjal natančnost prepoznavne in izpisa dogodkov na Instagramu med avtomatizirano aplikacijo in 26-timi dijaki. Ključni kazalniki vključujejo pokritost objav, število podvojenih dogodkov ter napake povezane z lokacijo, datumom in vsebino.

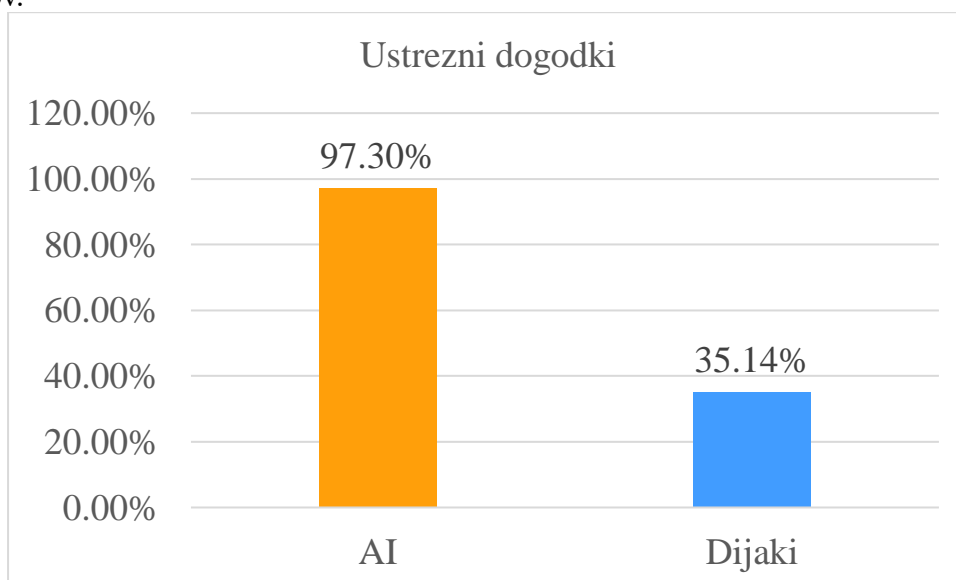
Aplikacija je pri eni objavi zaradi pomanjkanja informacij zapisala napačen naslov dogodka. Medtem, ko so dijaki v povprečju imeli veliko večje težave, ki so razvidne v tabeli.

	AI	Dijaki
Pokritost objav	37/37	22/37
Podvojenost dogodkov	0/37	2/37
Napačna/ni lokacija	0	4
Napačen/ni datum	0	2
Napačna vsebina	1	4
Skupaj OK	36/37	13/37

Tabela 1: Povprečna natančnost

Iz teh podatkov sem izračunal p-vrednosti pokritosti objav (0.0000000217459), podvojenosti dogodkov (0.002242507) in skupno primernost dogodkov (0.000000000000289591). S tem lahko potrdim, da je aplikacija znatno natančnejša, kot fizični vnos dijakov.

Graf (Graf 1) procentualno primerja število ustreznih dogodkov aplikacije in povprečje dijakov.



Graf 1: Ustrezni dogodki

6.1.2. Cena

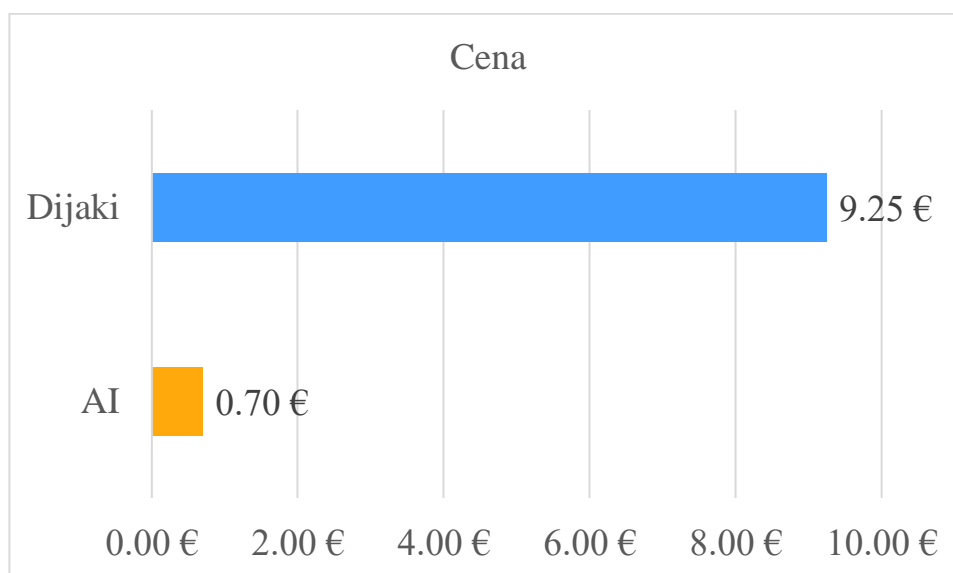
Za računanje cene, ki bi jo moral plačati dijakom za delo sem uporabil minimalno bruto urno postavko, ki trenutno znaša 7,34€³⁴. Postavko sem množil s porabljenim časom vsakega dijaka.

	AI	Dijaki
Čas dela (minute)	22	76
Cena	0,70 €	9,25 €

Tabela 2: Cena

Cena programa, opravljanja iste naloge, kot dijaki pa je znesla 0,70€, če ne upoštevamo brezplačne ravni storitev. Podrobnosti o izračunu stroškov najdemo v poglavju 6.3.

Iz teh podatkov, da je pregled in objava z aplikacijo znatno cenejša kot z zaposlovanjem študenta ali dijaka za delo lahko potrdimo ($p = 0.00000000000000000289796$).



