VIŠJA STROKOVNA ŠOLA ACADEMIA

MARIBOR

Primerjava ogrodij Flask, Bottle ter Falcon za izdelavo REST API-jev z uporabo umetne inteligence

Kandidat: Alen Cvetko

Vrsta študija: študent izrednega študija

Študijski program: Informatika

Mentor predavatelj: mag. Ervin Schaff

Mentor v podjetju: Bojan Janžek, dipl. inž. rač. in inf.

Lektorica: Maja Lampret, dipl. slov. (UN)

Maribor, 2024

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Podpisani/a Alen Cvetko, sem avtor/ica diplomskega dela z naslovom Primerjava ogrodij Flask, Bottle ter Falcon za izdelavo REST API-jev z uporabo umetne inteligence, ki sem ga napisal/a pod mentorstvom mag. Ervina Schaffa.

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, september 2024 Podpis študenta:

ZAHVALA

Zahvalil bi se mentorju g. Ervinu Schaffu za podporo in nasvete med pisanjem diplomskega dela.

Prav tako bi se zahvalil svojim staršem za podporo med študijem.

POVZETEK

Diplomsko delo preučuje zmogljivost in uporabo treh priljubljenih Python ogrodij za razvoj API-jev: Flask, Falcon in Bottle, z namenom ugotoviti, katero izmed njih ima največjo uporabno vrednost. Ta raziskava je pomembna tako za razvijalce kot tudi za podjetja, saj olajša izbiro ustreznega orodja za razvoj zmogljivih in hitro odzivnih API-jev. Da bi prišli do teh ugotovitev, smo se lotili razvoja primerljivih API-jev z vsakim od teh ogrodij. Za začetek smo pripravili integrirano razvojno okolje in načrtovali koncept API-jev, ki jih bomo razvili. Vsak API je bil zasnovan tako, da podpira osnovne funkcionalnosti CRUD operacij (Create, Read, Update, Delete). Po izdelavi API-jev smo izvedli meritve zmogljivosti, kjer smo merili odzivni čas, število poslanih zahtev in prepustnost. Poleg tega smo ocenili enostavnost implementacije in razširljivost posameznega ogrodja.

Nadaljevali smo z analizo podpore skupnosti, kjer smo pregledali aktivnosti razvijalcev na platformah Stack Overflow in Reddit ter dostopnost učnih gradiv. Flask se je izkazal za orodje z največjo podporo in dostopnostjo virov, medtem ko Falcon ponuja visoko specializirano podporo. V okviru raziskave smo prav tako preizkusili uporabo umetne inteligence ChatGPT 4.0 kot orodje za učenje uporabe ogrodja Bottle. Ugotovili smo, da je ChatGPT 4.0 učinkovit za učenje osnovnih in naprednih tehnik razvoja API-jev, čeprav so dodatni viri koristni za poglobljeno razumevanje. Med učenjem smo uporabljali ChatGPT za pridobivanje dodatnih informacij in nasvetov glede najboljših praks pri razvoju API-jev, kar je olajšalo proces učenja in povečalo učinkovitost dela.

Na podlagi teh meritev in ugotovitev smo zaključili, da ima Flask večjo podporo skupnosti kot Falcon, medtem ko Bottle izstopa po hitrosti odziva. Flask je dokazal svojo vrednost s svojo prilagodljivostjo in robustnostjo, ki omogočata enostavno integracijo z različnimi orodji in razširitvami. ChatGPT 4.0 se je izkazal za učinkovito orodje pri učenju uporabe ogrodja Bottle. Prav tako je bilo ugotovljeno, da je mogoče v manj korakih razviti API z ogrodjem Flask z uporabo umetne inteligence. Nadaljnje raziskave bi lahko vključevale poglobljeno analizo zmogljivosti pod različnimi obremenitvami, kvalitativno analizo podpore skupnosti ter primerjavo razvoja kompleksnejših API-jev. To bi zagotovilo celovito razumevanje prednosti in omejitev vsakega ogrodja v različnih primerih uporabe. Takšen pristop bi lahko pomagal razvijalcem pri izbiri najprimernejšega orodja glede na specifične zahteve njihovih projektov.

Ključne besede: Flask, Falcon, Bottle, API, ChatGPT 4.0, Python ogrodja, umetna inteligenca.

ABSTRACT

Comparison of Flask, Bottle, and Falcon Frameworks for Developing REST APIs Using Artificial Intelligence

This thesis examines the performance and use of three popular Python frameworks for API development: Flask, Falcon, and Bottle, aiming to determine which of them offers the greatest utility. This research is significant for both developers and businesses as it facilitates the selection of an appropriate tool for developing efficient and responsive APIs. To reach these conclusions, we developed comparable APIs with each of these frameworks. We started by setting up an integrated development environment and planning the API concepts to be developed. Each API was designed to support basic CRUD operations (Create, Read, Update, Delete). After developing the APIs, we performed performance measurements, where we measured response time, the number of requests sent, and throughput. Additionally, we evaluated the ease of implementation and the scalability of each framework.

We continued with an analysis of community support, where we examined developer activities on two platforms Stack Overflow and Reddit, and the availability of learning resources. Flask proved to be the tool with the most support and accessible resources, while Falcon offers highly specialized support. As part of the research, we also tested the use of artificial intelligence, specifically ChatGPT 4.0, as a tool for learning the Bottle framework. We found that ChatGPT 4.0 is effective for learning both basic and advanced API development techniques, although additional resources are beneficial for a deeper understanding. During the learning process, we used ChatGPT to obtain additional information and advice on best practices for API development, which facilitated the learning process and increased work efficiency.

Based on these measurements and findings, we concluded that Flask has greater community support than Falcon, while Bottle excels in response speed. ChatGPT 4.0 proved to be an effective tool for learning how to use the Bottle framework. It was also found that it is possible to develop an API with the Flask framework in fewer steps using artificial intelligence. Further research could include an in-depth analysis of performance under various loads, a qualitative analysis of community support, and a comparison of the development of more complex APIs. This would provide a comprehensive understanding of the advantages and limitations of each framework in different use cases.

Keywords: Flask, Falcon, Bottle, API, ChatGPT 4.0, Python frameworks, artificial intelligence.

KAZALO VSEBINE

KAZALO SLIK

KAZALO TABEL

1 UVOD

V sodobnem digitalnem svetu, kjer spletne aplikacije igrajo osrednjo vlogo pri povezovanju ljudi, informacij in storitev, je razvoj teh aplikacij postal ključnega pomena. S tem se povečuje tudi potreba po orodjih, ki omogočajo hitrejši in učinkovitejši razvoj spletnih rešitev. V tem kontekstu so se REST API-ji uveljavili kot nepogrešljivi gradniki sodobnih spletnih aplikacij, saj omogočajo enostavno in učinkovito integracijo različnih platform in storitev.

Ogrodja za izdelavo REST API-jev, kot so Flask, Bottle in Falcon, razvijalcem ponujajo širok spekter možnosti. Ta ogrodja omogočajo izdelavo back-end in front-end aplikacij v programskem jeziku Python, ki je zaradi svoje enostavnosti, prilagodljivosti in široke uporabnosti izjemno priljubljen med razvijalci. Vsako izmed teh ogrodij prinaša svoje prednosti in izzive, ali na področju zmogljivosti, enostavnosti uporabe ali podpore skupnosti. Hkrati pa se umetna inteligenca, zlasti napredni modeli, kot je ChatGPT 4.0, vse bolj uveljavlja kot orodje, ki lahko bistveno pospeši proces učenja in razvoja v svetu programiranja.

1.1 Opis področja in opredelitev problema

Področje raziskovanja v tem diplomskem delu se osredotoča na razvoj spletnih aplikacij s poudarkom na implementaciji REST API-jev. REST API-ji so ključni element pri povezovanju različnih spletnih storitev in aplikacij, omogočajo prenos podatkov med strežniki in odjemalci ter so temelj za večino sodobnih spletnih aplikacij. S pojavom različnih ogrodij za Python, kot so Flask, Bottle in Falcon, se je razvijalcem odprla široka paleta ogrodij za ustvarjanje teh API-jev. Vsako od teh ogrodij ima svoje specifike in prednosti, kar lahko vpliva na končno zmogljivost, enostavnost uporabe in podporo skupnosti.

Problem, ki ga nameravamo raziskati, je povezan s primerjavo omenjenih ogrodij in učinkovitostjo njihove uporabe pri razvoju REST API-jev. Glede na to, da je na trgu veliko različnih ogrodij, se postavlja vprašanje, katero izmed njih izbrati za določen projekt, saj lahko ta izbira pomembno vpliva na hitrost razvoja, zmogljivost aplikacije ter podporo pri reševanju težav. Poleg tega je pomembno tudi vprašanje, kako lahko umetna inteligenca, kot je ChatGPT 4.0, prispeva k učenju in uporabi teh ogrodij. V praksi se mnogi razvijalci soočajo z izzivi pri učenju novih tehnologij, zato je smiselno raziskati, ali lahko UI učinkovito nadomesti tradicionalne vire znanja in mentorstvo.

1.2 Namen, cilji in osnovne trditve

Namen diplomskega dela je raziskati in primerjati ogrodja Flask, Bottle in Falcon za izdelavo REST API-jev z uporabo umetne inteligence. Posebna pozornost bo namenjena ugotavljanju, kako učinkovito lahko umetna inteligenca, kot je ChatGPT 4.0, podpira razvoj in učenje teh ogrodij ter kako se ta ogrodja razlikujejo v smislu zmogljivosti, enostavnosti uporabe in podpore skupnosti.

Cilji diplomskega dela so:

- Primerjava zmogljivosti ogrodij Flask, Bottle in Falcon: analizirati in primerjati zmogljivost REST API-jev, izdelanih z vsakim izmed teh ogrodij, s poudarkom na odzivnem času in učinkovitosti.
- Ocenjevanje enostavnosti učenja: preučiti, kako lahko ChatGPT 4.0 kot orodje za umetno inteligenco pomaga razvijalcem pri učenju in uporabi ogrodij Bottle in Flask.
- Analiza podpore skupnosti: preučiti, kako obsežna in koristna je podpora skupnosti za vsako izmed teh ogrodij, kar je ključno za dolgoročno vzdrževanje in razvoj projektov.
- Raziskava uporabe umetne inteligence pri razvoju API-jev: oceniti, ali je ChatGPT 4.0 zadosten vir za učenje in pomoč pri razvoju API-jev z izbranimi ogrodji.

Osnovne trditve (hipoteze):

- H1: API, izdelan z ogrodjem Bottle, ima hitrejši odzivni čas v primerjavi z API-jem, izdelanim z Flask-om in Falcon-om.
- H2: Ogrodje Flask ima večjo podporo skupnosti kot ogrodje Falcon.
- H3: Za učenje uporabe ogrodja Bottle je dovolj ChatGPT 4.0 brez potrebe po drugih virih.
- H4: Z uporabo umetne inteligence lahko v manj korakih izdelamo API, ki je izdelan z Flask, kot podoben API, izdelan z Bottle.

1.3 Predpostavke in omejitve

Pred začetkom diplomskega dela nisem imel izkušenj programiranja v jeziku Python. Imam pa izkušnje iz drugih objektno orientiranih programskih jezikov, kot so C++, Java in C#, tako da so bile osnove podobne, nekaj sintakse pa sem se moral priučiti. Prav tako nisem imel izkušenj z ustvarjanjem lastnih API-jev.

Prva predpostavka je, da mora moje razumevanje delovanja REST API-jev in sintakse jezika Python dosegati zadosten nivo, da lahko izdelam lasten API na podlagi dokumentacije. Domneva se, da bo ChatGPT 4.0 kot umetna inteligenca zagotavljal dovolj relevantnih in natančnih informacij za učenje in razvoj z uporabo ogrodij.
Pričakuje se, da bodo izvedeni testi za merjenje zmogljivosti (odzivnega časa) API-jev omogočili natančno in primerljivo analizo zmogljivosti med izbranimi ogrodji.
Testiranje zmogljivosti API-jev bo izvedeno na istem računalniku, kot bo strežnik, kar pomeni, da rezultati morda ne bodo popolnoma odražali zmogljivosti v produkcijskih okoljih z velikim številom uporabnikov.

1.4 Uporabljene raziskovalne metode

1. Pregled literature:
Najprej bomo izvedli obsežen pregled obstoječe literature, dokumentacije in spletnih virov, povezanih z ogrodji Flask in Falcon. Ta pregled bo vključeval analizo tehničnih specifikacij, primerjalnih študij in uporabniških izkušenj, kar nam bo omogočilo boljše razumevanje prednosti, slabosti in področij uporabe teh ogrodij. Poleg tega bomo raziskali literaturo o uporabi umetne inteligence, zlasti ChatGPT 4.0, pri programiranju in učenju novih tehnologij.

2. Eksperimentalna metoda:
Eksperimentalni del dela bo vključeval praktično implementacijo REST API-jev z uporabo ogrodij Flask, Bottle in Falcon. Vsak API bo zasnovan z enakimi funkcionalnostmi (GET, POST, PUT in DELETE zahteve), da bo mogoče natančno primerjati njihovo zmogljivost. Po implementaciji bomo izvedli teste za merjenje odzivnega časa in učinkovitosti API-jev v različnih scenarijih obremenitve.

3. Kvantitativna analiza:
Za primerjavo zmogljivosti API-jev bomo uporabili kvantitativno analizo, ki bo vključevala merjenje odzivnih časov, obdelavo podatkov in statistično primerjavo rezultatov. Ta analiza bo omogočila oceno, katero ogrodje je najbolj učinkovito glede na specifične kriterije zmogljivosti.

4. Kvalitativna analiza:

Kvalitativna analiza bo osredotočena na oceno enostavnosti uporabe in podpore skupnosti za posamezno ogrodje. To bomo dosegli z analizo dokumentacije, dostopnosti virov za učenje ter aktivnosti in podpore na spletnih forumih in platformah, kot so Reddit, Stack Overflow in uradne strani ogrodij.

