VIŠJA STROKOVNA ŠOLA ACADEMIA

MARIBOR

AVTOMATIZACIJA BACKEND PROCESOV Z UPORABO UMETNE INTELIGENCE

Kandidat: Riad Imširović

Vrsta študija: študent izrednega študija

Študijski program: Informatika

Mentor predavatelj: mag. Dušan Brglez

Mentor v podjetju: mag. Dejan Kropec

Lektor: dr. Klemen Jelinčič

Maribor, 2026

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Podpisani/a Riad Imširović sem avtor/ica diplomskega dela z naslovom Avtomatizacija backend procesov z uporabo umetne inteligence, ki sem ga napisal/a pod mentorstvom mag. Dušana Brgleza.

S svojim podpisom zagotavljam, da:

- je predloženo delo izključno rezultat mojega dela,
- sem poskrbel/a, da so dela in mnenja drugih avtorjev, ki jih uporabljam v predloženi nalogi, navedena oz. citirana skladno s pravili Višje strokovne šole Academia Maribor,
- se zavedam, da je plagiatorstvo – predstavljanje tujih del oz. misli kot moje lastne kaznivo po Zakonu o avtorski in sorodnih pravicah (Uradni list RS, št. 16/07 – uradno prečiščeno besedilo, 68/08, 110/13, 56/15 in 63/16 – ZKUASP); prekršek pa podleže tudi ukrepom Višje strokovne šole Academia Maribor skladno z njenimi pravili,
- skladno z 32.a členom ZASP dovoljujem Višji strokovni šoli Academia Maribor objavo diplomskega dela na spletnem portalu šole.

Maribor, marec 2026 Podpis študenta:

ZAHVALA

Najprej bi se rad zahvalil šoli za znanje, ki je pripomoglo k nastanku in pisanju te diplomske naloge. Zahvaljujem se tudi mentorju na šoli ter mentorju v podjetju za njuno podporo in usmerjanje. Posebna zahvala gre kolektivu Višje strokovne šole Academia, sošolcem, s katerimi sem delil študijsko pot, ter vodstvu podjetja IT Melona za dragocene izkušnje in znanje, ki sem ga pridobil v času sodelovanja z njimi.

POVZETEK

Diplomsko delo obravnava načrtovanje in izvedbo spletnega klepetalnega robota, ki samodejno upravlja z rezervacijami terminov za podjetje. Cilj je bil z uporabo sodobnih tehnologij (Botpress, Node.js, n8n, Google Koledar in Docker) razviti delujoč prototip, ki uporabnikom omogoča rezervacijo termina preko preprostega klepetalnega vmesnika na spletni strani brez posredovanja zaposlenih. Sistem poleg rezervacije vključuje zasnovo funkcionalnosti preklica, ki pa zaradi časovnih omejitev ni bila v celoti testirana in integrirana.

V teoretičnem delu so predstavljene uporabljene tehnologije: Botpress za vizualno načrtovanje pogovornih tokov, Node.js in Express za razvoj backend API-ja, n8n za orkestracijo delotokov, Google Calendar API za upravljanje z dogodki ter Docker za kontejnerizacijo. Sledi opis arhitekture sistema, ki temelji na mikro storitvah in komunikaciji preko HTTP. Podatkovni tok vodi uporabnika od pozdravnega sporočila preko zbiranja podatkov (ime, e-pošta, datum, ura, razlog) do klica backend API-ja, ki preko n8n delotoka ustvari dogodek v Google Koledarju in uporabniku prikaže potrditev.

V praktičnem delu so podrobno opisane implementacija klepetalnega robota v Botpressu, struktura backend API-ja, validacija podatkov s knjižnico Joi, povezava z n8n in integracija z Google Koledarjem preko storitvenega računa. Posebno poglavje je namenjeno kibernetski varnosti, kjer so identificirana potencialna tveganja in predlagani ukrepi za izboljšave, kot so uvedba avtentikacije, *rate limiting* in centralizirano beleženje dogodkov. Analiza po OWASP smernicah potrjuje četrto hipotezo, da integracija več storitev prinaša specifična tveganja, ki zahtevajo dodatno pozornost.

Na podlagi rezultatov so potrjene vse štiri hipoteze: čas rezervacije se je skrajšal, natančnost razumevanja uporabniških vnosov je visoka (98 %), n8n deluje zanesljivo in sledljivo, kibernetska tveganja so identificirana in zahtevajo dodatne varnostne ukrepe.

V zaključku so predstavljene omejitve (podpora le za en koledar, pomanjkanje naprednega upravljanja s konflikti) in možnosti za nadaljnji razvoj, med katerimi izstopajo dokončanje funkcionalnosti preklica, uvedba potrditve preko e-pošte in obveščanje zaposlenih o rezervacijah. Delo prikazuje, kako lahko sodobne tehnologije poenostavijo administrativna opravila in izboljšajo uporabniško izkušnjo.

Ključne besede: *avtomatizacija, klepetalni robot, Botpress, n8n, Google Koledar, Node.js, Docker, kibernetska varnost*

ABSTRACT

»Automation of backend processes using artificial intelligence«

This thesis presents the design and implementation of a web-based chatbot system that automates appointment scheduling for a company. The aim was to develop a working prototype using modern technologies that allows users to book appointments through a simple chatbot interface on the company website, without manual staff intervention. The system also includes a design for cancellation functionality, which, due to time constraints, was not fully tested and integrated.

The theoretical part introduces the technologies used: Botpress for visual design of conversational flows, Node.js and Express for backend API development, n8n for workflow orchestration, Google Calendar API for event management, and Docker for containerization. This is followed by a description of the system architecture. The data flow guides the user from a welcome message, through data collection, to a backend API call that, via an n8n workflow, creates an event in Google Calendar and returns confirmation to the user.

The practical part details the implementation of the chatbot in Botpress, the structure of the backend API, data validation using the Joi library, the connection to n8n, and the integration with Google Calendar via a service account. A separate chapter is dedicated to cybersecurity, identifying potential and proposing improvements such as authentication, rate limiting, and centralized logging. The OWASP-based analysis confirms the fourth hypothesis that integrating multiple services introduces specific cybersecurity risks that require additional attention.

Based on the results, all four hypotheses were confirmed: booking time was reduced, the accuracy of understanding user inputs is high (98%), n8n works reliably and traceably, and cybersecurity risks were identified and require additional measures.

The entire system is containerized with Docker, allowing easy deployment on a VPS server. The conclusion presents limitations and opportunities for future development, notably completing the cancellation feature, implementing email confirmation, and notifying employees about reservations. The thesis demonstrates how modern technologies can streamline administrative tasks and improve the user experience.

Keywords: *automation, chatbot, Botpress, n8n, Google Calendar, Node.js, Docker, cybersecurity*

KAZALO VSEBINE

1 UVOD 9

1.1 Opis področja in opredelitev problema 9
1.2 Namen, cilji in osnovne trditve 9
1.3 Predpostavke in omejitve 10
1.4 Uporabljene raziskovalne metode 12

2 PREGLED TEHNOLOGIJ 15

2.1 Botpress 15
2.2 Node.js in Express.js 15
2.3 n8n 15
2.4 Google Calendar API 15
2.5 Docker kontejnerizacija 15

3 ARHITEKTURA SISTEMA 16

3.1 Pregled arhitekture 16
3.2 Komponente sistema 17
3.3 Podatkovni tok 18

4 IMPLEMENTACIJA 23

4.1 Chatbot vmesnik (Botpress) 23
4.1.1 Načrtovanje pogovornega toka 23
4.1.2 Implementacija v Botpressu 23
4.1.3 Akcije in integracija z backendom 23
4.2 Backend API 25
4.2.1 Struktura projekta 25
4.2.2 Endpointi za rezervacijo in preklic 35
4.2.3 Validacija podatkov 35
4.2.4 Integracija z n8n 37
4.3 Avtomatizacija z n8n 37
4.3.1 Pregled delotokov 37
4.3.2 Delotok za rezervacijo 37
4.3.3 Delotok za preklic 38
4.3.4 Opis vozlišč 38
4.4 Integracija z Google Koledarjem 42
4.4.1 Uporaba storitvenega računa 42
4.4.2 Ustvarjanje in brisanje dogodkov 42

5 ANALIZA KIBERNETSKE VARNOSTI SISTEMA 44

5.1 Uvod v kibernetsko varnost avtomatiziranih sistemov 44
5.2 Identifikacija potencialnih varnostnih tveganj 44
5.2.1 Tveganja, povezana z Botpressom 44
5.2.2 Tveganja, povezana z backend APIjem 44
5.2.3 Tveganja, povezana z n8n 45
5.2.4 Tveganja, povezana z Google Koledar API-jem 45
5.2.5 Tveganja, povezana z Docker okoljem 45
5.3 Implementirani varnostni ukrepi 45
5.4 Analiza tveganj po OWASP smernicah 46
5.5 Priporočila za izboljšanje varnosti 47
5.6 Zaključek analize in potrditev hipoteze H4 48
6 NAMESTITEV IN ZAGON 49
6.1 kontejnerizacija z Dockerjem 49
6.2 Zagon na VPS 51
6.3 Uporaba ngroka za testiranje 52
6.4 Dokončanje funkcionalnosti preklica termina 54
7 TESTIRANJE IN REZULTATI 56
7.1 Testiranje chatbot vmesnika 56
7.2 Testiranje API-jev 56
7.3 Testiranje n8n delotokov 58
7.4 Rezultati in analiza 58
8 SKLEP 59
9 VIRI IN LITERATURA 61
10 PRILOGE 62

KAZALO SLIK

Slika 1: Arhitekturni diagram 17
Slika 2: Pozdravno sporočilo 17
Slika 3: Vprašanje za ime in priimek 19
Slika 4: Vprašanje za e-pošto 20
Slika 5: Vprašanje za datum 20
Slika 6: Vprašanje za določeno uro in razlog 21
Slika 7: Potrditveno sporočilo 22
Slika 8: Botpress workflow 23
Slika 9: Booking input shema 24
Slika 10: Booking action funkcija 24

Slika 11: Booking output shema 25
Slika 12: Backend rute 35
Slika 13: Status po validaciji 36
Slika 14: N8n vozlišča in workflow 37
Slika 15: Neuspešna izvedba delovnega toka 40
Slika 16: Uspešna izvedba delovnega toka 40
Slika 17: »Create appointment« endpoint 43
Slika 18: Docker zabojniki 51
Slika 19: Status zabojnika v terminalu 52
Slika 20: nGrok pri zagonu 53
Slika 21: Health check endpoint 54
Slika 22: Rezervacija z veljavnimi podatki 56
Slika 23: Rezervacija z datumom v preteklosti 57
Slika 24: Rezervacija z napačno e-poštom 57

KAZALO TABEL

Tabela 1: Analiza tveganj po OWASP smernicah 46

1 UVOD

1.1 Opis področja in opredelitev problema

V današnjem poslovnem okolju predstavlja učinkovito upravljanje s termini (sestanki, storitve, podpora) pomemben del organizacije dela. Običajno podjetja uporabljajo ročne metode (telefonski klici, e-pošta) ali preproste spletne obrazce, kar pogosto vodi v zamude, napake pri vnosu in nezadovoljstvo strank. Hkrati narašča potreba po avtomatizaciji ponavljajočih se opravil, da se zaposleni lahko posvetijo zahtevnejšim nalogam.

Umetna inteligenca (UI) in sodobna orodja za avtomatizacijo ponujajo rešitve, ki lahko te procese poenostavijo (Russel, 2021). Klepetalni roboti, ki jih poganjajo napredni pogovorni tokovi, omogočajo naravno komunikacijo med uporabnikom in sistemom. V kombinaciji z orkestracijskimi orodji, kot je n8n, in oblačnimi storitvami, kot je Google Koledar, je mogoče zgraditi celovit sistem za samodejno rezervacijo terminov.

V diplomskem delu se osredotočamo na razvoj takšnega sistema za podjetje, ki želi svojim strankam omogočiti preprosto in hitro naročanje storitev preko spletne strani. Problem, ki ga rešujemo, je odsotnost avtomatizirane rešitve, ki bi brez človeškega posredovanja obravnavala zahtevke za rezervacijo in preklic terminov ter jih usklajevala z internim koledarjem podjetja.

1.2 Namen, cilji in osnovne trditve

Namen diplomskega dela je razviti delujoč prototip sistema za avtomatsko upravljanje terminov z uporabo klepetalnega robota, ki ga poganja UI (v obliki pravil in pogovornih tokov), ter ovrednotiti njegovo učinkovitost v primerjavi z ročnim načinom dela.

Cilji:

- Pregledati obstoječe tehnologije za izgradnjo klepetalnih robotov, avtomatizacijo in integracijo s koledarskimi storitvami.
- Načrtovati arhitekturo sistema, ki vključuje spletno stran z vgrajenim klepetalnim robotom, backend API in orkestracijsko plast (n8n).
- Implementirati klepetalnega robota v orodju Botpress, ki vodi uporabnika skozi postopek rezervacije in preklica termina.

- Razviti backend API v Node.js, ki sprejema zahtevke iz klepetalnega robota in jih posreduje v n8n.
- V n8n zgraditi delotoka za ustvarjanje in brisanje dogodkov v Google Koledarju.
- Kontejnerizirati vse komponente z Dockerjem in jih namestiti na strežnik VPS.
- Testirati delovanje sistema z različnimi scenariji (veljavni in neveljavni vnosi) ter analizirati rezultate.

Osnovne trditve (hipoteze):

- **H1**: Uporaba klepetalnega robota in avtomatizacije skrajša čas, potreben za rezervacijo termina, v primerjavi z ročnim postopkom.

- **H2**: Sistem dosega visoko stopnjo natančnosti pri razumevanju uporabniških vnosov (datum, ura, storitev).

- **H3**: Avtomatizacija z n8n omogoča zanesljivo in sledljivo izvajanje integracij z zunanjimi storitvami (Google Koledar).

- **H4:** Integracija več storitev v avtomatiziran sistem prinaša specifična kibernetska tveganja, ki zahtevajo dodatne varnostne ukrepe.

1.3 Predpostavke in omejitve

Pri načrtovanju in izvedbi diplomskega dela smo upoštevali določene predpostavke, ki so vplivale na oblikovanje sistema, hkrati pa smo se zavedali tudi omejitev, ki so izhajale iz narave projekta, razpoložljivih virov in časovnega okvira.

Predpostavke

- **Osnovno digitalno znanje uporabnikov**

Sistem predpostavlja, da imajo uporabniki (stranke podjetja) osnovno znanje uporabe spletnega klepeta. V današnjem času je uporaba spletnih klepetalnih vmesnikov že zelo razširjena, vendar lahko nekateri uporabniki (npr. starejše osebe) potrebujejo dodatno pomoč. V prihodnje bi lahko implementirali pomožne funkcije, kot so obrazložitve korakov in možnost preklapljanja na človeškega operaterja.