Graf 2: Cena

6.2. OCENITEV KOREKCIJE NAPAK APLIKACIJE

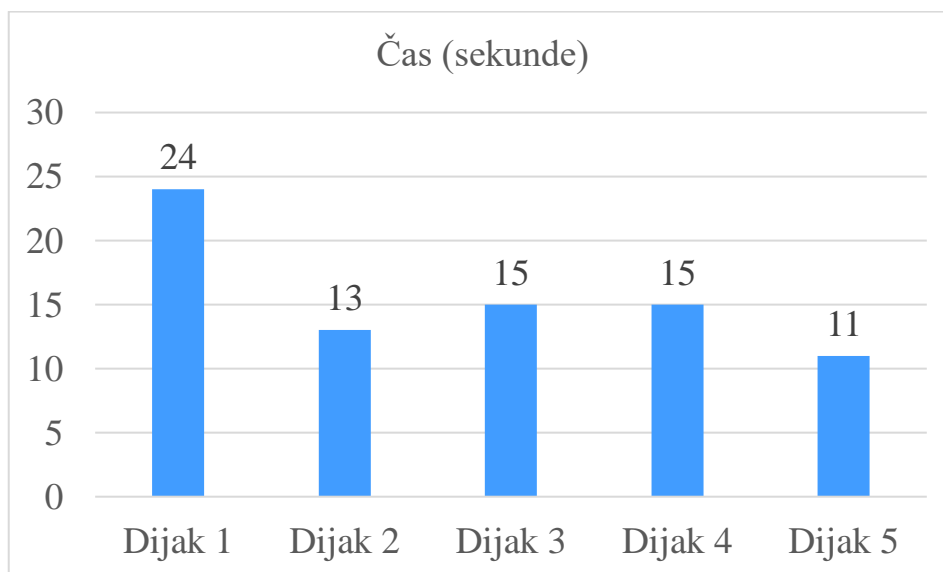
Tretji test je bil ustvarjen z namenom, da ocenimo stroške vzdrževanja aplikacije v primeru napačnih vnosov.

Potekal je tako, da sem v spletni aplikaciji naredil stran, ki prikaže dogodke, ki so bili objavljeni danes. Pridobil sem 5 dijakov 2. letnika programa tehnik računalništva. Od tega 3 fante in 2 dekleti. Dobili so sledeča navodila: »Zaposlen si v podjetju, kjer je tvoja

³⁴ <https://www.studentski-servis.com/studenti/novice/minimalna-urna-postavka/> (19.2.2025)

naloga identificirati napačne dogodke«. Preverjal sem njihovo hitrost pri identifikaciji napak pri vsebini.

Vsi dijaki so uspešno določili, katera objava vsebinsko ni primerna. Pri tem so povprečno porabili 15.6 sekund (Graf 3). S takšno hitrostjo bi za 1 polno uro dela potrebovali 231 delovnih dni oz. več kot 11 mesecev in pol. Glede na to, bi bil letni znesek takšnega vzdrževanja 7.63€, če delavec prejema minimalno bruto študentsko urno postavko glede na opravljeno uro dela.



Graf 3: Čas identifikacije objave

6.3. IZRAČUN STROŠKOV

Najdražje orodje, ki ga uporabljam je ChatGPT API, ki za popolno strukturiranje in preverjanje 37 dogodkov porabi žetone v vrednosti 0,56\$³⁵, kar je pretvorjeno 0,51€. Za strganje objav iz Instagrama odštejemo približno 0,15\$³⁶, kar je pretvorjeno 0,14€ za strganje 60 rezultatov. Če ne štejemo mesečne brezplačne ravni soritve, ki je 5\$. Storitve Microsoft Azure Vision so precej dostopne posameznikom, zato bi zanje za predelavo 60 objav moral plačati okoli 0,05\$, kar je pretvorjeno 0,046€, vendar so bile zame brezplačne, ker brezplačnih kreditov še nisem uporabil.³⁷ Storitve Google Places API je tudi precej dostopna, zato sem njej odštel le 0,0012\$, zato bom vrednost zanemarl.³⁸

³⁵ <https://platform.openai.com/docs/pricing> (19.2.2025)

³⁶ <https://console.apify.com/actors/shu8hvrXbJbY3Eb9W/input> (19.2.2025)

³⁷ <https://azure.microsoft.com/en-us/pricing/details/cognitive-services/computer-vision/> (6.4.2025)

³⁸ <https://developers.google.com/maps/billing-and-pricing/pricing> (6.4.2025)

7. RAZPRAVA

Hipoteza 1: Izdelan program lahko učinkoviteje prepozna dogodke iz družbenih omrežij kot ročni vnos uporabnikov.

Hipotezo lahko **potrdimo**.

Glede na rezultate pridobljene v poglavju 6.1.1 je razvidno, da je aplikacija dosegla bistveno boljše rezultate pri pokritosti dogodkov iz družbenih omrežij kot dijaki. Prav tako so rezultati pokazali, da se je aplikacija pri točnosti podatkov zmotila le pri enem dogodku, medtem ko so dijaki naredili več napak.

Hipoteza 2: Sistem umetne inteligence bo dosegel vsaj 80 % natančnost pri prepoznavanju uličnega naslova, mesta ter datuma dogodkov.

Hipotezo lahko **potrdimo**.

Glede na rezultate pridobljene v poglavju 6.1.1 je razvidno, da je aplikacija dosegla skupno natančnost 97.30 %, ki presega zadani cilj. V prihodnosti, bi zagotovo lahko teste izvajali skozi daljše časovno obdobje, saj se objave lahko precej razlikujejo.

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

Hipotezo lahko **potrdimo**.