5. Uporaba umetne inteligence:

Pri raziskavi bomo uporabili ChatGPT 4.0 za podporo pri učenju in implementaciji API-jev z ogrodjema Bottle in Flask. Proces bomo dokumentirali in ocenili, kako učinkovito umetna inteligenca pomaga pri razvoju in učenju v primerjavi s tradicionalnimi metodami učenja. Poseben poudarek bo na vprašanju, ali lahko ChatGPT 4.0 nadomesti druge vire znanja in mentorstvo.

2 UMETNA INTELIGENCA

2.1 Uvod v umetno inteligenco

Umetna inteligenca (v nadaljevanju UI) je področje računalništva, ki se ukvarja z razvojem sistemov, ki lahko izvajajo naloge, ki običajno zahtevajo človeško inteligenco, kot so razumevanje jezika, prepoznavanje vzorcev, učenje in odločanje. V zadnjih desetletjih je UI postala pomembna tehnologija, ki spreminja različne industrije in izboljšuje naš vsakdanjik (Artificial Intelligence, 2024).

2.2 Zgodovina in izvor umetne inteligence

Zgodovina UI sega v sredino 20. stoletja, z začetkom raziskav na področju ustvarjanja strojev, ki bi lahko posnemali človeško miselno delovanje. Prvi pomembni koraki so bili narejeni z razvojem logičnih teorij in algoritmov, ki bi lahko reševali matematične probleme. Pomemben mejnik je bil razvoj **Turingovega stroja** in ideja o testu, ki bi lahko določil, ali stroj razmišlja kot človek (Turingov test).

Prvo programsko opremo, ki se je lahko štela za UI, so razvili v petdesetih letih prejšnjega stoletja. **Strojno učenje** je igralo pomembno vlogo pri nadaljnjem razvoju UI, saj omogoča, da se sistemi izboljšujejo s pridobivanjem izkušenj in analiziranjem velikih količin podatkov brez potrebe po vnaprej določenih navodilih.
V zadnjih letih so napredki v **strojni opremi** (kot so CPU-ji) in dostopnost velikih količin podatkov (Big Data) omogočili hiter razvoj **globokega učenja** (Deep Learning), ki je osnova mnogih sodobnih UI aplikacij, vključno z naprednimi nevronskimi mrežami, ki omogočajo prepoznavanje slik, obdelavo naravnega jezika in še veliko več (Artificial intelligence, 2024).

2.3 Karakteristike UI

Umetna inteligenca se od tradicionalnih računalniških sistemov razlikuje po več značilnostih:

- Strojno učenje (angl.: Machine Learning): sistemi UI se učijo iz podatkov, prepoznavajo vzorce in na tej podlagi sprejemajo odločitve. Ta sposobnost omogoča, da UI nenehno izboljšuje svoje delovanje.
- Naravno jezikovno procesiranje (NLP − angl.: Natural Language Processing): NLP omogoča računalnikom razumevanje in ustvarjanje človeškega jezika, kar je ključno za aplikacije, kot so virtualni asistenti in chat boti.

− Generativna UI (angl.: Generative AI): Generativne UI tehnologije omogočajo ustvarjanje novih vsebin, kot so besedila, slike, glasba in celo video posnetki. Ti modeli so sposobni učenja in formiranja podatkov iz različnih virov (Artificial intelligence, 2024).

2.4 Uporaba umetne inteligence v sodobnem svetu

Umetna inteligenca (UI) je postala pomemben del sodobnega življenja, saj njena uporaba vpliva na različne industrije in vsakodnevne aktivnosti. V zdravstvu UI pomaga pri diagnosticiranju bolezni, izboljšanju zdravljenja in razvoju novih zdravil. Na finančnem področju UI prispeva k zaznavanju prevar, ocenjevanju kreditnih tveganj in avtomatizaciji trgovanja z delnicami in obveznicami. Poleg tega se UI uporablja tudi v trženju, kjer pomaga pri analizi podatkov, ciljnem oglaševanju in ustvarjanju vsebin, prilagojenih uporabnikom.

V vsakdanjem življenju srečujemo UI v pametnih pomočnikih, kot sta Siri in Alexa, ter v avtonomnih avtomobilih, ki temeljijo na globokem učenju za zaznavanje in interpretacijo okolice. V igrah pa UI omogoča bolj realistično in prilagojeno izkušnjo z neigralnimi liki, ki se odzivajo na interakcije igralcev.

Na področju zakonodaje je UI prav tako pomembna tema, saj se države po svetu trudijo vzpostaviti regulacije, ki bi zagotovile varno in odgovorno uporabo te tehnologije. Evropska unija je s sprejetjem Akta o umetni inteligenci leta 2024 postavila temelje za urejanje tega področja, medtem ko ZDA še vedno iščejo primerne zakonodajne rešitve (Artificial Intelligence, 2024).

2.5 Generativna umetna inteligenca in ChatGPT-4

Generativna umetna inteligenca (UI) predstavlja eno izmed najnaprednejših vej umetne inteligence, ki se osredotoča na ustvarjanje novih vsebin. Gre za tehnologijo, ki presega zgolj analiziranje in prepoznavanje vzorcev, saj omogoča računalnikom ustvarjanje povsem novih in edinstvenih informacij. Generativna UI se uporablja v številnih aplikacijah, od ustvarjanja umetniških del do sinteze glasbe in generiranja naravnega jezika, kar odpira nova področja uporabe v različnih panogah (Generative AI use cases, 2024).

Eden izmed najbolj znanih primerov generativne UI je ChatGPT-4, ki ga je razvilo podjetje OpenAI. ChatGPT-4 je velik jezikovni model, ki temelji na arhitekturi transformatorjev. Usposobljen je bil z analizo ogromnih količin besedilnih podatkov, kar mu omogoča generiranje koherentnih in smiselnih besedil na podlagi vhodnih podatkov uporabnika.

ChatGPT-4 se uporablja v številnih aplikacijah, vključno s pomočjo pri pisanju, ustvarjanju vsebin, avtomatizaciji pogovorov s strankami in celo pri raziskavah (GPT-4 Research, 2023).
Ena ključnih zmožnosti GPT-4 je njegova sposobnost generiranja besedila na podlagi podrobnih navodil, kar omogoča ustvarjanje prilagojenih in kompleksnih vsebin, kot so pisanje esejev, programskih kod, ustvarjanje zgodb in zagotavljanje podrobnih odgovorov na specifična vprašanja (Documentation, 2024).
Uporaba generativne umetne inteligence predstavlja pomemben korak naprej pri integraciji umetne inteligence v vsakdanje življenje in poslovanje. Njene aplikacije so široke in vključujejo izboljšanje uporabniške izkušnje, pospeševanje raziskovalnih procesov in ustvarjanje novih oblik zabave. Vendar pa prinaša tudi izzive, povezane z etiko, varnostjo in odgovorno uporabo, saj lahko generativni modeli ustvarjajo vsebine, ki so lahko napačne ali zavajajoče (Generative AI use cases, 2024).

2.5.1 Stroški uporabe ChatGPT 4.0

Stroški uporabe ChatGPT 4.0 so odvisni od več dejavnikov − način uporabe, različica produkta ter dodatne storitve in funkcije. ChatGPT 4.0 ponuja tri glavne cenovne modele: Osnovna verzija (Base), Naročniška verzija (Subscription) in Plačilo po uporabi (Pay-as-you-go).

Osnovna verzija (angl. Base)

Osnovna verzija je namenjena uporabnikom, ki potrebujejo osnovne funkcionalnosti in manjšo obremenitev. Pogosto je brezplačna ali zelo nizkocenovna, vendar ponuja omejen dostop do virov in funkcionalnosti. Ta verzija je primerna za manjše projekte ali osebno uporabo.

Naročniška verzija (angl. Subscription)

Naročniška verzija je namenjena podjetjem in posameznikom, ki redno uporabljajo ChatGPT 4.0 in potrebujejo boljše zmogljivosti in podporo. Stroški naročnine se običajno gibljejo od nekaj deset do več sto evrov na mesec, odvisno od obsega uporabe in dodatnih storitev. Naročniški modeli pogosto vključujejo določeno število klicev na mesec, dostop do naprednih funkcij, boljšo podporo strankam in višje prioritete pri obdelavi zahtev.

Plačilo po uporabi (angl. Pay-as-you-go)

Model plačila po uporabi je primeren za uporabnike, ki potrebujejo fleksibilnost in želijo plačevati le za dejansko uporabo storitev. Stroški so odvisni od števila klicev API-ja in obsega obdelave podatkov. Cene se lahko začnejo pri nekaj centih na klic, vendar lahko hitro narastejo z večanjem obsega uporabe. Ta model je posebej koristen za projekte z nepredvidljivo obremenitvijo ali za tiste, ki ne potrebujejo stalnega dostopa do storitev.

Dodatni stroški za napredne funkcionalnosti

Uporaba naprednih funkcionalnosti in specifičnih nastavitev modela ChatGPT 4.0 lahko prinese dodatne stroške. To vključuje večjo obdelavo podatkov, prilagoditev modela specifičnim potrebam uporabnika ali dodatne funkcionalnosti, kot so posebni varnostni in zasebnostni ukrepi (ChatGPT Pricing, 2024) (API Pricing, 2024).

3 REST API-JI

3.1 Uvod v API-je

»API pomeni programski vmesnik za aplikacije (angl. Application Programming Interface); gre za vmesnik, ki omogoča komunikacijo med spletno stranjo (ali mobilno aplikacijo) in logiko v ozadju. Preprosto povedano, API deluje kot posrednik, ki sprejme zahtevo uporabnika, jo pošlje sistemu v ozadju, nato pa uporabniku posreduje odgovor sistema. Za boljše razumevanje lahko API primerjamo z natakarjem oziroma natakarico v restavraciji, ki razume različna naročila strank in nato deluje kot posrednik med strankami in kuharji v kuhinji.« (Chan, 2019)

3.2 Opis REST

REST (Representational State Transfer) je arhitekturni slog, ki ga je leta 2000 prvič predstavil Roy Fielding v svoji disertaciji.

REST API omogoča komunikacijo med dvema programskima opremama preko omrežja ali v isti napravi. REST najpogosteje uporablja HTTP zahteve za pridobivanje in objavo podatkov med odjemalcem in srežnikom.

REST omogoča nemoteno komunikacijo, če sta odjemalec in strežnik napisana v različnih programskih jezikih ali uporabljata različne operacijske sisteme. Odjemalec lahko zahteva vire v jeziku, ki ga strežnik razume, strežnik pa odgovori z virom v jeziku, ki ga odjemalec lahko uporablja. Viri so najpogosteje v obliki JSON ali XML. Vsaka interakcija z virom poteka prek HTTP metod, kot so GET, POST, PUT in DELETE. Poglavje o virih sledi v nadaljevanju (What is a REST API?, 2024).

3.3 Šest načel REST arhitekture:

Da je API lahko »RESTful« se mora držati teh načel:

- Enoten vmesnik (angl. Uniform Interface)
- Odjemalec – Strežnik (angl. Client – Server)
- Komunikacija brez stanja (angl. Stateless)
- Predpomnilnik (angl. Cacheable)
- Plastni sistem (angl. Layered System)
- Koda na zahtevo (angl. Code on Demand) (opcijsko)

3.3.1 Enoten vmesnik (angl. Uniform Interface)

Enoten vmesnik pomeni, da vsi odjemalci in strežniki komunicirajo na enak način – enotna komunikacija.

Naslednje štiri omejitve lahko dosežejo enoten vmesnik REST:

- Identifikacija virov – vsak vir, ki je vključen v interakcijo med odjemalcem in strežnikom mora biti enolično identificiran s strani vmesnika (z URI).
- Manipulacija virov prek predstavitev – viri morajo biti enotno predstavljeni v odgovoru strežnika (HTML, JSON, XML). Odjemalci dobijo v predstavitvi vira dovolj informacij (s pomočjo metapodatkov) za spreminjanje in brisanje vira, če bi to želeli.
- Samoopisna sporočila – vsaka predstavitev vira mora imeti informacije za opis, kako obdelati sporočilo. Zagotoviti mora tudi informacije o dodatnih dejanjih, ki jih odjemalec lahko izvaja na viru.
- Hipermedij kot motor stanja aplikacije – strežnik s pošiljanjem hiperpovezav v predstavitvi vira doseže, da lahko odjemalec odkrije informacije o vseh drugih povezanih virih (RESTful API, 2024) (What is a RESTful API?, 2024).

3.3.2 Odjemalec – Strežnik (angl. Client – Server)

Ta značilnost omogoča, da se komponente odjemalca in strežnika razvijajo neodvisno drug od drugega. Odjemalec in strežnik sta lahko na različnih platformah, kar pa ne vpliva na njuno sodelovanje. Med razvijanjem komponent pa moramo biti pozorni, da se vmesnik/pogodba med njima ne prekine (RESTful API, 2024).

3.3.3 Komunikacija brez stanja (angl. Stateless)

Značilnost kumunikacije brez stanja je, da strežnik ne hrani nobenih podatkov o stanju odjemalca med zahtevami. Vsaka zahteva, ki jo odjemalec pošlje strežniku, mora vsebovati vse potrebne informacije za obdelavo zahteve. To načelo poenostavi obdelavo na strežniški strani in omogoča večjo skalabilnost sistema, saj strežnik ni obremenjen z vzdrževanjem stanja med sejami (RESTful API, 2024) (What is REST?, 2024).

3.3.4 Predpomnilnik (angl. Cacheable)

REST API-ji omogočajo predpomnjenje odgovorov strežnika, če so ti označeni kot predpomnjeni in imajo navedeno časovno obdobje. To pomeni, da lahko odjemalec uporablja predpomnjene odgovore za podobne zahteve, kar zmanjša obremenitev strežnika in poveča hitrost odziva. Predpomnjenje je posebej koristno pri ponavljajočih se zahtevah, kjer se podatki redko spreminjajo (RESTful API, 2024).

