AdvanDEB Platforma

Potpuno obrazloženje arhitekture
Svi dijelovi platforme i integracija

Verzija 12.25.1 - Dizajnerske odluke i obrazloženja

AdvanDEB razvojni tim

12. prosinca 2025.

Abstract

Ovaj dokument pruža sveobuhvatno obrazloženje svake važne arhitekturalne odluke u sklopu AdvanDEB platforme: autentikacija i upravljanje korisnicima, temeljna funkcionalnost sustava Knowledge Builder, dizajn Modeling Assistant-a, integracijska arhitektura te izbori u podatkovnom modelu. Izvještaj objašnjava razloge iza svake dizajnerske odluke, razmatra alternativne pristupe i dokumentira kompromise koji su oblikovali arhitekturu platforme. Dokument služi kao mjerodavni referentni izvor za razumijevanje zašto je platforma izgrađena baš na ovaj način, te podržava buduće odluke o razvoju i održavanju.

Važna napomena: Ovaj dokument prikazuje trenutačni arhitekturalni dizajn, a ne konačnu arhitekturu sustava. Očekuje se da će se arhitektura razvijati kako AdvanDEB projekt bude napredovao kroz implementaciju, testiranje i stvarnu upotrebu. Ovdje dokumentirane dizajnerske odluke mogu biti revidirane na temelju izazova u implementaciji, povratnih informacija korisnika, zahtjeva za performansama ili novih tehničkih uvida koji se pojave tijekom razvoja.

Contents

1 Sažetak za rukovodstvo

1.1 Svrha i opseg

Ovaj dokument pruža obrazloženje arhitekture koja stoji iza implementacijskih izbora. Svaka važna arhitekturalna odluka promatra se kroz više perspektiva: problem koji rješava, razmatrane alternative, učinjeni kompromisi te posljedice za budući razvoj platforme.

Sveobuhvatni obuhvat: Dokument pokriva sve glavne komponente platforme:

- **Autentikacija i upravljanje korisnicima**: Google OAuth, JWT tokeni, uloge temeljene na sposobnostima (capability-based), API ključevi, zapisivanje audita, jedinstvena prijava (SSO) između komponenti
- **Knowledge Builder**: Polu-nadzirana izgradnja baze znanja kroz interakciju chatbota s dokumentima i postojećim znanjem, trostupanjski model znanja (činjenice / stilizirane činjenice / grafovi), okvir AI agenata s RAG pristupom
- **Modeling Assistant**: Sučelje chatbota s LLM-om i RAG-om za konverzacijski razvoj modela, pretraživanje baze znanja, scenarijima vođen tijek rada
- **Integracijska arhitektura**: API ugovori, upravljanje verzijama, strategije sinkronizacije podataka
- **Podatkovni model**: Dizajn MongoDB sheme, struktura kolekcija, obrasci odnosa, mehanizmi atribucije

Svaka odluka odražava realnost izgradnje znanstvene istraživačke platforme za 10–20 beta korisnika uz planirano skaliranje na otprilike 100 korisnika, pri čemu se prednost daje jednostavnosti i brzoj iteraciji nad preuranjenom optimizacijom.

1.2 Znanstveni kontekst: AdvanDEB istraživački projekt

AdvanDEB platforma služi specifičnoj znanstvenoj misiji: unapređenju teorije dinamičkog energetskog budžeta (Dynamic Energy Budget, DEB) kroz modeliranje temeljeno na transportnim mrežama (tDEB). Razumijevanje ove misije ključno je za vrednovanje arhitekturalnih odluka, jer je dizajn platforme izravno usmjeren na podršku istraživačkoj metodologiji.

Znanstveni izazov: DEB modeli kvantitativno opisuju prikupljanje i raspodjelu energije u organizmima, omogućujući povezivanje okolišnih i bioloških razina. Pokazali su se ključnima za rješavanje društvenih izazova od proizvodnje hrane i očuvanja bioraznolikosti do zaštite ekosustava, korištenja resursa i učinaka onečišćivala na organizme. Ipak, moderni DEB modeli imaju ograničenja: stečenu energiju odmah pohranjuju u rezerve i raspodjeljuju je pomoću tzv. „k-pravila" u fiksnim omjerima između reprodukcije i ostalih procesa. Takvi modeli dobro opisuju dugoročne procese, ali se otežano nose s kratkoročnom dinamikom i ne uspijevaju uvjerljivo povezati stanja modela s fiziologijom organizma.

tDEB inovacija: Uvođenje transportnih mreža (primjerice krvotoka) u DEB modele adresira ove nedostatke. Uspješne primjene na kitovima i ribama pokazuju velik potencijal za široku primjenjivost. Ipak, brojna neriješena pitanja još uvijek priječe širu primjenu DEB modela temeljenih na transportnim mrežama (tDEB).

Ciljevi AdvanDEB projekta: Projekt adresira ključna pitanja: (i) provjeriti i osigurati usklađenost s poznatim biološkim principima, (ii) definirati jasna pravila korištenja energije tijekom reprodukcije, (iii) testirati općenitost i definirati domene primjenjivosti tDEB pristupa, (iv) analizirati odnose između tDEB-a i drugih teorija te (v) razviti formalne skupove pretpostavki i objedinjenu terminologiju.

Inovativni istraživački pristup: AdvanDEB će koristiti postojeće resurse, poput AmP baze zajednice DEB istraživača koja sadrži oko 4.500 vrsta, primjenjujući i klasične metode i AI-potpomognute tehnike za identifikaciju domena primjenjivosti tDEB-a, istraživanje općenitosti unutar i između filogenetskih skupina te formalizaciju tDEB teorije. Dodatno, AdvanDEB može doprinijeti ujedinjenju DEB teorije s Metaboličkom teorijom ekologije, stvarajući potencijal za promjenu paradigme u bioenergetici i otvarajući nove istraživačke pravce.

Uloga platforme u istraživanju: AdvanDEB platforma podržava ovu znanstvenu misiju kroz nove pristupe pregledu literature i izgradnji baze znanja. Platforma mora omogućiti: sustavno izdvajanje bioloških činjenica iz znanstvene literature, organizaciju heterogenog znanja o organizmima kroz filogenetske skupine, povezivanje znanja s modelnim parametrima i pretpostavkama, AI-potpomognuto otkrivanje obrazaca i općenitosti među vrstama te kolaborativnu kuraciju od strane stručnjaka uz strogu sljedivost.

Arhitektura platforme — posebice trostupanjski model znanja (Činjenice → Stilizirane činjenice → Grafovi znanja), okvir AI agenata za obradu dokumenata i spremanje grafa u MongoDB — izravno podržava ove istraživačke zahtjeve. Dizajnerske odluke daju prednost znanstvenoj strogosti, sljedivosti znanja i podrški za istraživačku analizu nad uobičajenim inženjerskim metrike izvedbe.

1.3 Misija i zahtjevi platforme

AdvanDEB platforma nalazi se na sjecištu upravljanja znanstvenim znanjem i računalnog modeliranja. Njena arhitektura mora služiti trima temeljnim zahtjevima: skalabilnom spremanju i upitu mrežno povezanih grafova znanja koji obuhvaćaju tisuće vrsta i bioloških odnosa, strogoj odgovornosti i sljedivosti koju zahtijeva znanstveno istraživanje (svaka činjenica mora imati pripisan izvor i kuratora) te fleksibilnoj integraciji između upravljanja znanjem i modeliranja, koja istraživačima omogućuje fluentno kretanje između pregleda literature, izdvajanja znanja i razvoja modela.

Odabir MongoDB-a kao temeljne baze podataka odražava ove zahtjeve za skalabilnost grafa znanja, dok se autentikacija, modeliranje i integracijska arhitektura nadograđuju na toj osnovi kako bi podržale kolaborativni znanstveni proces.

1.4 Arhitektonska filozofija

Arhitektura se vodi četirima temeljna principa koji su proizašli iz pažljive analize misije platforme. Prvo, znanstveni integritet ima prednost nad praktičnošću — svaki dio znanja mora biti sljediv do svojeg autora, a svaka radnja mora ostaviti trag u auditu. Taj se princip očituje u zahtjevima za autentikacijom, sustavima atribucije i nepromjenjivom zapisivanju događaja.

Drugo, sustav daje prednost suradnji nad kontrolom. Umjesto stvaranja krutih hijerarhija koje korisnike stavljaju u unaprijed zadane okvire, model dozvola temeljen na sposobnostima omogućuje da se pristupne mogućnosti korisnika razvijaju kako se razvijaju njihove potrebe. Istraživač može započeti kao osnovni kurator, kasnije zatražiti

pristup agentima za masovno izdvajanje znanja te naposljetku steći status recenzenta kroz demonstriranu stručnost. Taj organski rast dopuštenja odražava način na koji se stvarne istraživačke suradnje razvijaju.

Treće, jednostavnost proizlazi iz integracije, a ne iz razdvajanja. Iako se platforma sastoji od više komponenti (Knowledge Builder i Modeling Assistant), one dijele istu infrastrukturu autentikacije, modele korisnika i konvencije o podacima. Korisnici doživljavaju jedinstvenu platformu, a ne skup odvojenih alata. Takva integracija smanjuje trenje, a pritom zadržava prednosti neovisnosti komponenti.

Četvrto, sigurnost se implementira po defaultu, a ne konfiguracijom. Korisnici ne mogu oslabiti zahtjeve za autentikacijom, administratori ne mogu isključiti zapisivanje audita, a dozvole su eksplicitne, a ne implicitne. Sustav je dizajniran tako da bude siguran i kada njime upravljaju osobe bez specijaliziranog znanja o sigurnosti, jer većina znanstvenih institucija nema namjenske sigurnosne timove.

Napomena o stabilnosti arhitekture: Iako ovi principi usmjeravaju sve dizajnerske odluke, konkretne implementacije opisane u ovom dokumentu nisu konačne. Kako AdvanDEB projekt bude napredovao kroz razvoj i implementaciju, praktično iskustvo može otkriti bolje pristupe, neočekivana ograničenja ili pokazati da su neke pretpostavke bile pogrešne. Ovaj dokument bilježi naše trenutačno arhitektonsko razmišljanje i ažurirat će se kako bi odražavao značajne promjene tijekom evolucije platforme.

1.5 Svrha dokumenta

Ovaj dokument odgovara na pitanja „zašto baš ovi arhitektonski obrasci” i "koje su alternative odbačene". Podržava arhitektonske revizije, tehnološke evaluacije i buduće dizajnerske odluke pružajući detaljno obrazloženje svake ključne odluke.