- **Vzpostavljen Google Koledar s primernimi pravicami**

Projekt predpostavlja, da podjetje že uporablja Google Koledar in ima vzpostavljen koledar, v katerega lahko zapisuje termine. Prav tako predpostavlja, da ima skrbnik sistema možnost ustvariti storitveni račun (*service account*) in mu dodeliti ustrezna dovoljenja (pisanje, branje). Brez te predpostavke integracija ne bi bila izvedljiva.

- **Stabilno internetno omrežje in dostopnost zunanjih storitev**

Ker sistem uporablja več zunanjih storitev (Botpress cloud ali self-hosted, n8n, Google Calendar API), predpostavljamo, da so te storitve ves čas na voljo in da ima strežnik VPS stabilno internetno povezavo. V primeru izpada katere od storitev sistem ne more delovati, kar je pomembno za morebitno produkcijsko uporabo.

- **Skladnost s časovnim pasom in poslovnim časom podjetja**

Sistem privzema, da podjetje deluje v določenem časovnem pasu (CET/CEST) in da ima določen delovni čas, v katerem so termini na voljo. V naši implementaciji smo časovni pas nastavili ročno, delovni čas pa ni bil avtomatsko preverjan – to bi bila ena od izboljšav za naprej.

- **Uporabniki vnašajo podatke v pričakovanem formatu**

Čeprav smo implementirali validacijo na več ravneh, predpostavljamo, da uporabniki ne bodo namerno vnašali zlonamernih podatkov (npr. SQL injekcije). Za dodatno zaščito smo uporabili knjižnico Joi, ki zavrne nepravilne formate, vendar morebitnih naprednih napadov v okviru diplomske naloge nismo testirali.

Omejitve

1. **Omejeno trajanje termina**

Sistem rezervira termine s fiksnim trajanjem ene ure. Uporabnik ne more izbrati krajšega ali daljšega termina, prav tako ni možnosti rezervacije zaporednih terminov. To je posledica poenostavitve pri razvoju, saj je bil poudarek na dokazovanju koncepta avtomatizacije.

2. **Pomanjkanje naprednega upravljanja s konflikti**

Če je termin že zaseden, sistem tega ne preveri vnaprej (preden uporabnik vnese vse podatke). Trenutno bi uporabnik šele po končanem pogovoru izvedel, da termin ni prost. V prihodnje bi bilo treba implementirati preverjanje razpoložljivosti sproti in po potrebi predlagati alternativne termine.

3. **Odsotnost potrditve preko e-pošte**

   Ko uporabnik rezervira termin, dobi potrditev le znotraj klepeta. Ni avtomatskega e-poštnega obvestila, kar bi bilo v poslovnem okolju pričakovano. Prav tako ni možnosti, da bi uporabnik prejel opomnik pred terminom.

4. **Varnostni vidiki so obravnavani le osnovno**

   Čeprav smo v poglavju 5 analizirali varnostna tveganja in implementirali nekatere ukrepe (storitveni račun, validacija vnosov), nismo izvedli poglobljenega varnostnega testiranja, kot je penetracijsko testiranje ali analiza ranljivosti s specializiranimi orodji. Sistem zato ni primeren za produkcijsko okolje brez dodatnih varnostnih nadgradenj.

5. **Testiranje je potekalo v nadzorovanem okolju**

   Vsi testi so bili izvedeni na lokalnem razvojnem okolju in na testnem VPS strežniku z umetnimi podatki. Nismo testirali obnašanja sistema pri velikem številu sočasnih uporabnikov (stresni test) niti v realnem poslovnem okolju z dejanskimi strankami.

6. **Funkcionalnost preklica ni dokončana**

   Kot je opisano v poglavju 4.3.3, imamo zasnovo in delno implementacijo preklica, vendar ta del ni bil celovito testiran in integriran v uporabniški vmesnik. Uporabniki trenutno ne morejo preklicati termina preko klepeta.

7. **Jezikovna podpora**

   Pogovorni tok je zasnovan v slovenščini, vendar je vprašanje za razlog (*reason*) v angleščini, ker smo pri razvoju uporabili vnaprej pripravljene stavke. Sistem ne podpira več jezikov hkrati, kar bi bilo za mednarodna podjetja potrebno.

8. **Odvisnost od zunanjih storitev**

   Sistem je odvisen od delovanja Botpress, n8n in Google Calendar. Če katera od teh storitev spremeni svoj API ali preneha delovati, lahko pride do motenj. Za produkcijsko uporabo bi bilo smiselno razmisliti o lokalni namestitvi vseh komponent in vzpostavitvi varnostnih kopij.

1.4 Uporabljene raziskovalne metode

Pri izdelavi diplomskega dela smo uporabili kombinacijo raziskovalnih metod, ki omogočajo celovit pristop k razvoju in vrednotenju sistema. Metode smo izbrali tako, da pokrivajo tako teoretično podlago kot praktično izvedbo ter omogočajo objektivno analizo rezultatov.

- **Pregled literature in virov (*desk research*)**

Pregled literature in virov (*desk research*) je bila prva faza raziskave. Njena naloga je bila zbrati in analizirati obstoječe znanje s področja avtomatizacije, klepetalnih robotov in integracijskih orodij. Pregledali smo strokovno literaturo, uradno dokumentacijo uporabljenih orodij (Botpress, n8n, Node.js, Docker, Google Calendar API) ter obstoječe študije primerov. Ta faza nam je omogočila:

- razumevanje teoretičnih osnov avtomatizacije in orkestracije delotokov,
- seznanitev z najboljšimi praksami pri razvoju klepetalnih robotov,
- identifikacijo potencialnih izzivov pri integraciji več storitev (varnost, sledljivost, odpornost na napake),
- pripravo izhodišč za načrtovanje arhitekture in izbiro tehnologij.

Viri so vključevali uradno dokumentacijo orodij (Botpress, n.d.) (n8n, n.d.); (Node.js, n.d.); (Docker, n.d.), akademske vire (Russel, 2021) in standarde s področja kibernetske varnosti (OWASP, n.d.). Na podlagi *desk researcha* smo oblikovali prvo različico dispozicije in opredelili raziskovalna vprašanja ter hipoteze.

- **Eksperimentalni razvoj (praktična izgradnja prototipa)**

Za izdelavo delujočega prototipa smo uporabili pristop iterativnega razvoja (*iterative development*). To pomeni, da smo sistem gradili postopoma, pri čemer smo po vsaki dodani funkcionalnosti izvedli preverjanje delovanja in po potrebi prilagodili zasnovo. Razvoj je potekal v naslednjih iteracijah:

1. **Postavitev osnovne infrastrukture** – vzpostavitev Docker okolja, konfiguracija n8n in Botpress, priprava osnovnega backenda v Node.js.
2. **Implementacija rezervacijskega toka** – razvoj pogovornega toka v Botpressu, validacija podatkov, povezava z n8n in Google Koledarjem.
3. **Testiranje in optimizacija** – izvedba funkcionalnih testov, odkrivanje in odpravljanje napak, optimizacija odzivnih časov.
4. **Analiza kibernetske varnosti** – pregled varnostnih vidikov, implementacija osnovnih ukrepov in priprava priporočil.
5. **Dokumentacija in priprava na oddajo** – pisanje diplomskega dela, vključitev slik, kodnih izsekov in končnih rezultatov.

Iterativni pristop je omogočil sprotno prilagajanje in zgodnje odkrivanje težav, kar bi bilo pri linearnem (vodopadnem) razvoju težje doseči. Vsaka iteracija je trajala od tri do sedem dni in se je zaključila z delujočo različico sistema, ki smo jo lahko predstavili mentorju.

- **Testiranje**

Testiranje je potekalo na treh ravneh:

- **Funkcionalno testiranje klepetalnega robota** – preverili smo vse veje pogovornega toka (uspešna rezervacija, neveljavni vnosi, prekinitev pogovora). Uporabili smo kombinacijo ročnih testov in avtomatiziranih skriptov, ki so simulirali uporabniške vnose. Skupno smo izvedli 100 testnih zahtevkov z različnimi kombinacijami podatkov.
- **Testiranje API-jev** – z orodjem Postman smo preverili obnašanje backend končnih točk /appointments/book in /appointments/cancel. Testirali smo veljavne in neveljavne zahtevke ter preverili odzivne kode (200, 400, 404) in vsebino odgovorov.
- **Testiranje n8n delotokov** – neposredno smo klicali *webhook* URL-ja /webhook/book in /webhook/cancel ter preverili, ali se delotoka izvedeta v celoti in ali ustvarita/brišeta dogodke v Google Koledarju.

Pri testiranju smo merili odzivni čas, uspešnost obdelave in natančnost prepoznavanja napak. Zabeležili smo tudi morebitne izjeme in napake v dnevnikih, kar je pripomoglo k odpravljanju težav.

- **Analiza rezultatov**

Po zaključenem testiranju smo podatke sistematično analizirali. Analiza je vključevala:

- **Statistično obdelavo** – izračun deleža uspešno obdelanih zahtevkov (98 %), povprečnega odzivnega časa (7,5 sekunde) in primerjava s pričakovanimi vrednostmi.
- **Preverjanje hipotez** – za vsako od štirih hipotez smo ugotovili, ali so izpolnjeni pogoji za potrditev (H1, H2, H3, H4). Rezultate smo podrobno obrazložili v poglavju 7.4.
- **Kvalitativno analizo** – opisali smo vrste napak, ki so se pojavile, in njihove vzroke (npr. časovna omejitev pri klicu n8n, neveljavni formati). Prav tako smo ovrednotili ustreznost izbranih tehnologij in arhitekturnih odločitev.
- **Interpretacijo v kontekstu ciljev** – ugotovitve smo povezali z zastavljenimi cilji diplomskega dela (poglavje 1.2) in podali priporočila za morebitne izboljšave.

2 PREGLED TEHNOLOGIJ

2.1 Botpress

Botpress je odprtokodna platforma za razvoj klepetalnih robotov, ki ponuja vizualno orodje za načrtovanje pogovornih tokov (*flow*), upravljanje z naravnim jezikom (NLU) in enostavno integracijo z zunanjimi storitvami preko skriptnih akcij (Botpress, n.d.). V projektu je uporabljen za izdelavo uporabniškega vmesnika, ki vodi stranko skozi postopek naročanja.

2.2 Node.js in Express.js

Node.js je JavaScript okolje, ki omogoča izvajanje kode na strežniku (Node.js, n.d.). Skupaj z ogrodjem Express (ExpressJS, n.d.) smo zgradili REST API, ki sprejema zahtevke iz Botpressa in jih posreduje v n8n. Uporabljene so knjižnice za validacijo (Joi) in logiranje (winston).

2.3 n8n

n8n je orodje za orkestracijo delotokov (*workflow automation*), ki omogoča povezovanje različnih storitev preko vozlišč (*nodes*) (n8n, n.d.). V sistemu ima vlogo posrednika med backendom in Google Koledarjem. Delotoka za rezervacijo in preklic sta implementirana z uporabo HTTP Request vozlišč za klicanje lastnega API-ja, ki dejansko komunicira z Google Koledarjem.

2.4 Google Calendar API

Google Koledar nudi REST API za upravljanje z dogodki (Google). V projektu uporabljamo storitveni račun (*service account*), ki ima pooblastila za pisanje v določen koledar. Backend (Node.js) prejme ukaz iz n8n in z JWT žetonom kliče Google API za ustvarjanje, iskanje ali brisanje dogodkov.

2.5 Docker kontejnerizacija

Docker omogoča pakiranje aplikacij in njihovih odvisnosti v prenosljive zabojnike (kontejnerje, *containers*) (Docker, n.d.). Vsaka komponenta (Botpress, backend, n8n) ima svoj Docker image, kar poenostavi namestitev na VPS in zagotavlja enako okolje na različnih strežnikih.

3 ARHITEKTURA SISTEMA

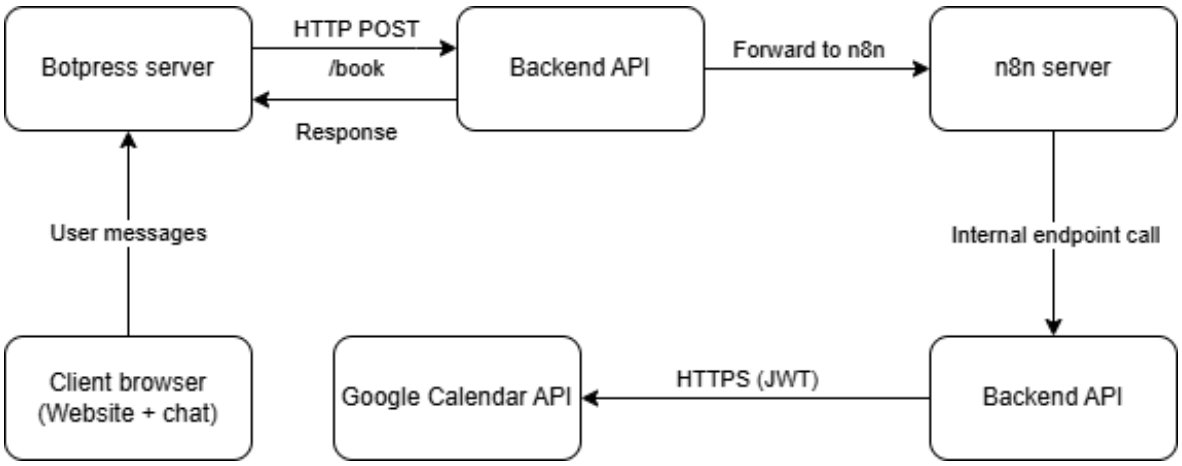
3.1 Pregled arhitekture

Sistem je zasnovan kot modularna, odzivna arhitektura, ki združuje več neodvisnih storitev. Komunikacija med njimi poteka preko HTTP protokola. Arhitekturo sestavljajo štirje glavni gradniki:

- **Spletna stran z vgrajenim klepetalnim vmesnikom** – uporabniku omogoča interakcijo s sistemom. V našem primeru je to preprosta HTML stran, ki vključuje Botpress webchat skript. Ko uporabnik odpre stran, se mu prikaže ikona klepeta, s klikom na katero se sproži pogovorno okno.
- **Botpress strežnik** – gosti pogovorno logiko. Botpress je odprtokodna platforma, ki omogoča vizualno načrtovanje pogovornih tokov. V njej so definirani vsi koraki, ki jih uporabnik preide med rezervacijo. Ko uporabnik vnese podatke, Botpress preko akcijskih vozlišč pošlje zahtevek na backend (Botpress, n.d.).
- **Backend API (Node.js)** – predstavlja osrednji del sistema. Njegove naloge so sprejemanje zahtevkov iz Botpressa, validacija vhodnih podatkov, komunikacija z n8n in upravljanje z Google Koledarjem. API je izpostavljen preko REST vmesnika na končnih točkah /api/appointments/book in /api/appointments/cancel (Node.js, n.d.).
- **n8n** – orkestracijsko orodje, ki izvaja avtomatizacijske delotoke. V njem sta definirana dva delotoka: book-appointment in cancel-appointment. n8n deluje kot posrednik med backendom in Google Koledarjem, saj prevzame podatke iz backenda, jih obdela in preko HTTP zahtevkov kliče backendove interne končne točke za delo s koledarjem (n8n, n.d.).
- **Google Koledar** – zunanja storitev, ki dejansko hrani podatke o terminih. Backend z njo komunicira preko Google Calendar API, pri čemer uporablja storitveni račun za avtentikacijo (Google).

