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

MLADI RAZISKOVALCI ZA RAZVOJ SAŠA REGIJE

RAZISKOVALNA NALOGA

AI PODCAST GENERATOR IZ ZAPISKOV

Tematsko področje: RAČUNALNIŠTVO

Avtorja:
Maj Kovač
Alen Strgar

Mentor:
Islam Mušić, prof.

Velenje, 2026

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

Mentor: Islam Mušić, prof.

Datum predstavitve: marec 2026

KLJUČNA DOKUMENTACIJSKA INFORMACIJA

ŠD ŠCV Elektro in računalniška šola, šolsko leto 2025/2026

KG podcast / umetna inteligenca / generator / programiranje

AV Maj Kovač / Alen Strgar

SA Islam Mušić

KZ 3320 Velenje, SLO, Trg mladosti 3

ZA ŠCV Elektro in Računalniška šola

LI 2026

IN AI PODCAST GENERATOR IZ ZAPISKOV

TD Raziskovalna naloga

OP

IJ sl

JI sl/en

AI Raziskovalna naloga obravnava razvoj sistema, ki avtomatizira pretvorbo slikovnih zapiskov v izobraževalni video z govorom, vžganimi podnapisi in interaktivnimi elementi. Sistem izvede OCR (optical character recognition), omogoči uporabniško verifikacijo besedila, generira intervjujski scenarij, ustvari govor s TTS (text-to-speech), iz zvočnega posnetka pridobi časovne žige za podnapise ter video sestavi z orodjem FFmpeg. Raziskava primerja čas izdelave učnega gradiva pri avtomatiziranem in ročnem postopku ter oceni vpliv verifikacije OCR na kakovost izpisa. Avtomatiziran postopek doseže povprečni čas izdelave 10 minut, ročni pa 92 minut, kar predstavlja 82 minut oziroma približno 89 % skrajšanje. Verifikacija OCR statistično značilno zniža CER (Character Error Rate) in WER (Word Error Rate) ter poveča subjektivno razumljivost na Likertovi lestvici. Rezultati potrjujejo, da sistem bistveno pohitri pripravo učnih videov, hkrati pa kontrolni pregled OCR opazno izboljša kakovost končne vsebine.

KEY WORDS DOCUMENTATION

ND ŠCV Electro and Computer School, academic year 2025/2026

CX podcast / artificial intelligence / generator / programming

AU Maj Kovač / Alen Strgar

AA Islam Mušić

PP 3320 Velenje, SLO, Trg mladosti 3

PB SCV Electro and Computer School

PY 2026

TI AI PODCAST GENERATOR FROM NOTES

DT Research project

LA sl

AL sl/en

AB This research project presents a system that automates converting photo notes into an educational video with narration, burned-in subtitles, and interactive elements. The pipeline performs OCR (optical character recognition), supports user verification of recognized text, generates an interview-style script, produces narration via TTS (text-to-speech), derives subtitle timecodes from the generated audio, and renders the final video using FFmpeg. The study compares production time between the automated and manual workflow and evaluates how OCR verification affects output quality. The automated workflow achieves an average production time of 10 min, while manual creation takes 92 min, yielding an 82 min or about 89% reduction. Adding OCR verification significantly reduces CER (Character Error Rate) and WER (Word Error Rate) and increases perceived comprehensibility on a Likert scale. The results confirm that the system substantially speeds up learning-video production, while OCR verification improves the quality of the final output.

KAZALO

KAZALO SLIK

1. Uvod

V zadnjih letih se je način ustvarjanja in shranjevanja učnih gradiv bistveno spremenil. Dijaki in študenti vse pogosteje namesto prepisovanja v digitalne zapiske uporabljajo telefone za fotografiranje table, strani zvezka ali natisnjenih gradiv. Takšne slike so sicer hitro ustvarjene, vendar pogosto ostanejo nestrukturirane v galeriji, kar otežuje njihovo učinkovito uporabo pri ponavljanju. Pretvorba slikovnih zapiskov v uporabno učno vsebino, kot so prepisi, povzetki, razlage ali preverjanje znanja, poteka večinoma ročno in je zato časovno zahtevna.

Ta raziskovalna naloga obravnava problem avtomatizacije procesa pretvorbe slikovnih zapiskov v multimedijsko učno gradivo. Cilj je razviti sistem, ki omogoča nalaganje slik zapiskov, izvedbo optičnega prepoznavanja besedila (OCR) ter generiranje razlage v obliki intervjujskega scenarija, govora (TTS), videa s podnapisi in kviz vprašanj iz prepoznanega in po potrebi popravljenega besedila.

Povezava: https://www.zapiskigpt.top/.

Prijavite se z emailom: testuporabnik@zapiski.gpt in geslom: Uporabnik123

1.1 Hipoteze

1. Avtomatizirana pretvorba slikovnih zapiskov v multimedijsko vsebino (video + podnapisi) zmanjša čas priprave učnega gradiva v primerjavi z ročnim postopkom.
2. Vključitev koraka verifikacije OCR (popravek prepoznanega besedila) opazno izboljša razumljivost končnega videa in kakovost podnapisov.
3. Interaktivni kviz med predvajanjem videa poveča aktivno sodelovanje uporabnika ter izboljša preverjanje razumevanja vsebine.
4. Sistem točk in odklepanja vsebin poveča motivacijo uporabnika za ponavljajočo uporabo aplikacije.
5. Zvoku dodati v ozadje tudi video pomaga učencu, da lahko fokus obdrži dlje časa.

2. Pregled objav

2.1 Optično prepoznavanje besedila (OCR)

Optično prepoznavanje besedila (OCR, *Optical Character Recognition*) je postopek, pri katerem iz slike ali skena dokumenta računalniški sistem prepozna znake (črke, številke, simbole) in jih pretvori v strojno berljivo besedilo. OCR se uporablja pri digitalizaciji dokumentov, iskanju po skenih, ekstrakciji podatkov iz računov, obrazcev ali fotografij zapiskov ter pri vseh primerih, ko je besedilo prisotno kot del slike.

2.1.1 Osnovni potek OCR

V splošnem OCR sistemi izvajajo več zaporednih faz. Tipična logika je naslednja:

1) Obdelava slike
Cilj je izboljšati kakovost vhodne slike in zmanjšati šum, ki povzroča napačno prepoznavo. Pogoste tehnike vključujejo poravnavo, pretvorbo slike v črno-belo in odstranjevanje šuma.

2) Zaznavanje besedila in postavitve
Sistem ugotovi, *kje* na sliki je besedilo. Pri dokumentih je to pogosto analiza po ravneh: stran > odstavek > vrstica > beseda > znak. V oblačnih OCR rešitvah (npr. Vision API) je posebej poudarjeno “dokumentno” zaznavanje gostega besedila, kjer rezultat pogosto vsebuje informacije o strukturi (odstavki, besede, prelomi).