Glede na rezultate pridobljene v poglavju 6.2 je razvidno, da je vzdrževanje in korekcija napak takšne aplikacije trivialna in ne predstavlja znatnega dodatnega dela. Stroški urne postavke za opravljeno delo so v primerjavi z ostalimi stroški obratovanja spletnih strani zanemarljivi, sploh če upoštevamo, da se takšno delo lahko preprosto opravi med drugimi vzdrževalnimi deli, ki so nujni za vsako spletno stran. Za bolj konkretne rezultate, bi lahko teste izvajali skozi daljše časovno obdobje, saj obstaja možnost, da bi pri avtomatskem vnosu prihajalo tudi do kompleksnejših vsebinskih napak. Trenutno je bila edina napaka napačno ime dogodka, kar so pri testiranju dijaki hitro in učinkovito rešili.

Seveda pa v tem primeru obstaja tudi možnost, da se algoritem prepoznavne dodela in te napake odpravi, kot je opisano v poglavju 5.4.

8. 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 kokoši in jajca. Iskal sem načine, kako so to rešila druga podjetja. Ugotovil sem, da če bom želel ta problem rešiti obstajata le dve možnosti. Vsakodnevno preverjanje novih dogodkov in ročni vnos teh ali aplikacija, ki dela to zame. Logična se mi je zdela samo ena in sicer strganje podatkov iz spleta in družbenih omrežij ter jih procesirati s programom. Začetna misel je bila »To bo enostavno, potrebujem samo program za pridobivanje informacij o dogodkih in drugo bo naredila umetna inteligenca«. Spoznal sem, da moramo umetni inteligenci postaviti precej točna navodila, če želimo doseči pričakovane rezultate.

Nadaljnje 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, kateri 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.

9. POVZETEK

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, Microsoft Azure Vision in Google Places API. V raziskovalni nalogi preverim, čas dela, pokritost objav, natančnost in ceno v primerjavi s 26 dijaki. V natančnost spadajo pravilen ulični naslov dogodka, pravilno mesto dogodka in pravilen datum začetka dogodka.

10. ZAHVALA

Zahvaljujem se mojemu mentorju Samu Železniku za začetne ideje, mnenja in vodstvu skozi projekt in dr. Nataši Meh Peer za lektoriranje naloge. Zahvaljujem se tudi 2. letniku, ki so mi pomagali pri primerjavi vnosa programa in fizičnemu vnosu ter preverjanju 60-tih objav. Najbolj se pa zahvaljujem svojemu dekletu, ki me je med raziskavo neizmerno podpirala in mi s poslušanjem pomagala rešiti velik del izzivov programa.

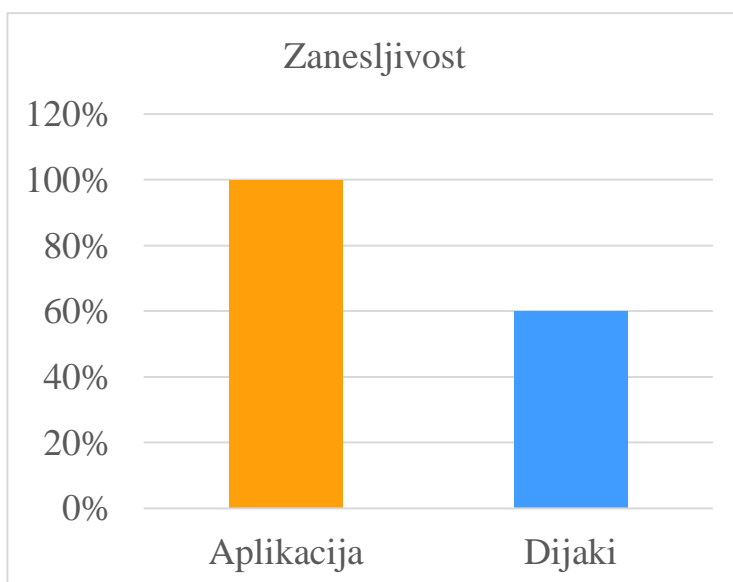
11. PRILOGE

Tukaj sem dodal rezultate prvega testiranja, točna navodila drugega testiranja in podrobneje razložil delovanje kode za namen replikacije mojih rezultatov. Zaradi same količine kode ni bilo najbolj primerno, da jo vključim v jedro.

11.1. REZULTATI PRVEGA TESTIRANJA

11.1.1. Zanesljivost

Podatke naj bi pridobil od 5 sošolcev. Na koncu dneva so le 3 izpeljali nalogo, medtem ko sta 2 nalogo opustila. Tako so dijaki dosegli 60% zanesljivosti, medtem ko je AI dosegel 100%. Nad temi rezultati nisem presenečen, saj je delo precej monotono, nezanimivo in precej časovno potratno.



Graf 4: Zanesljivost 1. testiranje

11.1.2. Natančnost

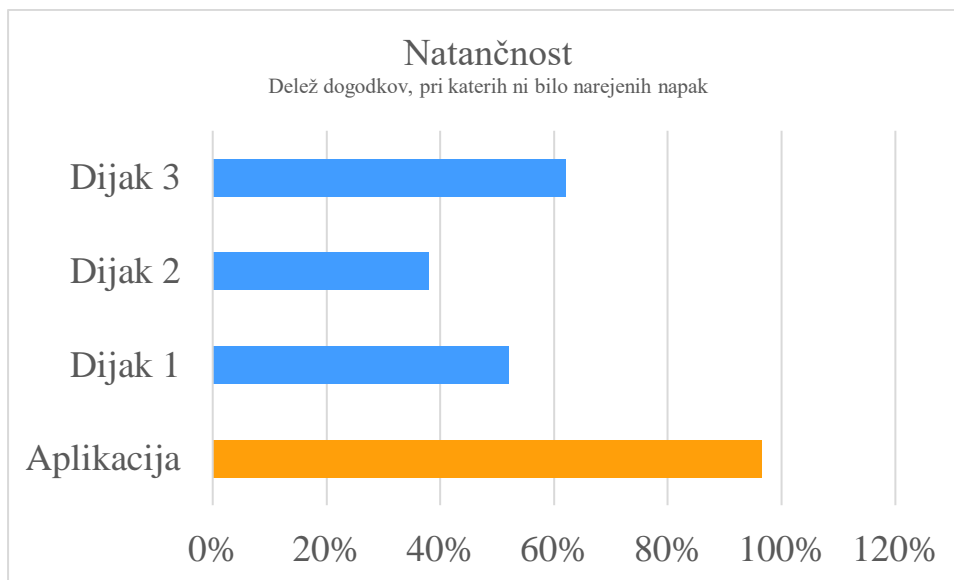
V raziskavi sem primerjal natančnost prepoznave in izpisa dogodkov na Instagramu med avtomatizirano aplikacijo in tremi dijaki. Ključni kazalniki vključujejo pokritost objav, število podvojenih dogodkov ter napake povezane z lokacijo, datumom in vsebino.

