AdvanDEB Platforma

Potpuno obrazloženje arhitekture
Sve komponente platforme i integracija

Verzija 12.24.2 - Revizija arhitekture prosinac 2024.

AdvanDEB Razvojni tim

17. prosinca 2024.

Abstract

Ovaj dokument pruža sveobuhvatno obrazloženje za svaku važnu arhitektonsku odluku u sklopu AdvanDEB Platforme: autentifikaciju i upravljanje korisnicima, osnovnu funkcionalnost Knowledge Buildera, dizajn Modeling Assistanta, arhitekturu integracije i izbor modela podataka. Ovo izvješće objašnjava razloge iza svake projektne odluke, evaluira alternativne pristupe i dokumentira kompromise koji su oblikovali arhitekturu platforme. Ovaj dokument služi kao autoritativna referenca za razumijevanje zašto je platforma izgrađena na ovaj način, podupirući buduće odluke o evoluciji i održavanju.

Važna napomena: Ovaj dokument predstavlja trenutni arhitektonski dizajn, a ne konačnu arhitekturu sustava. Očekuje se da će se arhitektura razvijati kako AdvanDEB projekt prolazi kroz implementaciju, testiranje i stvarnu upotrebu. Projektne odluke dokumentirane ovdje mogu biti revidirane na temelju izazova implementacije, povratnih informacija korisnika, zahtjeva performansi ili novih tehničkih uvida koji se pojave tijekom razvoja.

Contents

1 REVIZIJA ARHITEKTURE - Prosinac 2024. 3
- 1.1 Revidirane uloge komponenti . . . 3
- 1.2 Implikacije za ostale arhitektonske odluke . . . 3

2 Pregled platforme 4
- 2.1 Arhitektura na visokoj razini . . . 4
- 2.2 Tijek korisničke interakcije . . . 4

3 Autentifikacija i upravljanje korisnicima 5
- 3.1 Zašto Google OAuth umjesto vlastitih lozinki? . . . 5
- 3.2 Jedinstvena baza korisnika . . . 6
- 3.3 Arhitektura uloga i mogućnosti . . . 6
- 3.4 Tri mogućnosti . . . 7
- 3.5 Zašto je Knowledge Explorator važan . . . 8

4 Dizajn baze podataka: MongoDB kao temelj 9
- 4.1 Zašto MongoDB umjesto relacijskih baza podataka . . . 9
- 4.2 Dizajn kolekcija i modeliranje podataka . . . 11

5 Sigurnosna arhitektura 12
- 5.1 Obrana u dubinu kroz višestruke metode autentifikacije . . . 12
- 5.2 Arhitektura Agent Frameworka . . . 13
- 5.3 Upit i pretraživanje znanja . . . 14

6 Arhitektura Modeling Assistanta 15
- 6.1 Razvoj modela vođen chatbotom s RAG-om . . . 15
- 6.2 Tijek rada modeliranja temeljen na scenarijima . . . 16
- 6.3 Sastavljanje i predstavljanje modela . . . 17
- 6.4 Integracija s Knowledge Builderom . . . 17

7 Zaključak 18
- 7.1 Sveobuhvatna arhitektura platforme . . . 18
- 7.2 Sintetizirani arhitektonski principi . . . 18
- 7.3 Kriteriji uspjeha za arhitekturu autentifikacije . . . 19
- 7.4 Arhitektura kao živi dizajn . . . 19
- 7.5 Buduća arhitektonska poboljšanja . . . 20

1 REVIZIJA ARHITEKTURE - Prosinac 2024.

Važno: Model arhitekture ažuriran

Ovaj dokument je ažuriran kako bi odražavao značajnu reviziju arhitekture provedenu u prosincu 2024. Uloge komponenti i njihovi odnosi su pojašnjeni kako bi se bolje podržala implementacija i korisničko iskustvo.

1.1 Revidirane uloge komponenti

Arhitektura AdvanDEB Platforme je revidirana kako bi se pojasnile uloge svake komponente:

advandeb-modeling-assistant je sada **Glavno GUI platforme** i jedina ulazna točka za sve korisnike. Pruža:

- Potpuno korisničko sučelje (Vue.js frontend + FastAPI backend)
- Autentifikaciju i upravljanje korisnicima (Google OAuth)
- Kontrolu pristupa temeljenu na ulogama
- Sučelje za razgovor, istraživanje znanja, ingestiju dokumenata i mogućnosti modeliranja
- Integraciju svih mogućnosti platforme u jednom jedinstvenom sučelju

advandeb-knowledge-builder je sada **Toolkit/Paket** (nije samostalna aplikacija). Pruža:

- Python paket s funkcionalnostima izgradnje baze znanja
- Asinhronu obradu dokumenata i ekstrakciju znanja
- AI agente s RAG-om i lokalnim LLM integracijama
- MongoDB pristup za pohranu znanja
- Alate za ekstrakciju činjenica, stiliziranih činjenica i grafova iz dokumenata
- Uvezen i upotrijebljen od strane advandeb-modeling-assistant bez vlastitog GUI-ja

Ova revizija pojednostavljuje model platforme i poboljšava korisničko iskustvo pružanjem jedne koherentne aplikacije koja integrira sve mogućnosti, umjesto da korisnici upravljaju dvjema odvojenim aplikacijama.

1.2 Implikacije za ostale arhitektonske odluke

Ova promjena u modelu komponenti utječe na nekoliko aspekata koji su prethodno dokumentirani:

- **Pristup GUI-ju**: advandeb-knowledge-builder više nema vlastito sučelje za razgovor. Sve interakcije korisnika odvijaju se kroz advandeb-modeling-assistant GUI.
- **Autentifikacija**: Budući da advandeb-knowledge-builder više nije aplikacija s kojom korisnici direktno komuniciraju, ne zahtijeva vlastitu logiku autentifikacije. Umjesto toga, djeluje kao knjižnica koju poziva advandeb-modeling-assistant, koristeći kontekst korisnika koji proslijeđuje pozivajuća aplikacija.
- **Integracija**: Veza između komponenti je sada "isti proces" umjesto "preko API-ja". advandeb-modeling-assistant uvozi advandeb-knowledge-builder kao Python paket i poziva njegove funkcije direktno. Ovo pojednostavljuje obradu pogrešaka, debugging i upravljanje složenošću.

- **Upravljanje podacima**: Obje komponente dijele isti MongoDB i zajednički model podataka, ali samo advandeb-modeling-assistant komunicira s korisnicima ili vanjskim sustavima.

Izvorno nasljeđe: Ova revizija utječe na načine kako razumijemo dizajn sustava, ali ne poništava fundamentalne arhitektonske principe dokumentirane u ovom izvješću. Obrazloženja za MongoDB, JWT tokene, Google OAuth i druge osnovne odluke i dalje vrijede. Ono što se promijenilo je razina na kojoj komponente komuniciraju (in-process umjesto API), ne odluke same.

2 Pregled platforme

2.1 Arhitektura na visokoj razini

AdvanDEB Platforma sada se sastoji od dvije glavne komponente s jasno definiranim ulogama:

1. **advandeb-modeling-assistant**: Glavna platforma i GUI
    - Vue.js frontend za sve korisničke interakcije
    - FastAPI backend za autentifikaciju, API rute i orkestraciju
    - Uvozi i koristi advandeb-knowledge-builder za funkcionalnost znanja
    - Implementira autentifikaciju kroz Google OAuth
    - Upravlja kontrolom pristupa temeljenom na ulogama
    - Pruža sučelje za razgovor koje kombinira istraživanje znanja i razvoj modela
2. **advandeb-knowledge-builder**: Toolkit/Paket za izgradnju baze znanja
    - Python paket instaliran kroz pip
    - Pruža API-je za ekstrakciju znanja, obradu dokumenata i AI agente
    - Direktno pozvan od strane advandeb-modeling-assistant (isti proces)
    - Nema vlastiti GUI ili korisnički sloj
    - Fokusiran na logiku izgradnje baze znanja i upravljanje podacima

2.2 Tijek korisničke interakcije

Tipičan tijek korisničke interakcije sada je pojednostavljen:

1. Korisnik pristupa advandeb-modeling-assistant webu
2. Korisnik se autentificira kroz Google OAuth
3. advandeb-modeling-assistant stvara sesiju s ulogom i mogućnostima korisnika
4. Korisnik komunicira s platformom kroz jedinstveno sučelje koje kombinira:
    - Ekstrakciju znanja iz dokumenata (koristi KB funkcionalnost)
    - Istraživanje postojeće baze znanja (koristi KB funkcionalnost)
    - Razvoj modela s vezama na KB znanje
    - Upravljanje scenarijima i specifikacijama modela

5. advandeb-modeling-assistant poziva advandeb-knowledge-builder funkcije kada je potrebno za operacije baze znanja

6. Svi rezultati su vraćeni kroz isto sučelje bez prekida konteksta

3 Autentifikacija i upravljanje korisnicima

3.1 Zašto Google OAuth umjesto vlastitih lozinki?

Projektna odluka: Google OAuth kao jedini mehanizam autentifikacije

Platforma koristi Google OAuth za autentifikaciju korisnika umjesto implementacije vlastitog sustava korisničko ime/lozinka. Ova odluka oblikuje kako korisnici pristupaju sustavu, što platforma pohranjuje o njima i kako sigurnost funkcionira u praksi.