Ključne arhitektonske odluke koje su dokumentirane uključuju: model s tri osnovne uloge, zajedničku arhitekturu baze podataka, pristup s JWT tokenima, odabir MongoDB-a radi skalabilnosti grafa znanja i principe bezdržavnog (stateless) dizajna. Svaka odluka uključuje kompromise prilagođene beta razmjeru (10–20 korisnika) uz mogućnost rasta do otprilike 100 korisnika.

Evolucija arhitekture: Odluke dokumentirane ovdje odražavaju naše trenutačno razumijevanje zahtjeva i ograničenja. Kako AdvanDEB projekt bude napredovao kroz faze implementacije, očekujemo da će praktično iskustvo otkriti prilike za optimizaciju, skrivene ograničavajuće faktore ili sugerirati alternativne pristupe. Ovaj će se dokument ažurirati kako bi odražavao značajne arhitektonske promjene, uz povijest verzija koja prati razvoj dizajnerskog promišljanja kako platforma sazrijeva.

1.6 Struktura dokumenta

Svaki odjeljak slijedi konzistentan analitički obrazac. Najprije jasno navodimo dizajnersku odluku, zatim objašnjavamo obrazloženje u narativnom obliku. Raspravljamo o alternativnim pristupima uz iskrenu procjenu njihovih prednosti i nedostataka — odbačene alternative često imaju stvarne prednosti, ali se ne uklapaju u naše specifične okvire. Kompromisi se eksplicitno analiziraju jer svaka arhitektonska odluka podrazumijeva odricanje od nečega radi postizanja nečeg drugog. Na kraju raspravljamo o ovisnostima i implikacijama, pokazujući kako svaka odluka utječe na ostale dijelove sustava.

2 Razmatranja opsega i dizajnerska ograničenja

2.1 Beta platforma: 10–20 korisnika, maksimalno 100 korisnika

Temeljni pokretač arhitektonskih odluka očekivani je opseg korisnika. AdvanDEB platforma cilja 10–20 korisnika tijekom početnog beta razdoblja, uz maksimalno očekivani rast do otprilike 100 korisnika. Taj razmjer duboko utječe na dizajnerske izbore na način koji se razlikuje od platformi koje ciljanju tisuće ili milijune korisnika.

Implikacije za arhitekturu:

Dizajn baze podataka – Kod 100 korisnika cijela se korisnička baza lako uklapa u memoriju. Uobičajene brige oko performansi MongoDB-a na velikim razmjerima (strategije shardanja, topologije replika setova, optimizacija indeksa za milijune dokumenata) praktično su irelevantne. Jednostavna instanca MongoDB-a s osnovnim indeksima pruža podmilisekundne upite. To nam omogućuje da damo prednost fleksibilnosti sheme nad ekstremnom optimizacijom upita.

Performanse autentikacije – Kod 100 korisnika, od kojih svaki možda šalje 10–50 zahtjeva u minuti (ukupno 1.000–5.000 zahtjeva/min), trošak autentikacije je zanemariv. Pristup s JWT validacijom pruža izvrsne performanse, ali iskreno, sesijska autentikacija temeljena na poslužitelju jednako bi dobro funkcionirala na ovom razmjeru. JWT biramo prvenstveno zbog čistoće arhitekture i buduće proširivosti, a ne zbog nužde u pogledu performansi.

Ograničavanje brzine (rate limiting) – Ograničenja broja zahtjeva postoje prvenstveno kao zaštita od pogrešnih skripti, a ne od zlonamjerne zlouporabe. Kod 10–20 pouzdanih akademskih korisnika profil rizika je prije nespretna pogreška nego namjerni napad. Velikodušna ograničenja (50–500 zahtjeva u minuti po korisniku) vjerojatno neće biti dosegnuta tijekom normalnog rada.

Operativna jednostavnost – Mali broj korisnika omogućuje jednostavnije operacije. Ručno odobravanje zahtjeva za dodatne sposobnosti je izvedivo — administratori koji pregledavaju 1–2 zahtjeva tjedno mogu to lako pratiti, dok bi 100 zahtjeva dnevno zahtijevalo automatizaciju. Analiza audit logova može se provoditi korištenjem osnovnih MongoDB upita, bez specijalizirane analitičke infrastrukture. Postupci sigurnosne kopije i povrata mogu ostati jednostavni, bez složene orkestracije oporavka od katastrofe.

Fokus na suradnju – Na ovom razmjeru svaki korisnik potencijalno može poznavati svakog drugog korisnika. Platforma može poticati stvarnu istraživačku suradnju, a ne anonimni masovni doprinos. Značajke poput dodjeljivanja recenzenata mogu uzimati u obzir individualnu stručnost, umjesto da se oslanjaju isključivo na algoritamsko uparivanje.

Ove karakteristike razmjera omogućuju nam da odgodimo određene arhitektonske složenosti koje su već u startu nužne većim platformama: balansiranje opterećenja (za početak je dovoljan jedan poslužitelj), replikaciju baze (isprva je dovoljna sigurnosna kopija), međuslojeve cachea (performanse baze su adekvatne), distribuirano praćenje (dovoljno je jednostavno logiranje) i napredni nadzor (dostatne su osnovne provjere zdravlja sustava). Te se sposobnosti mogu izgraditi kao modularni dodaci kada rast to opravda, umjesto da se platforma od početka prekomjerno inženjerski optereti.

3 Arhitektura sustava: temeljni izbori

3.1 Dvo-komponentna arhitektura

Dizajnerska odluka: Razdvojiti Knowledge Builder i Modeling Assistant

Platforma je arhitektonski podijeljena na dvije integrirane, ali neovisne aplikacije, umjesto na jedan monolitni sustav ili potpunu mikroservisnu arhitekturu.

Odluka da se izgrade dvije odvojene aplikacije proizlazi iz spoznaje da su kuracija znanja i izgradnja modela temeljno različite aktivnosti s različitim tehničkim zahtjevima. Knowledge Builder se usredotočuje na polu-nadziranu izgradnju baze znanja — kuratori putem sučelja chatbota komuniciraju s dokumentima, AI agenti izdvajaju kandidatske činjenice, a sustav održava veze s postojećim znanjem radi osiguranja konzistentnosti. To je radno intenzivna i spremišno zahtjevna operacija s kompleksnim zahtjevima za obradu prirodnog jezika i konverzacijsku AI. Suprotno tome, Modeling Assistant je pretežito aplikacija za čitanje koja konzumira kurirano znanje kroz konverzacijsko istraživanje radi podrške izgradnji scenarija i konceptualizaciji modela. Njegovi se tehnički zahtjevi vrte oko performansi upita, prolaska kroz znanje putem RAG metodologije i konverzacijske interakcije za razvoj modela.

Pokušaj da se oba slučaja upotrebe posluže iz jedne monolitne aplikacije prisilio bi na kompromise koji štete objema stranama. Jedinstveno korisničko sučelje moralo bi biti ili dovoljno moćno za naprednu kuraciju (što bi preopteretilo povremene korisnike) ili dovoljno jednostavno za istraživanje (što bi frustriralo napredne korisnike). Raspodjela resursa postaje problematična kada se obrada dokumenata natječe s izvršavanjem modela za CPU i memoriju. Brzina razvoja pati jer timovi koji rade na značajkama kuracije ometaju timove koji razvijaju mogućnosti modeliranja.

Razmatrali smo potpunu mikroservisnu arhitekturu sa zasebnim servisima za autentikaciju, obradu dokumenata, spremanje znanja, izvršavanje upita, sastavljanje modela i simulaciju. To bi pružilo maksimalnu fleksibilnost i neovisnu skalabilnost svakog segmenta. Međutim, operativna složenost u ovoj fazi bila bi prevelika. Distribuirane transakcije između servisa uvode načine kvara koji ne postoje u integriranim aplikacijama. Mrežna latencija između servisa dodaje režiju svakoj operaciji. Razvojni tim bi više vremena provodio upravljajući ugovorima između servisa i orkestracijom deploya nego gradeći funkcionalnosti koje korisnici trebaju.

Pristup s dvije komponente predstavlja promišljeni kompromis. Knowledge Builder i Modeling Assistant mogu se razvijati, testirati i isporučivati neovisno. Oni se skaliraju neovisno — KB s volumenom dokumenata, MA s opterećenjem simulacija. Ipak, dijele infrastrukturu autentikacije i modele korisnika, izbjegavajući probleme konzistentnosti koji muče istinski distribuirane sustave. Ako budući rast to zahtijeva, svaka se komponenta dodatno može razložiti na mikroservise bez utjecaja na drugu.

Ova arhitektura prihvaća činjenicu da se integracijski ugovori između KB-a i MA moraju pažljivo održavati. Promjene koje prekidaju kompatibilnost zahtijevaju koordinirane deploye. Integracijsko testiranje složenije je od jediničnog. Ti su troškovi stvarni, ali upravljivi i znatno manji od troškova koje izbjegavamo time što ne gradimo monolit niti preuranjene mikroservise.

3.2 Zajednička arhitektura baze podataka

Dizajnerska odluka: Jedna instanca MongoDB-a za sve podatke platforme

Knowledge Builder i Modeling Assistant dijele jednu MongoDB bazu podataka za sve podatke: grafove znanja, činjenice, dokumente, scenarije, modele, korisnike, uloge i audit logove.

Zajednička arhitektura baze podataka odražava odabir MongoDB-a kao temeljnog spremišta podataka platforme, prvenstveno odabranog zbog zahtjeva za skaliranjem grafa znanja i baze znanja. Kada je MongoDB baza za upravljanje znanjem (ključna funkcija platforme), korištenje istog sustava za upravljanje korisnicima, podatke o modeliranju i presjeke između komponenti pruža jaku konzistentnost i operativnu jednostavnost.

Zajednička baza osigurava trenutnu konzistentnost između komponenti. Kada korisnik stvori činjenicu u KB-u, ona je odmah dostupna MA-u za izgradnju scenarija. Kada scenarij referencira ID činjenice, MA upite izvršava nad istom bazom koju koristi KB, čime se uklanjaju kašnjenja sinkronizacije. Kada se promijeni uloga korisnika, obje komponente odmah vide nove dozvole. Takva jaka konzistentnost moguća je jer MongoDB ACID transakcije obuhvaćaju više kolekcija unutar jedne baze — ažuriranje uloge korisnika, zapis audita i poništavanje cachea dozvola događaju se atomarno.