	Aplikacija	Dijak 1	Dijak 2	Dijak 3
Natančnost	56/58	30/58	23/58	36/58
	96.55%	51.72%	39.65%	62.00%

Tabela 3: Natančnost 1. testiranje

Iz teh rezultatov je razvidno, da je aplikacija znatno preseгла človeško prepoznavo dogodkov na Instagramu ($p = 0.009$).

Med dijaki so bile razlike v natančnosti velike ($\pm 22,35\%$), pri čemer je Dijak 3 dosegel najboljši rezultat (62 %), medtem ko je bil Dijak 2 najmanj natančen (39,65 %).



Graf 5: Natančnost 1. testiranje

11.1.3. Cena

Za računanje cene, ki bi jo moral plačati študentu za delo sem uporabil minimalno bruto urno postavko, ki trenutno znaša 7,34€³⁹. Postavko sem množil z porabljenim časom vsakega dijaka.

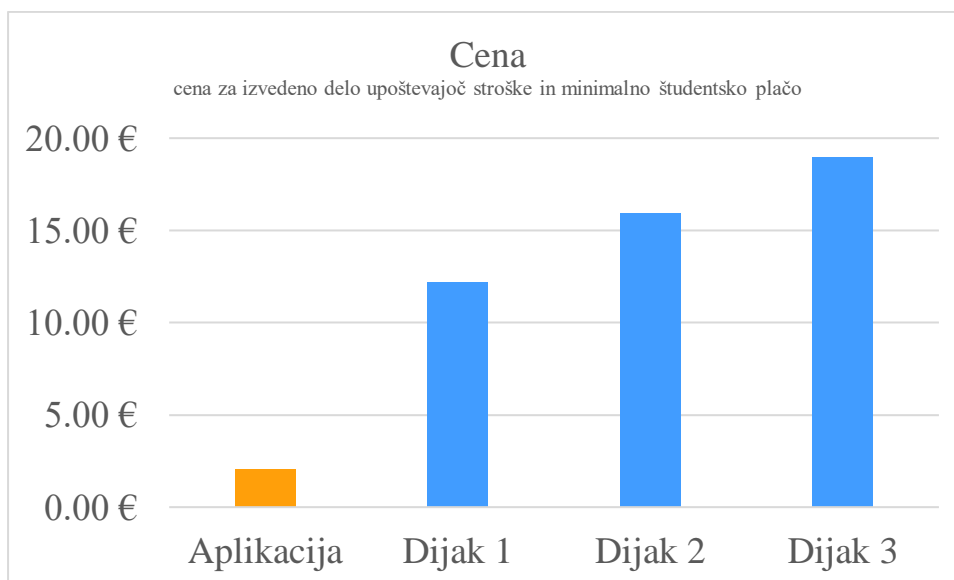
	Aplikacija	Dijak 1	Dijak 2	Dijak 3
Čas (minute)	29	100	130	155
Cena	2.05 €	12.18 €	15.93 €	18.94 €

Tabela 4: Cena 1. testiranje

Cena programa, opravljanja iste naloge, kot dijaki pa je znesla 2,05€, če ne upoštevamo brezplačne ravni storitev.

Iz teh podatkov, da je pregled in objava z aplikacijo znatno cenejša kot z zaposlovanje študenta ali dijaka za delo ($p = 0.009$).

³⁹ <https://www.studentski-servis.com/studenti/novice/minimalna-urna-postavka/>



Graf 6: Cena 1. testiranje

11.2. NAVODILA DIJAKOM

Tukaj so točna navodila, ki sem jih posredoval dijakom:

Zaposleni ste v podjetju. Vaša naloga je razbrati čim več dogodkov iz Instagram profilov. Najbolje bo, da profile obiščete na telefonu, saj se vam tako nebo potrebno prijavljati na Instagram preko šolskega računalnika.

Na vsakem profilu boste pogledali **prvih 6 objav** in preverili, če gre za dogodek, **ki bo v prihodnosti**. Preskočite dogodke, ki so že bili. Dogodki se **ne smejo ponavljati**.

Med delom se je potrebno **ŠTOPATI**.

Če vam zmanjka polj za vpis jih dodate sami. Zadevo si lahko po želji poenostavite.

Prvi 3-je dijaki, ki bodo našli največ dogodkov v najkrajšem možnem času dobijo nagrado.