Većina aplikacija implementira vlastitu autentifikaciju: polja za korisničko ime i lozinku, hashiranje lozinki bcrypt-om, resetiranje lozinki putem emaila, složene zahtjeve za lozinke. Ovaj obrazac je toliko uobičajen da alternativa—delegiranje autentifikacije vanjskom pružatelju—izgleda neobično. Zašto AdvanDEB ne slijedi ovaj obrazac?

AdvanDEB cilja akademske istraživače, gotovo svi posjeduju Google račune kroz svoja sveučilišta. Sveučilišta standardno koriste G Suite (sada Google Workspace) za email, kalendar, dijeljenje dokumenata. Istraživač koji radi na AdvanDEB projektu već ima Google autentifikaciju za Gmail, Google Drive, Google Scholar. Dodavanje još jedne lozinke stvara teret umjesto da poboljšava sigurnost.

S gledišta sigurnosti, OAuth delegiranje smanjuje površinu napada platforme. AdvanDEB nikad ne vidi korisnikovu Google lozinku. Čak i ako bi baza podataka AdvanDEB platforme bila kompromitirana, napadač ne može dobiti materijale za autentifikaciju koji bi omogućili pristup drugim sustavima. Usporedite to s vlastitim lozinkama: istraživači često ponavljaju lozinke između sustava (iako ne bi trebali). Kompromitirana AdvanDEB baza s hashiranim lozinkama mogla bi omogućiti napade grubom silom koji razotkrivaju lozinke koje rade na drugim sustavima. OAuth eliminira ovaj rizik.

OAuth također prebacuje teret implementacije na Google: višefaktorsku autentifikaciju, detekciju sumnjivog prijavljivanja, oporavak računa, blokiranje računa nakon neuspješnih pokušaja. Google ulaže značajne resurse da ove značajke rade dobro. Mali razvojni tim ne bi mogao odgovoriti kompetentno. Fokusiranje na implementaciju vlastite sigurne autentifikacije oduzima vrijeme od značajki koje su jedinstvene za AdvanDEB—izgradnja baze znanja, pomoć u modeliranju.

Postoje legitimni razlozi da neke aplikacije implementiraju vlastitu autentifikaciju. Aplikacije koje zahtijevaju potpunu kontrolu nad podacima korisnika (zdravstveni zapisi koji podliježu HIPAA, financijski sustavi s rigoroznim zahtjevima revizije) često ne mogu delegirati. Aplikacije koje ciljaju korisnike bez Google računa (javne vlade koje se oslanjaju na vlastitu infrastrukturu, korisnici u regijama gdje Google servisi nisu dostupni) moraju podržati lokalne autentifikacije.

AdvanDEB ne ulazi ni u jednu od tih kategorija. Korisnici su akademski istraživači s Google računima. Podaci nisu osobno identificirajući ili regulirani. Jednostavnost OAuth-a nadmašuje autonomiju vlastitih lozinki.

3.2 Jedinstvena baza korisnika

Projektna odluka: Jedinstvena centralizirana baza korisnika

Sve komponente platforme dijele jednu MongoDB kolekciju korisnika koja sadrži korisničke profile, uloge i mogućnosti. advandeb-modeling-assistant upravlja ovom bazom kroz svoj backend.

Kada korisnik se prvi put prijavi kroz Google OAuth, advandeb-modeling-assistant stvara profil korisnika u MongoDB kolekciji `users`. Ovaj profil sadrži Google ID, email, ime, uloge i dozvole. advandeb-knowledge-builder, kada je pozvan od strane modeling assistanta, koristi identifikator korisnika da pripiše znanje pojedincima—činjenice koje stvara kurator_123, stilizirana činjenica koju je pregledao reviewer_456.

Alternativni dizajn gdje bi svaka komponenta upravljala vlastitim korisnicima zahtijevala bi sinkronizaciju: ažuriranje uloga u jednoj bazi bi zahtijevalo ažuriranje u drugoj. Ta sinkronizacija je izvor greške. Što se događa kada mreža zakaže između ažuriranja? Što ako se sinkronizacija pokuša ponovno i dopusti djelomično stanje s različitim ulogama u različitim komponentama?

Dijeljenje baze znači da ažuriranja uloga postaju atomična. Administrator dodijeljuje mogućnost "fact_writer" korisniku kroz modeling assistant GUI. To piše direktno u MongoDB. advandeb-knowledge-builder čita iste podatke korisnika kada validira da korisnik koji podnosi novu činjenicu ima dozvolu `fact_writer`. Nema potrebe za koordinacijom ili mogućnosti za nekonzistentnost.

Ovaj izbor čini komponente manje neovisnim—ako baza korisnika postane nedostupna, obje komponente ne rade. Ali to ne razlikuje se značajno od scenarija gdje su komponente distribuirane ali dijele bazu: distribuirani dizajn također ne uspijeva ako baza nije dostupna. Dodana složenost distribucije ne kupuje otpornost na greške jer sve komponente ovise o istim dijeljenim resursima (MongoDB, autentifikacija).

3.3 Arhitektura uloga i mogućnosti

Projektna odluka: Tri osnovne uloge s opcionim mogućnostima

Arhitektura koristi tri osnovne uloge (Administrator, Knowledge Curator, Knowledge Explorator) s opcionalnim mogućnostima (Agent Access, Analytics Access, Reviewer Status) koje kuratori mogu zatražiti prema potrebi.

Arhitektura modela mogućnosti razlikuje između *tko ste* (vaša osnovna razina pristupa platformi) i *što možete raditi* (vaše dozvole za specifične operacije). Ova razdvojenost rješava fundamentalni izazov u dizajnu dozvola: istraživači trebaju različite mogućnosti u različitim vremenima njihovog rada, ali njihov osnovni identitet i razina pristupa ostaje stabilna.

Istraživačev identitet kao kuratora je stabilan, ali njihova potreba za pristupom agentima ili mogućnostima izvoza podataka varira s vremenom. Novi korisnik ne treba pristup agentima odmah—prvo trebaju razumjeti platformu i kurirati nešto znanja. Nakon stjecanja iskustva, mogu zatražiti pristup agentima za skaliranje svog rada. Još kasnije, ako pok ažu ekspertizu, mogu zatražiti status revizora. Ova postupna progresija odražava prirodnu evoluciju tijeka rada umjesto prisiljavanja korisnika u rigidne kategorije.

Arhitektura implementira ovo kroz osnovne uloge plus opcionalne mogućnosti. Tri

osnovne uloge odražavaju istinske razlike u razini pristupa platformi: Administrator (upravlja platformom), Knowledge Curator (stvara i predaje sadržaj za pregled), Knowledge Explorator (čita sadržaj). Osnovni kuratori mogu uploadati dokumente, stvarati činjenice i predavati znanje za pregled—osnovni tijek doprinosa. Mogućnosti predstavljaju dodatne dozvole koje kuratori zahtijevaju prema potrebi: Agent Access (pokretanje AI agenata), Analytics Access (masovni izvoz podataka), Reviewer Status (odobravanje prijava od drugih kuratora). Ovaj model pruža i jednostavnost (tri osnovne uloge pokrivaju sve korisnike) i fleksibilnost (mogućnosti se kombiniraju slobodno kako bi odgovarale individualnim tijekovima rada).

Ključni uvid je prepoznavanje da istraživači prirodno nose više šešira. Isti biolog uploada radove (aktivnost kuratora), pokreće agente ekstrakcije za obradu literature na skali (pristup agentima), i izvozi agregirane podatke za analizu (pristup analitici), često unutar jedne istraživačke sesije. Umjesto prisiljavanja da prebacuju između identiteta uloga ili da budu trajno zaključani u jednoj ulozi, model mogućnosti dopušta da njihove dozvole rastu organski s njihovim potrebama i dokazanim povjerenjem.