Alternativne arhitekture s odvojenim bazama po komponenti zahtijevale bi kompleksnu sinkronizaciju. Kada bi KB i MA svaka imale vlastitu instancu MongoDB-a, stvaranje činjenice u KB-u zahtijevalo bi tok događaja radi replikacije u MA. Tijekom zastoja u replikaciji, korisnici MA-a ne bi vidjeli nove činjenice. Bilo bi potrebno rješavanje konflikata ako bi obje komponente mijenjale istu entitet znanja. Promjene dozvola korisnika tražile bi obrasce konačne konzistentnosti sa svojom složenošću.

Središnji autentikacijski servis s odvojenim bazama znanja predstavlja još jednu alternativu. Svaka bi komponenta imala vlastitu MongoDB bazu za domenske podatke, i pozivala bi zajednički auth API za provjeru korisnika. Takav pristup zadržava neovisnost komponenti, ali fragmentira graf znanja — MA ne može izravno upućivati upite KB kolekcijama znanja; mora koristiti HTTP API-je uz dodatnu latenciju. Još je problematičnije to što upiti koji presijecaju znanje i podatke o modeliranju (npr. "prikaži scenarije koji koriste ove činjenice") postaju skupi višekorak API pozivi.

Pristup zajedničkoj bazi optimizira primarni slučaj uporabe platforme: integrirano upravljanje znanjem i modeliranje. Upiti kroz komponente ("prikaži scenarije izgrađene na ovim činjenicama", "koje činjenice podržavaju ovaj parametar modela") izvršavaju se učinkovito unutar jedne baze. Granice transakcija uključuju i znanje i korisničke operacije — stvaranje činjenice i zapis audita događaju se atomarno. Sigurnosne kopije i povrat lakši su kad je cjelokupno stanje platforme u jednoj bazi.

Ova odluka eksplicitno povezuje KB i MA na podatkovnoj razini. Dijele evoluciju sheme, moraju koordinirati migracije baze i dijele načine kvara (prekid rada baze utječe na obje). Međutim, ta veza već postoji na logičkoj razini — MA se temeljno oslanja na znanje iz KB-a. Učinivši vezu eksplicitnom kroz zajedničku bazu zapravo pojednostavljujemo sustav: komponente su namjerno usko integrirane, umjesto da se tretiraju kao neovisno isporučivi mikroservisi.

Odvojene kolekcije unutar zajedničke baze osiguravaju logičko razdvajanje: KB primarno radi s facts, stylized_facts, documents, knowledge_graphs; MA primarno radi sa scenarios, models, results. Kolekcije users, api_keys i audit_logs dijele obje komponente. Takva organizacija na razini kolekcija omogućuje i buduće shardanje baze ako bude

potrebno — kolekcije znanja mogu se dijeliti prema taksonomskim skupinama (kralježnjaci vs. beskralježnjaci) ili tipu organizma (vodeni vs. kopneni), dok kolekcije korisnika mogu ostati nešardane.

3.3 Paket advandeb-shared-utils

Dizajnerska odluka: Python paket sa zajedničkom logikom autentikacije

Kreirati verzionirani Python paket koji backend servisi KB-a i MA-a uvoze za sve funkcionalnosti autentikacije, autorizacije i audita.

Zajednički paket utilitarnih funkcija utjelovljuje DRY (Don't Repeat Yourself) princip na arhitektonskoj razini. Logika generiranja JWT tokena nije trivijalna — uključuje kriptografsko potpisivanje, formatiranje claimova, rukovanje istekom i sigurnosne razmatranja. Pisanje tog koda dvaput (jednom za KB, jednom za MA) udvostručuje testiranje, povećava mogućnost pogrešaka i udvostručuje trošak održavanja kada se otkriju sigurnosni problemi. Još podmuklije, dupli cirani kod ima sklonost divergenciji tijekom vremena dok developeri mijenjaju jedan dio, a zaboravljaju drugi, što dovodi do suptilnih razlika u ponašanju komponenti.

Paket sadrži sve što mora biti identično u obje komponente: generiranje i validaciju JWT-ova, integraciju s Google OAuth klijentom, Pydantic modele za korisnike i uloge, funkcije za provjeru dozvola (has_base_role, has_capability), hashiranje i validaciju API ključeva, formatere za audit logove i utilitarne funkcije za spajanje na MongoDB. Kod koji se legitimno može razlikovati između komponenti — poslovna logika, API rute, modelske klase specifične za komponentu — namjerno ostaje izvan paketa.

Ta je granica ključna. Zajednički kod stvara čvrstu vezu, što je prihvatljivo za presječne brige poput autentikacije, ali štetno za poslovnu logiku. Ako KB-ova obrada dokumenata treba posebne sposobnosti, one pripadaju KB-u, a ne shared-utils paketu gdje bi nepotrebno stvarale ovisnosti za MA. Paket pruža primitive (provjeri ima li korisnik sposobnost X), a ne politike (kurator smije učitavati dokumente). Politike se sastavljaju iz primitiva unutar koda specifičnog za pojedinu komponentu.

Verzioniranje paketa omogućuje kontroliranu evoluciju. KB i MA privremeno mogu koristiti različite verzije paketa tijekom migracija, čime se izbjegava "big-bang" nadogradnja. Semantičko verzioniranje prenosi informaciju o kompatibilnosti — patch verzije popravljaju bugove, minor verzije dodaju značajke unatrag kompatibilno, major verzije signaliziraju lomljive promjene. CI/CD cjevovodi mogu automatski testirati obje komponente s novim verzijama paketa prije nego što se puste u upotrebu.

Alternativa s dupliciranim kodom naizgled je jednostavnija — nema paketa za objavu, nema koordinacije verzija, svaki tim kontrolira vlastiti kod. Ta je jednostavnost iluzorna. Kada sigurnosna revizija otkrije ranjivost u rukovanju JWT-ovima, popravke treba pažljivo replicirati u oba koda. Kada se dodaju nove sposobnosti, logiku provjere dozvola treba sinkronizirano nadograditi. Noćna mora debugiranja kada se komponente ponašaju malo drugačije zbog divergirajućeg koda daleko nadmašuje trošak održavanja zajedničkog paketa.

4 Arhitektura autentikacije

4.1 Google OAuth 2.0 kao primarna autentikacija

Dizajnerska odluka: Koristiti Google OAuth 2.0 za autentikaciju u web sučelju

Korisnici se autentificiraju putem Google OAuth-a, a ne putem klasičnih korisničkih imena/lozinki ili drugih identitetskih pružatelja.

Odabir Google OAuth-a odražava pragmatičnu procjenu naše korisničke baze i institucionalne stvarnosti. Ciljana publika AdvanDEB-a sastoji se prvenstveno od akademskih istraživača, od kojih većina već posjeduje Google račune (Gmail ili Google Workspace). Tražiti od tih korisnika da kreiraju novi par korisničko ime–lozinka stvara trenje koje smanjuje usvajanje platforme. Zamor od lozinki je stvaran — korisnici ponovno koriste iste lozinke na više mjesta (sigurnosni rizik) ili zaboravljaju jedinstvene lozinke (opterećenje podrške). Google OAuth u potpunosti uklanja to trenje i delegira sigurnost lozinki organizaciji s daleko većom sigurnosnom ekspertizom od naše.

Sa sigurnosnog stajališta, Google OAuth znači da mi nikada ne vidimo, ne pohranjujemo niti prenosimo korisničke lozinke. Našu bazu podataka napadač ne može kompromitirati radi lozinki jer ih jednostavno nemamo. Korisnici koji uključe dvofaktorsku autentikaciju na svom Google računu automatski prenose tu zaštitu na AdvanDEB. Googleov sigurnosni tim nadzire kompromitirane vjerodajnice, neuobičajene obrasce prijave i brute-force napade — usluge koje bismo sami teško mogli implementirati.

Faktor vjerodostojnosti ne treba podcijeniti. Akademske institucije i financijeri projekata sve više proučavaju sigurnosne prakse. Izjava "koristimo Google OAuth i nikad ne rukujemo lozinkama" daleko je uvjerljivija od "implementirali smo vlastitu pohranu lozinki pomoću bcrypta", bez obzira na tehničku primjerenost potonje. Institucionalne e-mail adrese (istraživač@univerzitet.hr) pružaju laganu provjeru identiteta, a potencijalna buduća integracija s ORCID-om lakše se ostvaruje kroz Googleov OAuth okvir.

Razmatrali smo podršku za više OAuth pružatelja — GitHub za developere, ORCID za istraživače, institucionalni SAML za sveučilišta. Svaki dodatni pružatelj umnožava teret testiranja i održavanja. Različiti pružatelji vraćaju podatke o korisniku u različitim formatima, što zahtijeva logiku normalizacije. Korisnici s više računa kod različitih pružatelja trebaju mehanizme povezivanja računa. Ove bi složenosti mogle biti opravdane za platformu s raznolikom publikom, no naša je korisnička baza koncentrirana u akademskoj zajednici, gdje Google dominira.

Autentikacija korisničkim imenom i lozinkom pružila bi potpunu kontrolu i neovisnost o vanjskim servisima. Međutim, zahtijevala bi implementaciju sigurne pohrane lozinki (relativno jednostavno), tijekova za resetiranje lozinki (složenije), verifikaciju e-maila (treba infrastrukturu za e-poštu), provjeru jačine lozinke (što korisnike iritira) i zaključavanje računa nakon ponovljenih neuspjelih pokušaja (opterećenje podrške). Još važnije, korisničko iskustvo pati jer se istraživači nerado prijavljuju s novim lozinkama na domenski specifične platforme.

OAuth pristup mijenja ovisnost o Googleu za jednostavnost i sigurnost. Hipotetski Googleov krah ili promjena politika mogli bi utjecati na našu platformu, ali taj se rizik čini znatno manjim od sigurnosnih rizika kućno razvijene autentikacije. Ako se okolnosti promijene, apstrakcija autentikacije omogućuje dodavanje alternativnih pružatelja bez narušavanja postojećih korisničkih računa.

4.2 JWT tokeni za upravljanje sesijama

Dizajnerska odluka: Izdavati JWT tokene nakon OAuth autentikacije

Nakon uspješne Google OAuth prijave, platforma izdaje JWT pristupne i osvježavajuće tokene umjesto održavanja sesija na poslužitelju.