Instagram profili:

<https://www.instagram.com/maestroevents.eu/>

https://www.instagram.com/trust_maribor/

<https://www.instagram.com/zmaucbar/>

https://www.instagram.com/mc_pekarna/

https://www.instagram.com/bar_dolcevita/

https://www.instagram.com/galahala_/

<https://www.instagram.com/pureljubljana/>

<https://www.instagram.com/studentstikampus/>

<https://www.instagram.com/euterpeevents/>

<https://www.instagram.com/bar.turistica.lounge/>

Vsak dogodek zapišite po sledeči strukturi:

1. Dogodek
Ime dogodka: Ulični naslov dogodka: Mesto: Začetek dogodka: Konec dogodka: Opis:

Primer za <https://www.instagram.com/p/DF5jWuTtL7x/> :

1. Dogodek
Ime dogodka: SEVERINA & IVAN ZAK Ulični naslov dogodka: Športna dvorana Leona Štuklja, Šegova ulica 112 Mesto: 8000, Novo Mesto Začetek dogodka: 12. april 2025 ob 20.00 Konec dogodka: 13. april 2025 ob 03.00 (Če ni naveden si izmislite primerno) Opis: Pridružite se nepozabnemu glasbenemu večeru v Športni dvorani Leona Štuklja v Novem mestu, kjer bosta 12. aprila 2025 ob 20.00 nastopila izjemna Severina in priljubljeni Ivan Zak. Dogodek obeta edinstveno vzdušje, polno energije, vrhunskih nastopov in priljubljenih hitov, ki bodo občinstvo navduševali vse do zgodnjih jutranjih ur. Koncert bo trajal do približno 03.00 zjutraj in je kot nalašč za vse ljubitelje pop in zabavne glasbe, ki si želijo večer preživeti ob odlični družbi in ritmih, ki bodo še dolgo odzvanjali v spominu.

11.3. SCRAPER.JS

Program pridobi podatke o objavah jih strukturira in preveri.

11.3.1. 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. 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';
import dotenv from 'dotenv';
import axios from 'axios';
import fs from 'fs-extra';
import path from 'path';
import ftp from 'basic-ftp';
import cron from 'node-cron';
import { isFuture, parseISO, parse, format, isValid, addDays } from "date-fns";
import { sl } from 'date-fns/locale';
import util from 'util';

import pkg from '@azure-rest/ai-vision-image-analysis';
const { default: createClient, ImageAnalysisClient } = pkg;
import { AzureKeyCredential } from '@azure/core-auth';

dotenv.config();

const scraperToken = process.env.SCRAPER_TOKEN;
const openAiApiKey = process.env.CHATGPT_API_KEY;
const azureEndpoint = process.env.AZURE_ENDPOINT;
const azureApiKey = process.env.AZURE_API_KEY;
const endpoint = process.env.EVENT_API_ENDPOINT;

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,
    remotePathComplete: process.env.FTP_REMOTE_PATH_COMPLETE,
};

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

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

if (!fs.existsSync(CACHE_DIR)) {
    fs.mkdirSync(CACHE_DIR, { recursive: true });
}

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

Slika 22: Uvoz knjižnic in pridobivanje okoljskih spremenljivk

11.3.2.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 s pridobljenimi podatki (uporabnike, objave, hashtag-e ali kraje), koliko rezultatov naj bi bilo vrnjenih (10 profilov * 6 objav na profil = 60 objav) ter nazadnje ali naj doda nadrejene 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 directUrls = JSON.parse(process.env.DIRECT_URLS);

    const input = {
        directUrls: directUrls,
        resultsType: "posts",
        resultsLimit: 5,
        searchType: "user",
        searchLimit: 250,
        addParentData: true
    };

    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 23: RunScraper() funkcija

11.3.3.processPosts() preverjanje ustreznosti objave

Funkcija processPosts() pridobi niz podatkov vseh pridobljenih objavah. V for zanki nato iz vsake objave posebej shrani ključne podatke, ki se bodo uporabljali pri procesu v spremenljivke (city_name, street_name, displayUrl, caption, url, ownerFullName, type, images, locationName, likesCount in hashtags). Program prvo uporabi funkcijo sanitizeOwnerFullName, zato da iz nje izbriše kakršnekoli neznane znake, kot so emotikoni. Nato pogleda, če je objava s tem URL-jem že bila procesirana. S funkcijo isEventProcessed().

```

async function processPosts(posts) {
    for (const post of posts) {
        let { businessAddress: { city_name, street_address } = {}, displayUrl, caption, url, ownerFullName, type, images, locationName, likesCount, hashtags } = post;

        logMessage(`⚠️ Hashtags: ${hashtags}`);
        logMessage(`⚠️ Business: ${city_name}, ${street_address}`);

        ownerFullName = sanitizeOwnerFullName(ownerFullName);
        logMessage(`💡 Processing post: ${url}`);

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

Slika 24: ProcessPosts() preverjanje ustreznosti

11.3.3.1. SanitizeOwnerFullName()

Funkcija iz imen odstrani emotikone ali kakšne druge nenavadne simbole, saj se bo to ime uporabilo za prikaz na spletni strani.

```
function sanitizeOwnerFullName(name) {  
    return name.replace(/([\u2700-\u27BF]|[\uE000-\uF8FF]|[\uD83C-\uDBFF\uDC00-\uDFFF])/g, '');  
}
```

Slika 25: SanitizeOwnerFullName()

11.3.3.2. 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 26: IsEventProcessed funkcija

11.3.4.ProcessPosts() shranjevanje slike

Nato preveri, če je objava tipa sidecar, kar pomeni, da vsebuje več slik. Vsaka slika se pogleda, če je že bila procesirana v preteklosti, če je nadaljuje na drugo sliko. Generira se ime za sliko. Pot do slike in generirano ime se pošlje v funkcijo downloadImage(), ki vrne naslov slike na mojem strežniku. Če naslova ni se slika preskoči.