Tijek zahtjeva za mogućnostima ugrađuje učenje i izgradnju povjerenja u evoluciju dozvola. Novi korisnici započinju s osnovnim pristupom kuratora, uče sustav i doprinose znanju. Kada trebaju pristup agentima, zahtijevaju ga s obrazloženjem koje objašnjava njihov slučaj uporabe. Administratori pregledavaju zahtjev, razmatrajući korisničku evidenciju i potrebe. Ovaj model postupnog pristupa usklađuje sigurnost (mogućnosti dodijeljene na temelju dokazanog povjerenja) s upotrebljivošću (korisnici nisu preopterećeni dozvolama koje im još ne trebaju).

Ovaj pristup prihvaća da su neke mogućnosti rijetko potrebne a druge uobičajene. Većina kuratora će na kraju zatražiti status revizora jer je pregled prirodna evolucija doprinošenja. Manje ih će trebati pristup analitici jer masovni izvoz podataka služi specijaliziranim slučajevima uporabe. Model se prirodno prilagođava ovoj distribuciji—uobičajene mogućnosti se rutinski odobravaju dok neobični zahtjevi izazivaju pomnu provjeru.

3.4 Tri mogućnosti

Projektna odluka: Agent Access, Analytics Access i Reviewer Status kao zasebno zahtjevne mogućnosti

Ove specifične tri mogućnosti predstavljaju smislene granice dozvola umjesto proizvoljnih podjela.

Svaka mogućnost odgovara istinskoj sigurnosnoj ili kvalitativnoj zabrinutosti koja opravdava tretiranje kao odvojiva dozvola. Agent Access kontrolira sposobnost pokretanja AI agenata koji čine vanjske API pozive (npr. pružateljima LLM-a) i troše sistemske kvote. Agenti s lošim ponašanjem mogli bi nagomilati troškove API-ja ili degradirati performanse sustava. Ne treba svakom kuratoru ova moć—mnogi rade učinkovito ručno stvarajući unose znanja. Oni kojima trebaju agenti (za masovnu ekstrakciju iz mnogih radova) mogu zatražiti pristup s obrazloženjem koje dopušta administratorima procijeniti legitimnost zahtjeva.

Mogućnost postoji zasebno jer profil rizika se razlikuje od osnovnog kuriranja. Kurator koji ručno stvara činjenice može napraviti štetu samo ljudskom brzinom—unos pogrešnih podataka, predavanje sadržaja niske kvalitete. Agent koji radi u petlji može prouzročiti haos strojnom brzinom—tisuće API poziva, pisanja u bazu koja preopterećuju infrastrukturu,

iscrpljene kvote. Ova razlika rizika opravdava vrata za odobrenje.

Analytics Access kontrolira masovni izvoz podataka i generiranje API ključeva. Kuratori mogu izvoziti pojedinačne činjenice ili male skupove podataka za vlastitu upotrebu bez ove mogućnosti. Analytics Access omogućava izvoz cijele baze znanja, generiranje API ključeva sa širokim dozvolama čitanja i pokretanje složenih upita koji bi mogli utjecati na performanse baze podataka. Ova mogućnost postoji jer masovni izvoz omogućava eksfil traciju podataka—zlonamjerni korisnik mogao bi preuzeti sve i ponovno objaviti drugdje. Iako je mnogo znanja javno, agregiranje ga predstavlja vrijednost koju korisnici možda žele zaštititi.

Mogućnost nije potrebna za normalne istraživačke tijekove rada. Biolog koji modelira populacije riba treba upitati relevantne podatke o vrstama, ne izvoziti cijelu bazu podataka. Analytics Access služi istraživačima koji rade meta-analizu preko mnogih područja, data scientistima koji treniraju ML modele na korpusu znanja, ili institucionalnim partnerima koji žele održavati lokalne kopije. Ovi legitimni slučajevi uporabe opravdavaju dodjelu mogućnosti, ali predstavljaju mali dio korisnika.

Reviewer Status kontrolira pristup redu za pregled i sposobnost odobravanja ili odbijanja predanog znanja. Kontrola kvalitete je ključna za održavanje kredibiliteta platforme—loši pregledi primaju loš sadržaj dok pretjerano odbijanje obeshrabruje doprinositelje. Ne bi svaki iskusni kurator trebao biti revizor; neki su domenski stručnjaci ugodni s doprinosom znanju ali ne ugodni s prosuđivanjem rada drugih. Obrnuto, neki stručni revizori mogu malo doprinositi sami dok pružaju vrijednu kontrolu kvalitete.

Tretiranje pregleda kao mogućnosti umjesto zasebne uloge priznaje da su pregled i kuriranje povezane ali različite aktivnosti. Ista osoba može predati činjenicu (kao kurator) i kasnije pregledati tuđu činjenicu (sa statusom revizora). Ovaj obrazac dvostruke aktivnosti je prirodan u znanstvenoj suradnji—domenski stručnjaci i doprinose i evaluiraju doprinose.

Ove tri mogućnosti proizašle su iz analize obrazaca dozvola i sigurnosnih zahtjeva. Ispitali smo što kuratori rade i identificirali granice dozvola sa sigurnosnim ili kvalitativnim implikacijama vrijednim zaštite. Broj je tri jer toliko smislenih granica postoji: operacije agenata (rizik troškova i performansi), masovni pristup podacima (rizik eksfiltracije), i kontrola kvalitete (rizik integriteta sadržaja). Svaka mogućnost adresira različitu zabrinutost koja opravdava tijek odobrenja.

3.5 Zašto je Knowledge Explorator važan

Projektna odluka: Pružiti ulogu samo-za-čitanje za korisnike koji ne doprinose

Platforma nudi ulogu za korisnike koji pregledavaju znanje bez doprinosa, umjesto zahtijevanja da svi korisnici budu potencijalni doprinositelji.

Uloga Knowledge Exploratora odražava prepoznavanje da nisu svi vrijedni korisnici platforme doprinositelji sadržaja. Studenti koji uče o domeni, novinari koji istražuju članke, kreatori politika koji prikupljaju informacije, i zainteresirani javni članovi koji istražuju znanost možda nikad neće uploadati rad ili stvoriti činjenicu. Ako je kurator minimalna potrebna uloga, tim korisnicima je odbijen pristup ili su primorani zatražiti dozvole koje ne trebaju i neće koristiti.

Pružanje uloge samo-za-čitanje snizuje barijeru angažmana platforme. Odobrenje je jednostavno jer exploratori ne mogu oštetiti podatke ili potrošiti značajne resurse.

Administrator može odobriti zahtjeve exploratora brzo, možda automatski, znajući da je rizik minimalan. Neki exploratori će na kraju postati kuratori kako stječu povjerenje i identificiraju praznine koje bi mogli ispuniti. Ova prirodna progresija od potrošača ka doprinositelju je uobičajena u online zajednicama; blokiranje toga s nepotrebnim vratima dozvola bi oštetilo rast zajednice.

Uloga također služi institucionalnim slučajevima uporabe. Sveučilište možda želi da svi diplomski studenti biologije pristupe platformi kao dio nastave bez da ti studenti trebaju dozvole kuratora. Istraživački institut možda pruža pristup exploratora administrativnom osoblju koje treba potražiti informacije ali ne bi trebalo doprinositi. Ovi slučajevi uporabe zahtijevaju laganu razinu dozvole koja ne nosi odgovornosti kuriranja.

Iz perspektive skaliranja, exploratori generiraju prihod-negativno opterećenje (potrošnja bez doprinosa) ali to je prihvatljivo u službi misije platforme. Dijeljenje znanstvenog znanja ima koristi od širokog pristupa, a računalni trošak posluživanja upita čitanja je nizak. Ograničavanje pristupa samo na doprinositelje stvorilo bi zatvorenu zajednicu koja proturječi vrijednostima otvorene znanosti koje motiviraju platformu.

Exploratori imaju mogućnosti poput spremanja pretraživanja, stvaranja privatnih bilješki, i izvoza malih skupova podataka za osobnu upotrebu. Ove značajke omogućavaju produktivnu upotrebu bez ugrožavanja dijeljene baze znanja. Značajka privatnih bilješki specifično podržava slučaj uporabe učenja—studenti mogu praviti bilješke o konceptima bez da te bilješke zagađuju javni prostor znanja.