3.3.5 Večplasten sistem (angl. Layered System)

Večplasten sistem omogoča uporabo večplastne sistemske arhitekture, kjer imajo različne plasti različne naloge oziroma funkcionalnosti. Odjemalec ne more ugotoviti, s katero plastjo je povezan, saj je za njega pomemben samo odgovor. Ta pristop omogoča večjo fleksibilnost in varnost, saj lahko spremenimo ali nadgradimo eno plast brez vpliva na celoten sistem (REST Architectural Constraints, 2024).

3.3.6 Koda na zahtevo (angl. Code on Demand) (opcijsko)

V nekaterih primerih lahko strežnik odjemalcu pošlje kodo, ki jo ta nato izvede. Ta funkcionalnost ni obvezna, vendar lahko izboljša razširljivost sistema, saj omogoča dinamično nalaganje funkcionalnosti v odjemalca. To omogoča odjemalcem, da pridobijo nove funkcionalnosti brez potrebe po posodobitvi celotne aplikacije (RESTful API, 2024).

3.4 Viri (angl. Resources) v REST API-jih

V REST API-jih so viri temeljni gradniki, ki predstavljajo vsebine, s katerimi odjemalci komunicirajo. Viri so lahko na primer podatki o uporabnikih, izdelki, slike, dokumenti ali kateri koli drugi podatki, ki jih strežnik ponuja. Vsak vir v REST API-ju ima enolični URI (Uniform Resource Identifier), ki omogoča dostop do tega vira. URI je kot naslov, ki ga odjemalec uporablja za interakcijo z določenim virom na strežniku. URI mora biti zasnovan tako, da je razumljiv in logičen, da omogoča preprosto navigacijo in interakcijo z viri (Read the Docs, 2024).

Na primer, če imamo REST API, ki upravlja podatke o knjigah, lahko ima vsaka knjiga svoj URI, npr. »**http://example.com/api/books/1**«, kjer **1** označuje ID knjige. Ta URI omogoča odjemalcu, da dostopa do specifične knjige in izvede različne operacije z njo, kot so pridobitev informacij, posodobitev podatkov ali izbris knjige.

»URI se imenuje tudi končna točka zahteve (engl. request endpoint) in strežniku jasno pove, kaj odjemalec zahteva.« (What is a RESTful API?, 2024)

3.4.1 Načini predstavitve virov (JSON, XML)

Viri v REST API-jih niso dostopni v svoji surovi obliki, ampak so predstavljeni v določenem formatu, ki omogoča odjemalcem, da te vire berejo, razumejo in uporabljajo. Dva najpogosteje uporabljena formata za predstavitev virov sta JSON (JavaScript Object Notation) in XML (eXtensible Markup Language).

JSON (JavaScript Object Notation)

JSON je lahek format za izmenjavo podatkov, ki je enostavno berljiv za ljudi in lahko obdelovan s strani računalnikov. JSON predstavlja podatke v obliki parov ime-vrednost, kjer je vsaka vrednost lahko preprost tip (npr. število, niz) ali kompleksna struktura (npr. objekt, seznam). Zaradi svoje enostavnosti in lahke sintakse je JSON postal najbolj priljubljen format za predstavitev virov v REST API-jih (What is JSON?, 2024).

JSON je zelo priljubljen tudi zato, ker se naravno ujema s strukturami podatkov v večini programskih jezikov, kar poenostavi delo razvijalcev. JSON omogoča hitro in učinkovito izmenjavo podatkov (JSON vs XML, 2024).

Slika 1: Primer JSON

Vir: (Lastni vir)

XML je format za izmenjavo podatkov, ki je bolj strukturiran in zmogljivejši kot JSON. XML uporablja oznake (angl. tags) za definiranje elementov in njihovega odnosa. Čeprav je XML bolj kompleksen in obsežen od JSON-a, je zelo fleksibilen in primeren za predstavitev bolj zapletenih podatkovnih struktur, kjer je potrebna natančna opredelitev hierarhije in atributov. XML je pogosto uporabljen v primerih, kjer je potrebna večja natančnost, validacija podatkov, ali kadar sistem zahteva bolj kompleksno strukturo podatkov (What is XML?, 2024).

Slika 2: Primer XML

Vir: (Lastni vir)

3.4.2 Metode virov v REST API-jih

Metode virov v REST API-jih se nanašajo na različne operacije, ki jih odjemalec lahko izvaja nad viri. Te operacije temeljijo na standardnih HTTP metodah CRUD (Create, Read, Update, Delete), ki so posebej zasnovane za interakcijo z viri.

Tabela 1: Metode virov v REST API-jih

Metoda	Opis
GET	Metodo uporabljamo, ko želimo pridobiti obstoječ vir.
POST	Metodo uporabljamo, ko želimo ustvariti nov vir.
PUT	Metodo uporabljamo, ko želimo posodobiti obstoječ vir.
DELETE	Metodo uporabljamo, ko želimo izbrisati vir.

Vir: (Read the Docs, 2024)

GET:

Metoda GET se uporablja za pridobivanje informacij o viru. Ko odjemalec pošlje zahtevo GET na določen URI, strežnik vrne predstavitev vira, ki lahko vključuje podatke v obliki JSON, XML ali drugega formata. Na primer, zahteva GET ***http://primer.com/api/books/1*** bo vrnila podatke o knjigi z ID-jem 1.

POST:

Metoda POST se uporablja za ustvarjanje novega vira na strežniku. Ko odjemalec pošlje zahtevo POST, strežnik ustvari nov vir na podlagi poslanih podatkov in vrne URI novega vira. Na primer, zahteva POST ***http://primer.com/api/books*** s podatki o novi knjigi bo ustvarila novo knjigo in vrnila URI nove knjige.

PUT:

Metoda PUT se uporablja za posodabljanje obstoječega vira. Ko odjemalec pošlje zahtevo PUT na določen URI, strežnik posodobi vir z novimi podatki. Na primer, zahteva PUT ***http://primer.com/api/books/1*** z novimi podatki bo posodobila knjigo z ID-jem 1.

DELETE:

Metoda DELETE se uporablja za brisanje obstoječega vira. Ko odjemalec pošlje zahtevo DELETE na določen URI, strežnik izbriše vir. Na primer, zahteva DELETE ***http:// primer.com/api/books/1*** bo izbrisala knjigo z ID-jem 1.

Vsaka od teh metod ima specifično vlogo pri manipulaciji z viri v REST API-jih. Uporaba teh metod na pravilen način zagotavlja, da so operacije nad viri jasne, predvidljive in skladne s principi REST. Pomembno je, da so metode virov enotno uporabljene, kar omogoča razvijalcem preprosto razumevanje in uporabo API-ja, ne glede na to, kateri viri so v igri.

Na primer, če uporabljamo metodo GET, vemo, da bomo samo pridobili podatke in ne bomo spremenili stanja vira. Če uporabljamo metodo POST, vemo, da ustvarjamo nov vir, in tako naprej. Ta doslednost v uporabi metod prispeva k enostavnosti uporabe in zanesljivosti REST API-jev.

3.5 HTTP Statusne kode v REST API-jih

V REST API-jih imajo HTTP statusne kode ključno vlogo pri obveščanju odjemalcev o rezultatu njihovih zahtev. Te kode so trimestne številke, ki jih strežnik vrne kot odgovor na zahtevo odjemalca. Vsaka statusna koda sporoča, ali je bila zahteva uspešno obdelana ali je prišlo do napake ali pa so potrebna dodatna dejanja.

3.5.1 Kategorije HTTP statusnih kod

HTTP statusne kode so razdeljene v pet glavnih kategorij, vsaka s specifičnim pomenom:

- **1xx: Informativne kode**
  - Te kode sporočajo, da je zahteva prejeta in da se nadaljuje obdelava.
- **2xx: Uspešne kode**
  - Te kode sporočajo, da je bila zahteva uspešno prejeta, razumljena in sprejeta.
- **3xx: Preusmeritvene kode**
  - Te kode sporočajo, da je za dokončanje zahteve potrebna dodatna akcija, običajno preusmeritev na drug URI.
- **4xx: Napaka odjemalca**
  - Te kode sporočajo, da je prišlo do napake zaradi napačne zahteve s strani odjemalca.
- **5xx: Napaka strežnika**
  - Te kode sporočajo, da je prišlo do napake na strani strežnika, ki ni mogel uspešno obdelati zahteve.

(HTTP Status Codes, 2024)

Tabela 2: Najpogosteje uporabljene HTTP statusne kode v REST API-jih

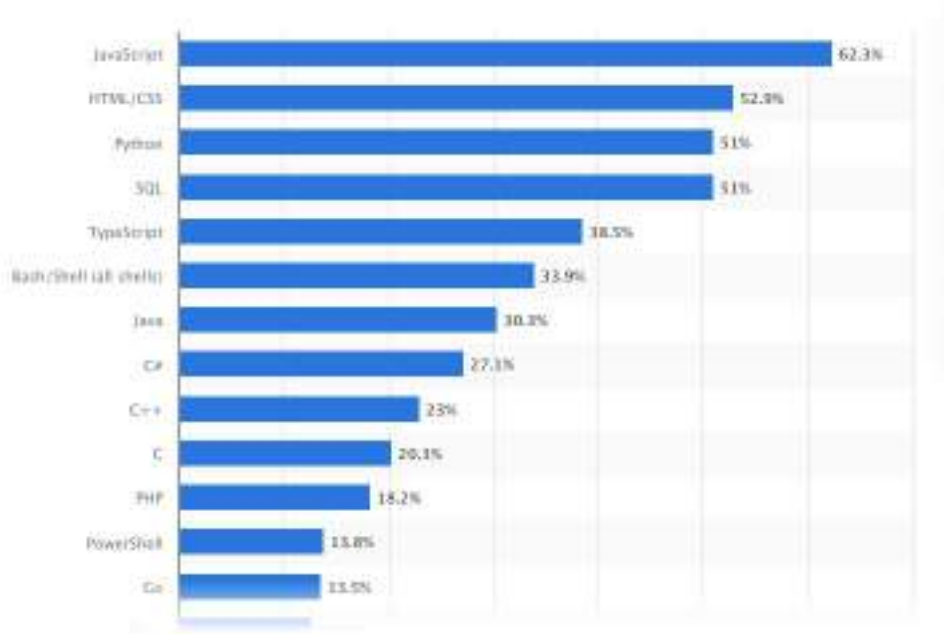
Statusne kode	Pomen	Opis
200 OK	Uspešno	Zahteva je bila uspešno obdelana, rezultat je poslan nazaj odjemalcu.
201 Created	Ustvarjeno	Zahteva je bila uspešno obdelana in ustvarjen je bil nov vir. Strežnik vrne URI novega vira.
204 No Content	Brez vsebine	Zahteva je bila uspešno obdelana, vendar ni nobene vsebine za vrnitev odjemalcu.
301 Moved Permanently	Trajno premaknjeno	Zahtevani vir je bil trajno premaknjen na nov URI. Odjemalec bi moral uporabiti nov URI za prihodnje zahteve.
400 Bad Request	Slaba zahteva	Zahteva je napačna ali ni veljavna. Strežnik ne more obdelati zahteve.
401 Unauthorized	Neavtorizirano	Odjemalec mora avtenticirati zahtevo, da dobi dostop do zahtevanega vira.
403 Forbidden	Prepovedano	Odjemalcu je prepovedano dostopati do zahtevanega vira, tudi če je avtenticiran.
404 Not Found	Ni najdeno	Zahtevani vir ni bil najden na strežniku.
500 Internal Server Error	Napaka strežnika	Prišlo je do splošne napake na strežniku. Zahteva ni bila uspešno obdelana.
501 Not implemented	Ni implementirano	Strežnik ne podpira funkcionalnosti, potrebne za izpolnitev zahteve.

Vir: (HTTP Status Codes, 2024)

4 PYTHON

Python je visokonivojski, interpretirani programski jezik, ki ga je razvil Guido van Rossum in je bil prvič izdan leta 1991. Zasnovan je bil za povečanje produktivnosti razvijalcev in enostavnosti vzdrževanja kode. Python je zaradi svoje preproste in berljive sintakse hitro postal priljubljen v različnih industrijah, kar je povzročilo njegov hiter razvoj in širitev. Danes je Python eden izmed najbolj priljubljenih programskih jezikov na svetu, saj podpira več programskih paradigem, kot so objektno usmerjeno, proceduralno in funkcijsko programiranje (Python Introduction, 2024).

Na spodnji sliki lahko vidimo, da je Python med prvimi petimi najbolj uporabljenimi programskimi jeziki na svetu v letu 2024.



Slika 3: Graf najbolj uporabljenih programskih jezikov na svetu v letu 2024

Vir: (Worldwide Developer Survey - Most Used Languages, 2024)

4.1 Ključne lastnosti Pythona

Enostavnost in berljivost: Python je zasnovan tako, da je koda enostavna za branje in pisanje, kar omogoča hitrejše učenje in razvoj aplikacij. To je še posebej uporabno za začetnike kot tudi za izkušene razvijalce, ki želijo učinkovito pisati in vzdrževati kodo (Python Introduction, 2024).

Vsestranskost: Python se uporablja v različnih industrijah in za različne namene, vključno s spletnim razvojem, znanostjo o podatkih, umetno inteligenco, avtomatizacijo, spletnim strganjem, razvojem iger in še več. Ta vsestranskost je omogočena s širokim naborom knjižnic in ogrodij, kot so Django, Flask, Pandas, NumPy in TensorFlow (What is Python?, 2024).

Obsežna standardna knjižnica: Python vključuje bogato zbirko modulov in knjižnic, ki razvijalcem omogočajo izvajanje različnih funkcij brez potrebe po pisanju dodatne kode (Python, 2024).

4.2 Uporaba Pythona v različnih panogah

Spletni razvoj: Python se pogosto uporablja za razvoj spletnih aplikacij, pri čemer ogrodja, kot sta Django in Flask, omogočata hitro in učinkovito gradnjo spletnih mest. Ta ogrodja ponujajo robustne funkcionalnosti, ki omogočajo varno in skalabilno razvijanje aplikacij (Web Development in Python Guide, 2024).