3) Prepoznavanje znakov/besed
V tej fazi sistem pretvori izrezane dele besedila v znake. Starejši pristopi so temeljili na primerjanju vzorcev, sodobni pa uporabljajo nevronske modele (npr. LSTM, nevronska mreža z dolgim kratkoročnim spominom) za prepoznavo vrstic in zaporedij znakov. Primer odprtokodnega pogona je Tesseract, ki od različice 4 naprej uporablja LSTM model za prepoznavanje.

4) Poprocesiranje
Končni izpis se izboljša z jezikovnimi pravili ali slovarji (npr. popravljanje očitnih napak, upoštevanje pogostih besed, prelomov vrstic). Ta korak je lahko koristen, vendar ima omejitve: pri strokovnih izrazih, imenih ali formulah lahko slovar povzroči napačne popravke.

2.1.2 OCR pri tiskanem besedilu in rokopisu

Pri uporabi OCR je velika razlika med tiskanim tekstom in rokopisom. Prepoznavanje rokopisa je praviloma težje zaradi različnih osebnih slogov pisave, različnih razmikov med vrsticami in besedami in razmazov. Zaradi teh faktorjev je pri skeniranju zapiskov realno pričakovati več napak kot pri tiskanih dokumentih.

2.1.3 Merjenje uspešnosti OCR

Za ocenjevanje OCR rezultatov se v praksi pogosto uporabljata:

- CER (Character Error Rate) – delež napak na ravni znakov,
- WER (Word Error Rate) – delež napak na ravni besed.

Ideja je: OCR izpis primerjaš z "referenčnim" (ročno pravilnim) besedilom in izračunaš, koliko zamenjav/izpustov/vstavitev se je zgodilo. CER je uporaben, ko so pomembne že majhne napake (npr. strokovni izrazi), WER pa je bolj za splošno razumljivost.

2.2 Generiranje besedila

Generiranje besedila iz učnih gradiv pomeni, da iz daljšega ali neurejenega vira (npr. OCR izpis iz zapiskov) ustvarimo strukturirano, razumljivo in ciljno usmerjeno besedilo, ki je bolj primerno za nadaljnjo uporabo (učenje, govor, kviz).

2.2.1 Povzetek proti scenariju

Iz vhodnega OCR generiranega teksta je potrebno ustvariti besedilo, ki je napisano za govor (scenarij). Potrebno ga je organizirati kot dialog med dvema osebama. Pomembno je, da pri tem koraku ohranimo ključne informacije in ustvarimo točno strukturo, iz katere je razvidno, kdo kaj reče.

2.2.2 Abstraktno generiranje

V sodobnih sistemih se v veliki meri uporablja abstraktno generiranje. To pomeni, da model ne le kopira dele izvornega besedila, ampak ga preoblikuje in napiše na novo. To je uporabno, ker lahko preuredi vrstni red informacij, po potrebi poenostavi jezik in odstrani ponavljanja.

2.3 Pretvorba besedila v govor

Tehnologija pretvorbe besedila v govor (TTS) je namenjena temu, da pisanemu besedilo, da glas. Za vhoden argument dobi besedilo, ki je lahko kakršne koli dolžine in ga pretvori v govorjen jezik. V zadnjih letih je TTS tehnologija zelo napredovala, kar pomeni, da je glas v generiranih besedilih v veliki meri zelo podoben človeškemu.

2.3.1 Tipične komponente TTS sistema

V mnogih neural TTS pristopih se konceptualno še vedno omenjajo tri ravni:

- analiza teksta: normalizacija besedila (številke, okrajšave), razbitje na stavke, izgovorjava;
- akustični model: iz besedila predvidi npr. mel-spektrogram,
- vocoder: iz akustične predstavitve ustvari zvočni signal.

2.4 Podnapisi in časovna poravnava govora

Podnapisi imajo v učnih vsebinah dvojno vlogo:

- dostopnost in razumljivost (hrupno okolje, slabše razumevanje govora),
- podpora učenju (vidiš in slišiš isto informacijo, lažje slediš pojmom).

Ključni problem podnapisov pri avtomatizaciji je časovna poravnava: podnapisi se morajo prikazati ob pravem času in biti berljivi.

2.4.1 Segmenti in besedni časovni žigi

Za podnapise so uporabni:

- časovni žigi na ravni segmenta (npr. stavek ali odsek),
- v nekaterih primerih tudi časovni žigi na ravni besede (bolj natančno, a zahtevnejše).

2.4.2 Pravila berljivosti

Dobri podnapisi niso samo pravilno časovno umeščeni, ampak so tudi berljivi:

- največ 2 vrstici,
- vrstico se lomi smiselno (po ločilih, pred vezniki, ne med členom in samostalnikom itd.),
- podnapisi ne smejo biti prekratki (da jih človek sploh ujame) in ne predolgi (da ne zaostajajo).

2.5 Generiranje videa

Pri generiranju videa gre za združitev vseh elementov (ozadje, zvok in podnapisi) v mp4 video. Za avtomatizacijo je najpogostejše orodje FFmpeg, ki je standard za procesiranje avdio/video vsebin in ponuja filtre za prekrivanje, risanje besedila, dodajanje podnapisov.

2.5.1 Vdelava podnapisov

Pri vdelavi podnapisov obstajata dve glavni možnosti:

- podnapisi se “zapečejo” v sliko videa. Prednost: deluje povsod, slabost: ne da se izklopiti.
- Podnapisi so ločeni v posebni datoteki (npr. MP4/MKV). Prednost: uporabnik jih lahko vklopi/izklopi, slabost: ni vedno enotne podpore na vseh platformah.

3. Pregled uporabljenih tehnologij

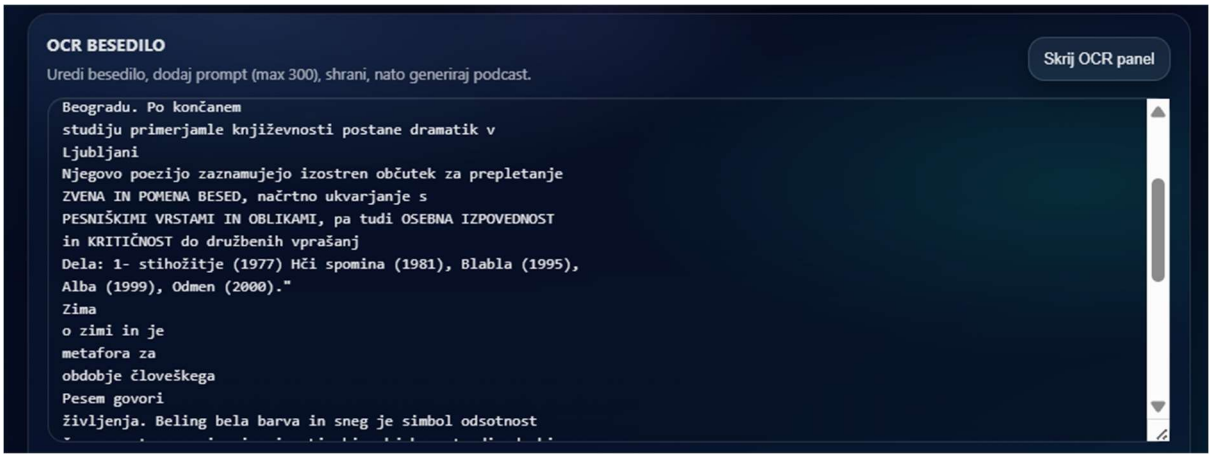
Sistem sva zasnovala tako, da kombinira oblačne storitve (OCR, generiranje besedila, TTS in jezikovni pregled podnapisov) ter lokalno obdelavo (transkripcija, sestava podnapisov in render videa). V nadaljevanju so tehnologije razdeljene po sklopih.