OAuth osigurava identitet korisnika, ali JWT tokeni osiguravaju stanje sesije. Kada se korisnik uspješno autentificira putem Googlea, znamo tko je on, ali trebamo mehanizam kojim taj korisnik može slati naknadne zahtjeve bez ponovne autentikacije. Tradicionalne web sesije pohranjuju stanje na poslužitelju (tipično u bazi ili Redis-u) i klijentu daju cookie sa ID-om sesije. Svaki zahtjev tada zahtijeva dohvat sesije kako bi se utvrdio identitet i dozvole korisnika.

JWT tokeni inverziraju taj model tako da kodiraju stanje sesije (ID korisnika, uloga, sposobnosti) izravno u kriptografski potpisan token. Poslužitelj generira token privatnim ključem, a svaki poslužitelj s odgovarajućim javnim ključem može ga validirati bez pristupa bazi. Provjera dozvola postaje čista CPU operacija — parsiranje tokena, provjera potpisa, izdvajanje claimova — koja se izvršava u mikrosekundama. Ova bezdržavna arhitektura uklanja usko grlo pohrane sesija koje ograničava tradicionalne sustave.

Implikacije na performanse su značajne. Tradicionalni sustav sa sesijama upućuje upit bazi ili Redis-u za svaki autentificirani zahtjev. Na 100 zahtjeva u sekundi, to je 100 upita prema spremištu sesija samo za autentikaciju, prije bilo kakve poslovne logike. JWT validacija ne zahtijeva I/O operacije, što je čini praktično besplatnom u odnosu na trošak poslovne logike. Još važnije, ovo omogućuje horizontalno skaliranje bez afiniteta na sesije. Zahtjevi se mogu usmjeriti na bilo koji backend poslužitelj bez obzira na to "posjeduje" li taj poslužitelj stanje sesije, jer stanje sesije zapravo ne postoji na poslužitelju.

Sigurnosna razmatranja oko JWT-a često se pogrešno tumače. Uobičajena kritika da se "JWT-ovi ne mogu opozvati" prenaglašena je — ispravno dizajnirana arhitektura JWT-a to ublažava kratkim vremenom valjanosti i osvježavajućim tokenima. Naši pristupni tokeni istječu nakon jednog sata, ograničavajući vremenski prozor u kojem ukradeni token ostaje valjan. Osvježavajući tokeni, koji traju 30 dana, pohranjeni su u bazi i mogu se odmah opozvati ako se otkrije kompromitacija. Korisničko iskustvo ostaje glatko jer je osvježavanje tokena automatsko i nevidljivo.

Bezdržavnost također omogućuje graciozno rukovanje promjenama dozvola. Kada se promijeni uloga korisnika, promjena stupa na snagu unutar jednog sata (kada trenutačni pristupni token istekne i osvježi se). Za većinu slučajeva uporabe taj je odmak prihvatljiv. Za hitne slučajeve (suspendirani korisnik, sigurnosni incident), administratori mogu forsirati odjavu poništavanjem osvježavajućih tokena, čime se zahtijeva trenutačna ponovna autentikacija.

Sadržaj tokena je base64-enkodiran JSON, čitljiv svakome tko ga presretne. To nije ranjivost ako se ispravno razumije — osjetljivi podaci nikada ne smiju biti stavljeni u JWT claimove. Naši tokeni sadrže samo ID korisnika, ulogu i sposobnosti, što su podaci koje korisnik ionako poznaje. Za prave tajne bila bi potrebna dodatna enkripcija, no u našem modelu autentikacije takvih tajni u tokenima nema.

Alternativni pristupi uključuju sesije na poslužitelju (jednostavno, ali slabo skalabilno), korištenje isključivo OAuth osvježavajućih tokena (zahtijeva komunikaciju s Googleom pri svakom zahtjevu) ili vlastite API sesijske tokene (loše iznova implementirani JWT). Pristup s JWT-om koristi godine industrijskog iskustva i dobro testirane biblioteke, što ga čini pragmatičnim izborom za bezdržavnu autentikaciju u distribuiranim sustavima.

4.3 API ključevi za programski pristup

Dizajnerska odluka: Podržati dugotrajne API ključeve uz JWT tokene

Platforma pruža autentikaciju API ključevima za skripte i automatizaciju, uz OAuth/JWT za interaktivne korisnike.

Istraživači često trebaju programski pristup platformi za masovne operacije, podatkovne pipelineove i ponovljive tijekove rada. Biolog može napisati Python skriptu za učitavanje 500 radova odjednom, a ekolog može imati cron posao koji svake noći izvozi ažurirane skupove podataka. OAuth tijekovi nisu prikladni za ove scenarije jer zahtijevaju interakciju u pregledniku kako bi se dovršio proces autentikacije. API ključevi rješavaju to tako da pružaju tajni token koji se može izravno ugraditi u skripte.

Dizajn našeg sustava API ključeva uči iz najboljih praksi industrije kakve primjenjuju platforme poput GitHuba i Stripea. Svaki ključ ima prepoznatljiv prefiks (advk_) nakon kojeg slijedi slučajno generiran niz znakova. Prefiks omogućuje brzo prepoznavanje u logovima i otkrivanje slučajnog izlaganja (npr. pretragom javnih repozitorija za "advk_"). U našoj bazi pohranjuje se samo hash tajnog dijela ključa, korištenjem SHA-256. Kada dolazni zahtjev sadrži API ključ, hashiramo ga i uspoređujemo s pohranjenim hashovima, na isti način kao što se provjeravaju lozinke.

Takav pristup znači da kompromitacija baze ne otkriva aktivne ključeve. Napadač koji pristupi bazi vidi hashove, ali iz njih ne može izvesti originalne ključeve. Korisnici vide ključ u čistom tekstu samo jednom, odmah nakon generiranja, uz eksplicitno upozorenje da ga sigurno pohrane. To je manje korisnički prijateljski od povratno dohvatljivih tokena, ali znatno sigurnije — ne postoji endpoint koji bi napadačima omogućio "prikaži moje API ključeve".

API ključevi imaju neovisni životni ciklus u odnosu na korisničke sesije. Istraživač može koristiti JWT tokene za interaktivan rad u web sučelju, a istodobno imati API ključeve za svoje skripte za učitavanje. Ako je API ključ kompromitiran, može se opozvati bez utjecaja na web pristup. Obrnuto, ako korisnik promijeni Google lozinku (što zahtijeva ponovnu autentikaciju), njegovi API ključevi nastavljaju raditi. Ta neovisnost ključna je za automatizaciju — cron poslovi ne bi smjeli prestati raditi zato što je istraživač uključio 2FA na Google računu.

Svaki API ključ nosi scopeove koji se automatski izvode iz uloge i sposobnosti vlasnika. Kurator s pristupom analitici dobiva API ključeve koji mogu čitati i pisati znanje te izvoziti podatke. Explorator dobiva ključeve koji mogu samo čitati objavljeno znanje. Scopeovi se periodički ponovno izračunavaju (mogli bismo ih provjeravati pri svakom zahtjevu, ali cacheiranje na sat vremena balansira sigurnost i performanse). Kada se promijene sposobnosti korisnika, mijenjaju se i dozvole povezanih API ključeva.

Automatsko isteknuće nakon 90 dana prisiljava periodičnu rotaciju ključeva, ograničavajući prozor u kojem kompromitirani, ali neotkriveni ključ ostaje valjan. Korisnici po želji mogu postaviti kraće vrijeme trajanja (korisno za privremenu automatizaciju). Vrijeme zadnje uporabe pomaže u identificiranju i brisanju napuštenih ključeva koji predstavljaju sigurnosni rizik.

Ograničavanje brzine vrši se po API ključu, a ne po korisniku, što omogućuje fino podešavanje. Istraživač može imati jedan ključ za masovno učitavanje (veći limit, dozvole za pisanje) i drugi za vanjske suradnike (niži limit, read-only). Takva separacija ciljeva nemoguća je u sustavima s jednim tokenom.

4.4 Jedinstvena prijava (SSO) između komponenti

Dizajnerska odluka: Isti autentikacijski tokeni vrijede u KB i MA

Korisnici se autentificiraju jednom kroz Knowledge Builder, a njihovi tokeni besprijekorno funkcioniraju u Modeling Assistantu bez zasebne prijave.

Jedinstvena prijava predstavlja odluku o korisničkom iskustvu s dubokim arhitektonskim implikacijama. Prirodni tijek rada AdvanDEB korisnika podrazumijeva fluentno kretanje između kuracije znanja i izgradnje modela. Istraživač može učitavati radove u KB, koristiti agente za izdvajanje podataka, a zatim se prebaciti na MA kako bi izgradio scenarije koristeći te podatke. Zahtijevati ponovnu autentikaciju na granici između KB-a i MA stvorilo bi umjetno trenje koje narušava viziju integrirane platforme.

Tehnički, SSO zahtijeva da obje komponente vjeruju istom JWT potpisnom ključu i istoj bazi korisnika. Kada KB izda JWT nakon OAuth autentikacije, potpisuje ga tajnim ključem spremljenim na siguran način (varijabla okoline, ne u kodu). MA posjeduje isti tajni ključ i stoga može validirati token koji je izdao KB. Token sadrži ID korisnika, ulogu i sposobnosti — sve što MA treba kako bi lokalno provjerio dozvole bez konzultiranja KB-a.

Ovaj model zajedničkog povjerenja povezuje KB i MA na sloju autentikacije. Moraju koordinirati rotaciju ključeva ako je potpisni ključ kompromitiran. Moraju se usuglasiti oko formata tokena i strukture claimova. Moraju identično interpretirati sposobnosti. Takve su veze prihvatljive jer je autentikacija inherentno zajednička briga — koncept "različitog identiteta" u KB-u i MA-u u osnovi bi narušio platformu.

Alternativa s odvojenom autentikacijom zahtijevala bi od korisnika dvije prijave, vjerojatno s istim Google računom, čime bi nastali identični korisnički zapisi u oba sustava. To pruža potpunu neovisnost komponenti, ali uz ozbiljan trošak u korisničkom iskustvu. Korisnici bi morali upravljati s dva skupa API ključeva. Promjene dozvola trebalo bi sinkronizirati. Audit tragovi bi bili fragmentirani. Iako nijedan od tih problema nije nerješiv, svi su u suprotnosti s ciljevima platforme.