```
if (type == "Sidecar"){
  if (Array.isArray(images)) {
    for (const imageUrl of images) {

      if (checkNoFutureEvent(imageUrl)) {
        logMessage(`⚠ Event with URL ${imageUrl} does not have future time. Skipping.`);
        continue;
      }

      logMessage(`💡 Processing image: ${imageUrl}`);
      const fileName = `${Date.now()}.jpg`;
      const uploadedUrl = await downloadImage(imageUrl, fileName);

      if (!uploadedUrl) {
        logMessage(`⚠ Failed to download image: ${imageUrl}`);
        continue;
      }
    }
  }
}
```

Slika 27: ProcessPosts() shranjevanje slike

11.3.4.1. DownloadImage()

Funkcija pridobi vrednosti in uporabi knjižnjico axios, ki pošlje GET zahtevek za nalaganje slike. responseType 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',
      timeout: 15000,
    });

    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 28: DownloadImage() funkcija

11.3.4.2. 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({})` vzpostavi povezavo s 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 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, retries = 3, delay = 5000) {
    const client = new ftp.Client();
    client.ftp.verbose = false;
    client.ftp.timeout = 10000;

    for (let attempt = 1; attempt <= retries; attempt++) {
        try {
            logMessage(`🔄 Attempt ${attempt}/${retries} to upload ${remoteFilename}`);

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

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

            client.close();
            return `${process.env.IMAGE_CACHE_URL}${remoteFilename}`;
        } catch (error) {
            logMessage(`⚠️ FTP Upload Attempt ${attempt} failed: ${error.message}`);

            if (attempt < retries) {
                logMessage(`🔄 Retrying in ${delay / 1000} seconds...`);
                await new Promise(res => setTimeout(res, delay));
            } else {
                logMessage(`❌ All ${retries} attempts failed. Skipping upload.`);
                return null;
            }
        } finally {
            client.close();
        }
    }
}
    
```

Slika 29: UploadImageToServerCache() funkcija

11.3.5.ProcessPosts() izpis besedila iz slike

Slika je bila uspešno dodana na spletni strežnik, kjer je dostopna na spletu. Pridobljena povezava se nato posreduje v funkcijo analizeImageWithAzure(), ki vrne besedilo ter prihodnje datume. Če besedilo ne vsebuje datumov v prihodnosti se slika preskoči.

```

const { text, futureDates, eventsNo } = await analyzeImageWithAzure(uploadedUrl);

if (!eventsNo || eventsNo == 0) {
    logMessage("No events found in this image!");
    logNoFutureEvent(imageUrl);
    continue;
}

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

if (!futureDates || futureDates.length === 0) {
    logMessage("No future events found in this image!");
    logNoFutureEvent(imageUrl);
    continue;
}
    
```

Slika 30: Sklic funkcije analyzeImageWithAzure()

11.3.5.1. AnalyzeImageWithAzure()

Funkcija pošlje povezavo do slike za analizo v Microsoft Azure Vision. Besedilo se nato preveri za datume v prihodnosti s funkcijo extractFutureDates().

```

async function analyzeImageWithAzure(imageUrl) {
    try {
        const result = await azureClient.path('/imageanalysis:analyze').post({
            body: { url: imageUrl },
            queryParams: { features: features },
            contentType: 'application/json'
        });

        const iaResult = result.body;
        let detectedText = "";

        if (iaResult.readResult && iaResult.readResult.blocks) {
            iaResult.readResult.blocks.forEach(block => {
                if (block.text) {
                    detectedText += block.text + " ";
                }
                else if (block.lines) {
                    block.lines.forEach(line => {
                        if (line.text) {
                            detectedText += line.text + " ";
                        }
                    });
                }
            });
        }

        const { foundFutureDates, foundDates } = extractFutureDates(detectedText);

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

        return { text: detectedText, futureDates: foundFutureDates, eventsNo: foundDates };
    } catch (error) {
        logMessage(`❌ Azure Vision API Error: ${error.message}`);
        return { text: "", futureDates: [] };
    }
}
    
```

Slika 31: AnalyzeImageWithAzure() funkcija

11.3.5.2. ExtractFutureDates()

Ta funkcija je precej obširna zaradi velike količine različnih zapisov datumov, zato jo bom razdelil na več delov. Prva slika prikazuje spremenljivke, ki shranjujejo 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 = [
    /(\d{1,2})\s+(\d{1,2})?(?:\s+(\d{4}))?/g,
    /(\d{1,2})\.\s*(\d{1,2})\.\s*(\d{4})?/g,
    /(\d{1,2})\.\s*(\d{1,2})?(?:\.\s*(\d{4}))?\.?/g,
    /(\d{1,2})\.\s*(januar|februar|marec|april|maj|junij|julij|avgust|september|oktober|november|december|jan|feb|mar|apr|maj|jun|jul|avg|sep|okt|nov|dec|januarja|februarja|marca|aprila|maja|junija|julija|avgusta|septembra|oktobra|novembra|decembra)\s*(\d{4})?/gi,
    /(\d{1,2})\s+(januar|februar|marec|april|maj|junij|julij|avgust|september|oktober|november|december|jan|feb|mar|apr|maj|jun|jul|avg|sep|okt|nov|dec|januarja|februarja|marca|aprila|maja|junija|julija|avgusta|septembra|oktobra|novembra|decembra)\s*(\d{4})?/gi,
    /(\d{1,2})?(?:st|nd|rd|th)?\s+(january|february|march|april|may|june|july|august|september|october|november|december|jan|feb|mar|apr|may|jun|jul|aug|sep|oct|nov|dec)\s*(\d{4})?/gi,
    /(\d{1,2})?(?:st|nd|rd|th)?\s+(january|february|march|april|may|june|july|august|september|october|november|december|jan|feb|mar|apr|may|jun|jul|aug|sep|oct|nov|dec)\s+(\d{1,2})?(?:st|nd|rd|th)?\s*(\d{4})?/gi
  ];
}
```

Slika 32: Variacije datumov v `extractFutureDates()` funkciji

Naslednji del funkcije vsebuje logiko. Najprej se določi trenutno leto ter ustvari prazen seznam `foundFutureDates` in spremenljivko `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 s številko ali z 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. Uporabi se `while` zanka za beleženje števila zaznanih datumov. S 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 `foundFutureDates`. Za beleženje števila vseh datumov se uporabi `while` zanka. Na koncu izpiše še koliko je unikatnih datumov na sliki. Funkcija nato vrne seznam vseh prihodnjih datumov in število unikatnih.