Strojno učenje in umetna inteligenca: Python je eden izmed glavnih jezikov za analizo podatkov in razvoj modelov strojnega učenja. Knjižnice, kot sta Pandas in TensorFlow, omogočajo analizo podatkov in razvoj naprednih algoritmov, ki jih uporabljajo podatkovni znanstveniki in raziskovalci (What is Python?, 2024).

Avtomatizacija: Python se pogosto uporablja za avtomatizacijo ponavljajočih se nalog, kot so obdelava datotek, spletno strganje in avtomatizacija testov. S tem razvijalci prihranijo čas in zmanjšajo možnost napak, kar omogoča bolj učinkovito delo (What is Python?, 2024).

4.3 Pregled ogrodij

Python podpira več ogrodij, ki omogočajo razvoj od preprostih do kompleksnih aplikacij. Ta poglavja obravnavajo tri ogrodja za spletne aplikacije: Falcon, Flask in Bottle.

4.3.1 Falcon.

Falcon je minimalistično ogrodje za izdelavo REST API-jev, ki se izvaja v Pythonu. Zasnovan je z namenom hitrega obdelovanja HTTP zahtev, kar omogoča razvoj visoko optimiziranih web storitev. Zaradi svoje lahke zasnove Falcon minimizira odvečne procese, kar omogoča hitrejše

odzivne čase, in je idealno za primer uporabe, kjer so zmogljivosti in nizke zakasnitve ključnega pomena (Falcon Documentation , 2024).

Falcon podpira tako sinhrono kot asinhrono programiranje, kar uporabnikom omogoča izbiro najustreznejšega pristopa glede na zahteve aplikacije. Ogrodje ponuja robustno podporo za različne vrste medijskih obravnav, vključno z zahtevami za JSON, URL-encoded formami in multipart podatki. Poleg tega Falcon omogoča preprosto upravljanje sredstev, saj nudi razširjen sistem usmerjanja in preusmerjanja zahtev, kar izboljša organiziranost in modularnost aplikacij (Falcon Documentation , 2024).

Eden ključnih razlogov za popularnost ogrodja Falcon je njegova sposobnost, da ostane izredno učinkovit, tudi ko je pod obremenitvijo, kar ga naredi idealnega za razvoj back-end storitev, ki so ključne za delovanje sodobnih spletnih aplikacij. Njegova skupnost je aktivna in nenehno razvija dodatke, ki razširjajo funkcionalnost ogrodja, obenem pa so na voljo obsežni viri in dokumentacija, ki novim uporabnikom olajšajo začetek uporabe (Falcon Documentation , 2024).

Prednosti ogrodja Falcon:
Hitrost: zasnovan je za visoko zmogljivost in je eden izmed najhitrejših ogrodij za web API-je v Pythonu. Optimiziran je za hitro obdelavo zahtev, kar je še posebej koristno pri obsežnih aplikacijah, kjer so odzivni časi bolj pomembni.
Lahek: Falcon vzdržuje minimalistično zasnovo. Minimalistično jedro zagotavlja, da je ogrodje lahko in ne zahteva veliko virov, kar omogoča boljšo zmogljivost in manjšo obremenitev strežnikov.
Enostavna uporaba: Kljub svoji zmogljivosti, ostaja relativno preprost za uporabo. Ponuja jasne in enostavne vzorce za razvoj API-jev, kar razvijalcem omogoča hitrejši razvoj in manjšo kompleksnost (Python Falcon: Introduction, 2024).

4.3.2 Flask

Flask je mikro ogrodje za izdelavo web aplikacij, ki se izvaja v Pythonu. Njegova glavna značilnost je prilagodljivost in minimalizem. Flask ne predpisuje arhitekture ali ogrodij, temveč zagotavlja preprosto jedro za začetek dela, na katerega lahko razvijalci nato dodajajo dodatne komponente po potrebi (Flask Documentation, 2024).

Flask podpira različne vrste zahtev, od preprostih statičnih strani do kompleksnih web aplikacij s polno funkcionalnostjo. Omogoča enostavno usmerjanje URL-jev in podporo za predloge, ki uporabljajo Jinja2 templating jezik. Poleg tega Flask podpira varne seje za shranjevanje podatkov med zahtevami, kar je izrednega pomena za aplikacije, ki zahtevajo avtentifikacijo in uporabniške seje (Flask Tutorial, 2024).

Eden od razlogov za popularnost ogrodja Flask je njegova izjemna razširljivost. Zahvaljujoč obsežnemu ekosistemu razširitve lahko razvijalci dodajajo funkcionalnosti, kot so obrazci, migracije baz podatkov, avtentifikacija in mnogo več. To omogoča ogrodju Flask, da raste in se prilagaja potrebam projektov, od majhnih osebnih blogov do velikih komercialnih web aplikacij. Njegova skupnost je ena izmed najaktivnejših v Python ekosistemu, kar zagotavlja, da so viri, podpora in dokumentacija vedno na voljo (Flask Documentation, 2024) (Flask Tutorial, 2024).

Prednosti ogrodja Flask:

Prilagodljivost in enostavnost: Flask omogoča razvijalcem, da začnejo z minimalno postavitvijo in po potrebi dodajajo funkcionalnosti. To je idealno za projekte, ki zahtevajo postopno skaliranje (Flask Documentation, 2024).

Velik ekosistem in podpora skupnosti: zahvaljujoč aktivni skupnosti in širokemu naboru razširitev, Flask omogoča razvijalcem dostop do širokega spektra orodij in virov, ki lahko pomagajo pri razvoju aplikacij (Flask Tutorial, 2024).

Dobro dokumentirano: Flask ima eno izmed najbolj obsežnih in dobro organiziranih dokumentacij med Python ogrodji, kar novim razvijalcem olajša učenje in uporabo ogrodja (Flask Documentation, 2024).

4.3.3 Bottle

Bottle je mikro web ogrodje za Python, ki je zasnovano za hitro razvijanje majhnih web aplikacij. S svojo preprosto in lahko zasnovo omogoča razvoj aplikacij z minimalnimi odvisnostmi, saj je celotno ogrodje implementirano v eni sami Pythonovi skripti. To preprostost ceni veliko število razvijalcev, ki iščejo hitre rešitve za manj zahtevne projekte GeeksForGeeks (Bottle Documentation, 2024) (Introduction to Bottle Web Framework - Python, 2024).

Bottle ponuja osnovne funkcionalnosti, kot so usmerjanje, predloge in dostop do podatkovnih baz, kar uporabnikom omogoča enostavno in učinkovito gradnjo aplikacij. Uporablja se lahko za različne namene, od preprostih osebnih projektov do prototipiranja in izobraževalnih orodij. Podpora za WSGI omogoča njegovo integracijo z drugimi Python ogrodji in orodji, kar dodatno povečuje njegovo uporabnost in prilagodljivost (Bottle Documentation, 2024).

Prednosti Bottle:

Enostavna uporaba in postavitev: Bottle je idealen za majhne aplikacije in prototipe, saj omogoča hitro postavitev z minimalno količino kode. Vsebuje vgrajeno podporo za usmerjanje, predloge in dostop do podatkovnih baz, kar poenostavi razvojni proces (Introduction to Bottle Web Framework - Python, 2024).

Lahkotnost in hitrost: zaradi svoje zasnove, ki vključuje minimalno število odvisnosti, je Bottle izredno hiter in učinkovit pri porabi sistemskih virov, kar ga naredi primernega za okolja z omejenimi viri (Bottle Documentation, 2024).

Fleksibilnost: kljub svoji enostavnosti, Bottle omogoča razvijalcem, da uporabljajo razširitve za dodajanje funkcionalnosti, kot so baze podatkov, avtentifikacija in piškotki. To ga naredi prilagodljivega za širši nabor uporabe (Bottle Documentation, 2024).

5 IZDELAVA API-JEV

5.1 Cilj API-ja

Cilj API-ja je bil, da lahko upravlja s tabelo uporabnikov, kjer so shranjeni podatki o njihovih imenih, priimkih in e-poštnih naslovih. API mora omogočati izvajanje osnovnih CRUD operacij − dodajanje novih uporabnikov (POST), pridobivanje podatkov o uporabnikih (GET) in posodabljanje obstoječih zapisov (PUT) ter brisanje uporabnikov (DELETE).

5.2 Vzpostavitev

Za izdelavo API-jev smo uporabljali orodje Visual Studio Code, zaradi priljubljenosti med programerji in enostavnostjo uporabe.

Pri vseh ogrodjih smo uporabljali enak način za shranjevanje podatkov. Izbrali smo orodje SQLAlchemy, ki je ORM (Object Relational Mapper). To nam je omogočalo enostavno interakcijo z bazo podatkov preko objektov v Pythonu, brez potrebe po pisanju neposrednih SQL poizvedb. SQLAlchemy nam omogoča definiranje podatkovnih modelov kot Pythonovih razredov, pri čemer se vsak razred preslika v tabelo v bazi podatkov. Tako lahko ustvarimo, preberemo, posodobimo ali izbrišemo podatke s pomočjo metod, ki jih ponuja SQLAlchemy, kar poenostavi delo z bazo podatkov in poskrbi za večjo berljivost in vzdrževanje kode (SQLAlchemy Documentation, 2024).
Za namestitev orodja SQLAlchemy smo se v direktoriju našega projekta pomaknili v terminal in vpisali naslednji ukaz:

```
pip3 install sqlalchemy
```

Slika 4: Namestitev ORM SQLAlchemy

Vir: (Lastni vir)

Po uspešni namestitvi knjižnice smo ustvarili nov dokument in ga poimenovali models.py, v katerega smo vpisali naslednjo kodo za uvoz komponent.

```
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.orm import sessionmaker, declarative_base
```

Slika 5: Uvoz komponent za SQLAlchemy

Vir: (Lastni vir)

Uvoz zajema naslednje komponente:

− *create_engine:* funkcija, ki se uporablja za inicializacijo povezave do podatkovne baze,

− *Column, Integer, String:* razredi, ki se uporabljajo za definicijo stolpcev v tabelah podatkovne baze, kjer *Integer* in *String* določata tip podatka,

− *sessionmaker:* razred za ustvarjanje sej, ki omogoča delo s transakcijami in poizvedbami v bazi

− *declarative_base:* razred, ki se uporablja za ustvarjanje osnovnega razreda za vse modele, ki so mapirani na tabelo v podatkovni bazi.

Nadaljevali smo s programom:

```
# 1. Ustvarimo motor, ki je povezan z lokalno SQLite bazo
engine = create_engine('sqlite:///users.db')

SessionLocal = sessionmaker(bind=engine)

# 2. Bazna klasa za modele
Base = declarative_base()

# 3. Definiramo model za uporabnika
class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True)
    name = Column(String, nullable=False)
    surname = Column(String, nullable=False)
    email = Column(String, nullable=False)


# 4. Ustvarimo vse tabele v podatkovni bazi
Base.metadata.create_all(engine)

# 5. Ustvarimo sessionmaker, ki je vezan na motor
Session = sessionmaker(bind=engine)
session = Session()
```

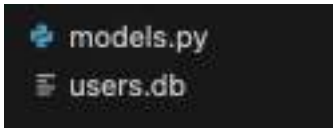
Slika 6: Telo datoteke models.py

Vir: (Lastni vir)

1. Vrstica ustvari povezavo do lokalne SQLite baze podatkov imenovane *users.db*, ki omogoča interakcijo Python aplikacije z bazo.
2. *Base* je osnova za vse modele v SQLAlchemy, ki omogoča definicijo struktur tabel v bazi podatkov s pomočjo Python razredov.
3. Definicija razreda *User,* ki predstavlja tabelo *users* v bazi, z atributi *id, name, surname,* in *email*, kjer *id* služi kot unikaten identifikator za vsak vnos.
4. Ukaz uporabi motor za ustvarjanje tabel v bazi podatkov na podlagi definicij v razredih, ki izhajajo iz *Base.*

5. Ustvari tovarno sej *Session*, ki omogoča izvajanje transakcij v bazi podatkov. Tovarna sej je vzorec oblikovanja, ki omogoča ustvarjanje objektov seje po potrebi, pri čemer vsaka seja omogoča izolirano delo z bazo podatkov. To pomeni, da vsaka transakcija ali poizvedba, izvedena v okviru seje, ne vpliva na druge seje ali na splošno stanje baze, dokler ni potrjena (committed). *bind=engine* poveže tovarno sej z motorjem baze podatkov (*engine*), kar omogoča sejam dostop do baze.; *session* pa je specifična instanca te seje, ki jo lahko uporabimo za delo s podatki (SQLAlchemy Documentation, 2024).

Ko smo program napisali, smo se ponovno postavili v terminal in zagnali program z ukazom "*python3 models.py*".
V našem direktoriju se nam je ustvarila datoteka baze users.db.



Slika 7: Direktorij po kreiranju baze

Vir: (Lastni vir)

5.3 Falcon

Za izdelovanje API-ja v ogrodju Falcon smo nadaljevali z namestitvijo knjižnice falcon z naslednjim ukazom v terminalu:

```
pip3 install falcon
```

Slika 8: Namestitev knjižnice Falcon

Vir: (Lastni vir)

Nadaljevali smo z kreiranjem datoteke resources.py, v katero smo uvozili knjižnici falcon in json ter komponente Session iz knjižnice sqlalchemy.orm ter SessionLocal in User iz lokalne datoteke models.py.

```
import falcon
import json
from sqlalchemy.orm import Session
from models import SessionLocal, User
```

Slika 9: Uvoz v datoteko Resources.py v ogrodju Falcon

Vir: (Lastni vir)

Nadaljevali smo z implementacijo CRUD metod. Metode so implementirane v dveh razredih – »UserListResource« in »UserResource«. »UserListResource« upravlja s seznamom uporabnikov. Metode v tem razredu ne delujejo na specifičnem uporabniku, ampak na kolekciji vseh.

V razredu »UserResource« pa metode upravljajo s specifičnimi zapisi uporabnikov, do katerih dostopamo preko primarnega ključa (ID-ja).

Falcon za poimenovanje metod, ki obravnavajo HTTP zahteve uporablja predpono »*on_*« sledi ime HTTP metode. Koda je razvidna na naslednji sliki po razlagi.