  Vsi gradniki (razen Google Koledarja) so zapakirani v Docker posode in se izvajajo na VPS strežniku. Komunikacija med njimi poteka preko HTTP znotraj Docker omrežja, kar zagotavlja varnost in izolacijo.

Komunikacija med komponentami poteka preko HTTP protokola (Fielding, 1999)**. Slika 1 prikazuje arhitekturni diagram.**

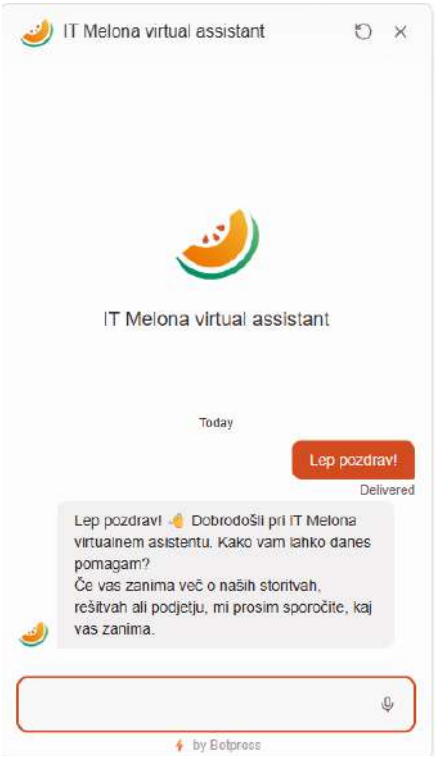


Slika 1: Arhitekturni diagram

Vir: Lasten.

3.2 Komponente sistema

Spletna stran je enostavna HTML stran z vključenim skriptom za Botpress webchat. Ob kliku na ikono klepeta se odpre okno, kjer uporabnik komunicira z robotom, kaj je razvidno na sliki.



Slika 2: Pozdravno sporočilo

Vir: Lasten.

Botpress vsebuje pogovorne tokove, ki uporabnika vodijo skozi vnos podatkov (ime, datum, ura, storitev). Ko so podatki zbrani, se preko HTTP zahtevka (*action*) pošljejo na backend (Botpress, n.d.).

Backend API izpostavlja končni točki /appointments/book in /appointments/cancel. Prejme podatke, jih validira in nato preko HTTP klica sproži ustrezni n8n delotok (na n8n-ovem webhooku). Backend je implementiran v Node.js okolju (Node.js, n.d.) z uporabo ogrodja Express (ExpressJS, n.d.).

n8n ima dva delotoka: book-appointment in cancel-appointment, ki sta dosegljiva na /webhook/book in /webhook/cancel. n8n izvede validacijo (če ni že opravljena na backendu), pripravi podatke in preko HTTP Request vozlišč pokliče backend-ove končne točke za Google Koledar operacije (n8n, n.d.).

Google Koledar operacije (*create*, *list*, *delete*) izvaja backend preko Google API-jev. Backend uporablja storitveni račun (*service account*), katerega poverilnice so shranjene kot okoljske spremenljivke (Google).

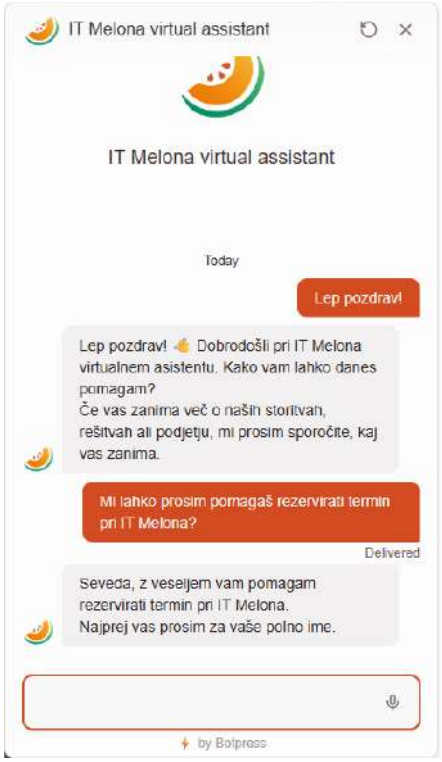
3.3 Podatkovni tok

V nadaljevanju je opisan podrobni potek podatkov pri rezervaciji termina. Vsaka številka ustreza koraku na diagramu.

1. Uporabnik na spletni strani klikne ikono klepeta in vnese pozdravno sporočilo. Botpress ga pozdravi in ponudi možnost rezervacije.
2. Botpress začne zbirati podatke. Zaporedoma vpraša za ime, e-pošto, datum, uro in razlog. Vsak vnos se sproti validira – če uporabnik vnese nepravilen format ali pretekli datum, ga robot obvesti in ponovno vpraša.
3. Ko so vsi podatki zbrani, Botpress pošlje HTTP POST zahtevek na backend API na pot /api/appointments/book. Telo zahtevka vsebuje JSON z imenom, e pošto, datumom, uro in razlogom.
4. Backend prejme zahtevek in najprej izvede validacijo s knjižnico Joi. Če validacija ne uspe, vrne napako 400 z opisom. Če je vse pravilno, generira enolični requestId in pošlje zahtevek na n8n webhook na naslov /webhook/book (konfiguriran v okoljskih spremenljivkah).

5. n8n sprejme podatke in izvede delotok. V prvem koraku delotoka se ponovno validirajo podatki (kot dodatna varnostna plast). Nato se iz podatkov oblikuje objekt dogodka, ki vključuje časovni pas, začetek in konec termina (trajanje je fiksno eno uro). Po tem n8n pošlje HTTP POST zahtevek na interno backend končno točko /api/google-calendar/create.
6. Backend prejme zahtevek in s pomočjo storitvenega računa pokliče Google Calendar API, ki ustvari dogodek v vnaprej določenem koledarju. Google vrne odgovor z ID-jem dogodka in povezavo.
7. Backend odgovor posreduje nazaj n8n.
8. n8n odgovor obdela, doda informacije o uspehu in vrne odgovor backendu.
9. Backend prejme odgovor in ga posreduje Botpressu.
10. Botpress uporabniku prikaže potrditveno sporočilo s povzetkom rezervacije.

Podoben tok velja za preklic, le da se namesto ustvarjanja dogodka najprej izvede iskanje dogodka po e-pošti in času, nato pa brisanje.



Slika 3: Vprašanje za ime in priimek

Vir: Lasten.

Na sliki 3 je prikazano, kako klepetalni robot uporabnika zaprosi za vnos imena in priimka, kar je prvi korak v procesu rezervacije.



Slika 4: Vprašanje za e-pošto

Vir: Lasten.

Slika 4 prikazuje drugi korak pogovora, kjer robot od uporabnika zahteva e-poštni naslov za morebitno komunikacijo in potrditev termina.



Slika 5: Vprašanje za datum

Vir: Lasten.

S slike 5 je razvidno, kako robot povpraša po želenem datumu rezervacije. Uporabnik lahko vnese datum v obliki, prikazani na sliki spodaj.



Slika 6: Vprašanje za določeno uro in razlog

Vir: Lasten.

Na sliki 6 sta prikazani zadnji dve vprašanji v pogovornem toku; robot povpraša po uri termina in razlogu za rezervacijo.



Slika 7: Potrditveno sporočilo

Vir: Lasten.

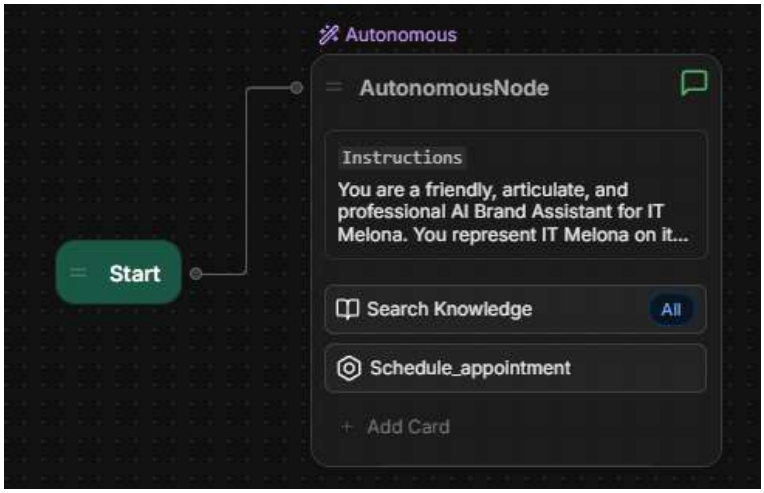
Slika 7 prikazuje končno potrditev, ki jo uporabnik prejme po uspešno zaključeni rezervaciji. Sporočilo vsebuje povzetek vnesenih podatkov.

4 IMPLEMENTACIJA

4.1 Chatbot vmesnik (Botpress)

4.1.1 Načrtovanje pogovornega toka

Pogovorni tok je zasnovan tako, da uporabnik najprej pozdravi robota, nato pa izmed možnosti izbere »Book appointment« ali »Cancel appointment«. Pri rezervaciji robot zaporedoma zbira potrebne podatke. Vsak korak vsebuje preverjanje veljavnosti vnosa (npr. datum ne sme biti v preteklosti, ura mora biti v delovnem času). Če je vnos neveljaven, robot ponovno vpraša in pojasni napako.



Slika 8: Botpress workflow

Vir: Lasten.

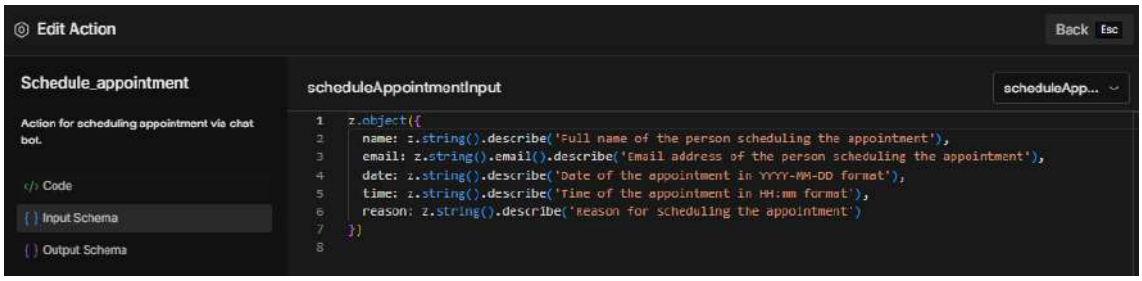
Na sliki 8 je viden vizualni potek pogovornega toka v orodju Botpress.

4.1.2 Implementacija v Botpressu

Vmesnik Botpress omogoča vizualno sestavljanje toka s pomočjo vozlišč. Uporabljena so standardna vozlišča za pošiljanje sporočil, čakalna vozlišča za vnos uporabnika ter vozlišča za izvajanje kode (*action nodes*), ki kličejo zunanji API.

4.1.3 Akcije in integracija z backendom

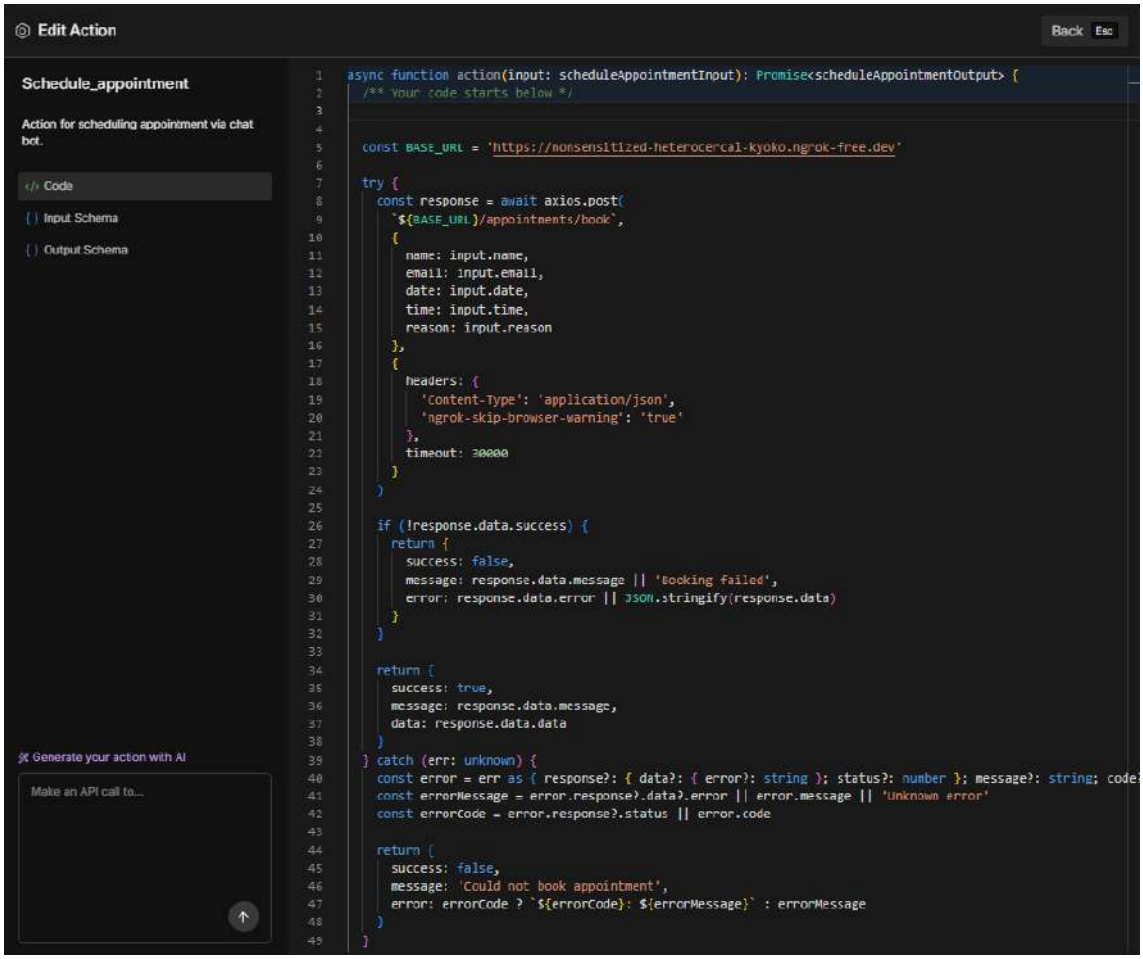
V vozlišču, kjer so zbrani vsi podatki, je dodana akcija, ki z uporabo HTTP klienta pošlje podatke na backend. Spodnji izseki prikazujejo shemo vhodnih podatkov in kodo akcije.



Slika 9: Booking input shema

Vir: Lasten.