```

const currentYear = new Date().getFullYear();
const foundFutureDates =
const uniqueDates = new Set();

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 (!isValid(parsedDate)) {
            continue;
        }
        const formattedDate = format(parsedDate, "yyyy-MM-dd'T'HH:mm:ssXXX");

        if (!uniqueDates.has(formattedDate)) {
            uniqueDates.add(formattedDate);
            if (isFuture(addDays(parsedDate, +1))) {
                foundFutureDates.push(formattedDate);
            }
        }
    }
});
const foundDates = uniqueDates.size;

return { foundFutureDates, foundDates };
}
    
```

Slika 33: Preostanek extractFutureDates() funkcije

11.3.6.ProcessPosts() pošiljanje podatkov za strukturiranje dogodka

Podatki se nato pošljejo za strukturiranje v ChatGPT API.

```

const eventDetails = await extractEventDetails(text, caption, eventsNo, ownerFullName, locationName,
hashtags, uploadedUrl, city_name, street_address);
    
```

Slika 34: ProcessPosts() pošiljanje podatkov za strukturiranje dogodka

11.3.6.1. ExtractEventDetails()

Funkcija uporablja ChatGPT API (GPT-4o) za izveček 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 in iz nabora izbere 1 kategorijo, ki se mu zdi najbolj primerna 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). Doda tudi vrednosti, kot so odpade, če je kje v besedilu to omenjeno vrednost spremeni na true, more, če je v eni sliki več dogodkov in formatira hashtag. Nažalost je ta del kode vsebinsko prevelik, da bi ga dodal v word dokument.

V tem delu kode prejme besedilo iz API-ja in ga očisti (trim()). Nato preveri, ali se podatki pravilno 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 35: Preverjanje JSON v extractEventDetails() funkciji

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.

```

    try {
      const parsedDate = parseISO(event.start);

      if (isFuture(parsedDate)) {
        return true;
      } else {
        logMessage(`✖ Event '${event.title || "Unnamed"}' is not in the future.`);
        return false;
      }
    } catch (error) {
      logMessage(`⚠ Error parsing date for event '${event.title || "Unnamed"}': ${error.message}`);
      return false;
    }
  });

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

```

Slika 36: Preverjanje datuma v `extractEventDetails()` funkciji

V kolikor dogodek še vedno vsebuje datum v prihodnosti, se kliče nova funkcija `refineEventWithChatGPT()`. Vanjo se pošlje vsak dogodek posebej. Te funkcije v priložo ne bom vključil, saj je precej podobna tej. Ključno je samo to, da je v tisti funkciji navodilo ChatGPT-ja, da išče dogodek na spletu in ga posodobi.

```

const verifiedEvents = [];

for (const futureEvent of futureEvents) {
  const refinedEvent = await refineEventWithChatGPT(futureEvent, location, ownerFullName);
  if (refinedEvent) {
    const withCity = await getCityFromPlace(refinedEvent);
    verifiedEvents.push(withCity);
  } else {
    logMessage("No output!!");
  }
}

return verifiedEvents;

```

Slika 37: Klicanje nove funkcije v `extractEventDetails()` funkciji

11.3.6.2. `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 ne bo 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=${address}&inputtype=textquery&fields=place_id&key=${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;
        const fullAddress = detailsResponse.data.result.formatted_address;
    }
}
    
```

Slika 38: Pridobivanje podatkov v `getCityFromPlace()` funkciji

11.3.6.3. `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 s `'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 39: Iskanje podatkov v `getCityFromPlace()` funkciji

11.3.6.4. GetCityFromPlace() vstavljanje podatkov

Če je spremenljivka foundCity definirana, se njena vrednost dodeli lastnosti city objekta event in se doda poln naslov v objekt full, 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;
  event.full = fullAddress;
  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 40: Vstavljanje podatkov v getCityFromPlace() funkciji

11.3.7.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 po končani operaciji spraznila. Najprej 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)) {
  const localFilePath = path.join(CACHE_DIR, fileName);

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

Slika 41: Dodajanje slike v mapo Slike v processPosts() funkciji

11.3.7.1. UploadImageToServerComplete()

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

```

async function uploadImageToServerComplete(localFilePath, remoteFilename) {
    const maxAttempts = 3;
    let attempt = 0;
    let result = null;

    while (attempt < maxAttempts) {
        attempt++;
        const client = new ftp.Client();
        client.ftp.verbose = false;
        try {
            await client.access({
                host: ftpConfig.host,
                user: ftpConfig.user,
                password: ftpConfig.password,
                port: ftpConfig.port,
                secure: false,
            });

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

            result = `${process.env.IMAGE_COMPLETE_URL}${remoteFilename}`;
            return result;
        } catch (error) {
            logMessage(`❌ FTP Upload Error on attempt ${attempt}: ${error.message}`);
            if (attempt >= maxAttempts) {
                return null;
            }
            // wait for 5 seconds before retrying
            await new Promise(res => setTimeout(res, 5000));
        } finally {
            client.close();
        }
    }
    return result;
}
    
```

Slika 42: UploadImageToServerComplete() funkcija

11.3.8.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 objave.

```

for (const event of eventDetails) {
    logMessage(`📄 Extracted future event: ${JSON.stringify(event)}`);
    addToDatabase({ completeUploadUrl, url, ownerFullName, type, likesCount, ...event });
}
    
```

Slika 43: Dodajanje v bazo v ProcessPosts() funkciji

11.3.8.1. 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 = process.env.EVENT_API_ENDPOINT;
    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 44: AddToDatabase() funkcija

11.3.8.2. ProcessPosts() drugi tipi

Pri drugih tipih objav, kot sta image in video je postopek identičen. Spremeni se zgolj kateri podatek se uporabi kot sliko za procesiranje.