Razred »UserListResource« :

- *on_get(self, req, resp)* : metoda pridobi vse uporabnike iz baze, jih pretvori v seznam slovarjev in nastavi kot odgovor,
    - »*self*« se nanaša na instanco trenutnega razreda,
    - »*req*« (request) predstavlja zahtevo, ki prihaja od uporabnika,
    - »*resp*« (response) je odgovor, ki ga aplikacija pošlje nazaj klientu,

– *on_post(self, req, resp)*: prejme podatke iz zahteve, ustvari nov objekt »*User*« in ga doda v bazo podatkov. Po uspešnem dodajanju se podatki potrdijo (commit) in posodobi se odziv z uspešnim sporočilom.

Razred »UserResource« :

– *on_get(self, req, resp, id)*: dodatni parameter »*id*« se uporablja za identifikacijo specifičnega uporabnika. Metoda poskuša najti uporabnika z danim ID-jem. Če uporabnik obstaja, se podatki vrnejo; če ne, se vrne napaka 404 (uporabnik ni najden),

– *on_put(self, req, resp, id)*: zahteva prejme podatke, ki se uporabijo za posodobitev obstoječega uporabnika. Če uporabnik z danim ID-jem ne obstaja, se vrne napaka 404.

```python
class UserListResource:
    def on_get(self, req, resp):
        session = SessionLocal()
        users = session.query(User).all()
        output = [
            {
            "id": user.id,
            "name": user.name,
            "surname": user.surname,
            "email" : user.email
            } for user in users
            ]
        session.close()
        resp.status = falcon.HTTP_200
        resp.media = output

    def on_post(self, req, resp):
        session = SessionLocal()
        data = json.load(req.bounded_stream)
        new_user = User(name=data["name"], surname=data["surname"], email=data["email"])
        session.add(new_user)
        session.commit()
        session.close()
        resp.status = falcon.HTTP_201
        resp.media = {"message": "User added!"}

class UserResource:
    def on_get(self, req, resp, id):
        session = SessionLocal()
        user = session.query(User).filter_by(id=id).first()
        if user:
            resp.media = {
                "id": user.id,
                "name": user.name,
                "surname": user.surname,
                "email": user.name
            }
        else:
            resp.status = falcon.HTTP_404
            resp.media = {"message": "User not found"}
        session.close()

    def on_put(self, req, resp, id):
        session = SessionLocal()
        user = session.query(User).filter_by(id=id).first()
        if user:
            data = json.load(req.bounded_stream)
            user.name = data["name"]
            user.surname = data["surname"]
            user.email = data["email"]
            session.commit()
            resp.status = falcon.HTTP_201
            resp.media = {"message": "User updated successfully"}
        else:
            resp.status = falcon.HTTP_404
            resp.media = {"message": "User not found"}
        session.close()

    def on_delete(self, req, resp, id):
        session = SessionLocal()
        user = session.query(User).filter_by(id=id).first()
        if user:
            session.delete(user)
            session.commit()
            resp.media = {"message": "User deleted successfully"}
        else:
            resp.status = falcon.HTTP_404
            resp.media = {"message": "User not found"}
        session.close()
```

Slika 10: Implementacija CRUD v ogrodju Falcon

Vir: (Lastni vir)

Po tem, ko je bila funkcionalnost programa napisana, je bil naslednji korak povezovanje metod s končnimi točkami (endpoints), preko katerih odjemalec komunicira z aplikacijo.Ustvarili smo še eno datoteko z imenom »app.py«, v katero smo uvozili knjižnici »falcon« in razrede »UserListResource« in »UserResource« iz datoteke »resources.py«. To nam omogoča uporabo funkcionalnosti, ki smo jo definirali v razredih, znotraj naše aplikacije.

```
import falcon
from resources import UserListResource, UserResource
```

Slika 11: Uvoz v datoteko app.py v ogrodju Falcon

Vir: (Lastni vir)

Ustvarili smo instanco aplikacije *falcon.App(),* ki služi kot osrednja točka za vso logiko usmerjanja in obdelave zahtev v naši aplikaciji.

Za vsak razred vira (»*UserListResource«* in »*UserResource*«) smo ustvarili instanco. Te instance so tiste, ki dejansko izvajajo logiko za vsako končno točko.

Določili smo poti, ki jih naša aplikacija prepoznava, in razrede, ki jih obdeluje. Vsaka pot je povezana z določeno instanco razreda, ki definira, kako se obdelujejo zahteve, ki prihajajo na te poti.

Pot »*/users*« je povezana z instanco »*userList*«, ki obdeluje zahteve za seznam vseh uporabnikov.

Pot »*/users/{id:int}*« je povezana z instanco »*userResource*«, ki obdeluje zahteve za specifičnega uporabnika, identificiranega z »id«.

```
# Instanca Falcon aplikacija
app = falcon.App()


userList = UserListResource()
userResource = UserResource()

# Poti za zahteve
app.add_route('/users', userList)
app.add_route('/users/{id:int}', userResource)
```

Slika 12: Koda datoteke app.py v Falcon ogrodju

Vir: (Lastni vir)

Ostal nam je še zadnji korak, ki je bil zagon aplikacije. Ustvarili smo še datoteko run.py, v katero smo uvozili funkcijo »*make_server*« iz modula »*wsgiref.simple_server*«, ki je del Pythonove standardne knjižnice ter instanco naše aplikacije iz datoteke app.py.

Slika 13: Uvod z datoteko run.py v ogrodju Falcon

Vir: (Lastni vir)

Funkcija »*make_server*« se uporablja za vzpostavitev WSGI strežnika. Natančneje, funkcija vrne instanco WSGI strežnika, ki je pripravljena sprejeti in obdelovati HTTP zahteve (WSGI Reference Library, 2024).

Strežnik smo nastavili tako, da se zažene na lokalnem naslovu (»127.0.0.1«, localhost), na vratih 8000.

S *httpd.serve_forever()* ukazom zaženemo strežnik, ki nenehno posluša zahteve na določenih vratih. Ta ukaz blokira preostalo izvajanje kode, kar pomeni, da bo strežnik ostal aktiven, dokler ga izrecno ne ustavimo (s prekinitvijo izvajanja v konzoli).

Slika 14: Zagon strežnika aplikacije v ogrodju Falcon

Vir: (Lastni vir)

Na naslednji sliki lahko vidimo primer odgovora na GET zahtevo, naslovljeno na endpoint »/users«.

Slika 15: Primer odgovora GET

Vir: (Lastni vir)

5.4 Flask

Po tem, ko smo izdelali prvo aplikacijo v ogrodju Falcon, je bilo precej lažje nadaljevati izdelavo aplikacije v Flask ogrodju, saj deluje na podoben način, le da ima svoje značilnosti.

Najprej smo morali ustvariti podatkovno bazo s SQLAlchemy, kot je opisano v poglavju 5.2. (Vzpostavitev).

Nadaljevali smo z namestitvijo flask z ukazom »pip3 install flask« v terminalu.

Za razliko od Flacon aplikacije, smo pri Flask aplikaciji samo ustvarili datoteko app.py. V njo smo uvozili razred »Flask« in funkcije »request« in »jsonify« iz knjižnice »flask«. Uvoz iz datoteke models.py ostaja enak kot pri ogrodju Falcon.

```
from flask import Flask, request, jsonify
from models import SessionLocal, User
```

Slika 16: Uvoz v datoteko app.py v Flask ogrodju

Vir: (Lastni vir)

Potem smo morali ustvariti instanco razreda Flask. V njem je argument »name«, ki je bližnjica, da Flask ve, kje najti vire, kot so predloge in statični dokumenti (Flask Quickstart, 2024).

```
app = Flask(__name__)
```

Slika 17: Instanca aplikacije Flask

Vir: (Lastni vir)

Flask uporablja dekorator »*@app.route()*« za povezovanje funkcij, ki obravnavajo HTTP zahteve, pri čemer v dekoratorju podamo argumente za URI pot in metodo. Pod dekoratorjem sledi funkcija, ki obdeluje zahtevo, podobno kot v ogrodju Falcon. Razlika je v tem, da v ogrodju Flask za pošiljanje odziva uporabljamo »*return jsonify()*«, ki vrne podatke v JSON formatu, skupaj s statusno kodo.

```python
@app.route('/users/<int:id>', methods=['GET'])
def get_user(id):
    session = SessionLocal()
    user = session.query(User).filter_by(id=id).first()
    if user:
        return jsonify({
            "id": user.id,
            "name": user.name,
            "surname": user.surname,
            "email": user.email
    }), 200
    else:
        return jsonify({"message": "User not found"}), 404
```

Slika 18: Primer GET metode v ogrodju Flask za pridobitev uporabnika po ID-ju

Vir: (Lastni vir)

Sledi slika kode implementacije vseh CRUD operacij.

Slika 19: Implementacija CRUD operacij v ogrodju Flask za upravljanje uporabnikov

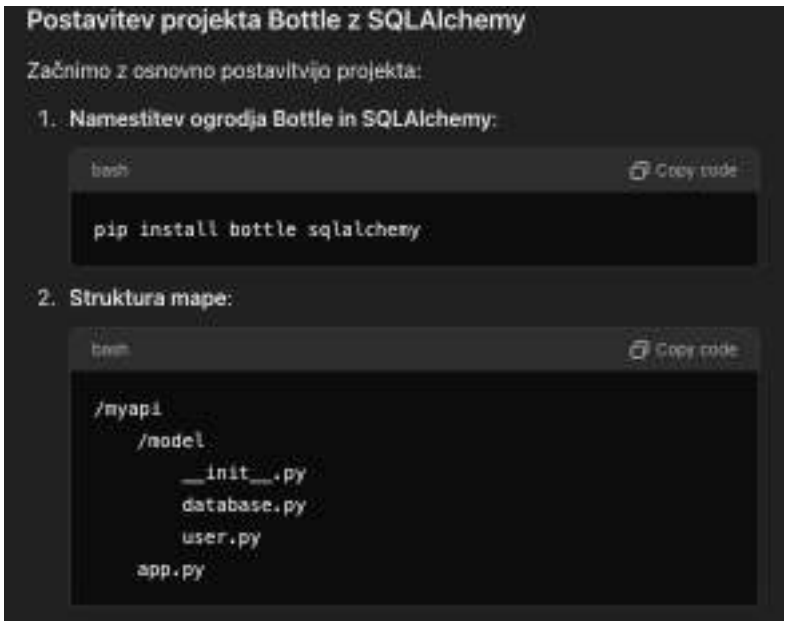
Vir: (Lastni vir)

Zadnji korak je zagon aplikacije. To storimo z uporabo metode *»run()«* na instanci aplikacije »app«. V tej metodi lahko določimo, na katerem naslovu (host) in vratih (port) bo aplikacija delovala, ter ali bo način za odpravljanje napak (debug mode) omogočen.

6 IZDELAVA REST API-JEV Z CHATGPT 4.0

6.1 Bottle

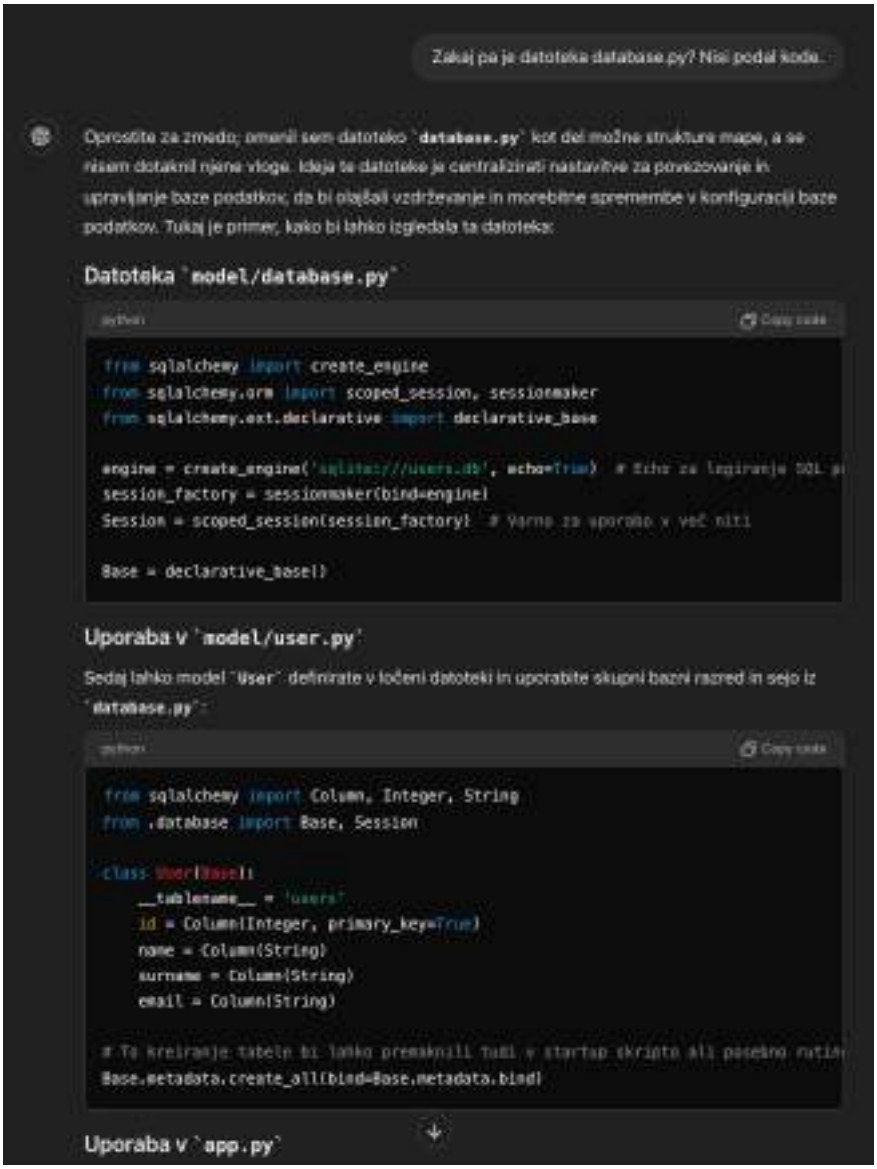
Izdelave API-ja v ogrodju Bottle smo se lotili s ChatGPT 4.0. Začeli smo z jasnim pozivom.
Poziv: »Želim, da si moj tutor za učenje in izdelavo REST API-ja v ogrodju Bottle v Pythonu. Želim izdelati aplikacijo, ki bo imela podatkovno bazo izdelano s SQLAlchemy in bo imela tabelo "users". V njej naj bodo atributi id, name, surname in email. Želim, da bom znal izdelati osnovne CRUD metode in dobro prakso ogrodja Bottle. Želim, da mi razlagaš v primerjavi z ogrodjem Falcon, saj sem v njem že domač. Prosim tudi, da upoštevaš, da se bo testirala učinkovitost, zato hočem izdelati učinkovit API.«
ChatGPT je takoj začel z delom in opisal nam je glavne razlike med ogrodjema Falcon in Bottle. Pristop je bil jasen in navdušujoč, saj je bilo razlaganje strukturirano in enostavno za sledenje. Nadaljeval je z osnovno postavitvijo projekta in ukazom za namestitev »bottle« in »sqlaclhemy« ter podal strukturo projekta.