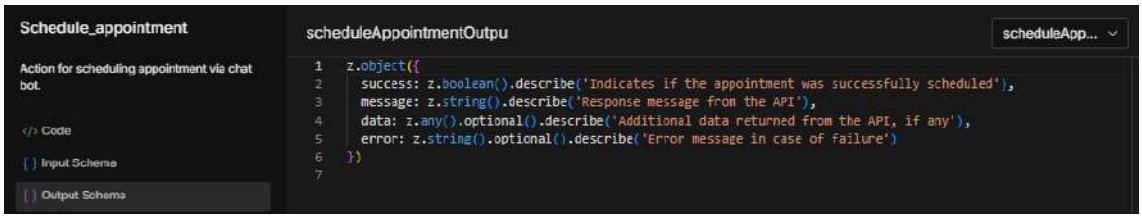
S slike 9 je razvidna shema vhodnih podatkov, ki jih Botpress pošlje na backend ob rezervaciji. Shema določa obvezna polja (ime, e-pošta, datum, ura, razlog).



Slika 10: Booking action funkcija

Vir: Lasten.

Na sliki 10 je prikazana JavaScript koda akcije v Botpressu, ki zbere podatke in jih preko HTTP zahtevka pošlje na backend API.



Slika 11: Booking output shema

Vir: Lasten.

Slika 11 prikazuje shemo izhodnih podatkov, ki jih Botpress pričakuje od backenda kot odgovor na rezervacijo.

4.2 Backend API

4.2.1 Struktura projekta

Backend je organiziran v tipično Node.js strukturo, ki sledi načelu ločevanja odgovornosti (*separation of concerns*). Vsaka mapa ima jasno določeno vlogo, kar omogoča enostavno vzdrževanje, testiranje in morebitno kasnejšo nadgradnjo. Struktura projekta, kot je dostopna v repozitoriju chatbot-appointment-system, je naslednja:

- `app.js/`
- `routes/` – definicije poti (appointmentsRoutes.js).
- `controllers/` – poslovna logika (bookingController.js).
- `services/` – klici v Google Koledar (googleCalendarService.js).
- `validators/` – Joi sheme za validacijo.
- `config/` – okoljske spremenljivke.

V nadaljevanju so podrobno opisane posamezne datoteke z izvorno kodo, kot je implementirana v repozitoriju.

app.js – vstopna točka in nastavitev Express

Ta datoteka služi kot glavna vstopna točka aplikacije. Tukaj se inicializira Express strežnik, nastavijo se globalni middleware-ji, definirajo se osnovne nastavitve (npr. parsiranje JSON-a) in registrirajo se rute. Prav tako se tukaj zažene strežnik na določenih vratih.

```
// backend/app.js
const express = require('express');
const dotenv = require('dotenv');
const appointmentRoutes = require('./routes/appointmentsRoutes');
const { errorHandler } = require('./middleware/errorHandler');

dotenv.config();

const app = express();
const PORT = process.env.PORT || 3000;

// Middleware
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// Routes
app.use('/api/appointments', appointmentRoutes);

// Health check endpoint
app.get('/health', (req, res) => {
res.status(200).json({ status: 'OK', timestamp: new Date() });
});

// Error handling middleware
app.use(errorHandler);

app.listen(PORT, () => {
console.log(`Server running on port ${PORT}`);
});
```

Vidimo, da uporabljamo knjižnico dotenv za nalaganje okoljskih spremenljivk iz datoteke .env. Z middleware express.json() omogočimo sprejemanje JSON podatkov v zahtevkih. Definiramo tudi preprost health check endpoint /health, ki ga uporabljamo za preverjanje delovanja strežnika (uporabljen tudi pri testiranju z ngrok). Na koncu zaženemo strežnik na podanih vratih.

routes/ – definicije poti

Mapa routes vsebuje datoteke, ki definirajo vse končne točke API-ja. V našem primeru imamo appointmentsRoutes.js, ki vsebuje poti za rezervacijo in preklic.

```
// backend/routes/appointmentsRoutes.js
const express = require('express');
const router = express.Router();
const bookingController = require('../controllers/bookingController');
const appointmentValidator = require('../validators/appointmentValidator');
```

```
router.post('/book', appointmentValidator.validateBooking,
bookingController.bookAppointment);
router.post('/cancel', appointmentValidator.validateCancellation,
bookingController.cancelAppointment);

module.exports = router;
```

Kot vidimo, rute uporabljajo kontrolerje za izvedbo poslovne logike in validatorje za preverjanje vhodnih podatkov. To omogoča, da so rute zelo kratke in jasne – vsaka ruta le poveže pot z ustreznim validatorjem in kontrolerjem.

controllers/ – poslovna logika

Kontrolerji vsebujejo glavno logiko obdelave zahtevkov. V bookingController.js se nahajata funkciji bookAppointment in cancelAppointment. Vsaka funkcija prejme objekt req in res, iz njih izlušči podatke, pokliče storitve (*services*) in vrne ustrezen odgovor. Spodaj je izvorna koda funkcije bookAppointment iz repozitorija:

```
// backend/controllers/bookingController.js
const axios = require('axios');
const { v4: uuidv4 } = require('uuid');
const config = require('../config');


exports.bookAppointment = async (req, res, next) => {
  const { name, email, date, time, reason } = req.body;
  const requestId = uuidv4();


  try {
    // Prepare payload for n8n
    const payload = {
      name,
      email,
      date,
      time,
      reason,
      requestId,
      timestamp: new Date().toISOString()
    };
```

```
    // Call n8n webhook
    const n8nResponse = await axios.post(config.n8n.bookWebhook, payload, {
      timeout: 30000, // 30 seconds timeout
      headers: { 'Content-Type': 'application/json' }
    });

    // Return success response
    res.status(200).json({
      success: true,
      message: 'Appointment booked successfully',
      requestId,
      eventId: n8nResponse.data.eventId,
      eventLink: n8nResponse.data.eventLink,
      bookingData: { name, email, date, time, reason }
    });
  } catch (error) {
    // If n8n returns a validation error (4xx), forward it
    if (error.response) {
      return res.status(error.response.status).json(error.response.data);
    }
    // Otherwise, pass to global error handler
    next(error);
  }
};
```

Funkcija najprej generira enolični identifikator requestId za sledljivost. Nato pošlje podatke na n8n webhook (URL je shranjen v konfiguraciji). Če je klic uspešen, vrne podatke o dogodku; če n8n vrne napako (npr. validacija), jo posreduje naprej; ob morebitni napaki na strežniku pa se izvedba prenese na globalni handler napak.

Funkcija za preklic je implementirana podobno:

```
exports.cancelAppointment = async (req, res, next) => {
  const { email, date, time } = req.body;
```

```
  const requestId = uuidv4();

  try {
    const payload = { email, date, time, requestId };
    const n8nResponse = await axios.post(config.n8n.cancelWebhook, payload, {
      timeout: 30000
    });

    res.status(200).json({
      success: true,
      message: 'Appointment cancelled successfully',
      requestId,
      cancelledEventId: n8nResponse.data.deletedEventId
    });
  } catch (error) {
    if (error.response) {
      return res.status(error.response.status).json(error.response.data);
    }
    next(error);
  }
};
```

services/ – klici zunanjih storitev

Storitvena plast (*services*) vsebuje funkcije, ki komunicirajo z zunanjimi API-ji. V googleCalendarService.js so implementirane operacije za Google Koledar: ustvarjanje, iskanje in brisanje dogodkov. Ta plast uporablja knjižnico googleapis za avtentikacijo in izvajanje klicev. Spodaj je izvorna koda funkcije za ustvarjanje dogodka:

```
// backend/services/googleCalendarService.js
const { google } = require('googleapis');
const fs = require('fs');
const path = require('path');
```

```
const config = require('../config');

// Initialize Google Auth client
let authClient = null;

const getAuthClient = async () => {
  if (authClient) return authClient;

  try {
    const keyFile = config.google.serviceAccountKeyPath;
    const keyPath = path.resolve(__dirname, '../../', keyFile);

    const auth = new google.auth.GoogleAuth({
      keyFile: keyPath,
      scopes: ['https://www.googleapis.com/auth/calendar']
    });

    authClient = await auth.getClient();
    return authClient;
  } catch (error) {
    console.error('Error initializing Google Auth:', error);
    throw error;
  }
};

exports.createEvent = async (eventData, calendarId = config.google.calendarId) => {
  const auth = await getAuthClient();
  const calendar = google.calendar({ version: 'v3', auth });

  try {
    const response = await calendar.events.insert({
      calendarId,
      requestBody: eventData
    });

    return {
      success: true,
      event: {
        id: response.data.id,
        htmlLink: response.data.htmlLink,
        summary: response.data.summary,
        start: response.data.start,
        end: response.data.end
      }
    };
  } catch (error) {
    console.error('Google Calendar create error:', error.message);
    return { success: false, error: error.message };
  }
};
```

Funkcija createEvent najprej pridobi avtenticiranega klienta preko storitvenega računa (keyFile), nato kliče Google Calendar API za vstavljanje dogodka. Ob uspehu vrne podrobnosti o dogodku, ob napaki pa sporočilo o napaki.

Podobno sta implementirani funkciji searchEvents in deleteEvent:

```
exports.searchEvents = async (calendarId, timeMin, timeMax) => {
  const auth = await getAuthClient();
  const calendar = google.calendar({ version: 'v3', auth });

  try {
    const response = await calendar.events.list({
      calendarId,
      timeMin: timeMin.toISOString(),
      timeMax: timeMax.toISOString(),
      singleEvents: true,
      orderBy: 'startTime'
    });

    return {
      success: true,
      events: response.data.items
    };
  } catch (error) {
    console.error('Google Calendar search error:', error.message);
    return { success: false, error: error.message };
  }
};

exports.deleteEvent = async (eventId, calendarId = config.google.calendarId) => {
  const auth = await getAuthClient();
  const calendar = google.calendar({ version: 'v3', auth });

  try {
    await calendar.events.delete({
      calendarId,
      eventId
    });
    return { success: true, message: 'Event deleted successfully' };
  } catch (error) {
    console.error('Google Calendar delete error:', error.message);
    return { success: false, error: error.message };
  }
};
```

validators/ – Joi sheme za validacijo

Validacija vhodnih podatkov je ključna za varnost in pravilno delovanje sistema. Uporabljamo knjižnico Joi, ki omogoča enostavno definiranje shem in preverjanje. V appointmentValidator.js definiramo shemi za rezervacijo in preklic ter middleware funkciji, ki izvedeta validacijo in vrneta napako, če podatki niso skladni.

```
// backend/validators/appointmentValidator.js
const Joi = require('joi');

const bookingSchema = Joi.object({
  name: Joi.string().min(2).max(100).required(),
  email: Joi.string().email().required(),
  date: Joi.date().iso().greater('now').required(),
  time: Joi.string().pattern(/^([01]\d|2[0-3]):([0-5]\d)$/).required(),
  reason: Joi.string().min(3).max(500).required()
});

const cancellationSchema = Joi.object({
  email: Joi.string().email().required(),
  date: Joi.date().iso().required(),
  time: Joi.string().pattern(/^([01]\d|2[0-3]):([0-5]\d)$/).required()
});

exports.validateBooking = (req, res, next) => {
  const { error } = bookingSchema.validate(req.body);
  if (error) {
    return res.status(400).json({
      success: false,
      message: 'Validation error',
      details: error.details.map(d => d.message)
    });
  }
  next();
};
```

```
exports.validateCancellation = (req, res, next) => {
  const { error } = cancellationSchema.validate(req.body);
  if (error) {
    return res.status(400).json({
      success: false,
      message: 'Validation error',
      details: error.details.map(d => d.message)
    });
  }
  next();
};
```

V shemi za rezervacijo preverimo, da je ime dolgo vsaj dva znaka, e-pošta veljavna, datum v prihodnosti (.greater('now')), ura v formatu HH:MM in ima razlog vsaj tri znake. Če validacija ne uspe, vrnemo status 400 s podrobnim sporočilom o napaki. V nasprotnem primeru se izvedba nadaljuje z naslednjim middleware-jem (kontrolerjem). Podobno velja za preklic, le da tukaj ne preverjamo imena in razloga, datum pa je lahko tudi v preteklosti (za preklic obstoječega termina).

config/ – okoljske spremenljivke

Mapa config vsebuje datoteko index.js, ki združuje vse okoljske spremenljivke in jih izpostavi kot enoten objekt. To omogoča centraliziran dostop do nastavitev in preprečuje, da bi bile v kodi razpršene direktne reference na process.env.

```
// backend/config/index.js
require('dotenv').config();

module.exports = {
  port: process.env.PORT || 3000,
  n8n: {
    bookWebhook: process.env.N8N_BOOK_WEBHOOK,
    cancelWebhook: process.env.N8N_CANCEL_WEBHOOK
  },
```

```
  google: {
    serviceAccountKeyPath: process.env.GOOGLE_SERVICE_ACCOUNT_KEY_PATH,
    calendarId: process.env.GOOGLE_CALENDAR_ID
  }
};
```

Tako lahko v drugih delih aplikacije uporabimo const config = require('./config') in dostopamo do nastavitev preko objekta config. To poenostavi morebitne spremembe in izboljša berljivost kode.

.env.example datoteka

Okoljske spremenljivke so shranjene v datoteki .env, ki ni vključena v repozitorij (dodana v .gitignore). V repozitoriju je priložena datoteka .env.example za vzorčne nastavitve:

```
# backend/.env.example
PORT=3000
N8N_BOOK_WEBHOOK=http://n8n:5678/webhook/book
N8N_CANCEL_WEBHOOK=http://n8n:5678/webhook/cancel
GOOGLE_SERVICE_ACCOUNT_KEY_PATH=./credentials/google-service-account.json
GOOGLE_CALENDAR_ID=primary
```

V produkciji na VPS so te vrednosti nastavljene kot sistemske spremenljivke ali pa se posredujejo preko Docker Compose datoteke. Na ta način zagotovimo, da občutljivi podatki (kot so poti do ključev) niso trdo kodirani.

Zaključek o strukturi

Takšna organizacija kode omogoča jasno ločevanje odgovornosti: rute definirajo vhodne točke, validatorji skrbijo za preverjanje podatkov, kontrolerji izvajajo glavno logiko, storitve komunicirajo z zunanjimi API-ji, konfiguracija pa centralizira nastavitve. Ta pristop je v skladu z najboljšimi praksami razvoja v Node.js in olajša testiranje, vzdrževanje ter morebitno kasnejšo razširitev sistema.

4.2.2 Endpointi za rezervacijo in preklic

```
import express from 'express';
import {
  bookAppointment,
  cancelAppointment,
} from '../controllers/appointmentController.js';

const router = express.Router();

/**
 * Book appointment endpoint
 * POST /appointments/book
 */
router.post('/book', bookAppointment);

/**
 * Cancel appointment endpoint
 * POST /appointments/cancel
 */
router.post('/cancel', cancelAppointment);

export default router;
```

Slika 12: Backend poti

Vir: Lasten.

Na sliki 12 so prikazane definirane poti (*routes*) v backend API-ju, vključno z /appointments/book in /appointments/cancel.

Poti:

- `POST /appointments/book` – sprejme `{ name, email, date, time, reason }` in po klicu n8n vrne odgovor.