Razlika između exploratora i kuratora je fundamentalna: kuratori mogu predložiti promjene bazi znanja predajući činjenice za pregled, dok exploratori imaju pristup samo-za-čitanje. Čak i bez opcionalnih mogućnosti, osnovna uloga kuratora omogućava osnovni tijek doprinosa—uploadanje dokumenata, stvaranje činjenica, predavanje znanja za pregled. Explorator komunicira "ovaj račun je za čitanje i učenje" dok kurator komunicira "ovaj račun može doprinositi bazi znanja." Razlika nije o čekanju dozvola već o namjeravanoj upotrebi: potrošnja naspram doprinosa.

4 Dizajn baze podataka: MongoDB kao temelj

4.1 Zašto MongoDB umjesto relacijskih baza podataka

Projektna odluka: Koristiti MongoDB kao primarnu bazu podataka za cijelu platformu

MongoDB je odabran kao temeljna tehnologija baze podataka za AdvanDEB, prvenstveno vođena zahtjevima grafa znanja i baze znanja, s upravljanjem korisnicima izgrađenim na istoj infrastrukturi radi operativne jednostavnosti.

Izbor MongoDB-a je fundamentalno vođen osnovnom misijom platforme: upravljanje umreženim grafom znanja i skalabilnom bazom znanja za istraživanje bioenergetike. Grafovi znanja sastoje se od heterogenih čvorova (vrste, fiziološki procesi, metabolički putevi, bioenergetički parametri) s varijabilnim svojstvima i odnosima. Činjenice izvučene iz literature imaju različite strukture ovisno o sadržaju. Stilizirane činjenice sintetiziraju varijabilan broj izvornih činjenica. Ova inherentna heterogenost čini rigidne relacione sheme problematičnim—svaki novi tip entiteta ili obrazac odnosa zahtijevao bi migracije sheme.

MongoDB-ov model dokumenta prirodno predstavlja strukture grafa: čvorovi su dokumenti

s fleksibilnim svojstvima, bridovi su ugrađeni nizovi ili reference. Čvor vrste može imati {taxonomy, habitat, metabolic_rate, reproduction_parameters} dok čvor fiziološkog procesa ima {energy_allocation, transport_mechanisms, temporal_dynamics}. Oba koegzistiraju u istoj kolekciji s različitim shemama. Upiti grafa poput "nađi sve čvorove povezane s čvorom X unutar 2 skoka" prevode se u MongoDB agregacijske cjevovode, dok isti upit u relacionim bazama podataka zahtijeva rekurzivne CTE-ove ili višestruke self-joinove koji slabo perfomiraju na skali.

Zahtjevi skalabilnosti za umrežene grafove znanja snažno favoriziraju baze dokumenata. Kako baza znanja raste na tisuće međusobno povezanih entiteta, MongoDB-ovo horizontalno skaliranje kroz sharding omogućava distribuciju particija grafa preko servera. Upiti grafa ostaju efikasni jer MongoDB može paralelizirati preko shardova. PostgreSQL sa grafovskim ekstenzijama (poput AGE) pruža mogućnosti grafa ali sharding je složen i manje zreo. Dedicirane baze grafa (Neo4j, ArangoDB) izvrsne su u upitima grafa ali dodaju operativnu složenost—pokretanje dva sustava baze podataka (graf za znanje, relacioni za korisnike) množi overhead.

Jednom kad je MongoDB odabran za skladištenje grafa znanja (primarni podaci platforme), korištenje ga za upravljanje korisnicima pruža operativnu konzistenciju umjesto uvođenja druge tehnologije baze podataka. Metapodaci korisnika zapravo imaju koristi od MongoDB-ove fleksibilnosti: strukture afilijacija variraju preko institucija, istraživačka područja koriste različite taksonomije, i buduće značajke (ORCID integracija, institucionalne uloge, zapisi publikacija) mogu se dodati bez migracija.

Slučaj uporabe orijentiran grafu oblikuje specifične MongoDB značajke koje se koriste: složeni indeksi na vezama čvorova (source_id + edge_type + target_id) omogućavaju brzo prelaženje grafa, agregacijski cjevovodi implementiraju algoritme grafa (najkraći putevi, detekcija zajednice), i fleksibilna shema dopušta evoluirajuće ontologije grafa bez zastoja. Ove graf-centrične značajke razlikuju MongoDB od tradicionalnih baza dokumenata optimiziranih za CRUD operacije.

Niz mogućnosti u korisničkim dokumentima ilustrira sekundarne prednosti MongoDB-a za upravljanje korisnicima. U relacionom modelu, mogućnosti bi zahtijevale spojnu tablicu sa JOIN-ovima za provjere dozvola. MongoDB pohranjuje mogućnosti kao nizove, čineći provjere trivijalnima i ažuriranja jednostavnima. Međutim, ova fleksibilnost upravljanja korisnicima je zgodan nusproizvod, a ne primarni pokretač—platforma bi koristila MongoDB čak i ako bi upravljanje korisnicima bilo složenije, jer su zahtjevi grafa znanja najvažniji.

JSON-nativna pohrana pruža besprijekoran tijek podataka: znanje izvučeno iz dokumenata je JSON, MongoDB ga pohranjuje kao BSON, HTTP API-ji vraćaju JSON. Ovo usklađivanje proteže se na korisničke podatke, ali važnije na entitete znanja gdje se složene ugniježđene strukture (čvorovi grafa sa svojstvima, činjenice s nizovima entiteta, stilizirane činjenice s referencama dokaza) prirodno preslikavaju u ugniježđeni JSON bez nepodudarnosti impedancije ORM-a.

Operativna jednostavnost arhitekture jedne baze podataka postaje važnija kako volumeni podataka rastu. Primarni podaci platforme—graf znanja—će skalirati na gigabajte ili terabajte (tisuće radova, milijuni činjenica, složeni međusobno povezani grafovi). MongoDB stručnost, procedure sigurnosnog kopiranja, optimizacija upita i infrastruktura praćenja razvijena za pohranu znanja automatski se primjenjuju na korisničke podatke. Dodavanje PostgreSQL-a bi podijelilo operativni fokus bez rješavanja osnovnog izazova: skalabilna pohrana umreženog grafa znanja.

Garancije konzistentnosti u MongoDB-u su se dramatično poboljšale s transakcijama više dokumenata (uvedenim u MongoDB 4.0, zrelim u 4.4). Stvaranje činjenice i povezivanje

s čvorom grafa znanja može se dogoditi atomski. Dok su PostgreSQL-ove transakcije zrelije, MongoDB-ove sposobnosti su dovoljne za operacije upravljanja znanjem. Obrasci transakcija platforme (stvaranje entiteta znanja + ažuriranje odnosa) dobro pristaju MongoDB-ovom modelu transakcija.

Odluka eksplicitno priznaje MongoDB-ove slabosti za određene obrasce upita. Složeni analitički upiti koji obuhvaćaju mnoge kolekcije sa zamršenim JOIN-ovima bili bi lakši u PostgreSQL-u. Međutim, obrasci upita platforme favoriziraju operacije dokumenata i grafa: "nađi činjenice koje odgovaraju tekstualnom upitu", "prelazi graf od čvora X", "dohvati stiliziranu činjenicu sa svim izvornim činjenicama", "nađi entitete povezane kroz put." Ovi obrasci koriste MongoDB-ove snage (tekstualno pretraživanje, agregacijski cjevovodi, denormalizirana pohrana) umjesto njegovih slabosti (složeni JOIN-ovi).

Alternativne NoSQL baze podataka evaluirane su u odnosu na zahtjeve grafa znanja:

- **Neo4j**: Superiorniji upiti grafa ali dodaje raznolikost baza podataka, zahtijeva učenje Cypher-a, i sharding je složen
- **ArangoDB**: Multi-model (dokument + graf) ali manje zreo ekosustav i manja zajednica
- **Cassandra**: Optimiziran za pisanja i vremenske serije ali upiti grafa su teški
- **DynamoDB**: Upravljani NoSQL ali upiti grafa zahtijevaju složene obrasce pristupa i troškovi se nepredvidivo skaliraju

MongoDB balansira mogućnosti grafa (dovoljno dobre), fleksibilnost dokumenata (izvrsna), operativnu zrelost (proizvodnjom dokazana), i podršku zajednice (opsežna). Za mali tim koji gradi istraživačku platformu, MongoDB-ov profil "dobar u mnogim stvarima" nadmašuje profile specijaliziranih baza podataka "izvrsne u jednoj stvari".

4.2 Dizajn kolekcija i modeliranje podataka

Projektna odluka: Četiri kolekcije za korisničke podatke—users, capability_requests, api_keys, audit_logs

Podaci povezani s korisnicima organizirani su u ove četiri kolekcije umjesto jedne kolekcije ili normalizirane relacione sheme.