Slika 20: Navodila za postavitev okolja s ChatGPT

Vir: (Lastni vir)

Sledila je definicija modela »User« s SQLAlchemy, v datoteki user.py ter vsebina funkcionalnosti API-ja v datoteki app.py. Vsebino datoteke database.py pa ni omenil, zato sem mu podal poziv za njegovo napako. Nadaljeval je z razlago in popravki v kodi, ki so prikazani v naslednji sliki.



Slika 21: ChatGPT pozabi kodo

Vir: (Lastni vir)

Sledilo je še nekaj popravkov zaradi napak, ki so se pojavile pri klicanju strežnika. Vse skupaj je bilo 5 napak (error-jev), ki pa jih je potem popravil in je bila aplikacija funkcionalna v celoti. Njegovi popravki so vključevali tudi podrobne razlage in navodila, ki so bila lahko sledljiva.

Slika 22: Popravki s strani ChatGPT

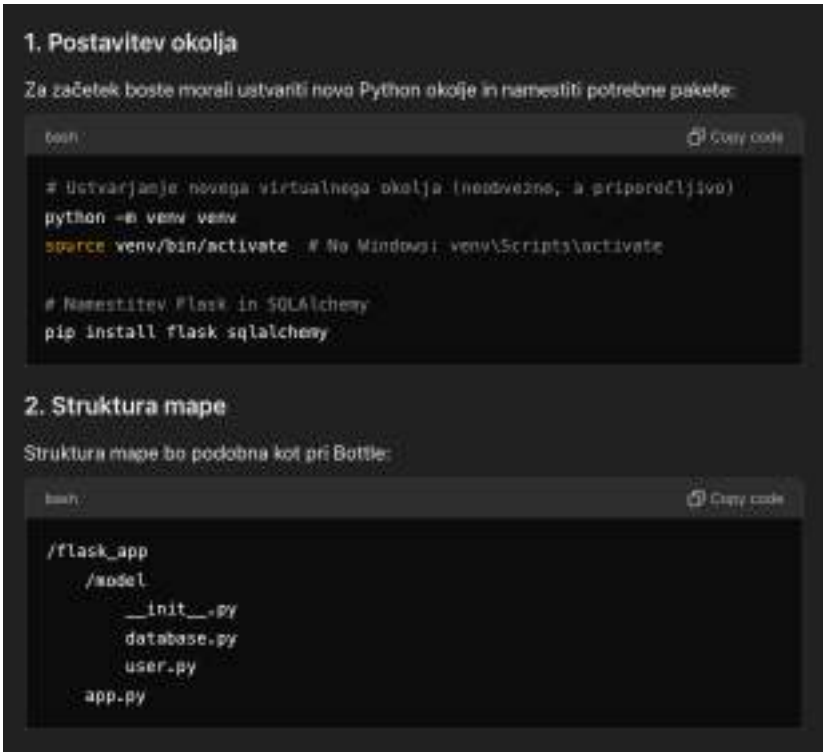
Vir: (Lastni vir)

Koda je bila pripravljena in lahko smo jo implementirali.

6.2 Flask

Za izdelavo primerljive Flask aplikacije s prejšnjo v ogrodju Bottle smo ostali v enakem klepetu s ChatGPT. Nadaljevali smo s pozivom: »Sedaj še izdelaj enako aplikacijo v ogrodju Flask.«

ChatGPT je enako začel z navodili za postavitev okolja in strukture datotek.



Slika 23: Navodila za postavitev okolja s ChatGPT za aplikacijo izdelano s Flask

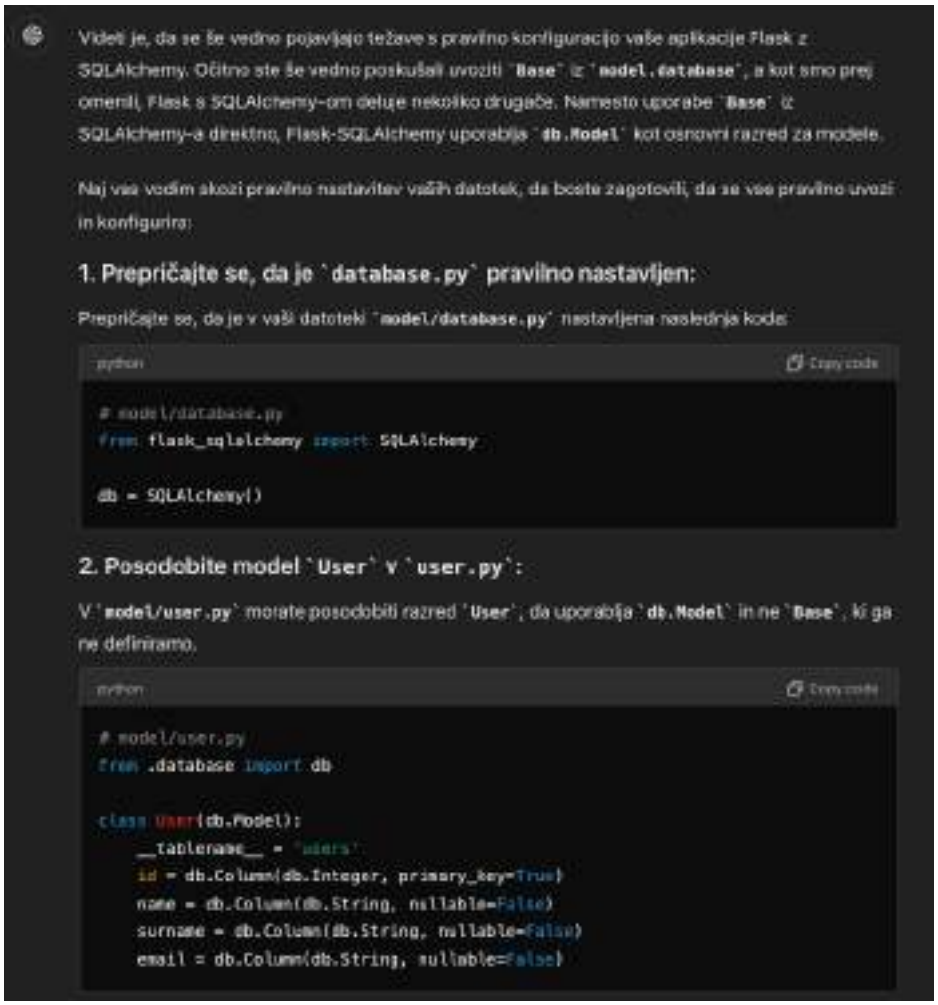
Vir: (Lastni vir)

Nadaljeval je s kodo za datoteke database.py, user.py in app.py. Pri tem primeru ni ostalo nič neobravnavanih datotek.

Slika 24: Navodila za kodo za aplikacijo Flask s ChatGPT

Vir: (Lastni vir)

Ko smo postavili okolje in vpisali kodo v Visual Studo Code smo opazili, da je pozabil na ukaz za namestitev knjižnice »*flask_sqlalchemy*«. To smo popravili z ukazom »*pip3 install flask_sqlalchemy*«. Ob klicanju strežnika so se pojavile ponovno napake pri konfiguraciji podatkovne baze. V klepet smo podali poziv z napako (error-jem) in dobili odgovor z razlago problema in podano rešitev.



Slika 25: ChatGPT odgovarja z razlago napake v kodi

Vir: (Lastni vir)

Koda je bila pripravljena in lahko smo jo implementirali.

7 TESTIRANJE IN RAZISKAVA

7.1 Merjenje zmogljivosti API-jev

Zmogljivost API-jev smo testirali z orodjem Postman, ki je priljubljeno orodje za razvoj in testiranje API-jev. Postman omogoča izvajanje različnih vrst testov, vključno z merjenjem zmogljivosti API-jev. Pri tem smo uporabili parametre **profil obremenitve** (angl. load profile), **število virtualnih uporabnikov (**angl. virtual users) in **trajanje testa** (angl. test duration).

Profil obremenitve določa, kako se obremenitev postopoma povečuje ali zmanjšuje med testom, kar pomaga razumeti, kako se API obnaša pod različnimi pogoji obremenitve.

Število virtualnih uporabnikov predstavlja število hkratnih uporabnikov, ki simulirajo obremenitev API-ja, kar nam omogoča merjenje zmogljivosti in odzivnosti pod različnimi scenariji uporabe.

Trajanje testa določa, kako dolgo traja test, kar nam omogoča opazovanje stabilnosti in zmogljivosti API-ja skozi čas.

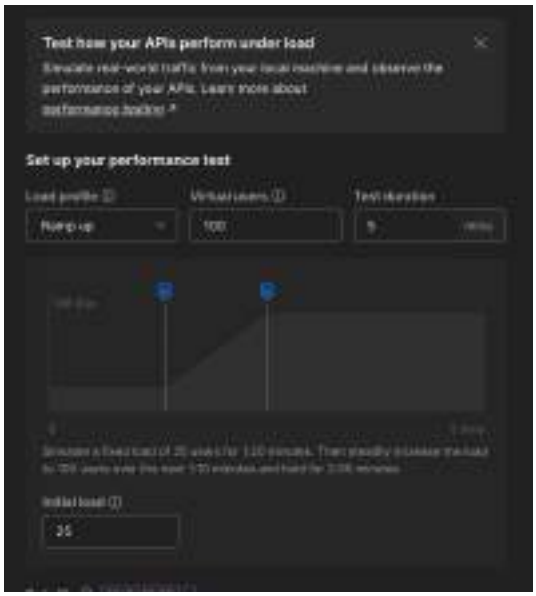
S temi parametri smo lahko natančno izmerili zmogljivost API-jev in pridobili vpogled v to, kako učinkovito obdelujejo zahteve ter kako se odzivajo pod različnimi pogoji obremenitve.

Naše nastavitve:

Profil obremenitve – Ramp up: Ramp up pomeni, da se obremenitev na API postopoma povečuje skozi čas. Na začetku testa se obremenitev začne z nizkim številom zahtev, nato pa se število zahtev postopoma povečuje, dokler ne doseže največjega števila virtualnih uporabnikov. To nam omogoča, da opazujemo, kako API ravna s povečanjem obremenitve.

Število virtualnih uporabnikov – 100: Nastavili smo 100 virtualnih uporabnikov, kar pomeni, da bo med testom simulirano, kot da 100 različnih uporabnikov hkrati pošilja zahteve našemu API-ju. To število je izbrano zato, da vidimo, kako dobro API obvladuje večje število hkratnih zahtev in kako se odziva na obremenitev.

Trajanje testa – 5 minut: Test smo nastavili na trajanje 5 minut, kar pomeni, da bo API obremenjen s simuliranimi zahtevami za celoten čas trajanja testa. To nam omogoča, da opazujemo stabilnost API-ja skozi čas in ugotovimo, ali se odzivni čas poslabša, ko se obremenitev podaljša.



Slika 26: Nastavitve za testiranje v programu Postman

Vir: (Lastni vir)

Med testiranjem smo se osredotočili na metode **GET**, **POST** in **UPDATE**. Metodo **DELETE** smo izpustili, saj je ta metoda zasnovana tako, da izbriše specifičen zapis na podlagi določenega id-ja. Pri testiranju z večjimi obremenitvami lahko vsakokratna izvedba DELETE metode izbriše samo en zapis, kar pomeni, da se pri večkratnem klicanju te metode hitro pojavijo napake, ko poskušamo izbrisati zapis, ki je že bil izbrisan ali ne obstaja več.

Prav tako smo v funkcijo, ki obravnava POST metodo, dodali en try blok za pravilen odziv v primeru praznega polja.

```
def on_post(self, req, resp):
    session = SessionLocal()
    data = json.load(req.bounded_stream)
    try:
        name = data["name"]
        surname = data["surname"]
        email = data["email"]
    except KeyError as e:
        resp.status = falcon.HTTP_404
        resp.media = {"message": f"Manjka: {str(e)}"}
        return
    new_user = User(name=name, surname=surname, email=email)
    session.add(new_user)
    session.commit()
    session.close()
    resp.status = falcon.HTTP_201
    resp.media = {"message": "User added!"}
```

Slika 27: Popravek v kodi

Vir: (Lastni vir)

Vsebina JSON, ki je bila poslana z metodo POST:

```
{
  "name": "new_user_name",
  "surname": "new_user_surname",
  "email": "new@email.com"
}
```

Vsebina JSON, ki je bila poslana z metodo PUT:

```
{
  "name": "updated_user_name",
  "surname": "updated_user_surname",
  "email": "updated@email.com"
}
```

Naši rezultati testiranja:

Tabela 3: Rezultati testiranja z orodjem Postman

Ogrodje	Število vseh zahtev	Prepustnost (zahtev/sekundo)	Povprečni čas (ms)
Falcon	26 720	87,44	327
Flask	13 722	44,63	1 222
Bottle − ChatGPT	25 947	84,94	324
Flask − ChatGPT	17 016	55,71	928

Vir: (Lastni vir)

7.1.1 Falcon

Falcon je pokazal visoko prepustnost in nizko povprečno odzivnost. To pomeni, da je Falcon zelo učinkovit pri obravnavi velikega števila zahtev in ohranja stabilno nizko odzivnost tudi pri visoki obremenitvi.