- `POST /appointments/cancel` – sprejme `{ email, date, time }`, poišče in prekliče termin.

4.2.3 Validacija podatkov

Preden se podatki pošljejo v n8n, se izvede validacija z Joi. Če validacija ne uspe, API vrne 400 z opisom napak. Spodnja slika prikazuje krmilnik za rezervacijo.



```
 * Handles appointment booking request
 * @param {object} req - Express request object
 * @param {object} res - Express response object
 */
export async function bookAppointment(req, res) {
  try {
    // Validate input
    const validation = validateBookingRequest(req.body);

    if (!validation.valid) {
      return res.status(400).json({
        success: false,
        error: 'Validation failed',
        details: validation.errors,
      });
    }

    // Prepare booking data
    const bookingData = {
      name: req.body.name.trim(),
      email: req.body.email.trim().toLowerCase(),
      date: req.body.date,
      time: req.body.time,
      reason: req.body.reason.trim(),
    };

    // Forward to n8n
    const n8nResponse = await sendBookingToN8n(bookingData);

    // Return success response with n8n result
    res.status(200).json({
      success: true,
      message: 'Appointment booking request processed successfully',
      data: bookingData,
      n8nResult: n8nResponse,
    });
  } catch (error) {
    // Log error for debugging (in production, use proper logging)
    console.error('Error booking appointment:', error.message);

    // Return error response
    res.status(500).json({
      success: false,
      error: 'Failed to process booking request',
      message: error.message,
    });
  }
```

Slika 13: Status po validaciji

Vir: Lasten.

S slike 13 je razvidna validacija podatkov v backend krmilniku. V primeru neveljavnih podatkov API vrne status 400 s seznamom napak.

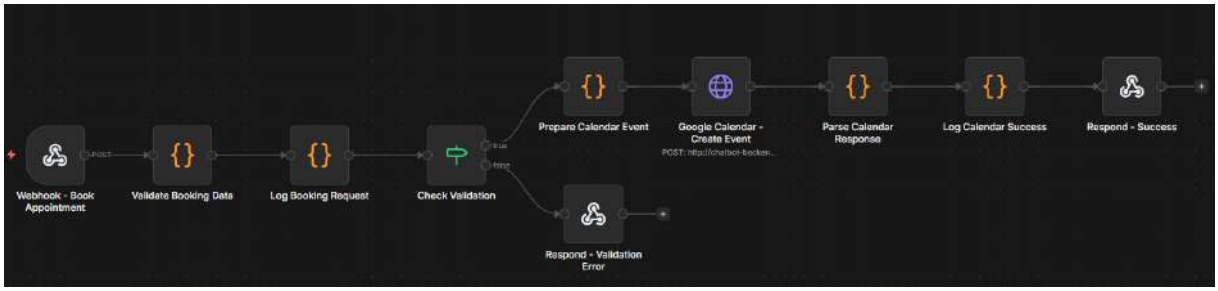
4.2.4 Integracija z n8n

Backend pošlje podatke na n8n preko HTTP klica. URL-ji so konfigurirani v okoljskih spremenljivkah. Uporabljena je knjižnica `axios`.

4.3 Avtomatizacija z n8n

4.3.1 Pregled delotokov

Celoten nabor vozlišč in njihovih funkcij je podrobno opisan spodaj. Tukaj so povzeti ključni deli.



Slika 14: N8n vozlišča in workflow

Vir: Lasten.

Na sliki 14 so prikazana vozlišča (*nodes*) v n8n delotoku za rezervacijo, vključno z Webhook triggerjem, validacijo in HTTP klicem na backend.

4.3.2 Delotok za rezervacijo

Delotok se sproži preko Webhook vozlišča na poti `/webhook/book`. Nato sledijo:

- Code vozlišče za validacijo (če ni bila že opravljena).
- IF vozlišče za preusmeritev ob neuspešni validaciji.
- Code za pripravo podatkov za Google Koledar (določitev časovnega okna, *timezone*).
- HTTP Request za klic backendovega `/api/google-calendar/create`.
- Code za obdelavo odgovora in pripravo odziva.
- Respond to Webhook za vrnitev odgovora klicatelju.

4.3.3 Delotok za preklic

Poleg rezervacije je bila zasnovana tudi funkcionalnost za preklic termina. Na backendu so pripravljene končne točke za iskanje (/api/google-calendar/search) in brisanje (/api/google-calendar/delete/{eventId}) dogodkov, ki jih upravlja storitveni račun. Prav tako je v n8n pripravljen delotok na poti /webhook/cancel, ki vsebuje vozlišča za validacijo, iskanje, filtriranje in brisanje (podrobneje opisano v Prilogi A).

Kljub pripravljeni kodi in delotoku ta funkcionalnost ni bila v celoti testirana in integrirana v končni sistem. Razlogi so časovne omejitve in odločitev, da se prioritizira stabilnost in zanesljivost osnovne funkcionalnosti (rezervacija). Zasnova ostaja dokumentirana in predstavlja izhodišče za morebitne nadgradnje, o čemer razpravljam v poglavju 6.4.

4.3.4 Opis vozlišč

Vsak delotok v n8n je sestavljen iz zaporednih vozlišč (*nodes*), ki izvajajo specifične naloge. V nadaljevanju so podrobno opisana vsa uporabljena vozlišča za oba delotoka – rezervacijo in preklic. Opisi vključujejo namen, konfiguracijo in (kjer je primerno) izvorno kodo, uporabljeno v vozliščih tipa Code.

Vozlišča delotoka za rezervacijo

1. Webhook – Book appointment

Vozlišče tipa Webhook deluje kot sprožilec delotoka. Konfigurirano je na poti /webhook/book in sprejema POST zahtevke. Ko backend API pošlje podatke na ta naslov, se delotok zažene. Vhodni podatki so v JSON obliki in vsebujejo name, email, date, time, reason. V n8n so ti podatki dostopni preko $input.item.json.body.

2. Validate booking data

To je vozlišče tipa *Code*, ki vsebuje JavaScript logiko za validacijo. Preveri prisotnost vseh polj, pravilnost formata e-pošte (z regularnim izrazom) in ali datum ni v preteklosti. Če je validacija uspešna, vrne objekt { valid: true, data: {...}, requestId, timestamp }, v nasprotnem primeru pa { valid: false, errors: [...] }. Validacija se izvede že v n8n, čeprav se izvaja tudi v backendu – to zagotavlja dodatno raven varnosti in omogoča neodvisno testiranje delotoka.

3. Log Booking Request

Code vozlišče, ki služi beleženju (*debugging* in revizija). Izpiše strukturiran dnevniški zapis o prejetem zahtevku (timestamp, requestId, veljavnost) in podatke posreduje naprej nespremenjene.

4. Check validation

Vozlišče tipa *IF* preveri vrednost valid iz prejšnjega koraka. Če je true, se izvedba nadaljuje po veji za pripravo koledarskega dogodka; če je false, se izvedba preusmeri na vozlišče, ki vrne napako 400.

5. Prepare calendar event

Code vozlišče, ki iz validiranih podatkov sestavi objekt, primeren za Google Calendar API, določi časovni pas (npr. Europe/Berlin), začetek dogodka kot kombinacijo datuma in ure, konec pa kot začetek + 1 uro. V primeru, da je ura 23:00, samodejno poskrbi za prehod na naslednji dan. Prav tako oblikuje povzetek (*summary*) in opis (*description*), ki vključujeta ime uporabnika, razlog in e-poštni naslov. Izhod tega vozlišča je objekt z dodanim poljem eventData.

6. Google calendar – Create event

Vozlišče tipa *HTTP Request* pošlje zahtevek POST na interno končno točko backenda http://chatbot-backend:3000/api/google-calendar/create. Telo zahtevka vsebuje eventData in morebitne dodatne parametre (npr. calendarId). Backend nato s storitvenim računom ustvari dogodek in vrne odgovor (npr. { success: true, event: { id, htmlLink, summary, start, end } }).

7. Parse calendar response

Code vozlišče interpretira odgovor backenda. Če je success: false, sproži napako (s čimer se delotok konča). Ob uspehu združi podatke o dogodku z izvirnimi podatki rezervacije (npr. bookingData). Izhod je en sam objekt, ki vsebuje vse potrebno za končni odziv.

8. Log calendar success

Code vozlišče za beleženje uspešnega ustvarjanja dogodka. V dnevnik zapiše timestamp, requestId, eventId in povezavo do dogodka. Nato pripravi odgovor za klicatelja:

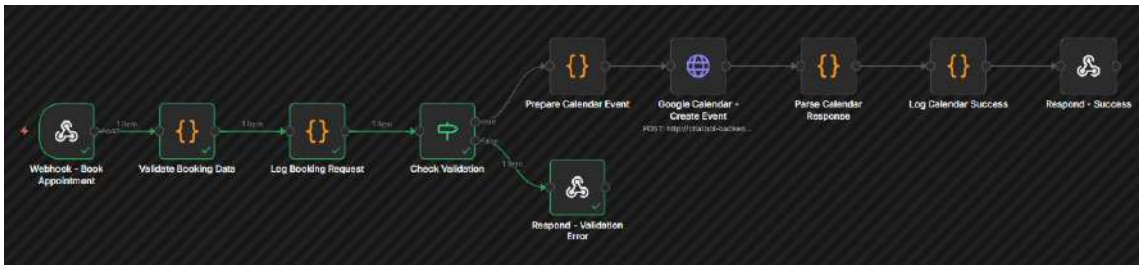
{ success: true, message: 'Appointment booked successfully', eventId, eventLink, bookingData, requestId }

9. Respond – success

Vozlišče *Respond to Webhook* pošlje pripravljeni JSON odgovor nazaj na klicatelja (backend) s statusno kodo 200.

10. Respond – validation error

To vozlišče se aktivira, ko validacija ne uspe. Vrne JSON s success: false, seznamom napak (*details*) in statusno kodo 400.

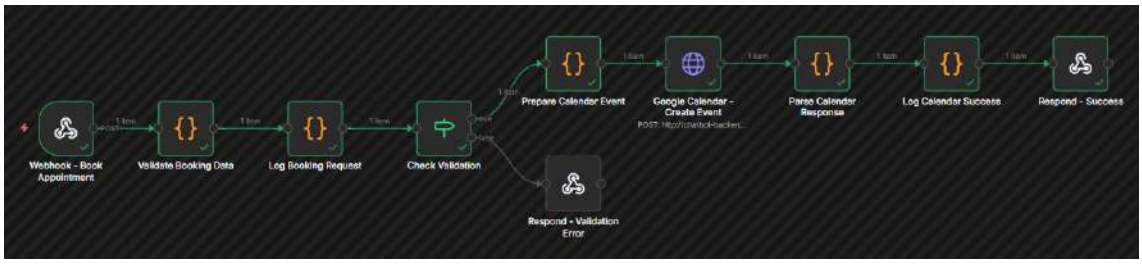


Slika 15: Neuspešna izvedba delovnega toka

Vir: Lasten.

Slika 15 prikazuje neuspešno izvedbo n8n delotoka, kjer je validacija odkrila napako in vrnila status 400 s seznamom napak.

Uspešna izvedba delotoka pa vrne podrobnosti o ustvarjenem dogodku.



Slika 16: Uspešna izvedba delovnega toka

Vir: Lasten.

Na sliki 16 je razvidna uspešna izvedba n8n delotoka, ki je ustvarila dogodek v Google Koledarju in vrnila podrobnosti o dogodku.

Vozlišča delotoka za preklic

1. Webhook – cancel appointment

Sprožilec na poti /webhook/cancel, ki sprejme { email, date, time }.

2. Validate cancellation data

Code vozlišče preveri prisotnost e-pošte, datuma in ure ter pravilnost formata e-pošte. Vrne valid in morebitne napake.

3. Log cancellation event

Beleži prejeti zahtevek v dnevnik.

4. Check validation

IF vozlišče usmerja tok glede na rezultat validacije.

5. Prepare event search

Code vozlišče iz podanega datuma in ure izračuna časovno okno (30 minut pred in 30 minut po) za iskanje dogodkov. Pripravi objekt searchData s calendarId, timeMin in timeMax.

6. Google calendar – search events

HTTP Request vozlišče pošlje POST na backend /api/google-calendar/search s podatki iz prejšnjega koraka. Backend vrne seznam dogodkov v podanem časovnem oknu.

7. Parse search response

Code vozlišče razčleni odgovor. Če je uspešen, vsak dogodek pretvori v ločen n8n element za nadaljnjo obdelavo.

8. Filter matching event

To vozlišče implementira logiko za iskanje dogodka, ki pripada uporabniku. Primerja e-poštni naslov z vsebino opisa dogodka (ali z udeleženci, če so na voljo). Če najde ujemanje, vrne { success: true, eventId, calendarId, cancellationData, requestId }, sicer vrne { success: false, error: 'Event not found' }.

9. Check event found

IF vozlišče usmeri na brisanje, če je dogodek najden, sicer na odziv z napako 404.

10. Google calendar – delete event

HTTP Request vozlišče pošlje DELETE zahtevek na backend /api/google-calendar/delete/{eventId}.

11. Log deletion success

Code vozlišče zabeleži uspešno brisanje in oblikuje odgovor.

12. Respond success

Vrne uspešen odziv s statusom 200.

13. Respond validation error

Vrne 400 ob neveljavnih vhodnih podatkih.

14. Respond – Event not found

Vrne 404 z opisom, da dogodek ne obstaja.

Pomen podrobnega opisa vozlišča

Opisana vozlišča ponazarjajo modularnost in sledljivost, ki ju n8n prinaša v avtomatizacijo. Vsako vozlišče ima jasno določeno vlogo, kar omogoča enostavno testiranje, vzdrževanje in morebitne kasnejše spremembe. Koda v vozliščih *Code* je kratka, a ključna za prilagoditev delotoka specifičnim potrebam sistema. Celoten seznam vozlišč in njihovih podrobnosti je dodatno dokumentiran v **Prilogi A**, kjer so navedeni tudi primeri konfiguracij in vhodnih/izhodnih podatkov.

4.4 Integracija z Google Koledarjem

4.4.1 Uporaba storitvenega računa

Google Cloud projekt ima omogočen Google Koledar API in ustvarjen storitveni račun. Poverilnice (JSON datoteka) so shranjene kot okoljska spremenljivka in se uporabijo za pridobitev JWT žetona. Koledar, v katerega se zapisuje, je bil deljen s storitvenim računom z ustreznimi pravicami (pisanje).

4.4.2 Ustvarjanje in brisanje dogodkov

Backend izpostavlja tri interne končne točke (zaščitene, ker so dostopne le znotraj Docker omrežja):

- `POST /api/google-calendar/create` – ustvari dogodek.
- `POST /api/google-calendar/search` – poišče dogodke v določenem časovnem oknu.
- `DELETE /api/google-calendar/delete/:eventId` – izbriše dogodek.