3.1 Oblačne storitve (Google)

3.1.1 Google Cloud Vision API (OCR)

Za optično prepoznavanje besedila iz naloženih slik uporabljava Google Cloud Vision OCR. OCR (Optical Character Recognition) je postopek računalniškega zaznavanja in ekstrakcije znakov iz slike ter njihove pretvorbe v strojno berljivo besedilo. V praksi to pomeni, da sistem iz vhodnih slik (npr. skenov, fotografij ali posnetkov zaslona) izvede detekcijo besedilnih regij, segmentacijo znakov oziroma besed ter prepoznavo vsebine z uporabo modelov za strojno učenje.

OCR predstavlja prvo fazo obdelovalne verige, zato je izhodni tekstovni rezultat osnova za vse nadaljnje korake (generiranje scenarija, TTS, generiranje podnapisov, generiranje kviza). Prepoznano besedilo se shrani in je uporabniku na voljo za pregled ter morebitne popravke. Sistem v nadaljnjih fazah uporablja potrjeno oziroma verificirano različico besedila.



Slika 1: Primer OCR odseka Vir: lasten

3.1.2 Gemini API (generiranje scenarija, naslova, kviza)

Za generiranje vsebine uporabljava Gemini API. Sistem po zaključeni OCR fazi vzame OCR besedilo oziroma uporabniško verificirano besedilo in ga posreduje modelu prek klica na Gemini API (prompt + vhodno besedilo). Model na podlagi posredovanega konteksta izvede generiranje izhoda, ki ga vrne kot strukturiran odgovor. Ta odgovor nato razdeliva v posamezne podatkovne dele (naslov, scenarij, kviz) in shraniva v posamezne elemente, ki jih aplikacija

uporablja v nadaljnjih korakih: naslov (če je potreben), scenarij v intervju formatu ter nabor kviz vprašanj, vezanih na vsebino.

3.1.3 Gemini API (jezikovni pregled podnapisov)

Gemini uporabiva tudi v fazi podnapisov. Ko podnapise lokalno časovno poravnava in razbijeva v posamezne ceuje, besedilo cuejev posredujeva modelu prek Gemini API, kjer model izvede jezikovno korekturo slovenskega besedila.

Pri tem ohraniva obstoječo strukturo in omejitve:

- časovnih žigov ne spreminjava,
- parafraziranje, dodajanje nove vsebine ali spreminjanje pomena ni dovoljeno,
- cilj je izključno popraviti slovnico, ločila, velike začetnice ter tipkarske napake.

Popravljeno besedilo se nato vrne v isti cue format in se uporabi v končni SRT/VTT datoteki.

3.1.4 Gemini TTS (pretvorba besedila v govor)

Za pretvorbo scenarija v govor uporabljava Gemini TTS prek Gemini API. V konfiguraciji je nastavljen TTS model tipa gemini-2.5-flash-tts. Sistem scenarij posreduje TTS modelu kot vhodno besedilo in prejme generiran zvočni izhod.

Uporabljava prebuilt glasova (npr. Glas A in Glas B). Ker ima TTS generiranje lahko omejitve glede dolžine vhodnega besedila in stabilnosti odziva, besedilo deliva na manjše odseke, generiranje izvajava segmentno ter v primeru napak uporabiva več poskusov (retry mehanizem).

Končni audio posnetek (združen iz posameznih segmentov) predstavlja osnovo za nadaljnje korake, predvsem za generiranje videa in sinhronizacijo podnapisov.

Odsek kode za TTS:

```
def _tts_request(client: genai.Client, text_chunk: str) -> bytes:
    resp = client.models.generate_content(
        model=TTS_MODEL,
        contents=text_chunk,
        config=types.GenerateContentConfig(
            response_modalities=["AUDIO"],
            speech_config=types.SpeechConfig(
                multi_speaker_voice_config=types.MultiSpeakerVoiceConfig(
                    speaker_voice_configs=[
                        types.SpeakerVoiceConfig(speaker="A", voice_config=types.VoiceConfig(
                            prebuilt_voice_config=types.PrebuiltVoiceConfig(voice_name=TTS_VOICE_A))),
                        types.SpeakerVoiceConfig(speaker="B", voice_config=types.VoiceConfig(
                            prebuilt_voice_config=types.PrebuiltVoiceConfig(voice_name=TTS_VOICE_B))),
                    ]
```

```
            )
        ),
    ),
)
return resp.candidates[0].content.parts[0].inline_data.data
```

3.2 Lokalna obdelava podnapisov

3.2.1 Faster-Whisper (lokalna transkripcija TTS audia)

Za transkripcijo audia in pridobivanje časovnih žigov uporabljava knjižnico faster_whisper (razred WhisperModel). Gre za optimizirano in hitrejšo implementacijo Whisper ASR pipelinea, primerno za lokalno izvajanje. Model na podlagi vhodnega zvočnega posnetka izvede prepoznavo govora in vrne seznam segmentov, kjer ima vsak segment besedilo ter začetni in končni čas. Po potrebi lahko vrne tudi časovne žige na nivoju posameznih besed.

3.2.2 Sestava podnapisnih enot in pravila berljivosti

Iz transkripcijskih segmentov lokalno zgradiva podnapisne enote. Podnapisna enota je najmanjši samostojen časovno omejen del podnapisov, ki vsebuje besedilo ter pripadajoč začetni in končni čas prikaza (timecode). Ena podnapisna enota je tipično prikazana kot 1 do 2 vrstici besedila na zaslonu.

Pri tem uporabljava pravila, ki izboljšajo berljivost:

- omejitev na največ 2 vrstici,
- omejitev znakov na vrstico in znakov na podnapisno enoto,
- minimalno trajanje podnapisne enote,
- normalizacija presledkov in osnovna stabilizacija segmentov.

3.2.3 Slovnični popravek podnapisnih enot prek Geminija

Ko so podnapisne enote sestavljene, besedilo posameznih enot posredujeva modelu prek Gemini API za slovnični popravek. Popravek poteka izključno na nivoju besedila, brez spreminjanja števila podnapisnih enot in brez posega v časovne žige.

V praksi sistem izvede naslednje korake:

- iz vsake podnapisne enote vzame samo besedilo (brez timecode-ov),
- besedilo pošlje v Gemini z navodili, da popravi slovnico, ločila, velike začetnice in tipkarske napake,
- prejme popravljeno besedilo v enaki strukturi,
- popravljeno besedilo zapiše nazaj v isto podnapisno enoto.