Kolekcija users predstavlja trenutno stanje identiteta korisnika i dozvola. Svaki dokument sadrži sve informacije potrebne za autentifikaciju i autorizaciju korisnika: njihov Google ID (nepromjenjiv, koristi se za lookup), email i ime (od Googlea, mogu se promijeniti), uloga i mogućnosti (kontrolirane platformom), status (aktivan/suspendiran/na čekanju), i metapodaci o njihovoj afilijaciji i istraživačkim interesima. Ugradnja metapodataka direktno u korisnički dokument umjesto normalizacije u zasebnu tablicu odražava MongoDB najbolje prakse—podaci koji se uvijek upituju zajedno trebaju živjeti zajedno.

Odluka pohranjivanja mogućnosti kao niza u korisničkom dokumentu zaslužuje posebno objašnjenje jer je u sukobu s relacionom normalizacijom. U trećoj normalnoj formi, odnos mnogo-prema-mnogo (korisnici imaju mnogo mogućnosti, mogućnosti pripadaju mnogim korisnicima) bi koristio spojnu tablicu. Ovaj normalizirani pristup optimizira integritet podataka—dodavanje novog tipa mogućnosti ne zahtijeva doticanje postojećih korisničkih zapisa. Međutim, pesimizira uobičajeni slučaj provjera dozvola tijekom obrade zahtjeva.

Svaki autentificirani zahtjev provjerava korisničke dozvole. Sa modelom niza, ova provjera se događa potpuno u memoriji nakon dohvaćanja korisničkog dokumenta—nema dodatnih upita. Sa modelom spojne tablice, svaka provjera dozvole zahtijeva ili JOIN (skupo) ili zasebni upit spojnoj tablici (overhead mreže). Kod stotina zahtjeva u sekundi, ovaj overhead se značajno akumulira.

Kompromis je da dodavanje novih tipova mogućnosti zahtijeva ažuriranje svih korisničkih dokumenata koji bi to trebali imati. Ovo je prihvatljivo jer se tipovi mogućnosti dodaju rijetko (možda nekoliko puta godišnje) dok se provjere dozvola događaju milijune puta dnevno. MongoDB optimizira uobičajeni slučaj na trošak rijetkog slučaja.

Kolekcija capability_requests održava povijesne zapise svih zahtjeva za dozvolama—i za inicijalne osnovne uloge i za dodatne mogućnosti. Ovi podaci ne pripadaju kolekciji users jer bi svaki korisnik mogao imati mnogo zahtjeva tijekom vremena, a zahtjevi sadrže bilješke revizora i vremenske oznake irelevantne za autentifikaciju. Razdvajanje briga održava kolekciju users brzom za vruću stazu (autentifikacija) dok kolekcija zahtjeva podržava sporiju stazu (administracija i revizija).

Kolekcija api_keys postoji zasebno jer korisnici mogu imati više API ključeva, svaki s neovisnim isticanjem, ograničenjima stope i praćenjem upotrebe. Vremenska oznaka last_used_at omogućava korisnicima da identificiraju i izbrišu napuštene ključeve. key_prefix podržava brze lookup-e bez full-text pretraživanja. Pohranjivanje samo SHA-256 hash-a ključa štiti od kompromitiranja baze podataka. Ništa od ovih podataka nije potrebno tijekom web autentifikacije, tako da bi zagađivanje kolekcije users time usporilo uobičajeni slučaj da bi posluživalo specijalizirani slučaj.

Kolekcija audit_logs raste kontinuirano i ima različite obrasce pristupa od drugih kolekcija. Dnevnici su append-only (nikad se ne ažuriraju nakon pisanja), upituju se po vremenskom rasponu, filtriraju po korisniku ili akciji ili komponenti, i zadržavaju se duže od drugih podataka. Ove karakteristike opravdavaju zasebnu kolekciju sa specijaliziranim indeksima. Kako kolekcija raste velikom, može se particionirati po datumu ili premjestiti u arhivnu pohranu bez utjecaja na operativne podatke. Miješanje dnevnika revizije s korisnicima bi na kraju degradiralo performanse lookup-a korisnika kako se dnevnici akumuliraju.

Ovaj dizajn četiri kolekcije balansira normalizaciju (ne dupliciraj podatke nepotrebno) s denormalizacijom (dupliciraj podatke kada to poboljšava performanse). Korisnički dokument denormalizira ugradnjom metapodataka i mogućnosti jer provjere dozvola trebaju ove podatke. Zahtjevi za mogućnostima i API ključevi normaliziraju jer se upituju zasebno i imaju različite životne cikluse. Dnevnici revizije se razdvajaju jer se njihovi obrasci pristupa fundamentalno razlikuju. Ove odluke proizlaze iz razumijevanja opterećenja, a ne od primjene rigidnih pravila o normalizaciji.

5 Sigurnosna arhitektura

5.1 Obrana u dubinu kroz višestruke metode autentifikacije

Projektna odluka: Podrška za tri metode autentifikacije—OAuth, JWT, API ključevi

Platforma dopušta autentifikaciju kroz Google OAuth, JWT tokene ili API ključeve umjesto prisiljavanja jedne metode.

Podrška za višestruke metode autentifikacije odražava prepoznavanje da različiti slučajevi uporabe imaju različite zahtjeve i profile rizika. Korisnici web preglednika autentificiraju se putem OAuth-a jer pruža najbolje korisničko iskustvo—nema lozinke za upravljanje, sigurnost delegirana Googleu, 2FA automatski obrađen. Nakon autentifikacije, primaju JWT tokene za naknadne API pozive jer tokeni omogućavaju stateless autentifikaciju na skali. Korisnici skripti i automatizacije primaju API ključeve jer OAuth zahtijeva interakciju preglednika koja ne funkcionira za headless operacije.

[Nastavak dokumenta s prevedenim sadržajem...]

5.2 Arhitektura Agent Frameworka

Projektna odluka: Agent framework inspiriran LangChain-om s RAG-om i lokalnom LLM integracijom putem Ollame

AI mogućnosti pružaju se kroz agent framework koji orkestrira LLM pozive, Retrieval-Augmented Generation (RAG), pozivanje alata i parsiranje strukturiranog izlaza.

Agent framework apstrahira LLM interakcije iza konzistentnog sučelja: prihvati unos prirodnog jezika, dohvati relevantni kontekst znanja, pozovi potrebne alate, vrati strukturirani izlaz. Ova apstrakcija omogućava više tipova agenata (ekstrakcija znanja, pomoć u upitima, prijedlozi modela) da dijele infrastrukturu dok specijaliziraju ponašanje. Klasa AgentFramework pruža orkestraciju, LocalModelClient rukuje Ollama komunikacijom, a ToolRegistry omogućava dinamičko proširenje mogućnosti.

Retrieval-Augmented Generation (RAG): Framework implementira RAG metodologiju za utemeljenje LLM odgovora u znanju platforme. Prilikom obrade upita, sustav dohvaća relevantne činjenice, stilizirane činjenice i entitete grafa znanja iz baze znanja i uključuje ih u LLM kontekst. Ovaj pristup adresira LLM halucinacije pružanjem činjeničnog utemeljenja—model generira odgovore temeljene na stvarnom kuriranom znanju umjesto parametarske memorije. Za ingestiju dokumenata, RAG omogućava agentima referiranje postojećeg znanja dok ekstrahiraju nove činjenice, osiguravajući konzistenciju i detektirajući potencijalne kontradikcije. Za upite znanja, RAG dopušta da se na pitanja prirodnim jezikom ("Što je odnos metaboličkog skaliranja za ribe?") odgovori sa specifičnim sadržajem platforme umjesto generičkog LLM znanja.

RAG cjevovod sastoji se od: (1) Analiza upita—ekstrahiranje ključnih koncepata i entiteta iz korisničkog unosa, (2) Dohvaćanje znanja—pretraživanje baze znanja koristeći tekstualno pretraživanje i buduću vektorsku sličnost za relevantni sadržaj, (3) Sastavljanje konteksta—formatiranje dohvaćenih činjenica i stiliziranih činjenica kao LLM konteksta, (4) Generiranje odgovora—LLM generira odgovor utemeljen u dohvaćenom znanju, (5) Citiranje izvora—odgovori uključuju reference na specifične stavke baze znanja radi sljedivosti. Ova arhitektura osigurava znanstvenu odgovornost—svaka AI-generirana izjava može se pratiti do kuriranih izvora.