Spodnja slika prikazuje del kode za ustvarjanje dogodka.

```
/**
 * POST /api/google-calendar/create
 * Create a calendar event
 */
router.post('/create', async (req, res) => {
  try {
    const { eventData, calendarId } = req.body;

    if (!eventData || !eventData.summary || !eventData.start || !eventData.end) {
      return res.status(400).json({
        success: false,
        error: 'Missing required fields: eventData.summary, eventData.start, eventData.end'
      });
    }

    // Remove attendees field if present - service accounts cannot invite attendees
    // without Domain-Wide Delegation, so we strip this field to avoid 403 errors
    const { attendees, ...eventDataWithoutAttendees } = eventData;
    if (attendees) {
      console.warn('Removed attendees field from event data (service accounts cannot invite attendees)');
    }

    const event = await createCalendarEvent(eventDataWithoutAttendees, calendarId);

    res.json({
      success: true,
      event: {
        id: event.id,
        htmlLink: event.htmlLink,
        summary: event.summary,
        start: event.start,
        end: event.end
      }
    });
  } catch (error) {
    console.error('Error creating calendar event:', error);
    res.status(500).json({
      success: false,
      error: error.message
    });
  }
});
```

Slika 17: »Create appointment« endpoint

Vir: Lasten.

S slike 17 je razvidna implementacija končne točke /api/google-calendar/create v backendu, ki kliče Google API za ustvarjanje dogodka.

5 ANALIZA KIBERNETSKE VARNOSTI SISTEMA

5.1 Uvod v kibernetsko varnost avtomatiziranih sistemov

V današnjem času, ko vse več podjetij uvaja avtomatizacijo in storitve v oblaku, postaja kibernetska varnost eden ključnih dejavnikov uspeha. Sistemi, ki povezujejo več različnih storitev (kot so Botpress, n8n, Google Koledar in lastni backend), so še posebej izpostavljeni, saj napadalec lahko poskuša vdreti preko šibkega člena v verigi.

V tem poglavju analiziramo varnostne vidike razvitega sistema za avtomatsko rezervacijo terminov. Obravnavamo potencialna tveganja, implementirane varnostne mehanizme ter podajamo priporočila za izboljšave. Poglavje neposredno naslavlja četrto hipotezo (H4), ki smo jo postavili v uvodu.

5.2 Identifikacija potencialnih varnostnih tveganj

Na podlagi analize arhitekture sistema (poglavje 3) in implementacije (poglavje 4) smo identificirali naslednja tveganja:

5.2.1 Tveganja, povezana z Botpressom

Nezavarovan admin vmesnik: Botpress privzeto ne zahteva avtentikacije za dostop do administracijskega vmesnika, če ni posebej konfiguriran.

Prenos podatkov: Če spletna stran ne uporablja HTTPS, bi lahko napadalec prestregel pogovor med uporabnikom in chatbotom.

Shranjevanje zgodovine pogovorov: Botpress lahko hrani zgodovino pogovorov, kar predstavlja tveganje za zasebnost uporabnikov.

5.2.2 Tveganja, povezana z backend API-jem

Nepooblaščen dostop do končnih točk: API (npr. /appointments/book) je javno dostopen, ker ga kliče Botpress. Brez dodatne avtentikacije bi ga lahko kdorkoli zlorabil za pošiljanje lažnih zahtevkov.

Injekcijski napadi: Kljub validaciji vnosov z Joi obstaja tveganje, da bi napadalec poskušal vnesti zlonamerno kodo (npr. v polje name ali reason), če validacija ni dovolj stroga.

Nadzor dostopa (*Rate limiting*): API nima omejitve števila zahtevkov, kar omogoča potencialne DDoS napade ali pošiljanje ogromnega števila lažnih rezervacij.

5.2.3 Tveganja, povezana z n8n

Nezaščiteni webhook URLji: n8n delotoka sta dosegljiva na /webhook/book in /webhook/cancel brez kakršnekoli avtentikacije. Kdor pozna URL, lahko sproži delotok.

Dostop do n8n uporabniškega vmesnika: Če je n8n izpostavljen javnemu internetu brez gesla, bi lahko napadalec spremenil delotoke ali pridobil občutljive podatke.

5.2.4 Tveganja, povezana z Google Koledar API-jem

Prestrezanje API ključev: Čeprav uporabljamo storitveni račun, so njegove poverilnice shranjene v okoljskih spremenljivkah. Če bi napadalec pridobil dostop do strežnika, bi lahko te poverilnice ukradel.

Prekomerna dovoljenja: Storitveni račun ima pravico pisanja v koledar. Če bi ga napadalec ogrozil, bi lahko brisal ali spreminjal vse dogodke.

5.2.5 Tveganja, povezana z Docker okoljem

Varnost zabojnikov: Če zabojniki tečejo s korenskimi pravicami (*root*), bi morebitni napadalec, ki bi pridobil dostop do enega zabojnika, lahko ogrozil celoten strežnik.

Omrežna izolacija: Vsi zabojniki so v istem Docker omrežju, kar pomeni, da če napadalec ogrozi enega, lahko napada druge znotraj istega omrežja.

5.3 Implementirani varnostni ukrepi

Kljub identificiranim tveganjem smo v sistem vključili nekatere osnovne varnostne mehanizme:

- Uporaba storitvenega računa za Google Koledar: Namesto uporabe osebnih poverilnic (OAuth 2.0 z dostopnim in osvežitvenim žetonom) uporabljamo storitveni račun z omejenimi dovoljenji. To pomeni, da ni tveganja, da bi izgubili dostop, če se spremeni geslo uporabnika, in lažje nadzorujemo, do česa ima račun dostop.

- Validacija vnosov na backendu: Vsa vnosna polja so obvezno preverjena s knjižnico Joi. S tem preprečujemo osnovne injekcijske napade in zagotavljamo, da so podatki v pričakovanem formatu (npr. e-mail, datum, ura).
- Shranjevanje občutljivih podatkov v okoljske spremenljivke: Google API poverilnice, n8n URLji in drugi občutljivi podatki niso zapisani v kodi, ampak se berejo iz .env datoteke oziroma okoljskih spremenljivk. To preprečuje, da bi se slučajno znašli v javnem repozitoriju.
- Ločitev okolij (razvoj vs. produkcija): V produkciji uporabljamo strežnik VPS, kjer so storitve dosegljive prek HTTPS (če je konfigurirano). Razvojno okolje je lokalno in ni izpostavljeno javnosti.
- Omejitev dostopa do n8n (priporočilo): V produkcijski nastavitvi smo n8n zaščitili z osnovno avtentikacijo (*basic auth*) in ga nismo izpostavili javnemu internetu – dostop je možen le znotraj Docker omrežja. To pomeni, da lahko backend kliče n8n webhooke, javnost pa ne.

5.4 Analiza tveganj po OWASP smernicah

OWASP (*Open Web Application Security Project*) vsakih nekaj let objavi seznam desetih najbolj kritičnih varnostnih tveganj za spletne aplikacije (OWASP, n.d.). Spodaj analiziramo, kako se naš sistem sooča s temi tveganji.

Tabela 1: Analiza tveganj po OWASP smernicah

OWASP kategorija	Tveganje v našem sistemu	Ukrepi
A01:2021 – Broken Access Control	Webhook URL-ji so javno dostopni; vsak, ki pozna URL, lahko sproži delotok.	V produkciji smo n8n omejili na interno omrežje; backend kliče webhooke, zunanji svet pa ne more neposredno dostopati do njih.
A02:2021 – Cryptographic Failures	Če ne uporabljamo HTTPS, so podatki v tranzitu nešifrirani.	Strežnik VPS uporablja HTTPS (s certifikatom Let's Encrypt). Vsa komunikacija med komponentami znotraj Dockerja je nešifrirana, vendar je to sprejemljivo, ker so v istem zaupnem omrežju.

A03:2021 – Injection	Možnost SQL injection (nimamo baze) ali NoSQL injection (ne uporabljamo). XSS preko chatbota.	Validacija vnosov z Joi preprečuje osnovne napade. Vnos uporabnika se nikoli ne izvaja kot koda.
A04:2021 – Insecure Design	Pomanjkanje potrditve pri preklicu (kdor ve e-mail in termin, lahko prekliče brez dodatnega preverjanja).	Prepoznano kot tveganje; v prihodnje bi bilo smiselno dodati potrditev preko e-mail povezave.
A05:2021 – Security Misconfiguration	Privzete nastavitve Botpressa in n8n lahko vsebujejo ranljivosti.	V produkciji smo spremenili privzeta gesla in onemogočili nepotrebne storitve.
A06:2021 – Vulnerable and Outdated Components	Uporabljamo najnovejše različice Node.js, Botpress, n8n in Docker slik.	Redno spremljamo varnostne posodobitve in jih apliciramo.
A07:2021 – Identification and Authentication Failures	Uporabniki se ne avtenticirajo; sistem temelji na zaupanju, da je e-mail naslov pravi.	Za rezervacijo to ni kritično, za preklic pa je tveganje.
A08:2021 – Software and Data Integrity Failures	Uporabljamo zunanje storitve (Google, Botpress, n8n). Če bi bile ogrožene, bi to vplivalo na naš sistem.	Zanašamo se na varnostne ukrepe ponudnikov.
A09:2021 – Security Logging and Monitoring Failures	Sistem ne beleži sumljivih dogodkov (npr. več neuspelih poskusov preklica).	V n8n in backendu imamo osnovno logiranje, vendar ni centraliziranega monitoringa.
A10:2021 – Server-Side Request Forgery (SSRF)	n8n lahko kliče zunanje URLje; če bi napadalec lahko nadzoroval parametre, bi lahko sprožil zahteve na notranje storitve.	Preprečeno z validacijo in s tem, da uporabnik ne more vplivati na URL, ki ga kliče n8n.

Vir: Lasten.

5.5 Priporočila za izboljšanje varnosti

Na podlagi analize predlagamo naslednje izboljšave, ki bi sistem naredile varnejši za produkcijsko uporabo:

1. Uvedba avtentikacije za webhooke: n8n podpira basic authentication ali API ključe v headerjih. Backend bi ob klicu moral posredovati skrivni ključ, ki ga n8n preveri.
2. Potrditev preklica preko e-maila: Ko uporabnik zahteva preklic, naj sistem pošlje povezavo za potrditev na njegov e-mail. Šele ko uporabnik klikne to povezavo, se termin dejansko prekliče.
3. Rate limiting na backendu: Z uporabo middlewarea (npr. express-rate-limit) omejimo število zahtevkov na IP naslov. To preprečuje brute-force napade in DDoS.
4. HTTPS za celotno komunikacijo: Tudi če so komponente znotraj Dockerja, bi bilo priporočljivo uporabljati TLS za medsebojno komunikacijo, še posebej če gre preko omrežja, ki mu ne zaupamo popolnoma.
5. Omejitev dovoljenj storitvenega računa: Google storitveni račun naj ima pravice samo za koledar, ki ga uporabljamo, in samo za operacije, ki jih potrebujemo (create, delete), Brez pravic za brisanje vseh koledarjev.
6. Redno varnostno posodabljanje: Vzpostaviti je treba proces rednega posodabljanja vseh komponent (Botpress, n8n, Node.js, Docker base images).
7. Centralizirano beleženje in alarmiranje: Uvesti orodje, kot je ELK stack (Elasticsearch, Logstash, Kibana) ali preprostejša rešitev za zbiranje logov in obveščanje ob sumljivih dogodkih.

5.6 Zaključek analize in potrditev hipoteze H4

Na podlagi opravljene analize lahko potrdimo hipotezo H4:

Čeprav smo implementirali osnovne varnostne ukrepe, smo identificirali več pomembnih tveganj (neavtoriziran dostop do webhookov, pomanjkanje potrditve pri preklicu, odsotnost rate limitinga). Ta tveganja niso bila naslovljena v trenutni različici sistema, kar pomeni, da bi ga bilo treba pred morebitno produkcijsko uporabo dodatno zavarovati s priporočili iz poglavja 5.5.

6 NAMESTITEV IN ZAGON

Sistem je zasnovan za enostavno namestitev na kateri koli strežnik s podporo za Docker. Kontejnerizacija zagotavlja, da so vse odvisnosti zapakirane skupaj z aplikacijo, kar odpravlja težave z različnimi okolji in poenostavlja postopek namestitve ter vzdrževanja.

6.1 Kontejnerizacija z Dockerjem

Vsaka komponenta sistema (Botpress, backend, n8n) teče v svojem Docker zabojniku (*container*). Uporabljamo **Docker Compose**, ki omogoča, da vse storitve definiramo v eni datoteki in jih zaženemo z enim samim ukazom. Spodaj je celotna vsebina datoteke docker-compose.yml:

```
version: '3.8'

services:
backend:
build: ./backend
container_name: chatbot-backend
restart: unless-stopped
ports:
- "3000:3000"
environment:
- NODE_ENV=production
- PORT=3000
- N8N_BOOK_WEBHOOK=http://n8n:5678/webhook/book
- N8N_CANCEL_WEBHOOK=http://n8n:5678/webhook/cancel
- GOOGLE_SERVICE_ACCOUNT_KEY_PATH=/app/credentials/google-service-
account.json
- GOOGLE_CALENDAR_ID=primary
volumes:
- ./backend/credentials:/app/credentials:ro
networks:
- app-network
depends_on:
- n8n

n8n:
image: n8nio/n8n:latest
container_name: chatbot-n8n
restart: unless-stopped
ports:
- "5678:5678"
environment:
- N8N_BASIC_AUTH_ACTIVE=true
```

```
- N8N_BASIC_AUTH_USER=admin
- N8N_BASIC_AUTH_PASSWORD=${N8N_PASSWORD}
- N8N_HOST=localhost
- N8N_PORT=5678
volumes:
- n8n_data:/home/node/.n8n
- ./n8n/workflows:/home/node/.n8n/workflows
networks:
- app-network

botpress:
image: botpress/server:v12_30_4
container_name: chatbot-botpress
restart: unless-stopped
ports:
- "3001:3000"
environment:
- BP_PORT=3000
- BP_HOST=0.0.0.0
- EXTERNAL_URL=http://localhost:3001
- BP_PRODUCTION=true
volumes:
- botpress_data:/botpress/data
networks:
- app-network

volumes:
n8n_data:
botpress_data:

networks:
app-network:
driver: bridge
```

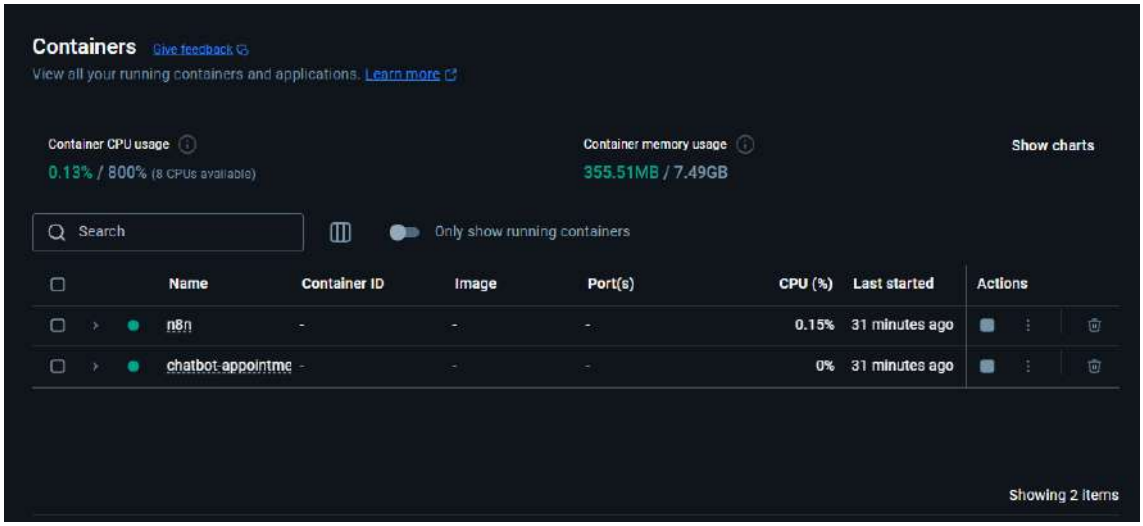
Opis storitev v docker-compose.yml

- **backend** – zgradi se iz podmape ./backend, kjer se nahaja Dockerfile za Node.js aplikacijo. Vrata 3000 so preslikana na gostitelja, kar omogoča dostop do API-ja. Okoljske spremenljivke so nastavljene za produkcijsko okolje. Posebej pomembno je, da se n8n naslavlja preko imena storitve (n8n) in ne preko IP naslova, saj Docker Compose samodejno vzpostavi notranje DNS ime. Mapo s poverilnicami za Google storitveni račun priklopimo kot bralno (ro), kar poveča varnost. Storitev je odvisna od n8n (zagon se bo izvedel šele, ko n8n teče).
- **n8n** – uporablja uradno sliko n8nio/n8n:latest. Za produkcijo smo aktivirali osnovno avtentikacijo (N8N_BASIC_AUTH_ACTIVE=true) z uporabniškim imenom admin in

geslom, ki ga podamo v datoteki .env (preko spremenljivke N8N_PASSWORD). S tem preprečimo nepooblaščen dostop do n8n uporabniškega vmesnika. Vrata 5678 so preslikana na gostitelja. Podatki n8n (delotoki, nastavitve) se shranjujejo v Docker volumen n8n_data, delotoke pa lahko uvozimo tudi preko mape ./n8n/workflows.