Konfiguracija cookie domene omogućuje besprijekorne prijelaze u pregledniku. Ako je KB hostan na kb.advandeb.org, a MA na ma.advandeb.org, cookie se može postaviti za nad-domenu advandeb.org, čineći ga dostupnim objema. Pristupni JWT token obično se pohranjuje u localStorage preglednika (dostupan kroz JavaScript), dok se osvježavajući token pohranjuje u HTTP-only cookie (zaštićen od XSS-a). Ova podjela pruža sigurnost u dubini — krađa pristupnog tokena iz localStoragea daje pristup samo tijekom jednog sata.

Specifične dozvole komponenti provjeravaju se lokalno korištenjem zajedničke logike provjere dozvola iz advandeb-shared-utils paketa. MA provjerava da je za izradu scenarija potrebna uloga kuratora, KB provjerava da je za pokretanje agenata potrebna sposobnost agent_access. Oba se oslanjaju na istu funkciju has_capability(), čime se osigurava konzistentna interpretacija. Audit log bilježi u kojoj je komponenti svaka radnja izvedena, omogućujući sigurnosnu analizu specifičnu po komponentama, ali uz zadržavanje jedinstvenog identiteta korisnika.

Ova SSO arhitektura pretpostavlja da KB i MA imaju slične sigurnosne zahtjeve. Ako buduće širenje doda komponentu s višim sigurnosnim potrebama (npr. modul za kliničke podatke), ta bi komponenta mogla zahtijevati odvojenu autentikaciju s dodatnim faktorima. Arhitektura to može podržati kroz zahtjeve za specifičnim sposobnostima — klinički modul mogao bi tražiti sposobnost koja se dodjeljuje tek nakon dodatne provjere. Osnovni SSO služi uobičajenim slučajevima, uz mogućnost iznimki za posebne potrebe.

5 Model uloga: dozvole temeljene na sposobnostima

5.1 Evolucija kroz tri verzije

Dizajnerska odluka: Osnovne uloge s opcionalnim sposobnostima

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

Arhitektura modela sposobnosti razlikuje između *tko ste* (vaša osnovna razina pristupa platformi) i *što možete raditi* (vaše dozvole za određene operacije). Ovo razdvajanje rješava temeljni izazov u dizajnu dozvola: istraživačima trebaju različite sposobnosti u različitim fazama rada, ali njihov temeljni identitet i razina pristupa ostaju stabilni.

Identitet istraživača kao kuratora je stabilan, ali se potreba za pristupom agentima ili mogućnostima izvoza podataka mijenja tijekom vremena. Novi korisnik ne treba odmah pristup agentima — najprije treba razumjeti platformu i kurirati dio znanja. Nakon stjecanja iskustva može zatražiti pristup agentima kako bi skalirao svoj rad. Kasnije, ako pokaže stručnost, može zatražiti status recenzenta. Takva postepena progresija odražava prirodan razvoj tijeka rada, umjesto da korisnike prisiljava u krute kategorije.

Arhitektura to ostvaruje kroz osnovne uloge uz opcionalne sposobnosti. Tri osnovne uloge odražavaju stvarne razlike u razini pristupa platformi: Administrator (upravlja platformom), Knowledge Curator (kreira sadržaj i šalje ga na recenziju), Knowledge Explorator (čita sadržaj). Osnovni kuratori mogu učitavati dokumente, stvarati činjenice i slati znanje na recenziju — temeljni tijek doprinosa. Sposobnosti predstavljaju dodatne dozvole koje kuratori traže prema potrebi: Agent Access (pokretanje AI agenata), Analytics Access (masovni izvoz podataka), Reviewer Status (odobravanje prijava drugih kuratora). Ovaj model pruža i jednostavnost (tri osnovne uloge pokrivaju sve korisnike) i fleksibilnost (sposobnosti se mogu slobodno kombinirati kako bi odgovarale pojedinačnim tijekovima rada).

Ključni uvid je prepoznavanje da istraživači prirodno nose više "kapa". Ista biologinja učitava radove (aktivnost kuratora), pokreće agente za izdvajanje kako bi obradila literaturu u velikom opsegu (agent access) i izvozi agregirane podatke za analizu (analytics access), često unutar jedne istraživačke sesije. Umjesto da je prisilimo da se prebacuje između identiteta uloga ili da zauvijek ostane zaključana u jednoj ulozi, model sposobnosti dopušta da joj se dozvole organski razvijaju u skladu s potrebama i iskazanim povjerenjem.

Tijek zahtjeva za sposobnostima ugrađuje učenje i izgradnju povjerenja u evoluciju dozvola. Novi korisnici počinju s osnovnim kuratorskim pristupom, uče sustav i doprinose znanju. Kada im zatreba pristup agentima, podnose zahtjev s obrazloženjem koje objašnjava slučaj uporabe. Administratori pregledavaju zahtjev, uzimajući u obzir dosadašnji rad i potrebe korisnika. Takav model postepeno rastućeg pristupa usklađuje sigurnost (sposobnosti se daju temeljem dokazanog povjerenja) s upotrebljivošću (korisnici nisu preplavljeni dozvolama koje još ne trebaju).

Ovaj pristup prihvaća da su neke sposobnosti rijetke, a druge česte. Većina će kuratora na kraju zatražiti status recenzenta jer je recenziranje prirodna evolucija doprinosa. Manje će ih trebati pristup analitici jer masovni izvoz podataka služi specijaliziranim slučajevima uporabe. Model se prirodno prilagođava toj distribuciji — uobičajene se sposobnosti rutinski odobravaju, dok neuobičajeni zahtjevi izazivaju dodatni nadzor.

5.2 Tri sposobnosti

Dizajnerska odluka: Agent Access, Analytics Access i Reviewer Status kao zasebno zatražive sposobnosti

Ove tri konkretne sposobnosti predstavljaju smisleno razgraničenje dozvola, a ne proizvoljne podjele.

Svaka sposobnost odgovara stvarnoj sigurnosnoj ili kvalitetetnoj brizi koja opravdava tretiranje te sposobnosti kao odvojene dozvole. Agent Access kontrolira mogućnost pokretanja AI agenata koji pozivaju vanjske API-je (primjerice pružatelje LLM-ova) i troše kvote sustava. Neispravno konfigurirani ili pogrešno korišteni agenti mogli bi generirati velike troškove API poziva ili degradirati performanse sustava. Nema svaki kurator potrebu za tom moći — mnogi učinkovito rade ručnim kreiranjem zapisa znanja. Oni kojima agenti trebaju (za masovno izdvajanje iz većeg broja radova) mogu zatražiti pristup uz obrazloženje koje administratorima omogućuje procjenu opravdanosti zahtjeva.

Sposobnost je odvojena jer se profil rizika razlikuje od osnovne kuracije. Kurator koji ručno unosi činjenice može činiti štetu samo ljudskim tempom — unošenjem pogrešnih podataka ili slanjem sadržaja slabije kvalitete. Agent koji stalno radi u petlji može izazvati štetu brzinom stroja — tisuće API poziva, masovni upisi u bazu koji preplavljuju infrastrukturu, potrošene kvote. Ova razlika u riziku opravdava dodatnu kontrolnu točku pri odobravanju.

Analytics Access upravlja masovnim izvozom podataka i generiranjem API ključeva. Kuratori mogu izvoziti pojedinačne činjenice ili male skupove podataka za vlastitu upotrebu i bez ove sposobnosti. Analytics Access omogućuje izvoz cjelokupne baze znanja, generiranje API ključeva s opsežnim read dozvolama i izvođenje kompleksnih upita koji mogu utjecati na performanse baze. Ova sposobnost postoji jer masovni izvoz olakšava izvlačenje podataka — zlonamjeran korisnik mogao bi sve preuzeti i objaviti na drugom mjestu. Iako je puno znanja javno, njegova agregacija predstavlja vrijednost koju korisnici možda žele zaštititi.

Sposobnost nije potrebna u uobičajenim istraživačkim tijekovima rada. Biolog koji modelira populacije riba treba upite nad relevantnim vrstama, a ne izvoz cijele baze. Analytics Access služi istraživačima koji rade meta-analize preko više domena, podatkovnim znanstvenicima koji treniraju modele strojnog učenja na korpusu znanja ili institucionalnim partnerima koji žele održavati lokalne kopije. Ti legitimni slučajevi uporabe opravdavaju dodjelu sposobnosti, ali čine mali udio korisnika.

Reviewer Status kontrolira pristup redu za recenziju i mogućnost odobravanja ili odbijanja dostavljenog znanja. Kontrola kvalitete ključna je za očuvanje vjerodostojnosti platforme — loše recenzije propuštaju nekvalitetan sadržaj, dok pretjerano odbijanje obeshrabruje doprinositelje. Nije svaki iskusni kurator nužno dobar recenzent; neki su stručnjaci za doprinos znanju, ali se ne osjećaju ugodno u procjeni tuđeg rada. Obrnuto, neki vrsni recenzenti možda sami malo doprinose, ali pružaju značajnu vrijednost kroz kontrolu kvalitete.

Tretiranje recenzije kao sposobnosti, a ne zasebne uloge, prepoznaje da su recenziranje i kuracija povezane, ali različite aktivnosti. Ista osoba može podnijeti činjenicu (kao kurator) i poslije recenzirati tuđu činjenicu (s Reviewer Status sposobnošću). Takav dvojak obrazac aktivnosti prirodan je u znanstvenoj suradnji — domenski stručnjaci i doprinose i evaluiraju doprinose drugih.

Ove tri sposobnosti proizašle su iz analize obrazaca dozvola i sigurnosnih zahtjeva.

Analizirali smo što kuratori rade i identificirali granice dozvola s implikacijama na sigurnost ili kvalitetu koje vrijedi posebno štititi. Broj je tri jer upravo toliko smislenih granica prepoznajemo: operacije agenata (rizik troška i performansi), masovni pristup podacima (rizik izvlačenja) i kontrola kvalitete (integritet sadržaja). Svaka sposobnost adresira zasebnu brigu koja opravdava tijek odobravanja.

5.3 Zašto je Knowledge Explorator važan

Dizajnerska odluka: Pružiti ulogu samo za čitanje za korisnike koji ne doprinose

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

Uloga Knowledge Explorator odražava spoznaju da ne doprinose svi vrijedni korisnici sadržajem. Studenti koji uče o domeni, novinari koji istražuju priče, donositelji politika koji prikupljaju informacije i zainteresirana šira javnost koja istražuje znanost možda nikada neće učitati rad ili stvoriti činjenicu. Ako je kurator minimalna potrebna uloga, ti korisnici su ili isključeni ili prisiljeni tražiti dozvole koje im ne trebaju i koje neće koristiti.