S tem izboljšava profesionalen videz in razumljivost podnapisov.

3.3 Lokalno generiranje videa

3.3.1 FFmpeg (priprava audia + render videa)

Za multimedijsko obdelavo uporabljava FFmpeg. FFmpeg je odprtokodno ukazno-vrstično orodje in zbirka knjižnic za obdelavo avdio in video datotek, ki omogoča pretvorbe formatov, kodiranje, dekodiranje, filtriranje ter sestavljanje več medijskih virov v enoten izhod.

FFmpeg deluje tako, da sistem sestavi in izvede ukaz, v katerem definira:

- vhodne datoteke (npr. background.mp4 + voice.wav; ali image.jpg + narration.mp3),
- filtre za obdelavo (npr. scale=1080:1920, crop=1080:1920),
- izhodni format (npr. MP4, MOV; ali WebM),
- parametre kodiranja (npr. video codec libx264, preset=fast; ali CRF=23, bitrate=6M).

FFmpeg nato prebere vhodne vire, izvede filtre in jih zakodira v končni izhod.

FFmpeg uporabljava za:

- pretvorbo TTS audia v WAV, da je transkripcija stabilna in hitrejša,
- združevanje video ozadja, audia in podnapisov v končni video,
- prilagoditev slike na ciljno resolucijo (1080×1920),
- nastavitev parametrov kodiranja (preset/CRF/bitrate) kot kompromis med hitrostjo renderja in velikostjo/kakovostjo.

3.3.2 Vdelani podnapisi (subtitle filter + fontsdir)

Podnapise dodajava prek FFmpeg subtitles filtra, ki deluje kot video filter v filtergraphu. FFmpeg pri tem prebere podnapisno datoteko, jo interpretira (SRT/VTT ali ASS) in nato besedilo podnapisov izriše neposredno na posamezne video frame. Rezultat so burn-in podnapisi, kar pomeni, da so podnapisi trajno vgrajeni v video in niso ločena podnapisna steza.

Pri tem uporabljava:

- možnost uporabe ASS ali “navadnih” podnapisov (npr. subtitles=subtitles.srt; ali subtitles=subtitles.ass)
- nastavitve pisave, velikosti, outline/shadow/margins
- fontsdir, če je potrebno zagotoviti pravilno pisavo na sistemu

FFmpeg nato pri renderju za vsak frame izračuna, kateri podnapis mora biti viden glede na časovne žige, ter ga izriše z izbranimi stilskimi nastavitvami.

Primer kode za podnapise:

```
def _render_video_with_audio_and_subs_progress(...):
    vf = f"{vf_resize},{_ffmpeg_filter_for_subtitles(sub_path, is_ass)}"
    cmd = [
        "ffmpeg", "-y", "-i", video_in, "-i", audio_in,
        "-vf", vf, "-c:v", "libx264", "-preset", VIDEO_PRESET, "-crf", str(VIDEO_CRF),
        "-c:a", "aac", "-b:a", VIDEO_AUDIO_BITRATE, "-shortest",
        "-progress", "pipe:1", "-nostats", out  path,
    ]
    p = subprocess.Popen(cmd, stdout=subprocess.PIPE, text=True)
    for line in p.stdout:
        if line.startswith("out  time  ms="):
            on_progress(ratio, eta)
```

Primer izdelanih podnapisov:

Slika 2: primer podnapisov Vir: lasten

3.3.3 Video ozadja (lokalna knjižnica ozadij)

Ozadja za video izbirava lokalno iz zbirke datotek (mapa z videi po žanrih). Sistem izbere random ozadje glede na izbran žanr in ga uporabi kot vizualno podlago za končni video.

3.4 Backend tehnologije (API, baza, ozadna opravila)

3.4.1 FastAPI (REST API strežnik)

Backend je implementiran v FastAPI, kar pomeni, da je aplikacija zgrajena kot HTTP API strežnik v Pythonu. FastAPI definira končne točke (endpointe) in podatkovne modele, ob vsaki zahtevi pa izvede validacijo vhodnih podatkov, pokliče ustrezno poslovno logiko in vrne odgovor v strukturirani obliki (tipično JSON).

FastAPI se izvaja prek ASGI strežnika. ASGI (Asynchronous Server Gateway Interface) je standardni vmesnik med Python aplikacijo in strežnikom, ki omogoča asinhrono obdelavo zahtev. ASGI strežnik (npr. Uvicorn ali Hypercorn) sprejme HTTP zahteve od odjemalcev, jih posreduje FastAPI aplikaciji in omogoča učinkovito obdelavo več sočasnih povezav, kar je posebej pomembno pri I/O operacijah (npr. nalaganje datotek, klici zunanjih API-jev, čakanje na job rezultate).

API skrbi za:

nalaganje slik in ustvarjanje jobov: endpoint sprejme naložene datoteke, shrani vhod, ustvari zapis joba (npr. job_id) in sproži obdelavo v ozadju ali prek delavcev

- ❖ vračanje stanja napredka in rezultatov: endpointi omogočajo polling ali pridobitev trenutnega statusa joba (npr. queued/running/failed/done) ter dostop do vmesnih ali končnih rezultatov
- ❖ kviz, tržnico in odklepanje vsebin: endpointi izpostavijo logiko generiranja/pridobivanja kviza, prikaz in nakup elementov v tržnici ter mehanizem odklepanja vsebin na podlagi pravil (npr. točke, nivo, nakup)
- ❖ upravljanje uporabnikov in točk: endpointi za registracijo/prijavo, avtentikacijo (npr. JWT), upravljanje profilov ter knjiženje in preverjanje stanja točk (npr. dodelitev za aktivnost, poraba ob odklepu)

3.4.2 SQLAlchemy (ORM) in relacijska baza

Za shranjevanje podatkov uporabljava relacijsko bazo prek SQLAlchemy ORM. SQLAlchemy je Python knjižnica, ki omogoča delo z relacijsko bazo na nivoju objektov, kjer so tabele predstavljene kot razredi, vrstice kot instance teh razredov, relacije med tabelami pa kot povezave med objekti. ORM plast skrbi za generiranje SQL poizvedb, mapiranje rezultatov nazaj v Python objekte ter upravljanje transakcij.

V praksi to pomeni, da aplikacija namesto ročnega pisanja SQL uporablja SQLAlchemy modele in query mehanizme, SQLAlchemy pa v ozadju izvaja INSERT/UPDATE/SELECT/DELETE operacije nad bazo. To omogoča bolj konsistentno podatkovno shemo, lažje vzdrževanje in centralizirano upravljanje relacij (npr. uporabnik → job-i, job → kviz).