7.1.2 Flask

Flask ima precej nižjo prepustnost in višji povprečni odzivni čas v primerjavi z ogrodjem Falcon. To kaže, da Flask ni tako zmogljiv pri obdelavi velikega števila zahtev in je manj primeren za visoko obremenjene aplikacije.

7.1.3 Bottle – ChatGPT 4.0

Bottle, z uporabo kode, ustvarjene s pomočjo ChatGPT, se je izkazal za zelo učinkovitega, z rezultati, ki so primerljivi z ogrodjem Falcon. Povprečni odzivni čas in prepustnost sta na visoki ravni, kar kaže, da je Bottle dober konkurent ogrodju Falcon, še posebej pri lažjih aplikacijah.

7.1.4 Flask – ChatGPT

Flask z uporabo kode, ustvarjene s pomočjo ChatGPT, je pokazal boljše rezultate kot naša aplikacija Flask, vendar še vedno zaostaja za ogrodjema Falcon in Bottle glede na zmogljivost. To pomeni, da je z nekaj optimizacijami in usmeritvami mogoče izboljšati zmogljivost Flask aplikacij.

Falcon se je izkazal kot najzmogljivejše ogrodje, s skoraj idealno kombinacijo visoke prepustnosti in nizke odzivnosti. Bottle, ki je bil izdelan s pomočjo ChatGPT, je pokazal impresivne rezultate, kar dokazuje, da je lahko zelo učinkovito ogrodje, še posebej v manjših in srednje velikih projektih. Flask, kljub svoji prilagodljivosti, se sooča z omejitvami glede zmogljivosti, čeprav lahko dodatne optimizacije, kot jih je predlagal ChatGPT, nekoliko izboljšajo njegove rezultate.
Sledijo še posnetki zaslona grafov primerjav.

Falcon − Bottle ChatGPT 4.0

Slika 28: Graf primerjave Falcon − Bottle ChatGPT 4.0

Vir: (Lastni vir)

Falcon − Flask ChatGPT 4.0

Slika 29: Graf primerjave testiranja Falcon − Flask ChatGPT 4.0

Vir: (Lastni vir)

Flask − Flask ChatGPT 4.0

Slika 30: Graf primerjave testiranja Flask − Flask ChatGPT

Vir: (Lastni vir)

7.2 Analiza podpore skupnosti

Za analizo podpore skupnosti smo pregledali dve ključni platformi, ki ju pogosto uporabljajo razvijalci − **Reddit** in **Stack Overflow**. Reddit je ena izmed največjih spletnih skupnosti, kjer se razvijalci povezujejo, razpravljajo o problemih in delijo nasvete. Stack Overflow pa je vodilna platforma za vprašanja in odgovore na področju programiranja, kjer razvijalci iščejo in ponujajo rešitve za različne tehnične izzive.

7.2.1 Falcon

Reddit: Falcon nima svoje posebne skupnosti na Redditu, vendar se omenja znotraj širših Python skupnosti, kot je r/Python. To pomeni, da kljub temu, da ni veliko usmerjene razprave o ogrodju Falcon, obstaja splošna podpora znotraj širših Python programerskih skupin (Reddit: falcon python, 2024).

Stack Overflow: Na Stack Overflow obstajajo specifične oznake za Falcon in falconframework, kjer je nekaj vprašanj (87 vprašanj za falcon in 201 za falconframework). To kaže na aktivnost skupnosti, vendar ni tako močna kot pri bolj priljubljenih ogrodjih (Stack Overflow: falcon, 2024) (Stack Overflow: falconframework, 2024).

Slika 31: Reddit Falcon

Vir: (Reddit: falcon python, 2024)

Slika 32: Stackoverflow Falcon

Vir: (Stack Overflow: falcon, 2024)

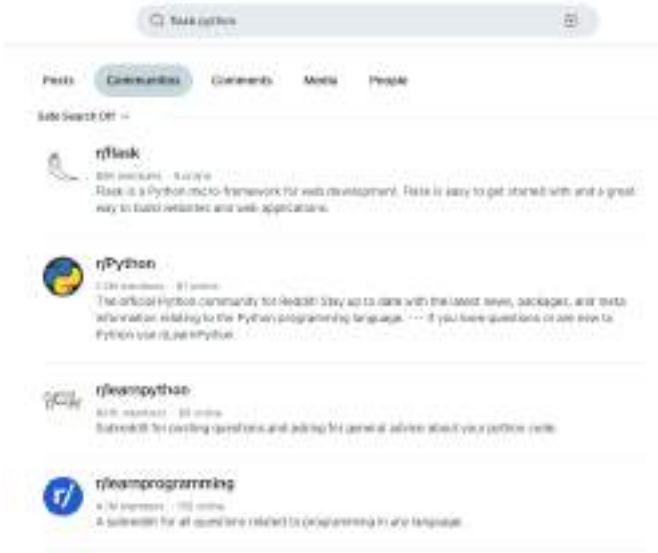
Slika 33: Stackoverflow FalconFramework

Vir: (Stack Overflow: falconframework, 2024)

7.2.2 Flask

Reddit: Flask ima svojo specifično skupnost znotraj r/Flask, ki ima 85 tisoč članov, kar kaže na zelo močno skupnost. Prav tako se Flask pogosto omenja znotraj širših Python skupnosti (Reddit: flask python, 2024).

Stack Overflow: Flask ima zelo močno prisotnost na Stack Overflow z več kot 55,774 vprašanji. To kaže na veliko priljubljenost in podporo v skupnosti (Stack Overflow: flask, 2024).



Slika 34: Reddit Flask

Vir: (Reddit: flask python, 2024)

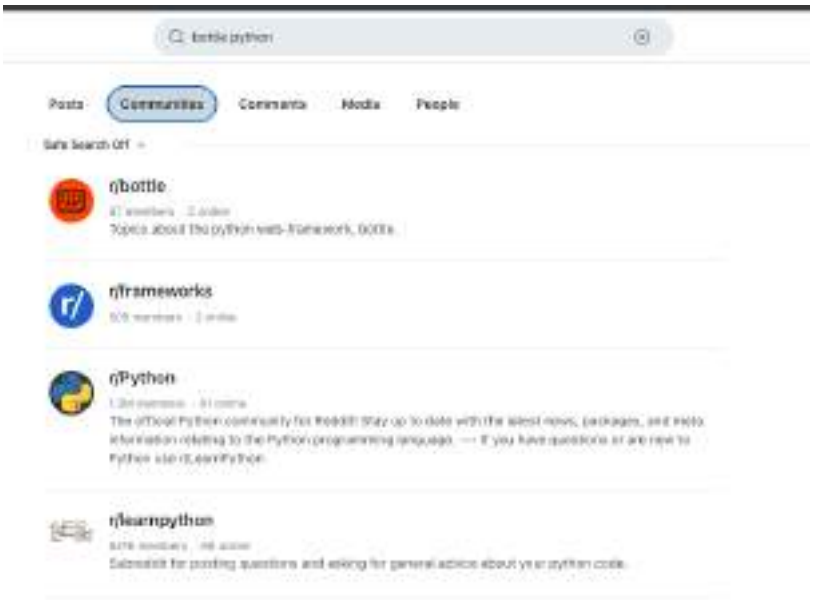
Slika 35: Stackoverflow Flask

Vir: (Stack Overflow: flask, 2024)

7.2.3 Bottle

Reddit: Bottle ima majhno, vendar specifično skupnost na Redditu z 87 člani. To kaže, da je Bottle manj priljubljen kot Flask ali Falcon, vendar ima še vedno nekaj podpore (Reddit: bottle python, 2024).

Stack Overflow: Bottle ima 1487 vprašanj na Stack Overflow, kar kaže, da je manj priljubljen kot Flask, vendar še vedno dovolj prisoten, da ima aktivno skupnost (Stack Overflow: bottle, 2024).



Slika 36: Reddit bottle python

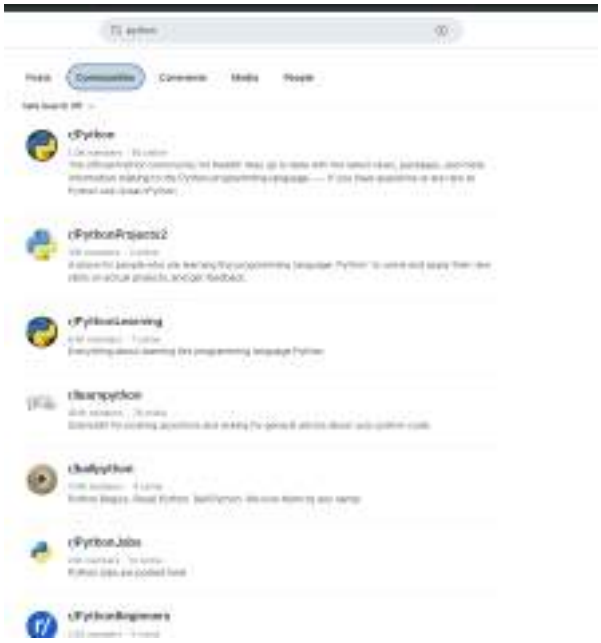
Vir: (Reddit: bottle python, 2024)

Slika 37: Stackoverflow bottle

Vir: (Stack Overflow: bottle, 2024)

7.2.4 Python

- **Reddit**: Python ima ogromno skupnost na Redditu, kjer je prisotnih več kot 1,3 milijonov članov. To pomeni, da ne glede na specifično ogrodje, je na voljo ogromna baza znanja in podpore znotraj splošne Python skupnosti.
- **Stack Overflow**: Python ima več kot 2,2 milijona vprašanj na Stack Overflow, kar je jasen pokazatelj izjemne podpore in priljubljenosti znotraj programerske skupnosti.



Slika 38: Reddit Python

Vir: (Reddit: python, 2024)

Slika 39: Stackoverflow Python

Vir: (Stack Overflow: python, 2024)

7.3 Ugotovitve

Flask ima najmočnejšo podporo skupnosti med našimi ogrodji, kar je vidno tako na Redditu kot na Stack Overflow. Falcon ima manjšo, vendar še vedno pomembno prisotnost, medtem ko Bottle uživa skromno, a še vedno prisotno podporo. Splošna Python skupnost je zelo močna in podpira vsa tri ogrodja, kar pomeni, da je pomoč na voljo tudi iz širše Python skupnosti.

Tabela 4: Analiza podpore skupnosti za ogrodja Flask, Bottle ter Falcon

	Falcon	**Flask**	**Bottle**
Število vprašanj (Avgust 2024)	Stack Overflow: 87 (falcon) + 201 (falconframework)	Stack Overflow: 55,774	Stack Overflow: 1487
Specifična skupnost	Reddit: ni posebne skupnosti	Reddit: r/Flask (85 tisoč članov)	Reddit: 87 članov (majhna skupnost)
Znotraj širših skupnosti	Omenjen v širših Python skupnostih	Omenjen v širših Python skupnostih	Manj omenjen v širših skupnostih

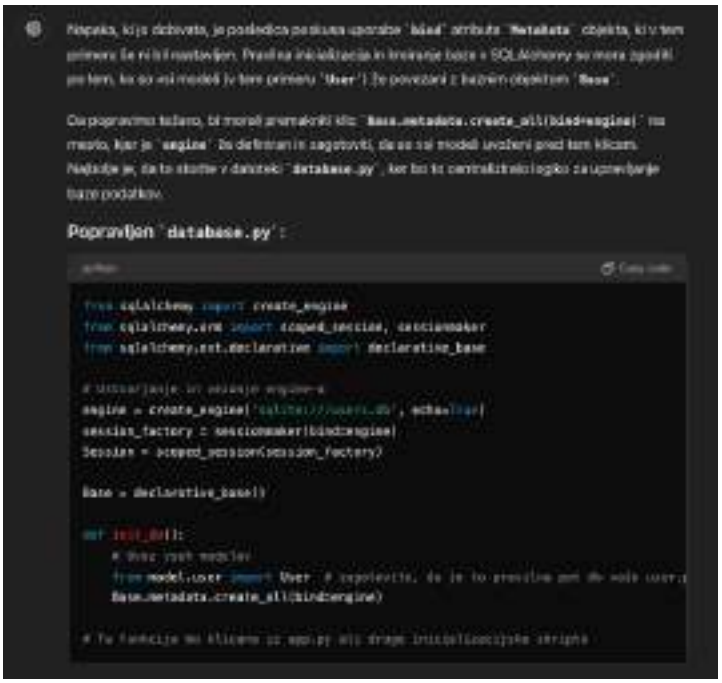
Vir: (Reddit: falcon python, 2024) (Reddit: flask python, 2024) (Reddit: bottle python, 2024) (Stack Overflow: falcon, 2024) (Stack Overflow: falconframework, 2024) (Stack Overflow: flask, 2024) (Stack Overflow: bottle, 2024)

7.4 Učinkovitost ChatGPT 4.0 pri izdelavi API-jev

7.4.1 Pozitivne Strani Uporabe ChatGPT

Jasne in strukturirane razlage: ena od glavnih prednosti uporabe ChatGPT je bila njegova sposobnost, da hitro in jasno predstavi glavne koncepte. Razlage so bile prilagojene našemu predznanju ogrodja Falcon, kar je olajšalo prehod na Bottle. ChatGPT je dosledno uporabljal primerjave med obema ogrodjema, kar nam je omogočilo boljše razumevanje specifičnih lastnosti Bottla.

Učinkovitost pri reševanju težav: ChatGPT je bil zelo učinkovit pri nudenju nasvetov o tem, kako optimizirati API za boljšo zmogljivost. To je bilo še posebej pomembno, saj je bila ena od ključnih zahtev našega projekta zagotoviti, da je API kar se da učinkovit. ChatGPT je predlagal rešitve, kot sta uporaba ustreznih HTTP metod in pravilno predpomnjenje odgovorov, kar je pripomoglo k izboljšanju odzivnega časa aplikacije.