Pružanje uloge samo za čitanje snižava prag za uključivanje u platformu. Odobravanje je jednostavno jer exploratori ne mogu oštetiti podatke niti značajno trošiti resurse. Administrator može brzo, pa čak i automatski, odobriti zahtjeve za explorator ulogu, znajući da je rizik minimalan. Neki će exploratori s vremenom postati kuratori kako steknu samopouzdanje i uoče praznine koje mogu popuniti. Taj prirodan prijelaz od potrošača prema doprinositelju čest je u online zajednicama; blokirati ga nepotrebnim dozvolama na ulazu značilo bi usporiti rast zajednice.

Uloga također služi institucionalnim slučajevima uporabe. Sveučilište može poželjeti da svi studenti diplomskog studija biologije imaju pristup platformi u sklopu nastave, bez da im je potreban kuratorski pristup. Istraživački institut može dati explorator pristup administrativnom osoblju koje treba pretraživati informacije, ali ne bi smjelo doprinositi sadržajem. Takvi slučajevi zahtijevaju lagan stupanj dozvola koji ne nosi odgovornosti kuracije.

Gledano iz perspektive skaliranja, exploratori generiraju opterećenje koje ne donosi izravan prihod (konzumacija bez doprinosa), ali to je prihvatljivo u službi misije platforme. Dijeljenje znanstvenog znanja profitira od šireg pristupa, a računalni trošak posluživanja upita za čitanje je nizak. Ograničavanje pristupa isključivo na doprinositelje stvorilo bi zatvorenu zajednicu u suprotnosti s idealima otvorene znanosti koji motiviraju platformu.

Exploratori imaju mogućnosti poput spremanja pretraga, stvaranja privatnih bilješki i izvoza manjih skupova podataka za osobnu upotrebu. Te značajke omogućuju produktivno korištenje, a da se pritom ne ugrozi zajednička baza znanja. Značajka privatnih bilješki posebno podržava potrebe učenja — studenti mogu zapisivati svoje komentare i tumačenja bez da ti zapisi ulaze u javni prostor znanja.

Razlika između exploratora i kuratora je temeljna: kuratori mogu predlagati promjene u bazi znanja slanjem činjenica na recenziju, dok exploratori imaju read-only pristup. Čak i bez dodatnih sposobnosti, osnovna kuratorska uloga omogućuje temeljni tijek doprinosa — učitavanje dokumenata, kreiranje činjenica, slanje znanja na recenziju. Explorator označava "ovaj račun je za čitanje i učenje", dok kurator označava "ovaj račun može doprinositi bazi znanja". Razlika nije u čekanju na dozvole, već u namjeni: konzumacija

nasuprot doprinosu.

6 Dizajn baze podataka: MongoDB kao temelj

6.1 Zašto MongoDB umjesto relacijskih baza

Dizajnerska odluka: Koristiti MongoDB kao primarnu bazu za cijelu platformu

MongoDB je odabran kao temeljna baza podataka za AdvanDEB, prvenstveno zbog zahtjeva grafa znanja i baze znanja, pri čemu je upravljanje korisnicima izgrađeno na istoj infrastrukturi radi operativne jednostavnosti.

Odabir MongoDB-a proizlazi iz temeljne misije platforme: upravljanje mrežno povezanim grafom znanja i skalabilnom bazom znanja za bioenergetska istraživanja. Grafovi znanja sastoje se od heterogenih čvorova (vrste, fiziološki procesi, metabolički putevi, bioenergetski parametri) s različitim svojstvima i odnosima. Činjenice izdvojene iz literature imaju različitu strukturu ovisno o sadržaju. Stilizirane činjenice sintetiziraju različit broj izvorišnih činjenica. Ova inherentna heterogenost čini krute relacijske sheme problematičnima — svaka nova vrsta entiteta ili obrazac odnosa zahtijevao bi migracije sheme.

MongoDB-ov dokumentni model prirodno predstavlja grafove: čvorovi su dokumenti s fleksibilnim svojstvima, a bridovi su ugniježđeni nizovi ili reference. Čvor vrste može imati {taxonomy, habitat, metabolic_rate, reproduction_parameters}, dok fiziološki proces može imati {energy_allocation, transport_mechanisms, temporal_dynamics}. Oba koegzistiraju u istoj kolekciji s različitim shemama. Graf upiti poput "pronađi sve čvorove povezane s čvorom X unutar 2 skoka" prevode se u MongoDB agregacijske pipelineove, dok isti upit u relacijskoj bazi zahtijeva rekurzivne CTE konstrukte ili više self-joinova koji loše skaliraju.

Zahtjevi skalabilnosti za mrežno povezane grafove znanja snažno pogoduju dokumentnim bazama. Kako baza znanja raste na tisuće međusobno povezanih entiteta, horizontalno skaliranje MongoDB-a kroz shardanje omogućuje raspodjelu dijelova grafa po poslužiteljima. Graf upiti ostaju učinkoviti jer MongoDB može paralelno raditi preko shardova. PostgreSQL s graf ekstenzijama (poput AGE) pruža graf sposobnosti, ali je shardanje složenije i manje zrelo. Namjenske graf baze (Neo4j, ArangoDB) izvrsne su za graf upite, ali povećavaju operativnu složenost — održavanje dva sustava baza (graf za znanje, relacijski za korisnike) udvostručuje trošak.

Kada je MongoDB jednom odabran za pohranu grafa znanja (primarni podatak platforme), korištenje iste tehnologije za upravljanje korisnicima pruža operativnu konzistentnost umjesto uvođenja druge baze. Metapodaci o korisnicima zapravo profitiraju od fleksibilnosti MongoDB-a: strukture afilijacija razlikuju se među institucijama, istraživačka područja koriste različite taksonomije, a buduće značajke (integracija s ORCID-om, institucionalne uloge, popisi publikacija) mogu se dodati bez migracija.

Graf-orijentirani slučaj uporabe oblikuje specifične MongoDB značajke koje koristimo: složeni indeksi na vezama čvorova (source_id + edge_type + target_id) omogućuju brzu translaciju kroz graf, agregacijski pipelineovi implementiraju graf algoritme (najkraći putevi, detekcija zajednica), a fleksibilna shema dopušta evoluciju ontologije grafa bez zastoja. Te graf-centrične značajke razlikuju MongoDB od tradicionalnih dokumentnih

baza optimiziranih za jednostavne CRUD operacije.

Polje capabilities u dokumentima korisnika ilustrira sekundarne koristi MongoDB-a za upravljanje korisnicima. U relacijskom modelu, sposobnosti bi zahtijevale međutablicu s JOIN-ovima radi provjere dozvola. MongoDB sposobnosti pohranjuje kao nizove, čime su provjere trivijalne, a ažuriranja jednostavna. Ipak, ta fleksibilnost u upravljanju korisnicima samo je ugodna nuspojava, a ne glavni razlog — platforma bi koristila MongoDB i da je upravljanje korisnicima složenije, jer su zahtjevi grafa znanja presudni.

JSON-native pohrana osigurava neprekinuti tok podataka: znanje izdvojeno iz dokumenata zapisuje se kao JSON, MongoDB ga pohranjuje kao BSON, HTTP API-ji vraćaju JSON. Ovo poravnanje proteže se i na podatke o korisnicima, ali je osobito važno za entitete znanja gdje kompleksne ugniježđene strukture (čvorovi grafa s svojstvima, činjenice s nizovima entiteta, stilizirane činjenice s referencama na dokaze) prirodno mapiraju na ugniježđeni JSON bez ORM nesklada.

Operativna jednostavnost arhitekture s jednom bazom postaje sve važnija kako volumeni podataka rastu. Primarni podaci platforme — graf znanja — skalirat će na gigabajte ili terabajte (tisuće radova, milijuni činjenica, kompleksni međusobno povezani grafovi). Ekspertiza u MongoDB-u, postupci sigurnosne kopije, optimizacija upita i nadzorna infrastruktura razvijeni za pohranu znanja automatski se primjenjuju i na podatke o korisnicima. Dodavanje PostgreSQL-a podijelilo bi operativni fokus, a ne bi riješilo glavni izazov: skalabilnu pohranu mrežno povezanog grafa znanja.

Jamstva konzistentnosti u MongoDB-u dramatično su poboljšana s uvođenjem transakcija nad više dokumenata (uvedene u MongoDB 4.0, zrele od 4.4). Stvaranje činjenice i povezivanje te činjenice s čvorom grafa znanja može se obaviti atomarno. Iako su transakcije u PostgreSQL-u zrelije, MongoDB-ove mogućnosti su dovoljne za operacije upravljanja znanjem. Obrasci transakcija na platformi (stvaranje entiteta znanja + ažuriranje odnosa) dobro se uklapaju u tranzakcijski model MongoDB-a.

Odluka eksplicitno priznaje slabosti MongoDB-a za određene obrasce upita. Kompleksni analitički upiti preko mnogih kolekcija s intrikatnim JOIN-ovima bili bi jednostavniji u PostgreSQL-u. No obrasci upita na platformi favoriziraju dokumentne i graf operacije: "pronađi činjenice koje odgovaraju tekstualnom upitu", "prođi kroz graf od čvora X", "dohvati stiliziranu činjenicu sa svim izvorišnim činjenicama", "pronađi entitete povezane putem zadane putanje". Ti obrasci koriste MongoDB-ove snage (tekstualno pretraživanje, agregacijske pipelineove, denormaliziranu pohranu) umjesto njegovih slabosti (kompleksni JOIN-ovi).

Alternativne NoSQL baze procijenjene su u odnosu na zahtjeve grafa znanja:

- **Neo4j**: Superiorni graf upiti, ali uvodi raznolikost baza, zahtijeva učenje Cyphera i ima kompleksno shardanje
- **ArangoDB**: Multi-model (dokument + graf), ali s manje zrelom ekosferom i manjom zajednicom
- **Cassandra**: Optimizirana za pisanja i time-series, ali graf upiti su teški
- **DynamoDB**: Upravljana NoSQL baza, ali graf upiti zahtijevaju kompleksne obrasce pristupa, a troškovi nepredvidljivo skaliraju

MongoDB balansira graf sposobnosti (dovoljno dobre), fleksibilnost dokumenata (izvrsna), operativnu zrelost (dokazana u produkciji) i podršku zajednice (opsežna). Za mali tim koji gradi istraživačku platformu, profil "dobar u mnogočemu" nadmašuje specijalizirane baze koje su "izvrsne u jednoj stvari".