V bazo shranjujeva:

- uporabnike in točke (npr. user tabela, stanje točk, zgodovina porabe/dodelitev)
- job-e (status, napredek, OCR/verified besedilo, scenarij, poti do audia/videa, podnapisne enote) (npr. job_id, queued/running/done, progress %, paths)
- odklepe (žanri, stili podnapisov, podcasti v tržnici) (npr. unlock tabela z user_id, item_type, item_id, timestamp)
- kviz in povezane podatke (npr. quiz tabela, vprašanja, odgovori, povezava na job_id in user_id)

3.4.3 Redis + RQ (job queue in worker)

Za dolgotrajna opravila (OCR, generiranje scenarija, TTS, transkripcija, podnapisi, video render) uporabljava Redis in RQ (Redis Queue). Redis deluje kot hiter in-memory podatkovni strežnik, ki v tem primeru služi kot čakalna vrsta opravil, RQ pa je Python knjižnica, ki omogoča dodajanje opravil v Redis vrsto ter njihovo izvajanje prek ločenih worker procesov.

Delovanje je takšno, da backend ob ustvarjanju joba nalogo doda v Redis vrsto, RQ worker pa jo prevzame in izvede v ozadju ločeno od HTTP zahtev. Med izvajanjem worker sproti posodablja status in napredek joba, backend pa ta podatek izpostavi prek API endpointa, da lahko frontend prikazuje faze obdelave.

3.4.4 Pydantic (validacija podatkov)

Za validacijo vhodnih in izhodnih podatkov uporabljava Pydantic. Pydantic je Python knjižnica, ki definira podatkovne modele kot razrede in ob ustvarjanju teh modelov avtomatsko izvede validacijo tipov, obveznih polj, formatov in omejitev. V FastAPI se Pydantic modeli uporabljajo kot request in response sheme, zato se podatki preverijo že ob vstopu v endpoint, preden se izvede poslovna logika.

Če podatki ne ustrezajo definiranemu modelu (npr. manjka obvezno polje, napačen tip ali neveljaven format), FastAPI samodejno vrne napako z opisom validacijskih problemov. To izboljša robustnost API-ja in zmanjša napake pri komunikaciji med frontend in backend, ker so strukture podatkov strogo definirane in konsistentne.

3.5 Avtentikacija in varnost

3.5.1 JWT (python-jose)

JWT žetoni (JSON Web Token) so podpisani tekstovni žetoni, ki vsebujejo podatke o uporabniku in se uporabljajo za avtentikacijo med odjemalcem (frontend) in backendom. JWT je sestavljen iz treh delov (header.payload.signature), kjer payload vsebuje trditve o uporabniku (npr. user_id, vloge, čas veljavnosti), signatura pa zagotavlja, da žeton ni bil spremenjen.

V najinem primeru JWT generira backend ob prijavi. Frontend nato JWT shrani (npr. v local storage ali secure cookie) in ga pri nadaljnjih zahtevah pošilja v Authorization headerju. Backend pri vsakem zaščitenem endpointu JWT preveri s python-jose, kar vključuje preverjanje podpisa, veljavnosti (exp) in po potrebi dodatnih pravic. Če je žeton veljaven, backend iz njega prebere identiteto uporabnika in na tej osnovi dovoli ali zavrne dostop do endpointa.

3.5.2 Hashiranje gesel (passlib + bcrypt)

Hashiranje je postopek, kjer iz vhodnega podatka (npr. gesla) izračunava enosmerno vrednost fiksne oblike, imenovane hash. Enosmerno pomeni, da iz hasha ni mogoče praktično rekonstruirati originalnega gesla. Hashiranje se uporablja zato, da se gesel nikoli ne shranjuje v čisti obliki, ampak samo v obliki, ki jo je mogoče preveriti, ne pa prebrati.

V praksi deluje tako, da backend ob registraciji vzame geslo, mu doda naključno vrednost (salt) in nato izvede bcrypt hashiranje prek passlib. V bazo se shrani samo rezultat (bcrypt hash). Ob prijavi backend ponovno izvede bcrypt postopek nad vnesenim geslom in primerja rezultat z

že shranjenim hashom. Če se ujemata, je geslo pravilno, brez da bi kadarkoli potrebovali originalno geslo iz baze.

3.6 Frontend tehnologije (uporabniški vmesnik)

3.6.1 Next.js/React

Frontend je implementiran v React/Next.js. React je JavaScript knjižnica za gradnjo uporabniških vmesnikov, kjer je aplikacija sestavljena iz komponent, ki se ponovno izrisujejo glede na stanje (state) in podatke (props). Next.js je ogrodje nad Reactom, ki doda strukturiran projekt, usmerjanje (routing), možnost strežniškega renderiranja (SSR), statično generiranje strani (SSG) ter API integracijo, kar poenostavi gradnjo produkcijskih spletnih aplikacij.

Delovanje je takšno, da uporabnik v brskalniku dostopa do Next.js aplikacije, ki prikazuje React komponente. Te komponente prek HTTP klicev komunicirajo z backend API-jem, pošiljajo podatke (npr. slike, prijavne podatke) ter prejemajo rezultate (npr. status joba, OCR besedilo, video URL, kviz). Frontend upravlja stanje uporabnika (npr. prijavljen uporabnik, JWT token), stanje joba (npr. queued/running/done) ter prikaz posameznih korakov v UI.

Slika 3: Uporabniški vmesnik Vir: lasten

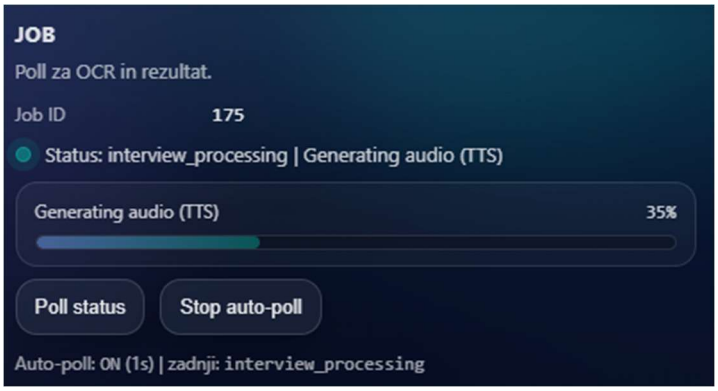
Uporabniški vmesnik omogoča:

- registracijo in prijavo: frontend pošlje podatke backendu, prejme JWT in ga uporablja za nadaljnje klice,
- nalaganje slik (tudi več slik): frontend naloži datoteke prek multipart/form-data in sproži ustvarjanje joba;
- prikaz napredka joba: frontend periodično preverja status prek API-ja in prikazuje trenutno fazo obdelave;
- pregled OCR besedila in urejanje verificiranega besedila: frontend prikaže prepoznano besedilo, omogoči uporabniku popravke in shrani potrjeno verzijo;
- predvajanje videa: frontend naloži končni video in omogoča predvajanje v brskalniku;
- kviz med predvajanjem: frontend prikazuje vprašanja sinhrono z vsebino in beleži uporabnikove odgovore;
- tržnico in odklepanje vsebin: frontend prikazuje elemente, preverja stanje točk ter omogoča odklepanje dodatnih možnosti (npr. žanri, stili podnapisov, podcasti).