Slika 40: ChatGPT pri reševanju napake

Vir: (Lastni vir)

Pomoč pri implementaciji CRUD metod: Pri implementaciji CRUD metod je ChatGPT podal natančna navodila, kako se lotiti vsakega koraka. Pravilno je strukturiral metode in predlagal uporabo knjižnice SQLAlchemy za interakcijo z bazo podatkov. Na primer:

- **POST metoda za dodajanje novega uporabnika:** ChatGPT je predlagal ustvarjanje metode, ki prejme podatke o novem uporabniku, jih preveri, nato pa jih shrani v bazo.
- **GET metoda za pridobivanje podatkov o uporabnikih:** poudaril je pomen optimizacije poizvedb in pravilnega oblikovanja odgovorov v JSON formatu.

Nasveti glede najboljših praks: ChatGPT je dosledno uporabljal najboljše prakse, kot sta ločitev poslovne logike od predstavitvene plasti in pravilno upravljanje z napakami. To je bilo še posebej pomembno za zagotovitev, da je koda berljiva in enostavna za vzdrževanje.

7.4.2 Negativne strani in napake

Kljub številnim prednostim je bilo nekaj težav in napak, ki smo jih zaznali med uporabo ChatGPT-ja:

Število napak: 5 napak je bilo ugotovljenih med delom. Napake so bile večinoma povezane s sintakso ali z določenimi predpostavkami, ki jih je ChatGPT naredil med pisanjem kode.

Primeri napak:

- **Sintaksne napake:** ChatGPT je v nekaterih primerih predlagal kodo z manjšimi sintaksnimi napakami, ki so zahtevale ročno popravljanje. Na primer, včasih je manjkala vejica ali pa je bil napačno uporabljen klic funkcije.
- **Neoptimalne rešitve:** v nekaterih primerih so bile predlagane rešitve zastarele ali neoptimalne glede na trenutne standarde. Na primer, predlagal je uporabo določene metode, ki ni bila več podprta v najnovejši različici knjižnice.

Časovni zamiki: včasih je prišlo do časovnih zamikov pri pridobivanju odgovorov, kar je nekoliko upočasnilo naš delovni tok. Vendar to ni bilo zelo pogosto in ni bistveno vplivalo na celoten potek dela.

Popravki: medtem ko je bil ChatGPT zelo uporaben, je bilo nekaj primerov, ko smo morali ročno prilagoditi ali prepisati dele kode, da bi zagotovili pravilno delovanje aplikacije. To je vključilo preverjanje in popravljanje sintaksnih napak ter prilagoditev kode specifičnim potrebam projekta.

Sklep: na splošno je bila uporaba ChatGPT-ja pri razvoju API-ja v ogrodju Bottle zelo pozitivna izkušnja. ChatGPT je znatno pospešil proces razvoja in nam omogočil, da smo se hitro prilagodili novemu ogrodju. Kljub manjšim napakam in potrebi po ročnih prilagoditvah, je bil ChatGPT zelo koristen kot tutor in pomočnik pri tem projektu. Izkušnja je bila koristna tudi z vidika nadgradnje našega obstoječega znanja, še posebej pri razumevanju razlik med ogrodjema Falcon in Bottle.

8 PREDLOGI ZA NADALJNJE RAZISKAVE

- **Poglobljena analiza zmogljivosti**

  Nadaljnje raziskave bi lahko vključevale bolj podrobno merjenje zmogljivosti pri različnih obremenitvah in scenarijih uporabe. Testiranje bi lahko zajemalo različne vrste poizvedb in zahtev ter vključilo bolj kompleksne in dinamične scenarije.

- **Kvalitativna analiza podpore skupnosti**

  Izvajanje intervjujev in anket z aktivnimi člani skupnosti Flask, Bottle in Falcon, da bi pridobili globlje vpoglede v kakovost podpore, dostopnost virov ter izzive, s katerimi se razvijalci srečujejo.

- **Učinkovitost učenja z uporabo ChatGPT**

  Izvajanje kontroliranih študij, kjer bi skupina uporabnikov uporabljala ChatGPT, druga skupina pa tradicionalne učne vire, bi omogočilo bolj objektivno oceno učinkovitosti ChatGPT kot učnega orodja.

- **Razvoj API-jev z večjo kompleksnostjo**

  Razvoj različnih vrst API-jev z različnimi zahtevami glede kompleksnosti in funkcionalnosti bi omogočil bolj celovito oceno enostavnosti in učinkovitosti razvoja z različnimi ogrodji.

9 SKLEP

Pri izdelavi diplomskega dela sem se osredotočil na analizo in primerjavo treh priljubljenih ogrodij za razvoj API-jev: Bottle, Flask in Falcon. Osrednji cilj raziskave je bil preveriti hitrost odziva API-jev, podporo skupnosti in učinkovitost pri učenju ter razvoju z uporabo teh orodij. Postavljene hipoteze so bile usmerjene v primerjavo teh ogrodij po ključnih merilih, ki so pomembna za razvijalce in podjetja, ki iščejo optimalne rešitve za svoje projekte.

Prva hipoteza (**H1**), ki je predpostavljala, da ima API, izdelan z ogrodjem Bottle, hitrejši odzivni čas v primerjavi z API-jem, izdelanim z ogrodjem Flask in ogrodjem Falcon, je bila **potrjena** v **poglavju 7.1**. Moje meritve so pokazale, da ogrodje Bottle resnično ponuja nekoliko hitrejši odzivni čas, kar ga postavlja v prednost v okoljih, kjer je hitrost ključnega pomena. Ta rezultat je še posebej pomemben za projekte, kjer se zahtevajo visoke hitrosti procesiranja in minimalni časi zakasnitve.

Druga hipoteza (**H2**), ki je domnevala, da ima ogrodje Flask večjo podporo skupnosti kot ogrodje Falcon, je bila prav tako **potrjena** v **poglavju 7.2**. Analiza spletnih platform StackOverflow in Reddit je pokazala, da Flask uživa precej večjo podporo in priljubljenost med razvijalci. To je pomembno pri izbiri orodja, saj večja podpora skupnosti pomeni lažji dostop do virov, hitrejše reševanje problemov in večje število razpoložljivih razširitev in modulov.

Tretja hipoteza (**H3**), da je za učenje uporabe ogrodja Bottle dovolj uporaba ChatGPT 4.0 brez potrebe po drugih virih, se je izkazala za napačno in je bila **zavržena** v **poglavju 6.1**. Med procesom učenja sem ugotovil, da je ChatGPT sicer izjemno uporaben pripomoček, vendar v določenih primerih ne more nadomestiti obsežnih dokumentacij in specializiranih učnih virov, ki so potrebni za globlje razumevanje in reševanje kompleksnih težav, s katerimi se razvijalci srečujejo pri delu z Bottle.

Četrta hipoteza (**H4**), ki je predvidevala, da lahko z uporabo umetne inteligence v manj korakih izdelamo API s Flask kot podoben API z Bottle, je bila **delno potrjena** v **poglavju 6.2**. Analiza postopkov je pokazala, da je Flask res bolj intuitiven in ponuja več pripravljenih rešitev, kar omogoča hitrejši razvoj API-jev. Vendar je treba upoštevati, da je učinkovitost teh korakov močno odvisna od specifičnih zahtev projekta in kompleksnosti implementacije.

Na podlagi teh ugotovitev lahko zaključim, da čeprav vsak izmed analiziranih ogrodij ponuja določene prednosti, je izbira najbolj primernega orodja odvisna od specifičnih potreb projekta. Bottle je izjemen pri doseganju hitrega odzivnega časa, Flask pa se izkaže kot odlična izbira zaradi močne skupnosti in enostavnosti uporabe. Falcon se je izkazal kot robustno in zmogljivo orodje, vendar z manjšo podporo, kar lahko predstavlja izziv za manj izkušene razvijalce.

Na koncu sem prišel do zaključka, da je za moje nadaljnje delo najbolj smiselna uporaba ogrodja Flask, predvsem zaradi njegove popularnosti, bogate podpore skupnosti in enostavnosti integracije s sodobnimi tehnologijami. Rezultati in ugotovitve te raziskave so mi pomagali sprejeti to odločitev, prav tako pa upam, da bodo koristili tudi drugim razvijalcem pri izbiri pravega orodja za njihove projekte.

10 VIRI

Artificial intelligence. (8. 9 2024). Pridobljeno iz IBM: https://www.ibm.com/think/artificial-intelligence

Artificial Intelligence. (9. 8 2024). Pridobljeno iz Builtin: https://builtin.com/artificial-intelligence

API Pricing. (9. 8 2024). Pridobljeno iz OpenAI: https://openai.com/api/pricing/

ChatGPT Pricing. (9. 8 2024). Pridobljeno iz OpenAI: https://openai.com/chatgpt/pricing/

Generative AI use cases. (9. 8 2024). Pridobljeno iz Google Cloud: https://cloud.google.com/use-cases/generative-ai?hl=en

GPT-4 Research. (14. 3 2023). Pridobljeno iz OpenAI: https://openai.com/index/gpt-4-research/

Documentation. (9. 8 2024). Pridobljeno iz OpenAI: https://platform.openai.com/docs/overview

Chan, J. (2019). *Python API Development Fundamentals : Develop a full-stack web application with Python and Flask.* Packt Publishing .

What is a REST API? (1. 8 2024). Pridobljeno iz Uptrends: https://www.uptrends.com/what-is/rest-api

What is a RESTful API? (1. 8 2024). Pridobljeno iz Amazon Web Services: https://aws.amazon.com/what-is/restful-api/

RESTful API. (1. 8 2024). Pridobljeno iz RESTfulAPI.net.: https://restfulapi.net/

What is REST? (1. 8 2024). Pridobljeno iz Codecademy: https://www.codecademy.com/article/what-is-rest

REST Architectural Constraints. (1. 8 2024). Pridobljeno iz RESTfulAPI.net: https://restfulapi.net/rest-architectural-constraints/

Read the Docs. (1. 8 2024). Pridobljeno iz RESTful API Design: https://restful-api-design.readthedocs.io/en/latest/resources.html

What is JSON? (10. 8 2024). Pridobljeno iz W3Schools: https://www.w3schools.com/whatis/whatis_json.asp

JSON vs XML. (10. 8 2024). Pridobljeno iz RESTfulAPI.net: https://restfulapi.net/json-vs-xml/

What is XML? (10. 8 2024). Pridobljeno iz Amazon Web Services: https://aws.amazon.com/what-is/xml/

HTTP Status Codes. (10. 8 2024). Pridobljeno iz RESTfulAPI.net: https://restfulapi.net/http-status-codes/

Worldwide Developer Survey - Most Used Languages. (15. 8 2024). Pridobljeno iz Statista: https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages

Python Introduction. (15. 8 2024). Pridobljeno iz W3Schools: https://www.w3schools.com/python/python_intro.asp

What is Python? (15. 8 2024). Pridobljeno iz GeeksforGeeks: https://www.geeksforgeeks.org/what-is-python/

Python. (15. 8 2024). Pridobljeno iz Python: https://www.python.org/

Web Development in Python Guide. (15. 8 2024). Pridobljeno iz BrowserStack: https://www.browserstack.com/guide/web-development-in-python-guide

Falcon Documentation . (15. 8 2024). Pridobljeno iz Falcon Documentation: https://falcon.readthedocs.io/en/stable

Python Falcon: Introduction. (15. 8 2024). Pridobljeno iz GeeksforGeeks: https://www.geeksforgeeks.org/python-falcon-introduction/

Flask Documentation. (15. 8 2024). Pridobljeno iz Flask: https://flask.palletsprojects.com/en/3.0.x/

Flask Tutorial. (15. 8 2024). Pridobljeno iz GeeksforGeeks: https://www.geeksforgeeks.org/flask-tutorial/

Bottle Documentation. (15. 8 2024). Pridobljeno iz Bottlepy: https://bottlepy.org/docs/dev/

Introduction to Bottle Web Framework - Python. (15. 8 2024). Pridobljeno iz GeeksforGeeks: https://www.geeksforgeeks.org/introduction-to-bottle-web-framework-python/

SQLAlchemy Documentation. (17. 8 2024). Pridobljeno iz SQLAlchemy: https://www.sqlalchemy.org/

WSGI Reference Library. (17. 8 2024). Pridobljeno iz Python Documentation: https://docs.python.org/3/library/wsgiref.html

Flask Quickstart. (17. 8 2024). Pridobljeno iz Flask Documentation: https://flask.palletsprojects.com/en/3.0.x/quickstart/

Reddit: falcon python. (28. 8 2024). Pridobljeno iz Reddit: https://www.reddit.com/search/?q=falcon+python&type=sr

Stack Overflow: falcon. (28. 8 2024). Pridobljeno iz Stack Overflow: https://stackoverflow.com/questions/tagged/falcon

Stack Overflow: falconframework. (28. 8 2024). Pridobljeno iz Stack Overflow: https://stackoverflow.com/questions/tagged/falconframework

Reddit: flask python. (28. 8 2024). Pridobljeno iz Reddit: https://www.reddit.com/search/?q=flask+python&type=sr

Stack Overflow: flask. (28. 8 2024). Pridobljeno iz Stack Overflow: https://stackoverflow.com/questions/tagged/flask

Reddit: bottle python. (28. 8 2024). Pridobljeno iz Reddit: https://www.reddit.com/search/?q=bottle+python&type=sr

Stack Overflow: bottle. (28. 8 2024). Pridobljeno iz Stack Overflow: https://stackoverflow.com/questions/tagged/bottle

Reddit: python. (28. 8 2024). Pridobljeno iz Reddit: https://www.reddit.com/search/?q=python&type=sr

Stack Overflow: python. (28. 8 2024). Pridobljeno iz Stack Overflow: https://stackoverflow.com/questions/tagged/python

Read the Docs. (10. 8 2024). Pridobljeno iz RESTful API Design: https://restful-api-design.readthedocs.io/en/latest/methods.html