6.2 Dizajn kolekcija i modeliranje podataka

Dizajnerska odluka: Četiri kolekcije za podatke o korisnicima — users, capability_requests, api_keys, audit_logs

Podaci povezani s korisnicima organizirani su u ove četiri kolekcije, a ne u jednu kolekciju ili strogo normaliziranu relacijsku shemu.

Kolekcija users predstavlja trenutačno stanje identiteta i dozvola korisnika. Svaki dokument sadrži sve informacije potrebne za autentikaciju i autorizaciju korisnika: Google ID (nepromjenjiv, koristi se za dohvat), e-mail i ime (dolaze iz Googlea, mogu se mijenjati), uloga i sposobnosti (pod kontrolom platforme), status (aktivan/suspendiran/na čekanju) te metapodatke o afilijaciji i istraživačkim interesima. Ugniježđivanje metapodataka izravno u dokument korisnika, umjesto normalizacije u zasebnu tablicu, odražava najbolje prakse MongoDB-a — podaci koji se uvijek dohvaćaju zajedno trebaju biti pohranjeni zajedno.

Odluka da se sposobnosti pohranjuju kao niz u dokumentu korisnika zaslužuje posebno obrazloženje jer je u suprotnosti s relacijskom normalizacijom. U trećoj normalnoj formi, odnos više-prema-više (korisnici imaju više sposobnosti, sposobnosti pripadaju više korisnika) rješava se međutablicom. Takav normalizirani pristup optimizira integritet podataka — dodavanje novog tipa sposobnosti ne zahtijeva izmjenu postojećih zapisa korisnika. Međutim, on kažnjava uobičajeni slučaj provjere dozvola tijekom obrade zahtjeva.

Svaki autentificirani zahtjev provjerava dozvole korisnika. U modelu s nizom, ta se provjera odvija u potpunosti u memoriji nakon što se dohvatí dokument korisnika — bez dodatnih upita. U modelu s međutablicom, svaka provjera dozvola zahtijeva JOIN (skup) ili zaseban upit prema međutablici (mrežno opterećenje). Kod stotina zahtjeva u sekundi, ta se dodatna opterećenja brzo akumuliraju.

Kompromis je taj da dodavanje novih tipova sposobnosti zahtijeva ažuriranje svih korisničkih dokumenata kojima ih želimo dodijeliti. To je prihvatljivo jer se tipovi sposobnosti dodaju rijetko (možda nekoliko puta godišnje), dok se provjere dozvola događaju milijune puta dnevno. MongoDB optimizira uobičajeni slučaj na račun rijetkog.

Kolekcija capability_requests održava povijesne zapise svih zahtjeva za dozvolama — i za početne osnovne uloge i za dodatne sposobnosti. Ti podaci ne pripadaju u kolekciju users jer jedan korisnik može tijekom vremena imati mnogo zahtjeva, a zahtjevi sadrže bilješke recenzenata i vremenske oznake nebitne za autentikaciju. Razdvajanje odgovornosti održava kolekciju users brzom za "vrući" put (autentikacija), dok kolekcija zahtjeva podržava sporiji put (administracija i audit).

Kolekcija api_keys postoji zasebno jer korisnici mogu imati više API ključeva, svaki s neovisnim istekom, ograničenjima brzine i praćenjem upotrebe. Polje last_used_at korisnicima omogućuje da identificiraju i izbrišu napuštene ključeve. Polje key_prefix omogućuje brze upite bez full-text pretraživanja. Pohranjivanje samo SHA-256 hasha ključa štiti od kompromitacije baze. Nijedan od tih podataka nije potreban tijekom web autentikacije, pa bi "zagađivanje" kolekcije users tim informacijama usporilo uobičajeni slučaj kako bi poslužilo specijalizirani.

Kolekcija audit_logs neprestano raste i ima drugačije obrasce pristupa od ostalih kolekcija. Logovi su samo-dodavani (append-only, bez ažuriranja nakon upisa), dohvaćaju se po vremenskom rasponu, filtriraju prema korisniku ili tipu akcije ili komponenti i zadržavaju dulje od drugih podataka. Te karakteristike opravdavaju zasebnu kolekciju sa specijaliziranim indeksima. Kako kolekcija raste, može se particionirati po datumu

ili premjestiti u arhivsku pohranu bez utjecaja na operativne podatke. Miješanje audit logova s korisnicima s vremenom bi degradiralo performanse dohvaćanja korisnika kako se logovi gomilaju.

Ovaj dizajn s četiri kolekcije balansira normalizaciju (ne dupliciraj podatke nepotrebno) i denormalizaciju (dupliciraj podatke kada to poboljšava performanse). Dokument korisnika denormalizira se ugniježđivanjem metapodataka i sposobnosti jer su ti podaci nužni za provjere dozvola. Zahtjevi za sposobnostima i API ključevi normalizirani su jer se upituju zasebno i imaju različite životne cikluse. Audit logovi su odvojeni jer su im obrasci pristupa temeljno drugačiji. Te odluke proizlaze iz razumijevanja opterećenja, a ne iz rigidne primjene pravila normalizacije.

7 Sigurnosna arhitektura

7.1 Višeslojna zaštita kroz više metoda autentikacije

Dizajnerska odluka: Podržati tri metode autentikacije — OAuth, JWT, API ključevi

Platforma dopušta autentikaciju putem Google OAuth-a, JWT tokena ili API ključeva, umjesto forsiranja jedne metode.

Podrška za više metoda autentikacije odražava spoznaju da različiti slučajevi uporabe imaju različite zahtjeve i profile rizika. Korisnici u web pregledniku autentificiraju se putem OAuth-a jer pruža najbolje korisničko iskustvo — nema lozinke kojom treba upravljati, sigurnost je delegirana Googleu, a 2FA se automatski obrađuje. Nakon autentikacije dobivaju JWT tokene za naknadne API pozive jer tokeni omogućuju bezdržavnu autentikaciju na većem razmjeru. Korisnici skripti i automatizacije dobivaju API ključeve jer OAuth zahtijeva interakciju u pregledniku koja ne radi u "headless" okruženju.

Ova raznolikost pruža višeslojnu zaštitu. Ako je OAuth privremeno nedostupan zbog Googleovog ispada, korisnici s API ključevima i dalje mogu programski pristupati platformi. Ako je osvježavajući JWT token korisnika ukraden, njegovo opozivanje ne poništava zasebno izdane API ključeve (pod pretpostavkom da oni također nisu kompromitirani). Ako se otkrije ranjivost u JWT biblioteci, API ključevi služe kao rezervni mehanizam dok se zakrpe ne primijene. Ne postoji jedna jedina točka kvara jer platformu opslužuju više neovisnih autentikacijskih putanja.

Svaka metoda ima ograničenja koja druge nadoknađuju. OAuth ovisi o dostupnosti i politikama Googlea. JWT-ovi se ne mogu trenutačno opozvati prije isteka. API ključevi zahtijevaju sigurno skladištenje kojim mnogi korisnici slabo upravljaju. Kombiniranje metoda omogućuje korisnicima da odaberu pristup koji odgovara njihovom sigurnosnom profilu i slučaju uporabe, dok platformi pruža rezervne opcije.

Implementacija dijeli logiku autorizacije kroz advandeb-shared-utils. Bilo da se korisnik autentificira OAuth+JWT putem preglednika ili API ključem putem skripte, iste funkcije provjere dozvola određuju njegov pristup. Ta konzistentnost sprječava sigurnosne rupe u kojima jedan put slučajno zaobilazi provjere koje drugi put provodi. Metoda autentikacije bilježi se u audit logovima, što omogućuje analizu imaju li različite metode različite profile sigurnosnih incidenata.

7.2 Ograničavanje brzine radi sprječavanja zlouporabe

Dizajnerska odluka: Ograničenja brzine po korisniku temeljena na sposobnostima, implementirana u Redis-u

Zahtjevi se ograničavaju ovisno o tome tko ih šalje i koje sposobnosti ima, uz praćenje u Redis-u.

Ograničavanje brzine služi višestrukim svrhama koje opravdavaju njegovu implementacijsku složenost. Prvo, sprječava nenamjerne denial-of-service situacije zbog pogrešnih skripti. Python skripta istraživača s beskonačnom petljom ne bi smjela iscrpiti konekcije prema bazi ili CPU vrijeme. Drugo, ograničava namjernu zlouporabu od strane zlonamjernih korisnika koji pokušavaju masovno čupati podatke ili preopteretiti infrastrukturu. Treće, omogućuje pravednu raspodjelu resursa — korisnici s višim sposobnostima (i većim legitimnim potrebama) dobivaju više limite, dok se istodobno sprječava da jedan korisnik monopolizira resurse.

Ograničenja temeljena na sposobnostima odražavaju realnost da različiti tipovi korisnika imaju legitimno različite potrebe. Knowledge Explorator koji pregledava platformu može generirati 50 zahtjeva u minuti klikajući po činjenicama i radovima. Knowledge Curator koji aktivno doprinosi može generirati 100 zahtjeva u minuti dok učitava dokumente i stvara zapise. Kurator s Analytics Access sposobnošću koji pokreće skripte za izvoz podataka legitimno može trebati 500 zahtjeva u minuti. Za našu beta bazu korisnika (10–20 korisnika inicijalno) ta su ograničenja velikodušna i vjerojatno neće biti dosegnuta tijekom normalnog rada. Primarno služe kao sigurnosne ograde protiv beskonačnih petlji ili bugovitih automatizacija, a ne kao obrana od sofisticiranih napada.

Redis implementira ograničavanje brzine jer se provjera mora izvršiti pri svakom zahtjevu uz minimalnu latenciju. Pohranjivanje brojača ograničenja brzine u MongoDB-u dodalo bi opterećenje i latenciju na bazu, čime bi se poništila svrha. Redis čuva brojače u memoriji, s vremenima pristupa u mikrosekundama, i podržava atomske operacije uvećanja koje sprječavaju utrke u distribuiranim sustavima. Ugrađena TTL (time-to-live) funkcionalnost automatski uklanja zastarjele brojače, učinkovito implementirajući klizne prozore ograničenja.

Konkretni algoritam je token bucket, pri čemu svaki korisnik ima "kantu" s tokenima, a svaki zahtjev troši jedan token. Tokeni se nadopunjuju konstantnom brzinom (prema njihovom limitu). To omogućuje kratke izboje iznad prosječne stope, ali sprječava trajno visoke stope zahtjeva. Korisnik s limitom od 200 zahtjeva u minuti može u jednoj minuti poslati 300 zahtjeva ako je u prethodnim minutama slao malo zahtjeva, ali ne može trajno održavati 300 zahtjeva u minuti. Ova fleksibilnost odgovara realnim obrascima korištenja (razdoblja intenzivne aktivnosti nakon kojih slijede mirnija razdoblja) i istodobno sprječava zlouporabu.