3.6.2 Komunikacija z backendom

Ker obdelava poteka v ozadju, frontend uporablja vzorec periodičnega preverjanja stanja (polling). Ko uporabnik naloži slike in backend ustvari job, frontend prejme identifikator joba in začne v rednih časovnih intervalih klicati API endpoint za status. Odgovor tipično vsebuje trenutni status (npr. queued, running, failed, done), trenutno fazo obdelave (npr. ocr, script, tts, transcribe, subtitles, render) ter numerični napredek ali dodatne metapodatke (npr. trenutni korak, morebitno opozorilo ali sporočilo o napaki).

Na podlagi tega frontend posodablja lokalno stanje aplikacije in ponovno izriše uporabniški vmesnik. V praksi to pomeni, da uporabnik vidi jasno označen korak, ki je trenutno v teku, prejšnji koraki so označeni kot zaključeni, naslednji kot čakajoči. Če backend vrne odstotek napredka, frontend lahko prikaže progress bar; če vrne samo faze, prikaže stepper oziroma seznam faz. Ko status preide v done, frontend preneha s pollingom in avtomatsko preklopi v prikaz rezultatov (npr. OCR besedilo, video predvajalnik, kviz). Če status preide v failed, frontend preneha s pollingom in prikaže napako ter opcijo za ponovni poskus ali popravek vhodnih podatkov.



Slika 4: Prikaz statusa Vir: lasten

4. Rezultati in diskusija

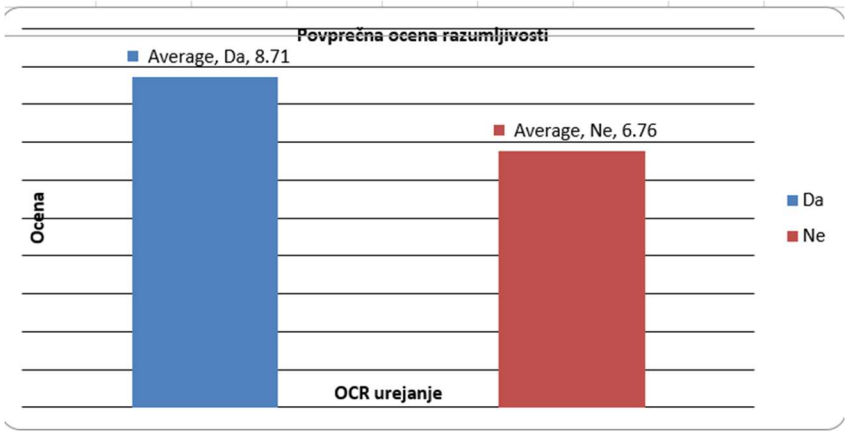
4.1 Hipoteza: Avtomatizirana pretvorba slikovnih zapiskov v video z govorom in podnapisi statistično značilno skrajša čas priprave učnega gradiva v primerjavi z ročnim postopkom.

Ta hipoteza je POTRJENA. Rezultati so pokazali, da povprečni čas izdelave videa z avtomatiziranim sistemom znaša 10 minut, medtem ko ročna priprava iste vsebine traja povprečno 92 minut. Razlika 82 minut predstavlja približno 89 % skrajšanje časa priprave. Avtomatizacija torej bistveno poveča učinkovitost produkcije učnih vsebin in zmanjša časovno obremenitev ustvarjalca.

4.2 Hipoteza: Vključitev uporabniške verifikacije in popravka OCR izpisa statistično značilno zmanjša število napak v podnapisih ter izboljša razumljivost generirane vsebine.

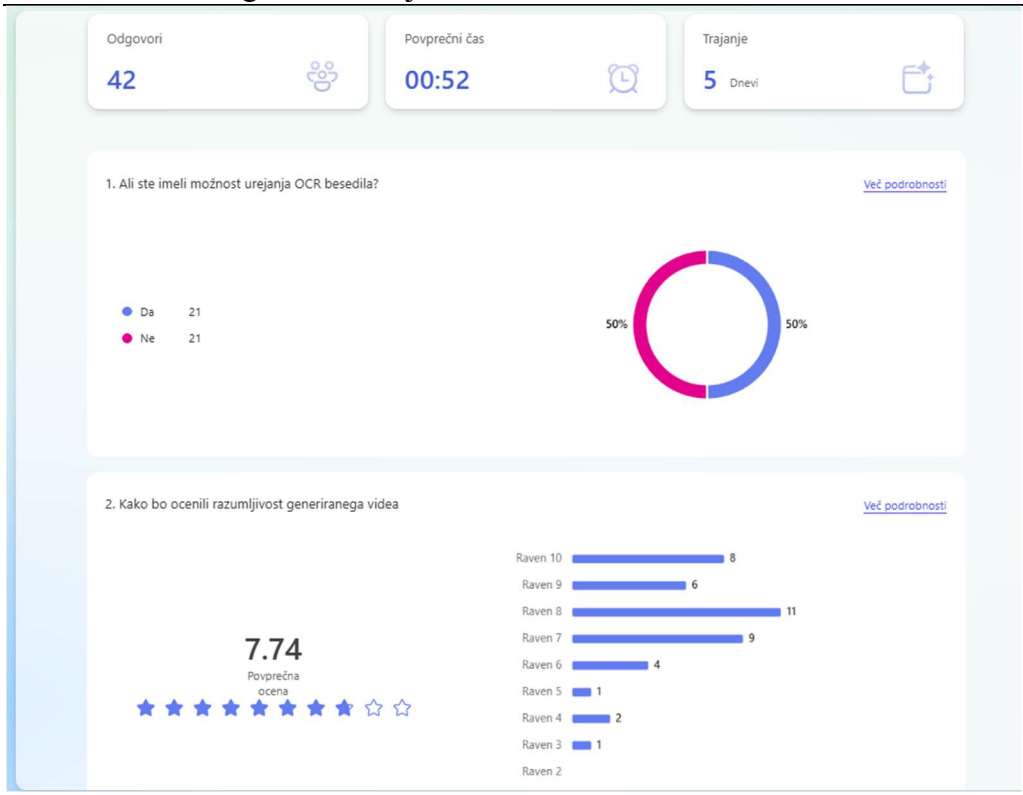
Ta hipoteza je POTRJENA. Z vključitvijo verifikacije so se vrednosti kazalnikov CER (Character Error Rate) in WER (Word Error Rate) statistično značilno znižale. Hkrati se je povečala povprečna subjektivna ocena razumljivosti vsebine. Rezultati potrjujejo, da kontrolni pregled besedila preprečuje širjenje napak skozi nadaljnje faze obdelave in neposredno vpliva na kakovost končnega videa.

Na spodnji sliki vidimo, da je skupina, ki je imela na opcijo urejanje OCR besedila, z rezultatom videa veliko bolj zadovoljna in ga bolje razume. Razumljivost videa je bila ocenjena z povprečno oceno 8,71, v primerjavi z skupino, ki napak v OCR bsedilo ni mogla popraviti in je povprečna ocena samo 6,76.