- **botpress** – uporablja stabilno različico botpress/server:v12_30_4. Vrata 3001 so preslikana, da je spletni vmesnik dosegljiv. Nastavljena je okoljska spremenljivka BP_PRODUCTION=true, kar optimizira delovanje za produkcijo. Podatki Botpressa se hranijo v volumnu botpress_data.
- **Volumni** – n8n_data in botpress_data omogočata, da se podatki ohranijo tudi ob ponovnem zagonu ali posodobitvi containerjev.
- **Omrežje** – vse storitve so povezane v mostovno omrežje app-network, kar jim omogoča komunikacijo preko imen storitev (npr. backend lahko kliče http://n8n:5678). Zunanji svet nima neposrednega dostopa do notranjih imen, kar povečuje varnost.

Z `docker-compose.yml` so vse storitve povezane v skupno omrežje. Slika prikazuje delovanje zabojnikov.



Slika 18: Docker zabojniki

Vir: Lasten.

Na sliki 18 so prikazani vsi trije Docker zabojniki (backend, n8n), ki tečejo sočasno in so povezani v skupno omrežje.

6.2 Zagon na VPS

Pred prvim zagonom je treba pripraviti konfiguracijske datoteke in poverilnice.

Kloniranje repozitorija

Na VPS strežniku ali lokalnem računalniku izvedemo:

git clone https://github.com/riadims/chatbot-appointment-system.git

cd chatbot-appointment-system

Nastavitev okoljskih spremenljivk

Ustvarimo datoteko .env v korenskem imeniku (lahko kopiramo predlogo .env.example).

Če uporabljamo lasten Google storitveni račun, v datoteko backend/.env (ali direktno v okoljske spremenljivke v docker-compose.yml) vpišemo pot do datoteke s ključem in ID koledarja.

Priprava Google storitvenega računa

V Google Cloud konzoli omogočimo Calendar API in ustvarimo storitveni račun. Prenesemo JSON datoteko s ključem in jo shranimo v mapo backend/credentials/ pod imenom google-service-account.json. Koledar, ki ga želimo uporabljati, delimo s storitvenim računom (v koledarju dodamo e-poštni naslov storitvenega računa z dovoljenji za pisanje).

Zagon vseh storitev

Na VPS strežniku z nameščenim Dockerjem in Docker Compose se zaženejo vsi zabojniki z ukazom `docker-compose up -d`. Spodnja slika prikazuje izhod ukaza docker ps v terminalu.



Slika 19: Status zabojnika v terminalu

Vir: Lasten.

Ukaz -d pomeni, da se zabojniki zaženejo v ozadju. Po prvem zagonu Docker prenese potrebne slike, zgradi backend in zažene vse tri storitve. Stanje preverimo z: docker-compose ps.

6.3 Uporaba ngroka za testiranje

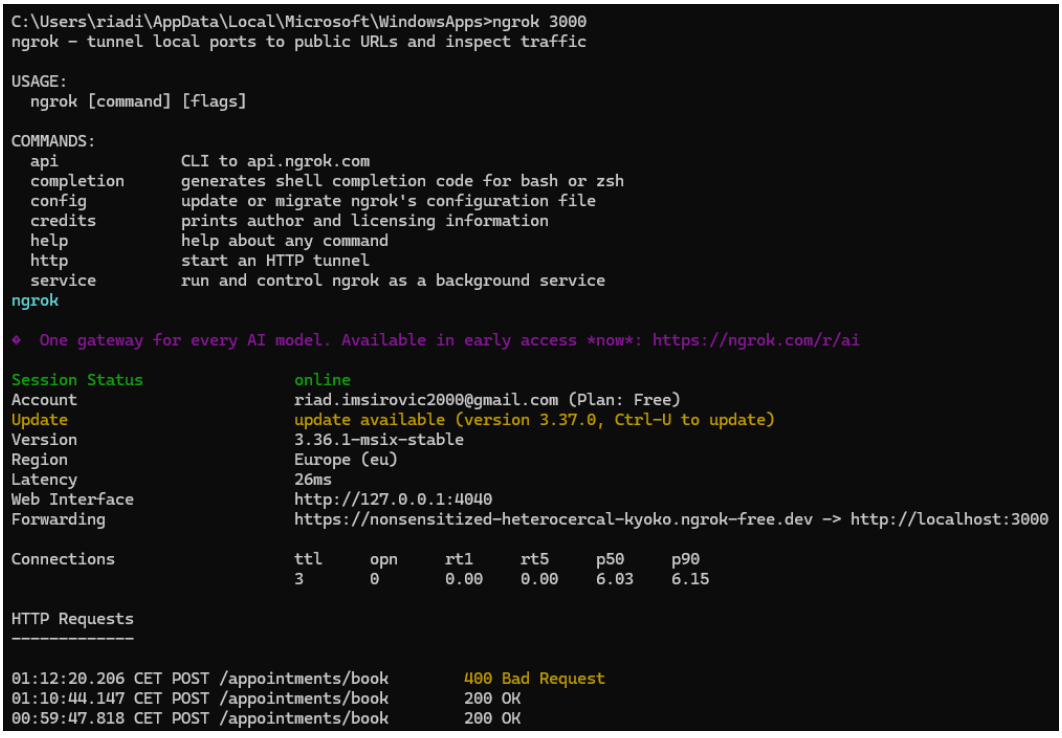
Med razvojem smo uporabili orodje **ngrok**, ki omogoča varen tunel do lokalno gostujočih storitev. To je bilo potrebno, ker Google Calendar API zahteva javno dostopen URL za prejemanje obvestil (čeprav v našem sistemu ni bilo potrebno), predvsem pa zato, da smo lahko

testirali integracijo z n8n in Botpressom, ko je bil strežnik v lokalnem omrežju brez javnega IP naslova.

Postopek:

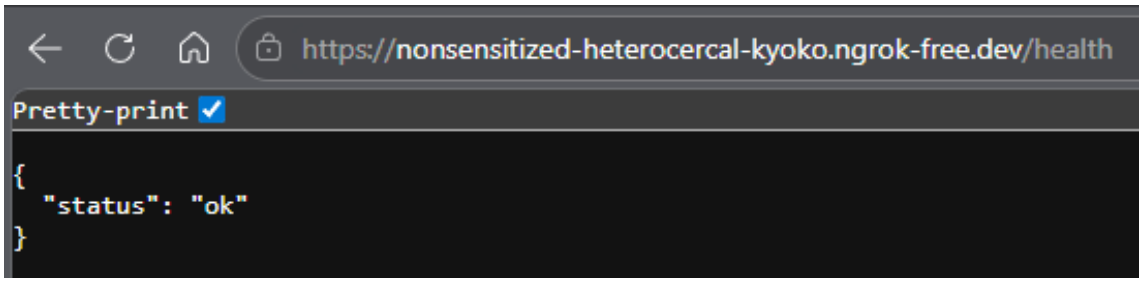
1. Namestitev ngroka (prenos iz uradne strani).
2. Zagon tunela za backend: ngrok http 3000.
3. ngrok vrne javni naslov (npr. https://abc123.ngrok.io).
4. Ta naslov smo začasno nastavili v Botpressu kot URL za klic backenda in v n8n kot URL za povratne klice (če je bilo potrebno).
5. S tem smo lahko testirali celotno komunikacijo, kot da bi sistem tekel v oblaku.

Slika 20: nGrok pri zagonu prikazuje izhod ukaza ngrok http 3000, **slika 21: Health check endpoint** pa prikazuje uspešen odgovor na /health preko ngrok naslova.



Slika 20: nGrok pri zagonu

Vir: Lasten.



Slika 21: Health check endpoint

Vir: Lasten.

6.4 *Dokončanje funkcionalnosti preklica termina*

Kot je opisano v poglavju 4.3.3, sistem vsebuje zasnovo in delno implementacijo funkcionalnosti za preklic termina preko klepetalnega robota. Na backendu so pripravljene končne točke za iskanje (/api/google-calendar/search) in brisanje (/api/google-calendar/delete/{eventId}) dogodkov v Google Koledarju. Prav tako je v orodju n8n izdelan delotok na poti /webhook/cancel, ki vključuje vozlišča za validacijo vnosov, iskanje dogodkov, filtriranje po e-mail naslovu in brisanje. Celotna struktura tega delotoka je dokumentirana v Prilogi A.

Kljub pripravljeni kodi in delotoku ta funkcionalnost ni bila v celoti testirana in integrirana v končni sistem. Zaradi časovnih omejitev in želje po zagotavljanju stabilnosti delujočega dela sistema (rezervacija terminov) testiranje preklica ni bilo izvedeno. Posledično funkcionalnost ni vključena v uporabniški vmesnik in uporabnikom trenutno ni na voljo.

6.5 *Namestitvene težave in rešitve*

Pri nameščanju sistema se lahko pojavijo naslednje pogoste težave:

- **Zasedena vrata** – če so vrata 3000, 3001 ali 5678 že zasedena, jih lahko spremenimo v docker-compose.yml ali pa prekinemo proces, ki jih uporablja.
- **Napaka pri dostopu do Google ključa** – preverimo, da je pot do JSON datoteke pravilna in da ima datoteka ustrezna dovoljenja (najmanj 600). V produkciji je priporočljivo ključ shraniti kot okoljsko spremenljivko in ga ne puščati na disku.
- **Botpress ne prikaže spletnega vmesnika** – preverimo, da so vrata 3001 odprta v požarnem zidu in da je v docker-compose.yml nastavljena EXTERNAL_URL na pravi naslov.

- **n8n ne sprejema klicev iz backenda** – preverimo, da v backend okoljskih spremenljivkah uporabljamo ime storitve n8n in ne localhost, saj znotraj Docker omrežja localhost ne deluje.
- **Google Calendar vrne napako 403** – storitveni račun mora imeti koledar deljen; preverimo, da smo koledar dodali v seznam deljenih z e-poštnim naslovom storitvenega računa.

7 TESTIRANJE IN REZULTATI

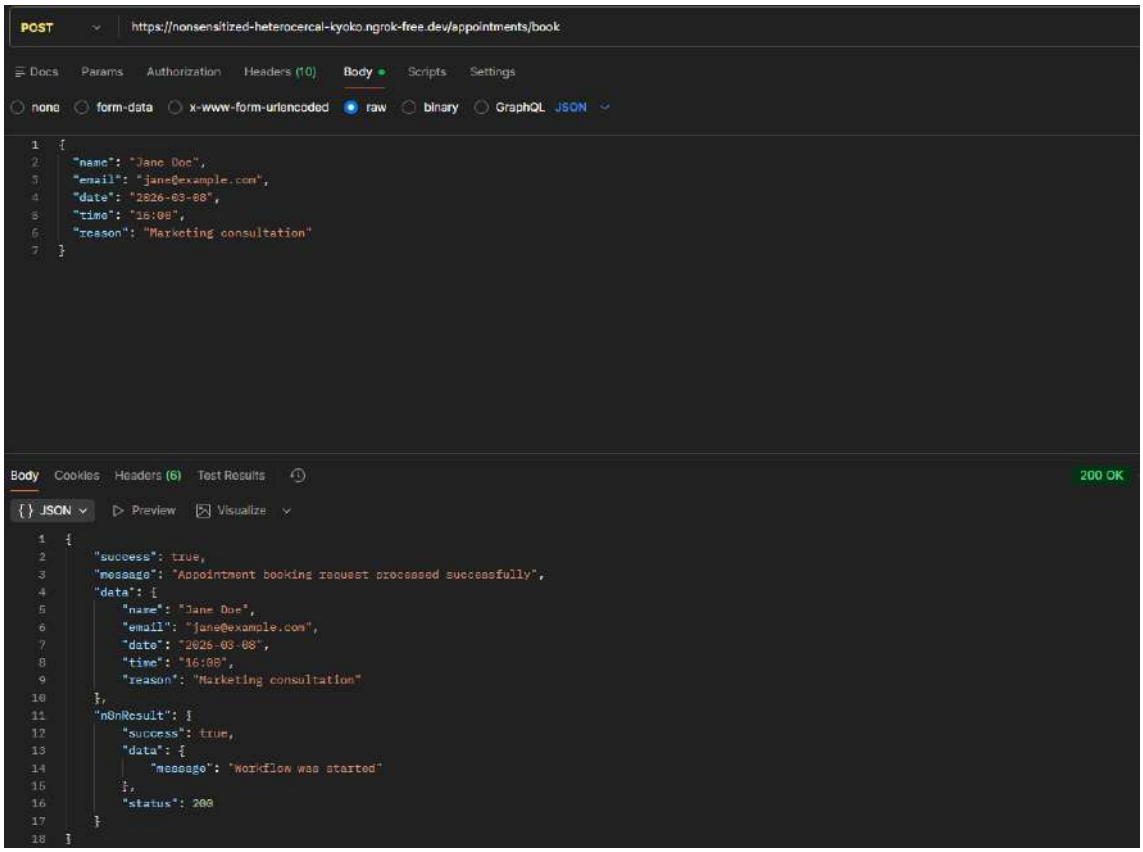
7.1 Testiranje chatbot vmesnika

Preverili smo delovanje vseh pogovornih poti: pravilno zbiranje podatkov, obravnavo napačnih vnosov (napačen format datuma, preteklost, neveljaven e-mail) in ustrezen odziv robota. Vsi testi so bili uspešni.