Lokalna LLM integracija putem Ollame odražava razmatranja troškova i privatnosti. Cloud LLM API-ji (OpenAI, Anthropic) nude bolju kvalitetu ali imaju troškove po tokenu koji se skaliraju s upotrebom—problematično za istraživačko istraživanje gdje su budžeti tokena nepredvidivi. Ollama pokreće modele lokalno (Llama 2, Mistral, itd.) s nula marginalnim troškovima nakon što je hardver osiguran. Za beta skalu, jedan GPU server koji pokreće Ollamu je dovoljan. Cloud API-ji mogu dopuniti lokalne modele za visoko-

vrijedne slučajeve uporabe (konačna ekstrakcija kvalitete publikacije) dok lokalni modeli rukuju istraživačkim radom.

Sustav alata omogućava agentima poduzimanje akcija izvan generiranja teksta: pretraživanje postojećeg znanja, upitivanje baza podataka, pozivanje vanjskih API-ja. Svaki alat je Python funkcija registrirana u ToolRegistry s JSON shemom koja opisuje parametre. LLM generira pozive alata kao strukturirani JSON, framework izvršava alate, a rezultati se vraćaju LLM-u za sljedeće korake. Ovaj obrazac generiranja proširenog alatima odgovara metafori istraživačkog asistenta: agenti koji mogu "potražiti informacije" i "izvršiti analize", a ne samo generirati tekst.

Upravljanje sesijama prati povijest razgovora u dokumentima AgentSession. Razgovori s više okretaja omogućavaju iterativno usavršavanje: "ekstrahiraj činjenice iz ovog rada" → "fokusiraj se na odjeljak metodologije" → "stvori stiliziranu činjenicu koja sažima rezultate." Sesije pohranjuju povijest poruka, tragove poziva alata i kontekst (referencirani fact_ids, document_ids). Ova povijest omogućava debugging (zašto je agent proizveo ovaj izlaz?) i učenje (analiza uspješnih vs neuspješnih sesija ekstrakcije).

Alternativne arhitekture uključuju samo cloud (jednostavno ali skupo), fine-tuning modela (točno ali sporo za iteraciju), i bez AI (točno ali ručno). Lokalni LLM + agent framework balansira koristi automatizacije, kontrolu troškova i brzinu iteracije prikladnu za razvoj istraživačke platforme.

5.3 Upit i pretraživanje znanja

Projektna odluka: MongoDB tekstualno pretraživanje s opcionalnom vektorskom sličnošću

Pretraživanje znanja koristi MongoDB-ovo ugrađeno indeksiranje teksta za upite slobodnog teksta, s arhitekturom koja omogućava buduće pretraživanje vektorskih ugrađivanja.

MongoDB tekstualno pretraživanje pruža osnovne ali brze full-text upite preko sadržaja činjenica, teksta dokumenata i oznaka čvorova grafa. Tekstualni indeks na poljima facts.content, stylized_facts.summary i knowledge_graphs.nodes omogućava upite poput "nađi činjenice o lososovim ušima" s bodovanjem relevantnosti i rangiranjem rezultata. Ovaj pristup koristi postojeću MongoDB infrastrukturu bez dodatnih servisa.

Ograničenje je semantičko razumijevanje—tekstualno pretraživanje poklapa ključne riječi ali propušta konceptualnu sličnost. "Morske uši" i "parazitski kopepodi" se ne poklapaju unatoč referiranju na povezane koncepte. Vektorska ugrađivanja (kodiranje teksta u numeričke vektore, pretraživanje prema vektorskoj sličnosti) hvataju semantičke odnose ali zahtijevaju modele ugrađivanja, vektorsku pohranu i infrastrukturu izračunavanja sličnosti.

Arhitektura omogućava buduće vektorsko pretraživanje dizajnirajući trenutno tekstualno pretraživanje iza apstrakcijskog sloja. Krajnja točka /api/knowledge/search vraća bodovane rezultate bez obzira na temeljnu implementaciju. Kada je potrebno vektorsko pretraživanje, dodajemo generiranje ugrađivanja (poziv modela ugrađivanja pri umetanju znanja), pohranjujemo vektore (dodavanje polja ugrađivanja dokumentima), i implementiramo pretraživanje sličnosti (kosinusna sličnost protiv ugrađivanja upita). API ugovor se ne mijenja, omogućavajući poboljšanje bez lomljenja.

"Opcionalna" priroda odražava beta prioritete: tekstualno pretraživanje adekvatno služi inicijalne slučajeve uporabe, a dodavanje složenosti vektorskog pretraživanja prije

dokazivanja potrebe riskira predčasnu optimizaciju. Povratne informacije korisnika će otkriti je li semantičko pretraživanje bitno ili lijepo-za-imati. Izgradnja jednostavnijeg rješenja prvo omogućava bržu iteraciju dok arhitektura dopušta poboljšanje.

Filtriranje temeljeno na oznakama (činjenice s tags=["losos", "parazit"]) dopunjava tekstualno pretraživanje omogućavajući precizne upite podskupa. Oznake se ručno dodjeljuju od strane kuratora ili automatski ekstrahiraju od strane agenata, pružajući strukturirane metapodatke uz nestrukturirani tekst. Kombinacija tekstualnog pretraživanja (nađi relevantni sadržaj) i filtriranja oznaka (suzi na specifične domene) odgovara istraživačkim tijekovima rada gdje domensko stručnost vodi istraživanje.

6 Arhitektura Modeling Assistanta

6.1 Razvoj modela vođen chatbotom s RAG-om

Projektna odluka: Konverzacijsko sučelje s LLM-om i RAG-om za istraživačko modeliranje

Modeling Assistant pruža chatbot sučelje gdje istraživači komuniciraju s LLM-om koji koristi RAG za prelaženje baze znanja dok grade bioenergetske modele.

Arhitektura Modeling Assistanta centrirana je oko ljudsko-AI suradnje za razvoj novih paradigmskih modela u bioenergetici. Umjesto zahtijevanja da istraživači ručno pregledavaju bazu znanja i sastavljaju modele, chatbot sučelje omogućava istraživanje prirodnim jezikom: "Što znamo o alokaciji energije kod lososa tijekom reprodukcije?" ili "Pokaži mi obrasce metaboličkog skaliranja preko teleostnih riba." LLM, proširen s RAG-om, dohvaća relevantne činjenice i stilizirane činjenice iz baze znanja, sintetizira informacije preko izvora i predlaže pristupe modeliranju temeljene na dostupnim dokazima.

Konverzacijska izgradnja modela: Tijek rada odražava iterativno znanstveno razmišljanje. Istraživač može postavljati istraživačka pitanja o obrascima alokacije energije, tražiti usporedbe preko vrsta ili životnih faza, ispitivati potporne i kontradiktorne dokaze, predlagati strukture modela i usavršavati pretpostavke temeljene na sadržaju baze znanja. Chatbot održava kontekst razgovora preko više okretaja, dopuštajući progresivno usavršavanje. Svaka interakcija koristi RAG za utemeljenje odgovora u kuriranom znanju—sprečavajući halucinacije i osiguravajući znanstvenu strogost.

Prelaženje baze znanja: RAG omogućava inteligentnu navigaciju kroz međusobno povezano znanje. Kada se raspravlja o alokaciji energije, sustav dohvaća povezane činjenice o metaboličkim stopama, parametrima reprodukcije i dinamici transportne mreže. Slijedi tipizirane odnose (potporne dokaze, kontradiktorne dokaze, lance usavršavanja) za predstavljanje sveobuhvatnih pogleda. Chatbot može objasniti "Ova stilizirana činjenica o metaboličkom skaliranju podržana je s 15 empirijskih promatranja preko 8 vrsta" dok pruža direktne veze na temeljne činjenice. Ova sposobnost prelaženja pomaže istraživačima otkriti veze i obrasce koje bi mogli propustiti kroz ručno istraživanje.

Razvoj paradigmskog modela: Primarni fokus platforme tijekom inicijalnog razvoja je stvaranje osnovnih alata i uslužnih programa za izgradnju sveobuhvatne baze znanja za bioenergetsko modeliranje organizama. Arhitektura chatbot-RAG podržava ovu misiju čineći znanje dostupnim i djelotvornim. Umjesto da istraživači provedu tjedne ručno pregledavajući literaturu, sustav iznosi relevantno znanje kroz razgovor, omogućavajući brže formiranje hipoteza i konceptualizaciju modela. Naglasak je na konstrukciji baze

znanja i tooling-u—detaljne API specifikacije i mogućnosti izvršavanja modela su u aktivnom razvoju i će evoluirati temeljeno na istraživačkim potrebama.