Slika 5: graf povprečne ocene razumljivosti Vir: lasten

Na spodnji sliki vidimo še skupen rezultat vseh odgovorov iz microsoft form spletne strani.

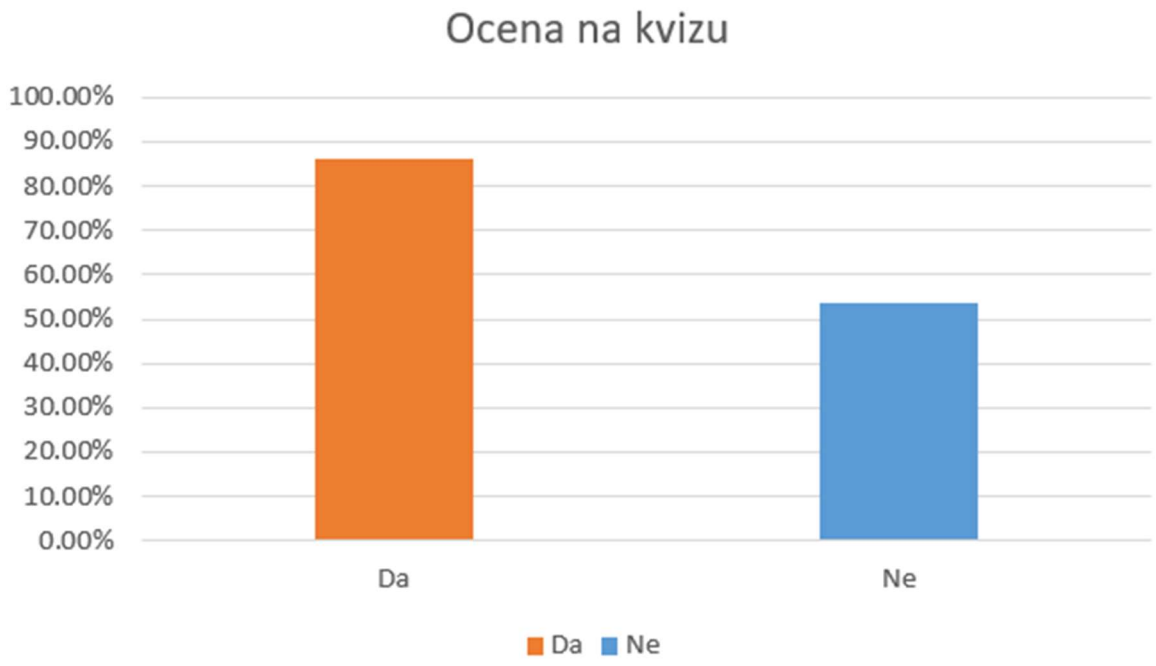


Slika 6: prikaz rezultatov kviza v Microsoft forms Vir: lasten

4.3 Hipoteza: Vključitev interaktivnega kviza med predvajanjem videa statistično značilno izboljša razumevanje učne vsebine in poveča uspešnost na končnem testu znanja.

Ta hipoteza je POTRJENA. Skupina uporabnikov, ki je uporabljala video z vmesnimi kvizi, je dosegla višjo povprečno uspešnost na končnem testu znanja v primerjavi s skupino brez interaktivnih elementov. Prav tako je bila stopnja dokončanja videa višja pri skupini z interaktivnim pristopom. Rezultati potrjujejo, da aktivno vključevanje uporabnika spodbuja boljše pomnjenje in razumevanje snovi.

Na podnji sliki je razvidno, da so uporabniki, ki so imeli med videom kvize z vprašanji na testu odgovorili veliko bolj pravilno. 21 uporabnikov je gledalo enak video, vendar jih je 11 imelo med videom vklopljen sproten kviz z vprašanji, 10 pa ga je imelo izklopljenega. V prilogi D lahko vidimo točna vprašanja in statistiko odgovorov.



Slika 7: Prikaz rezultatov ocene na kvizu Vir: lasten

4.4 Hipoteza: Uvedba sistema točk in odklepanja vsebin statistično značilno poveča pogostost ponovne uporabe aplikacije ter uporabniško angažiranost.

Ta hipoteza je POTRJENA. Po uvedbi gamifikacijskih mehanizmov se je povečalo povprečno število ustvarjenih vsebin ter povprečni čas uporabe aplikacije na sejo. Analiza podatkov kaže na jasen vedenjski premik v smeri pogostejše in daljše uporabe sistema, kar potrjuje učinkovitost motivacijskih elementov pri digitalnih učnih okoljih.

To sva razbrala iz števila ustvarjenih videov v podatkovni bazi. Primerjala sva številke ustvarjenih videov pred in po dodatku gamifikacijskih mehanizmov in številke so prikazale, da se je po dodatku povprečno število ustvarjenih videov na uporabnika zvišalo.

4.5 Hipoteza: Zvoku dodati v ozadje tudi video pomaga učencu, da lahko fokus obdrži dlje časa.

Ta hipoteza je POTRJENA. Kot je razvidno v raziskavi Evaluation of TikTok as a mode of instruction in teaching science, ki jo je objavil The Research Probe 30. 6. 2025 (Priloga E), so bili študenti pri učenju z uporabo kratkih, vizualno podprtih videov visoko angažirani, kar je eden ključnih pogojev za daljše vzdrževanje pozornosti.

Poleg večje angažiranosti so rezultati pokazali tudi izboljšanje učne uspešnosti: v “TikTok” skupini se je uspešnost od pred testa do po testne meritve izboljšala iz “zadovoljivo” v “zelo

zadovoljivo", medtem ko je tradicionalna skupina ostala na "zadovoljivi" ravni v obeh meritvah. To nakazuje, da kombinacija zvoka in enostavnih vizualnih elementov ne pomaga le pri ohranjanju pozornosti, ampak tudi pri učinkovitejšem priklicu informacij. V kontekstu tvoje naloge to pomeni, da dodan video deluje kot vizualna opora, ki zmanjšuje monotonost poslušanja, krepi zanimanje in podpira daljše spremljanje vsebine.

Hipoteze so se izkazale za uspešne in rezultati so bili v vseh primerih precej jasni. Pri hipotezi 4.1 je avtomatizirana pretvorba zapiskov v video močno skrajšala pripravo gradiva: povprečno je trajala 10 minut, ročni postopek pa 92 minut, kar pomeni približno 89 % prihranek časa. Pri hipotezi 4.2 se je pokazalo, da uporabniška verifikacija OCR pomembno izboljša kakovost, saj se je število napak v podnapisih (CER/WER) statistično značilno zmanjšalo, hkrati pa se je povečala razumljivost vsebine. Pri hipotezi 4.3 je vključitev interaktivnih kvizov izboljšala razumevanje snovi: uporabniki so dosegli boljše rezultate na končnem testu in pogosteje dokončali video, kar kaže na večjo aktivno vključenost. Pri hipotezi 4.4 je sistem točk in odklepanja vsebin povečal angažiranost, saj so uporabniki v aplikaciji preživeli več časa in ustvarili več vsebin na sejo, kar potrjuje motivacijski učinek gamifikacije. Pri hipotezi 4.5 se je pokazalo, da dodan preprost video v ozadju ob zvočni razlagi pomaga pri ohranjanju pozornosti, saj poveča angažiranost uporabnikov in podpira boljši učni učinek v primerjavi z zgolj tradicionalnim pristopom.