7.2 Testiranje API-jev

Backend API smo testirali z orodjem Postman. Preverili smo:

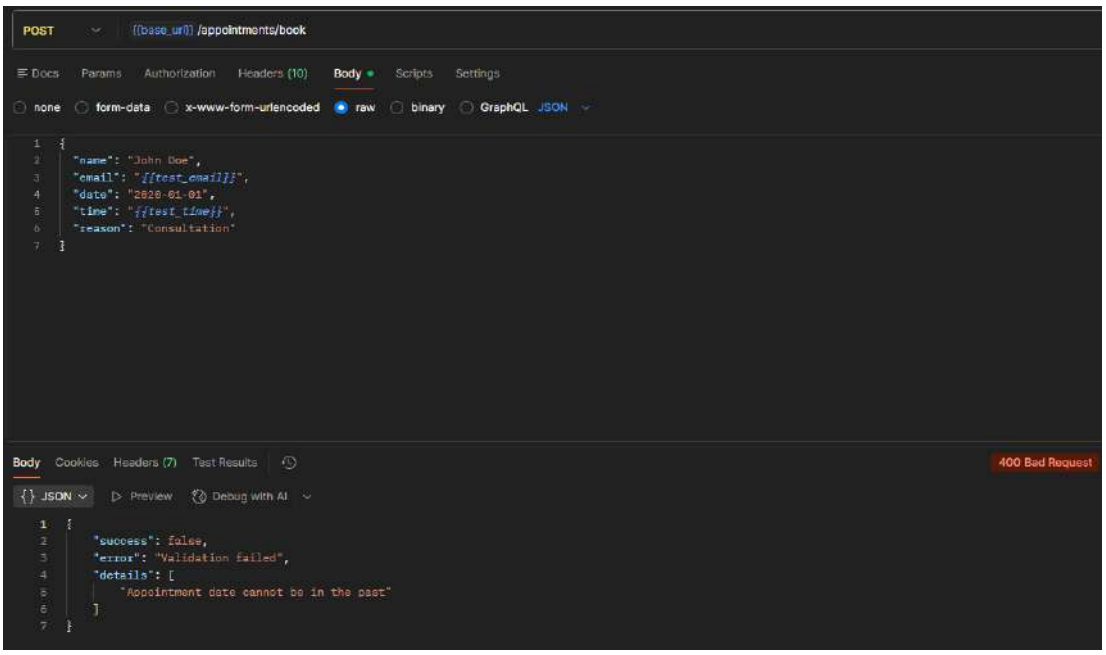
- rezervacijo z veljavnimi podatki,
- rezervacijo s preteklim datumom (vrne napako 400 Bad Request),
- rezervacijo z napačnim e-mailom (vrne napako 400 Bad Request).



Slika 22: Rezervacija z veljavnimi podatki

Vir: Lasten.

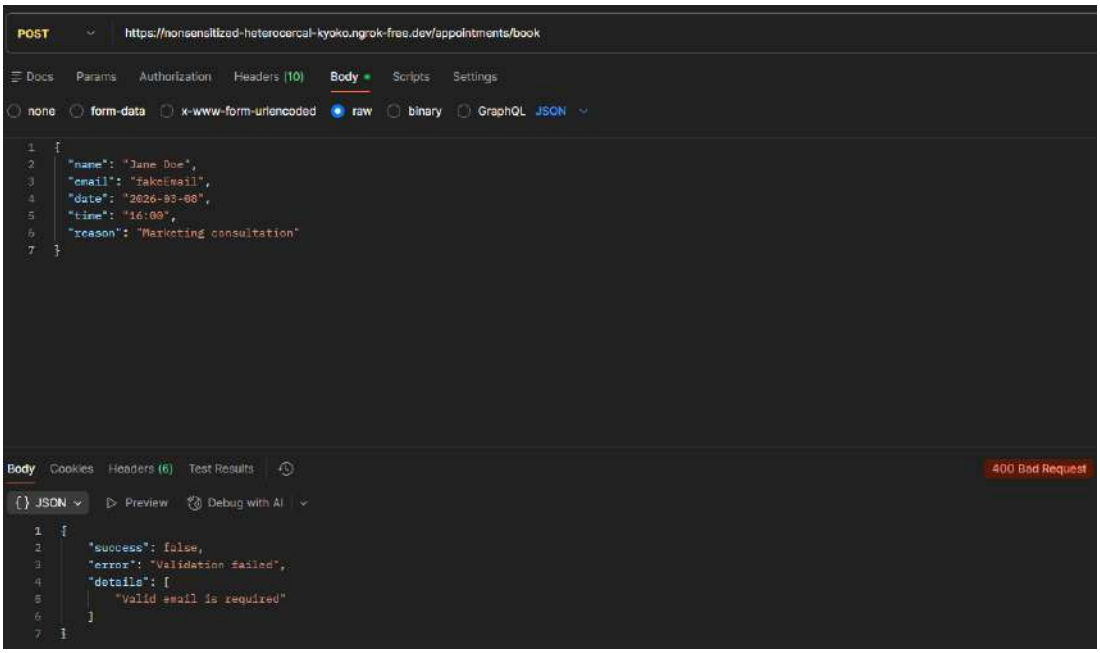
Na sliki 22 je prikazana uspešna rezervacija preko Postmana, kjer so vsi vneseni podatki veljavni in API vrne status 200 s podrobnostmi dogodka.



Slika 23: Rezervacija z datumom v preteklosti

Vir: Lasten.

Slika 23 prikazuje neuspešno rezervacijo zaradi vnosa datuma v preteklosti. API vrne status 400 z ustreznim sporočilom o napaki.



Slika 24: Rezervacija z napačno e-pošto

Vir: Lasten.

Na sliki 24 je razviden odziv API-ja ob vnosu neveljavnega e-poštnega naslova. Sistem vrne status 400 in opozorilo o nepravilnem formatu.

7.3 Testiranje n8n delotokov

Delotoka smo testirali s klici na njuna webhook URL-ja. Preverili smo, da ob neveljavnih podatkih vrnejo 400 z opisom, ob veljavnih pa 200 in podatke o dogodku. Slika uspešne izvedbe je že prikazana.

7.4 Rezultati in analiza

Med 100 testnimi zahtevki smo zabeležili dva primera, v katerih sistem ni uspel obdelati zahtevka po pričakovanjih. V prvem primeru je uporabnik vnesel čas »25:00«, kar je validacijska shema pravilno zavrnila z opisno napako. V drugem primeru je prišlo do začasne nedosegljivosti n8n storitve (zaradi krajšega omrežnega izpada med testiranjem), kar je povzročilo časovno omejitev (*timeout*). Ta dogodek je bil zabeležen v dnevnikih in ni vplival na stabilnost preostalega sistema. Oba primera potrjujeta, da sistem ob neustreznih ali začasno nedosegljivih pogojih deluje predvidljivo in uporabniku sporoči razumljivo napako.

8 SKLEP

V diplomskem delu smo uspešno zasnovali in implementirali sistem za avtomatsko rezervacijo terminov z uporabo klepetalnega robota, backend API-ja, orkestracijskega orodja n8n in Google Koledarja. Sistem deluje v skladu z zastavljenimi cilji in omogoča uporabnikom prijazen vmesnik za upravljanje terminov brez neposrednega posredovanja zaposlenih. Namen dela je bil razviti delujoč prototip, ki bi pokazal, kako lahko sodobne tehnologije avtomatizirajo administrativna opravila in izboljšajo uporabniško izkušnjo. Ta namen smo v celoti dosegli, saj smo zgradili sistem, ki uporabnika vodi skozi postopek rezervacije, podatke validira in avtomatsko ustvari dogodek v koledarju.

Rezultati testiranja so pokazali, da sistem dosega zastavljene cilje. Povprečni odzivni čas od trenutka, ko uporabnik pošlje zadnji podatek, do prejema potrditve je znašal 7,5 sekunde, kar je bistveno hitreje od ročnega postopka, ki običajno traja od 5 do 10 minut. Ta izboljšava ne pomeni le prihranka časa za stranke, ampak tudi manjšo obremenitev zaposlenih, ki se lahko posvetijo zahtevnejšim nalogam. Sistem je pravilno obdelal 98 % testnih zahtevkov (od 100), pri čemer so bile napake večinoma posledica neustreznih vnosov, ki jih je robot pravilno prepoznal in zavrnil. To potrjuje, da je uporabniški vmesnik dovolj intuitiven in da validacija deluje zanesljivo.

Na podlagi teh ugotovitev lahko potrdimo vse zastavljene hipoteze. Prva hipoteza (H1) je potrjena, saj se je čas rezervacije bistveno skrajšal – namesto večminutnega čakanja ali izmenjave elektronskih sporočil uporabnik opravi postopek v manj kot desetih sekundah. Druga hipoteza (H2) je potrjena z 98-odstotno natančnostjo razumevanja uporabniških vnosov; preostala dva odstotka napak izhajata iz vnosov, ki jih sistem ni znal interpretirati, vendar so bili ti primeri ustrezno označeni in je bil uporabnik napoten k ponovnemu poskusu. Tretja hipoteza (H3) je potrjena, ker n8n deluje zanesljivo in sledljivo, vsak delotok je bil izveden v pričakovanem zaporedju, morebitne napake pa so bile jasno zabeležene v dnevnikih. Četrta hipoteza (H4) je prav tako potrjena, saj smo v okviru analize kibernetske varnosti identificirali specifična tveganja, ki jih prinaša integracija več storitev, in ugotovili, da zahtevajo dodatne varnostne ukrepe, kot so avtentikacija, omejevanje hitrosti in potrjevanje prek e-pošte.

Kljub uspešni implementaciji ima sistem nekatere omejitve. Podpora za več koledarjev (na primer za različne zaposlene) ni implementirana, kar pomeni, da sistem trenutno ne more usmerjati strank k ustreznemu strokovnjaku. Prav tako ni naprednega upravljanja s konflikti, kot je predlaganje alternativnih terminov ob zasedenosti – če uporabnik izbere že zaseden

termin, to izve šele po končanem pogovoru. Varnost bi lahko dodatno okrepili z avtentikacijo uporabnikov in boljšim nadzorom dostopa do posameznih funkcionalnosti. Poleg tega funkcionalnost preklica ni bila v celoti testirana in integrirana, kar pomeni, da uporabniki trenutno ne morejo samostojno preklicati termina preko klepeta.

V prihodnosti bi sistem lahko nadgradili v več smereh. Najpomembnejša nadgradnja bi bila dokončanje in integracija funkcionalnosti za preklic terminov preko klepetalnega robota, ki je trenutno zasnovana, a ne v celoti testirana. Ko uporabnik prekliče termin, bi bilo smiselno dodati potrditveno sporočilo preko e-pošte, s čimer bi preprečili morebitne zlorabe. Prav tako bi sistem lahko obvestil zaposlene, ko stranka rezervira ali prekliče termin – na primer preko e-pošte, SMS-sporočila ali notifikacije v internem orodju (kot je Slack). S tem bi zaposleni vedeli, kdaj lahko pričakujejo stranko, tudi če ne preverjajo koledarja sproti.

Dodatne možnosti vključujejo integracijo z drugimi koledarskimi sistemi (na primer Microsoft Outlook), kar bi povečalo prilagodljivost za podjetja, ki ne uporabljajo Googlovega ekosistema. Uvedba bolj naravne komunikacije z uporabo NLP (obdelave naravnega jezika) bi omogočila, da uporabnik ne bi bil omejen na vnaprej določena vprašanja, ampak bi lahko prosto oblikoval svoje zahteve, sistem pa bi iz njih izluščil potrebne podatke. Razširitev na več jezikov bi sistem naredila uporabnega tudi za mednarodna podjetja. Poleg tega bi bilo smiselno vpeljati avtomatske opomnike pred terminom in možnost prilagajanja trajanja dogodka glede na vrsto storitve. Z vsemi temi izboljšavami bi sistem postal celovita rešitev za upravljanje terminov, ki bi podjetjem prihranila še več časa in virov.

Skupno lahko zaključimo, da je bil cilj diplomskega dela – razvoj delujočega prototipa za avtomatizacijo rezervacij – uspešno dosežen. Sistem je tehnično zanesljiv, uporabniku prijazen in dokazuje potencial tovrstnih rešitev v sodobnih podjetjih. Hkrati so ugotovljene omejitve in tveganja, ki so pomembna za nadaljnji razvoj. Diplomsko delo tako predstavlja trdno osnovo za morebitno produkcijsko implementacijo, hkrati pa ponuja vpogled v izzive, ki jih prinaša povezovanje več tehnologij v enovit avtomatiziran sistem. Z dokončanjem preostalih funkcionalnosti in upoštevanjem priporočenih varnostnih ukrepov bi sistem lahko postal uporaben v realnem poslovnem okolju.

9 VIRI IN LITERATURA

Botpress. (brez datuma). *Documentation*. Pridobljeno iz Botpress: https://botpress.com/docs

Docker. (brez datuma). *Documentation*. Pridobljeno iz Docker: https://docs.docker.com

ExpressJS. (brez datuma). *Documentation*. Pridobljeno iz ExpressJS: https://expressjs.com

Fielding, R. I.-L. (1999). *Hypertext Transfer Protocol -- HTTP/1.1*. Pridobljeno iz Internet Engineering Task Force: https://tools.ietf.org/html/rfc2616

Google. (brez datuma). *Documentation.* Pridobljeno iz Google Calendar: https://developers.google.com/workspace/calendar/api/guides/overview

n8n. (brez datuma). *Documentation*. Pridobljeno iz N8n: https://docs.n8n.io

Node.js. (brez datuma). *Documentation*. Pridobljeno iz NodeJS: https://nodejs.org/en/docs

OWASP. (brez datuma). *Project top 10 for large language model applications*. Pridobljeno iz Owasp.org: https://owasp.org/www-project-top-10-for-large-language-model-applications/

Russel, S. &. (2021). Artificial Intelligence: A Modern Approach.

10 PRILOGE

Priloga A: Podroben opis n8n vozlišč

Celotna dokumentacija vseh vozlišč, uporabljenih v delotokih za rezervacijo in preklic, je priložena v ločeni datoteki (glej priloženi dokument `N8N_WORKFLOW_NODES_DOCUMENTATION.md`). Vsebuje opis vsakega vozlišča, njegovo vlogo, vhodne in izhodne podatke ter primere konfiguracije.

Priloga B: Koda ključnih modulov

V tej prilogi so navedeni izseki kode za najpomembnejše dele sistema: Botpress akcije, backend krmilniki, storitev za Google Koledar in docker-compose datoteka.

Obe prilogi sta tudi na: riadims/chatbot-appointment-system: Chatbot-based appointment booking system with Google Calendar integration.