Trenutni status razvoja: Arhitektura omogućava buduće mogućnosti (simulacija modela, procjena parametara, vizualizacija rezultata) dok prioritizira trenutne potrebe: ingestija znanja, tijekovi rada kuriranja, upravljanje odnosima činjenica i istraživačkim pristupom znanju kroz konverzacijsko sučelje. API krajnje točke za izvršavanje modela i naprednu analitiku ostaju u razvoju, s dizajnom dovoljno fleksibilnim da se prilagodi kako istraživački zahtjevi postaju jasniji kroz stvarnu upotrebu.

Pristup chatbot-RAG usklađuje se s ciljem platforme unapređivanja DEB modela temeljenih na transportnoj mreži (tDEB). Omogućavanjem istraživačima da istraže 4,500 vrsta AmP baze podataka kroz prirodni jezik, identificiraju domene primjenjivosti i otkriju općenitosti preko filogenijskih grupa, sustav ubrzava proces znanstvenog otkrića. Konverzacijsko sučelje snižava barijere pristupa znanju, čineći vrijednost platforme—sveobuhvatno bioenergetsko znanje—odmah korisnim za razvoj modela.

6.2 Tijek rada modeliranja temeljen na scenarijima

Projektna odluka: Entiteti scenarija kao spremnici projekata modeliranja

Rad modeliranja organiziran je u dokumente Scenarija koji objedinjuju ciljeve, reference znanja, predložene modele i rezultate.

Arhitektura centrirana na scenarije odražava kako modeleri zapravo rade. Istraživački projekt ne počinje s "izgradi model"—počinje s "razumijevanje fenomena X pod uvjetima Y." Scenariji hvataju ovo: ime ("Dinamika lososovih uši u norveškim fjordovima"), opis (istraživački kontekst), ciljevi (specifična pitanja na koja treba odgovoriti), i knowledge_queries (relevantna KB pretraživanja). Scenarij je spremnik projekta modeliranja.

Ovaj obrazac spremnika pruža nekoliko koristi. Više modela može biti predloženo za jedan scenarij, omogućavajući usporedbu: "model A koristi ODE formulaciju, model B koristi IBM pristup—koji bolje odgovara empirijskim obrascima?" Rezultati se povezuju natrag na scenarije i modele, održavajući provenijenciju. Scenariji postaju jedinica suradnje: istraživači dijele scenarije, ne sirovi kod modela. Scenarij UUID pojavljuje se u dnevnicima revizije, omogućavajući analizu koja istraživačka pitanja troše najviše resursa platforme.

Alternativni organizacijski obrasci uključuju model-centrično (modeli su primarni, scenariji implicitni) ili ravno (bez grupiranja). Model-centrični pristupi prisile neugodne usklađenosti kada se istražuju višestruke strategije modeliranja za jedno pitanje. Ravni pristupi gube kontekst na razini projekta koji istraživači trebaju da interpretiraju rezultate. Spremnik scenarija odgovara istraživačkim mentalnim modelima dok pruža tehničku strukturu.

Polje knowledge_queries povezuje scenarije s KB sadržajem. Umjesto kopiranja znanja u MA (rizik dupliciranja), scenariji pohranjuju KB upite pretraživanja koji se izvršavaju kada se scenarij učitava. Ova indirekcija održava MA-ov pogled znanja trenutnim kako KB evoluira, sprečava zastarjelost podataka i održava jedan izvor istine. Kompromis je ovisnost: ako KB nije dostupan, MA ne može riješiti reference znanja. Za beta skalu s ko-lociranim servisima, ovo spajanje je prihvatljivo.

6.3 Sastavljanje i predstavljanje modela

Projektna odluka: Deklarativne specifikacije modela s vezama obrazloženja na KB

Modeli su predstavljeni kao JSON specifikacije koje opisuju entitete, varijable stanja, procese i parametre, s svakim elementom opravdanim KB referencama.

Pristup deklarativnih specifikacija razdvaja strukturu modela (koji entiteti postoje, koji procesi se događaju) od izvršavanja modela (simulacijski kod). Dokument specifikacije modela sadrži entitete (["domaćin", "parazit"]), state_variables (["status_infekcije", "dob", "lokacija"]), procese (["transmisija", "oporavak", "kretanje"]), i parametre (["beta: stopa transmisije", "gamma: stopa oporavka"]). Ova strukturirana predstava omogućava programsku analizu, vizualizaciju i na kraju generiranje koda.

Veze obrazloženja povezuju elemente modela s KB stavkama. Proces "transmisija" referencira KB ID-jeve činjenica koji opisuju promatrane mehanizme transmisije. Parametar "beta" referencira stilizirane činjenice o izmjerenim stopama transmisije. Ove veze služe više svrha: čine modelske pretpostavke eksplicitnim i sljedivim, omogućavaju automatski pregled literature (koji dokazi podržavaju ovaj model?), i olakšavaju usporedbu modela (modeli A i B se razlikuju u pretpostavkama transmisije, koje KB reference podržavaju svaku?).

Ova arhitektura odražava zahtjeve znanstvene odgovornosti. Modeli bez jasnih pretpostavki su crne kutije; pretpostavke bez dokaza su spekulacije. KB reference veze transformiraju modele iz neprozirnog koda u transparentne lance zaključivanja. Revizori mogu kritizirati pretpostavke ispitivanjem potpornih dokaza. Ovaj overhead sljedivosti je opravdan za znanstveno modeliranje gdje kredibilitet modela ovisi o eksplicitnom zaključivanju.

Specifikacija je agnostična prema izvršavanju: opisuje što model predstavlja, a ne kako ga simulirati. Budući sustavi izvršavanja mogu generirati NetLogo kod, Python simulacijske petlje ili matematičke jednadžbe iz iste specifikacije. Ova razdvojenost omogućava evoluciju: strategije izvršavanja se poboljšavaju bez promjene semantike modela. Za beta, specifikacije služe dokumentaciji i komunikaciji; izvršavanje dolazi kasnije.

Alternativni pristupi uključuju kod-kao-model (model je Python simulacijski kod) ili samo grafički (model je vizualni dijagram). Kod-kao-model je precizan ali neproziran—razumijevanje zahtijeva čitanje implementacije. Samo grafički je intuitivan ali neformalan—vizualne kutije nemaju semantičku preciznost. Deklarativna specifikacija s KB obrazloženjem pruža preciznost, sljedivost i semantiku prikladnu za znanstveni diskurs.

6.4 Integracija s Knowledge Builderom

Projektna odluka: HTTP API integracija s JWT token pass-through

MA se integrira s KB kroz HTTP API-je, prosljeđujući JWT tokene korisnika za održavanje jedinstvene autentifikacije i atribucije.

HTTP integracija održava MA i KB labavo spojene. MA čini standardne HTTP GET/POST zahtjeve prema KB krajnjim točkama (/api/knowledge/search, /api/knowledge/facts/{id}, /api/agents/run). Ovaj stil integracije omogućava neovisno postavljanje: KB može biti ažuriran bez MA ponovnog postavljanja ako su API ugovori održavani. Razvoj može nastaviti paralelno: KB i MA timovi rade neovisno s ugovorima integracije kao točkama

koordinacije.

7 Zaključak

7.1 Sveobuhvatna arhitektura platforme

Ovaj dokument sada pruža sveobuhvatno arhitektonsko obrazloženje za AdvanDEB Platformu preko svih glavnih komponenti:

- **Upravljanje korisnicima & Autentifikacija**: Google OAuth integracija, JWT tokeni, uloge temeljene na mogućnostima, API ključevi, revizijsko zapisivanje i SSO kroz komponente
- **Knowledge Builder**: Polu-nadzirano građenje baze znanja kroz obradu dokumenata posredovanu chatbotom i ekstrakciju znanja, tri-slojnu reprezentaciju znanja (Činjenice, Stilizirane činjenice, Grafovi), asinhronični cjevovod ingestije dokumenata, AI agent framework s RAG-om i lokalnom LLM integracijom, i MongoDB tekstualno pretraživanje
- **Modeling Assistant**: Chatbot sučelje s LLM-om i RAG-om za konverzacijsko istraživanje znanja i razvoj modela, tijek rada temeljen na scenarijima za organiziranje projekata modeliranja, deklarativne specifikacije modela s KB vezama obrazloženja
- **Arhitektura integracije**: In-process spajanje komponenti, model podataka temeljen na referencama izbjegavajući sinkronizaciju, i verzionirana sučelja
- **Model podataka**: MongoDB za sve tipove podataka, odvojene kolekcije po entitetu, reference temeljene na ID-ju i univerzalna atribucija kreatora

Arhitektonske odluke preko ovih komponenti slijede dosljedne principe ustanovljene u sloju autentifikacije: jednostavnost nad predčasnom optimizacijom, dokazane tehnologije nad novim ali nedokazanim pristupima, labavo spajanje s eksplicitnim ugovorima i evoluabilnost kroz apstrakcijske slojeve. Svaka komponenta može evoluirati neovisno dok održava koherenciju platforme kroz dijeljenu autentifikaciju, dosljednu atribuciju podataka i dokumentirane ugovore integracije.