5. ZAKLJUČEK

V raziskovalni nalogi sva razvila in analizirala sistem za avtomatizirano pretvorbo slikovnih zapiskov v izobraževalni video z govorom, podnapisi in interaktivnimi elementi. Raziskava je bila usmerjena v primerjavo avtomatiziranega postopka z ročno pripravo učnega gradiva ter v oceno vpliva posameznih funkcionalnosti sistema na kakovost vsebine, razumevanje snovi in uporabniško angažiranost.

Pri analizi časovne učinkovitosti sva ugotovila, da avtomatiziran sistem bistveno skrajša proces izdelave učnega videa. Povprečni čas od nalaganja slik do končnega generiranega videa znaša 10 minut, medtem ko ročna priprava vključuje več ločenih faz (prepis zapiskov, priprava scenarija, snemanje govora, izdelava podnapisov, montaža videa) in traja povprečno 92 minut. Integracija optičnega prepoznavanja znakov (OCR), generiranja besedila, sinteze govora (TTS) in avtomatskega renderiranja videa omogoča enoten, tehnološko podprt proces, ki zmanjšuje potrebo po ročnem delu.

Kakovost vsebine je neposredno povezana z natančnostjo prepoznave besedila. Analiza je pokazala, da se brez verifikacije napake iz OCR faze prenašajo v nadaljnje faze generiranja scenarija, govora in podnapisov. Vključitev uporabniške verifikacije predstavlja pomemben kontrolni mehanizem, ki stabilizira celoten sistem in poveča razumljivost končnega izdelka.

Rezultati so prav tako pokazali, da interaktivni elementi, kot so vmesni kvizi med predvajanjem videa, pomembno vplivajo na učni proces. Aktivna vključitev uporabnika spodbuja globljo obdelavo informacij in izboljšuje uspešnost na testu znanja. Poleg tega je uvedba sistema točk in odklepanja vsebin vplivala na vedenjske vzorce uporabnikov ter povečala pogostost uporabe aplikacije.

Na podlagi vseh ugotovitev lahko sklepava, da avtomatizirana pretvorba zapiskov v video predstavlja učinkovito, časovno optimizirano in pedagoško utemeljeno rešitev za ustvarjanje digitalnih učnih vsebin.

6. Povzetek

Raziskava opisuje razvoj in analizo sistema, ki avtomatizira pretvorbo slikovnih zapiskov v učne videoposnetke, vključujoč govor, podnapise in interaktivne elemente, kot so kvizi. Sistem združuje več tehnologij: OCR za prepoznavo besedila iz slik, generiranje strukturiranega scenarija in kvizov prek Gemini API, TTS za pretvorbo besedila v govor, lokalno sestavo in časovno poravnavo podnapisov ter njihovo slovnično korekcijo. Končni video se generira s pomočjo FFmpeg, ki omogoča združevanje ozadij, zvoka in podnapisov v enoten multimedijski izdelek.

Backend aplikacije temelji na FastAPI za upravljanje API zahtev, SQLAlchemy za relacijsko bazo, Redis in RQ za obdelavo dolgotrajnih opravil ter Pydantic za validacijo podatkov, medtem ko frontend uporablja Next.js/React za prikaz uporabniškega vmesnika, upravljanje stanja in interakcijo z uporabniki.

Avtomatiziran sistem bistveno skrajša čas priprave učnega gradiva, izboljša natančnost in berljivost vsebine z vključitvijo preverjanja OCR izpisa ter spodbuja boljše razumevanje snovi s pomočjo interaktivnih kvizov. Hkrati sistem omogoča večjo angažiranost uporabnikov prek mehanizmov točk in odklepanja vsebin, kar povečuje pogostost in trajanje uporabe aplikacije.

7. Viri in literatura

[1] Chan J. C. K. et al. | In-lecture quizzes improve online learning for university and community college students
https://pmc.ncbi.nlm.nih.gov/articles/PMC11965562/ (16. 2. 2026)
[2] FastAPI | Background Tasks (asinhrona opravila po HTTP odgovoru)
https://fastapi.tiangolo.com/tutorial/background-tasks/ (15. 2. 2026)
[3] FFmpeg | HowToBurnSubtitlesIntoVideo (subtitles/ass filter, vžgani podnapisi)
https://trac.ffmpeg.org/wiki/HowToBurnSubtitlesIntoVideo (13. 2. 2026)
[4] Google Cloud | Detect and extract text from images (Cloud Vision OCR)
https://docs.cloud.google.com/vision/docs/ocr (10. 2. 2026)
[5] Google Cloud | Gemini-TTS (Cloud Text-to-Speech)
https://docs.cloud.google.com/text-to-speech/docs/gemini-tts (11. 2. 2026)
[6] Li M. et al. | Examining the effectiveness of gamification as a tool promoting teaching and learning (meta-analiza)
https://pmc.ncbi.nlm.nih.gov/articles/PMC10591086/ (16. 2. 2026)
[7] Netflix | Timed Text Style Guide: General Requirements (line breaks, max 2 lines ipd.)
https://partnerhelp.netflixstudios.com/hc/en-us/articles/215758617-Timed-Text-Style-Guide-General-Requirements (14. 2. 2026)
[8] SQLAlchemy | SQLAlchemy ORM (uradna dokumentacija)
https://docs.sqlalchemy.org/orm/ (16. 2. 2026)
[9] SYSTRAN | faster-whisper (CTranslate2 implementacija Whisper)
https://github.com/SYSTRAN/faster-whisper (12. 2. 2026)
[10] The Research Probe | Evaluation of TikTok as a mode of instruction in teaching science
https://media.neliti.com/media/publications/661682-evaluation-of-tiktok-as-a-mode-of-instru-f747909f.pdf (30. 6. 2025)

8. Zahvale

Vesela sva, da sva lahko raziskovalno nalogo izpeljala, kot sva si zadala in izpolnila svoje cilje. Najprej bi se rada zahvalila mentorju Islamu Mušiću za njegovo pomoč in podporo pri delu. Za pomoč pri testiranju bi se rada zahvalila tudi sošolcem in prijateljem. Na koncu se zahvaljujeva še družini in bližnjim, za vso pomoč pri najinem raziskovalnem delu.

9. Priloge

Priloga A: koda frontenda

Page.js

Priloga B: koda backend

Main.py

Priloga C: koda workerja

Tasks.py

Priloga D: Vprašanja kviza (Hipoteza 4.3)

Priloga_D_Biblija_Kviz.pdf

Priloga E: Raziskava o TikToku kot način učenja

Raziskava