11.3.8.3. ProcessPosts() konec

Ko je dogodek v popolnosti sprocesiran se doda v datoteko processedEvents.log z funkcijo markEventAsProcessed(). Po končanem procesiranju vseh pridobljenih objav se izbriše tudi mapa cache tako na lokalni napravi kot tudi na strežniku.

```
markEventAsProcessed(url);

}
await clearCache();
logMessage(`✅ Cache cleared successfully.`);
await deleteAllImagesFromServer();
}
```

Slika 45: ProcessPosts() konec

11.3.8.4. MarkEventAsProcessed()

Po obdelavi in dodajanju 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 46: MarkEventAsProcessed() funkcija

11.4. 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.

11.4.1. 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
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 47: Nastavitve API v add-event.php

11.4.2. Pripravljanje podatkov za uporabo

Ta koda najprej preveri, ali vhodni podatki vsebujejo vsa zahtevana polja. 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.

```
$requiredFields = ["title", "description", "address", "start", "end", "city", "categories", "full", "odpade", "more"];
foreach ($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']);
$cityQuery = "SELECT id FROM kraji WHERE ime = '$city' LIMIT 1";
$cityResult = $link->query($cityQuery);
if ($cityResult && $cityResult->num_rows > 0) {
    $rowCity = $cityResult->fetch_assoc();
    $kraj_id = $rowCity['id'];
} else {
    echo json_encode(["error" => "City not found"]);
    exit;
}

$full = $link->real_escape_string($data['full']);
$odpade = $link->real_escape_string($data['odpade']);
$more = $link->real_escape_string($data['more']);
$hashtags = $link->real_escape_string($data['hashtags']);
$performers = $link->real_escape_string($data['performers']);
$datum_objave = date('Y-m-d H:i:s');

$image = isset($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']) : "";

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

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

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

Slika 48: Pripravljanje podatkov za uporabo v `add-event.php`

11.4.3. Preprečevanje podvojenih dogodkov in zagotavljanje kakovosti

Ta koda preverja, ali je nov dogodek podvojen z obstoječim v bazi. Najprej se inicializira spremenljivka `$duplicateEventId` na `null`. Nato se izvede SQL poizvedba, ki pridobi vse že vpisane dogodke. Če poizvedba vrne rezultate, se ustvari asociativno polje `$newEvent`, ki vsebuje podatke o novem dogodku. 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, tip objave, nato vrednost `more`, če sta pa obe vrednosti enaki se izbere tistega, ki ima več izvajalcev, če je pa še to enako pa tistega z daljšim opisom. Pogleda tudi, če je vrednost `odpade` `true`, kar pomeni, da se mora dogodek odstraniti iz baze ne glede na vse.

ime, če imata dogodka skupnega vsaj enega izvajalca. Če je to vse res potem gre za podvojen dogodek.

```
function isDuplicateEvent($existingEvent, $newEvent) {
    $fullAddress1 = $existingEvent['full'];
    $fullAddress2 = $newEvent['full'];

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

    $sameAddress = ($fullAddress1 === $fullAddress2);
    $sameDateTime = ($existingDateTime === $newDateTime);

    $existingPerformers = array_filter(explode(" ", strtolower($existingEvent['performers'])));
    $newPerformers = array_filter(explode(" ", strtolower($newEvent['performers'])));
    $commonPerformers = array_intersect($existingPerformers, $newPerformers);
    $performersSimilarity = (count($commonPerformers) >= 1);

    return $sameAddress && $sameDateTime && $performersSimilarity;
}
```

Slika 50: *IsDuplicateEvent()* funkcija

11.4.4. Dodajanje dogodka in ravnanje s ponavljajočimi se dogodki

Če se podvojen dogodek najde in je novi dogodek bolj kakovosten od prejšnjega se vse vrednosti posodobijo.

```
if ($duplicateEventId) {
    $updateDuplicateQuery = "UPDATE dogodki SET
        ime = '$title',
        naslov = '$address',
        opis = '$description',
        slika = '$image',
        zacetek_dog = '$startFormatted',
        konec_dog = '$endFormatted',
        datum_objave = '$datum_objave',
        kraj_id = '$kraj_id',
        organizacija = '$organization',
        url = '$eventUrl',
        full = '$full',
        type = '$type',
        likes = '$likes',
        more = '$more',
        hashtags = '$hashtags',
        performers = '$performers'
    WHERE id = '$duplicateEventId'";
    if (!$link->query($updateDuplicateQuery)) {
        echo json_encode(["error" => "Database error updating duplicate event: " . $link->error]);
        exit;
    }
    $eventIdForMapping = $duplicateEventId;
} else {
    $insertEventQuery = "INSERT INTO
    dogodki (ime, naslov, opis, slika, zacetek_dog, konec_dog, datum_objave, kraj_id, organizacija, url, full, type
    , likes, more, hashtags, performers)
    VALUES ('$title', '$address', '$description', '$image', '$startFormatted', '$endFormatted', '$datum_objave', '$kraj_id
    ', '$organization', '$eventUrl', '$full', '$type', '$likes', '$more', '$hashtags', '$performers')";
    if ($link->query($insertEventQuery)) {
        $eventIdForMapping = $link->insert_id;
    } else {
        echo json_encode(["error" => "Database error: " . $link->error]);
        exit;
    }
}

if ($duplicateEventId) {
    $deleteMappingQuery = "DELETE FROM dogodek_kategorija WHERE dogodek_id = '$eventIdForMapping'";
    $link->query($deleteMappingQuery);
}
```

Slika 51: Dodajanje in ravnanje z dogodki v *add-event.php*

11.4.5. 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'];
        $insertMappingQuery = "INSERT INTO dogodek_kategorija (dogodek_id, kategorija_id) VALUES ('$eventIdForMapping', '
$skategorija_id')";
        $link->query($insertMappingQuery);
    }
}

echo json_encode(["success" => "Event added successfully"]);

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

Slika 52: Dodajanje kategorij v *add-event.php*