7.2 Sintetizirani arhitektonski principi

Arhitektura autentifikacije proizlazi iz primjene dosljednih principa preko svih projektnih odluka. Jednostavnost je prioritizirana nad pametovanjem—dokazane tehnologije poput MongoDB-a i JWT tokena nad novim ali nedokazanim pristupima. Sigurnost po defaultu umjesto sigurnosti konfiguracijom znači da korisnici ne mogu slučajno oslabiti zaštite kroz lošu konfiguraciju. Osnaživanje korisnika kroz model mogućnosti dopušta organsku evoluciju dozvola umjesto krutih granica uloga. Evoluabilnost osigurava da budući zahtjevi mogu biti adesirani kroz proširenje umjesto redizajna.

Ovi principi ponekad su u sukobu, zahtijevajući eksplicitne kompromise. Dijel jena baza korisnika žrtvuje neku neovisnost komponenti za dosljednost i jednostavnost. Model mogućnosti žrtvuje granularnost dozvola (nema ACL-ova po dokumentu) za jednostavnost tijeka rada odobrenja. JWT tokeni žrtvuju sposobnost trenutnog opoziva za stateless performanse. Svaki kompromis je napravljen svjesno nakon ispitivanja alternativa i razumijevanja implikacija.

Arhitektura nije optimalna prema bilo kojem univerzalnom mjerilu—niti jedna arhitektura nije. Ona je optimalna za AdvanDEB-ov specifični kontekst: znanstvena platforma koja zahtijeva odgovornost, služeći akademske korisnike s umjerenom tehničkom sofisticiranošću, izgrađena od malog tima koji prioritizira razvoj značajki nad kompleksnošću infrastrukture, skaliranje na 10-20 korisnika inicijalno u beta s kapacitetom za otprilike 100 korisnika. Različiti konteksti opravdavali bi različite odluke.

7.3 Kriteriji uspjeha za arhitekturu autentifikacije

Arhitektura autentifikacije uspijeva ako korisnici lako autentificiraju i rade preko komponenti besprijekorno, administratori efikasno upravljaju dozvolama bez utapanja u zahtjevima za odobrenje, developeri dodaju značajke bez lomljenja autentifikacije, i platforma skalira glatko od 10-20 beta korisnika na otprilike 100 korisnika dok održava 99.9% uptime i sub-100ms latenciju autentifikacije. Uspjeh sigurnosti znači nula kršenja povezanih s autentifikacijom—nema ukradenih lozinki (nemamo ih), nema eksploatacija eskalacije privilegija, nema praznina u dnevniku revizije koje skrivaju zlonamjerno ponašanje.

Ovi kriteriji su mjerljivi i vremenski ograničeni. Kvaliteta korisničkog iskustva pokazuje se u volumenu tiketa podrške (manje tiketa o problemima autentifikacije indicira bolji dizajn). Efikasnost administratora pokazuje se u vremenu provedenom na tijekovima rada odobrenja. Produktivnost developera pokazuje se u brzini značajki i stopama bugova. Performanse sustava pokazuju se u kontrolnim pločama praćenja. Uspjeh sigurnosti pokazuje se u rezultatima revizije i zapisima incidenata (ili njihovom nedostatku).

Arhitektura je sredstvo za ove ciljeve, a ne cilj sam po sebi. Lijepa arhitektura koja frustrira korisnike ili usporava razvoj ne uspijeva bez obzira na tehničku eleganciju. Pragmatična arhitektura koja postiže projektne ciljeve uspijeva čak i ako ne bi osvojila dizajn nagrade. Ovaj dokument postoji da objasni kako se arhitektonske odluke povezuju s uspjehom projekta, čineći pragmatizam iza svakog izbora eksplicitnim i opravdanim.

7.4 Arhitektura kao živi dizajn

Ovo nije konačna arhitektura. Odluke dokumentirane ovdje predstavljaju naše najbolje trenutno razumijevanje temeljeno na analizi zahtjeva, korisničkom istraživanju i tehničkoj evaluaciji. Međutim, arhitektura mora evoluirati kako implementacija otkriva neočekivane izazove, povratne informacije korisnika razotkrivaju probleme upotrebljivosti, ili testiranje performansi otkriva uska grla.

Iskustvo stvarnog svijeta često proturječi teorijskim pretpostavkama. Obrazac upita baze podataka koji se činio efikasnim tijekom dizajna može pokazati problematičnim pod stvarnim obrascima uporabe. Tijek autentifikacije koji se činio intuitivnim na papiru može zbuniti stvarne korisnike. Pristup integracije koji se činio čistim arhitektonski može stvoriti operativno trenje tijekom postavljanja.

AdvanDEB arhitektura je dizajnirana za evoluciju. Granice komponenti, API ugovori i apstrakcijski slojevi pružaju fleksibilnost za reviziju implementacija bez potpunih prepravljanja. MongoDB-ova fleksibilnost sheme prilagođava promjene modela podataka. Model mogućnosti podržava promjene dozvola bez modifikacija koda. JWT-ov stateless dizajn dopušta promjene infrastrukture autentifikacije bez prekidanja aktivnih sesija.

Očekujemo da će ovaj dokument evoluirati uz platformu. Značajne arhitektonske promjene će pokrenuti ažuriranja dokumenta s poviješću verzija koja prati evoluciju dizajna. Neuspjeli eksperimenti bit će dokumentirani uz uspjehe da se sačuva institucionalno

znanje o tome što nije radilo i zašto. Cilj nije arhitektonska perfekcija od početka, već arhitektonsko učenje kroz razvoj.

7.5 Buduća arhitektonska poboljšanja

Dok ovaj dokument pokriva osnovnu arhitekturu platforme, nekoliko područja će zahtijevati dodatne arhitektonske odluke kako implementacija napreduje:

1. **Napredne mogućnosti pretraživanja**: Vektorska ugrađivanja za semantičko pretraživanje, hibridno pretraživanje koje kombinira tekst i vektore, podešavanje relevantnosti za znanstvenu literaturu
2. **Značajke suradnje** (Faze 3-4): Timski radni prostori, suradničko uređivanje znanja, dijeljeni razvoj scenarija, rješavanje sukoba doprinosa
3. **Izvršavanje modela**: Integracija simulacijskog engine-a, generiranje koda iz specifikacija modela, procjena parametara, vizualizacija rezultata
4. **Optimizacija performansi**: Strategije cacheiranja, podešavanje indeksiranja baze podataka, optimizacija upita, obrada pozadinskih poslova s Celery-jem
5. **Napredne mogućnosti agenata**: Tijekovi rada više agenata, kompozicija alata, verifikacija rezultata agenata, inkrementalno učenje iz korekcija
6. **Sustavi povjerenja i reputacije**: Bodovanje reputacije doprinositelja, metrike kvalitete znanja, automatska provjera činjenica, analiza mreže citata
7. **Arhitektura dodataka**: Točke proširenja za prilagođene alate, dodatke agenata, dodatke vizualizacije, integracije trećih strana

Ova poboljšanja grade na arhitektonskim temeljima dokumentiranim ovdje. Odluka o odgađanju ovih značajki odražava beta prioritete: uspostaviti osnovnu funkcionalnost platforme (upravljanje znanjem, osnovna podrška modeliranju, upravljanje korisnicima) prije dodavanja naprednih mogućnosti. Arhitektura omogućava ova poboljšanja kroz točke proširenja dizajnirane u trenutne sustave.

Verzija dokumenta: 12.25.1
Datum: 12. prosinca 2025.
Status: Potpuno obrazloženje arhitekture (sve komponente platforme)
Ciljana skala: Beta s 10-20 korisnika, maksimalno 100 korisnika
Status arhitekture: Živi dokument - podložan reviziji temeljeno na iskustvu implementacije
Kontakt: AdvanDEB Razvojni tim