Alternativni pristupi poput ograničavanja u bazi (presporo), ograničavanja na razini proxyja (nema uvid u korisničke sposobnosti) ili potpunog izostanka ograničenja (poziva na zlouporabu) odbačeni su jer ne uspijevaju uravnotežiti konkurentne ciljeve performansi, pravednosti i fleksibilnosti.

7.3 Sveobuhvatno zapisivanje audita

Dizajnerska odluka: Logirati sve upise, događaje autentikacije i promjene dozvola

Platforma zapisuje svaku operaciju koja mijenja stanje ili sigurnosni profil sustava, pohranjujući logove u kolekciju audit_logs.

Audit logiranje služi potrebama znanstvenog integriteta, sigurnosnog nadzora, regulatorne usklađenosti i korisničke podrške, koje zajedno opravdavaju njegov operativni trošak. Za znanstveni integritet, svaki doprinos znanju mora se moći pripisati svom autoru. Ako se u bazi pojavi sporna činjenica, audit logovi pokazuju tko ju je stvorio, kada i s koje IP adrese. Ta sljedivost gradi povjerenje — korisnici koji u dobroj vjeri doprinose znanju nisu kompromitirani djelovanjem loših aktera jer su doprinosi individualno pratljivi.

Sigurnosni nadzor zahtijeva logiranje radi detekcije kompromitiranih računa s neuobičajenim ponašanjem. Kurator koji inače kreira 10 činjenica dnevno, a odjednom stvara 1000 na sat, sugerira kompromitaciju računa. Korisnik koji pristupa platformi iz nove države zaslužuje dodatnu provjeru. Takvi obrasci postaju vidljivi tek analizom ponašanja kroz vrijeme. Nadzor u stvarnom vremenu može aktivirati alarme za administratore, dok povijesni logovi omogućuju forenzičku analizu nakon otkrivanja incidenta.

Zahtjevi usklađenosti koje postavljaju financijeri, etička povjerenstva i regulativa zaštite podataka često nalagu zapisivanje audita. GDPR zahtijeva praćenje tko je pristupio osobnim podacima i kada. Financijeri projekata žele dokaz da su istraživački podaci obrađivani prema protokolima. Buduće certifikacije (HIPAA ako se dodaju zdravstveni podaci, FedRAMP za državnu upotrebu) zahtijevat će sveobuhvatno logiranje. Izgradnja audit mogućnosti od samog početka daleko je lakša od naknadnog dodavanja kada certifikacija postane nužna.

Korisnička podrška profitira od audit logova pri odgovaranju na pitanja tipa "što se dogodilo s mojim podacima?". Korisnik prijavi da je njegov nacrt činjenice nestao — logovi pokazuju da ju je jučer slučajno izbrisao. Korisnik ne može pristupiti značajci za koju tvrdi da mu je bila dostupna — logovi pokazuju da su mu sposobnosti ukinute prošli tjedan uz dokumentirano obrazloženje. Ti se svakodnevni slučajevi podrške brzo razrješavaju uz dobre logove, a bez njih se pretvaraju u frustrirajuće misterije.

Odluka da se logiraju upisi, ali ne i čitanja, rezultat je analize koristi i troška. Operacije pisanja relativno su rijetke i visokog utjecaja. Pogrešno stvorena činjenica može dovesti istraživače u zabludu. Nepažljivo generiran API ključ može omogućiti izvlačenje podataka. Te operacije opravdavaju trošak pohrane i performansi logiranja. Operacije čitanja mnogo su učestalije (možda i 100 puta češće od pisanja) i manjeg pojedinačnog utjecaja. Logiranje svakog čitanja generiralo bi golemi volumen logova s ograničenom sigurnosnom vrijednošću — čitanje javnih podataka tipično nije sigurnosni događaj. Za naše beta okruženje s 10–20 korisnika, volumen logova ostaje upravljiv čak i uz sveobuhvatno logiranje upisa, što omogućuje detaljnu forenzičku analizu bez zabrinutosti za pohranu.

Iznimka kod ne-logiranja čitanja su neuspjeli pokušaji autorizacije. Ako korisnik pokuša pristupiti nečemu za što nema dozvolu, taj se pokušaj logira. Time se otkrivaju situacije konfuzije oko dozvola (korisnik očekuje pristup koji nema, što sugerira UX problem) i potencijalni napadači koji sondiraju ranjivosti. Uspješna čitanja javnih podataka ne logiraju se, ali neovlašteni pokušaji da im se pristupi logiraju se, pružajući vidljivost sigurnosti bez utapanja u buci.

Logovi su nepromjenjivi nakon upisa — zapisi se ne mogu ažurirati niti brisati. To

sprječava manipulaciju dokazima kada kompromitirani račun pokuša obrisati tragove zlonamjernog ponašanja. Nepromjenjivost logova zahtijeva pažljiv dizajn kako bi se izbjeglo logiranje osjetljivih podataka (nikada ne logirati lozinke, čak ni hashirane; nikada ne logirati pune API ključeve, već samo prefikse). Ako se osjetljivi podaci ipak slučajno zalogiraju, čitav log zapis treba rotirati iz dostupnog prostora umjesto editiranja.

8 Skalabilnost i performanse

8.1 Bezdržavna arhitektura za horizontalno skaliranje

Dizajnerska odluka: Pohranjivati stanje sesije u JWT tokenima, ne na poslužiteljima

Stanje autentikacije živi u kriptografski potpisanim tokenima, a ne u pohranama sesija na poslužitelju.

Bezdržavna arhitektura proizašla je iz spoznaje da pohrana sesija postaje usko grlo tradicionalnih web aplikacija na većem razmjeru. Svaki zahtjev u sustavu sa sesijama upućuje upit pohrani sesija (baza ili Redis) kako bi dohvatila identitet i dozvole korisnika. Na 1000 zahtjeva u sekundi, to znači 1000 dohvaćanja sesije u sekundi prije bilo kakve poslovne logike. Taj strop propusnosti pojavljuje se bez obzira na to koliko aplikacijskih poslužitelja dodamo, jer se svi natječu za pristup istoj pohrani sesija.

Bezdržavna JWT autentikacija uklanja to usko grlo tako da kodira stanje sesije u sam token. Dokument korisnika dohvaća se jednom tijekom autentikacije, a njegova ključna polja (ID korisnika, uloga, sposobnosti) enkodiraju se u JWT potpisan tajnim ključem. Naknadni zahtjevi šalju taj token, a bilo koji aplikacijski poslužitelj može ga validirati koristeći tajni ključ, bez upita prema bazi. Validacija zahtijeva samo CPU vrijeme (provjera kriptografskog potpisa), kojeg obično imamo više nego I/O vremena.

Ova arhitektura omogućuje horizontalno skaliranje kako broj korisnika raste. Za početno beta okruženje (10–20 korisnika) dovoljan je jedan aplikacijski poslužitelj. Kako se približavamo 100 korisnika, dodatni se poslužitelji mogu trivijalno dodavati bez brige o afinitetu sesija. Load balancer može nasumično raspoređivati zahtjeve jer nijedan poslužitelj ne održava stanje sesije. Ovo dizajnersko načelo — bezdržavno od početka — znači da za skaliranje ne trebamo arhitektonske promjene, već samo dodatne resurse.

Bezdržavni model mijenja mogućnost trenutačnog opoziva za skalabilnost. Tradicionalnu sesiju moguće je obrisati iz pohrane sesija, čime se pristup trenutačno prekida. JWT ostaje valjan do isteka, čak i ako je korisnik suspendiran. To se ublažava kratkim vremenom trajanja tokena (1 sat za pristupne tokene) i opozivim osvježavajućim tokenima. Suspendiranje korisnika blokira osvježavanje, pa on gubi pristup unutar najviše jednog sata. Za hitne slučajeve administratori mogu dodati JTI (jedinstveni identifikator) tokena na listu opoziva, iako to ponovno uvodi potrebu za jednim upitom prema bazi.

Pristup pretpostavlja da se dozvole korisnika mijenjaju rijetko u odnosu na učestalost zahtjeva. Korisnik koji šalje 100 zahtjeva u minuti, a kojem se dozvole mijenjaju jednom dnevno, postiže golemu korist od bezdržavne provjere. Da se dozvole mijenjaju svake minute, cacheiranje inherentno u JWT tokenima postalo bi problematično. Ta je pretpostavka u našem slučaju realna — promjene uloga su administrativne akcije koje se događaju povremeno, a ne kontinuirano.

8.2 Trajanje tokena: balans između sigurnosti i praktičnosti

Dizajnerska odluka: Pristupni tokeni od 1 sata, osvježavajući tokeni od 30 dana

Pristupni tokeni brzo istječu, dok osvježavajući traju dulje, balansirajući sigurnosni rizik s praktičnošću za korisnike.

Pristup s dvama tipovima tokena različitog trajanja odražava temeljnu napetost u dizajnu autentikacijskih sustava. Korisnici žele trajne sesije — jednom se prijaviti i ostati prijavljeni neograničeno dugo. Sigurnost traži kratkotrajne vjerodajnice — ukradeni token treba vrlo brzo prestati vrijediti. Dvo-token pristup pomiruje te zahtjeve razdvajanjem odgovornosti: pristupni tokeni služe autorizaciji (kratko traju, često se koriste), a osvježavajući tokeni služe ponovnoj autentikaciji (dulje traju, rijetko se koriste).

Pristupni tokeni autoriziraju pojedinačne zahtjeve. Uključeni su u svaki API poziv, vidljivi u logovima zahtjeva i potencijalno izloženi JavaScriptu u pregledniku. Ta izloženost opravdava kratko vrijeme trajanja. Ako se pristupni token ukrade putem XSS napada ili kompromitiranog log filea, napadač dobiva pristup samo do isteka — jedan sat. To ograničava štetu u odnosu na tokene koji vrijede neograničeno. Konkretnije, trajanje od jednog sata balansira tipično trajanje sesije (korisnici koji aktivno rade rijetko imaju više od sata pauze između zahtjeva) s prozorom sigurnosne izloženosti.

Osvježavajući tokeni omogućuju dobivanje novih pristupnih tokena bez ponovne autentikacije. Pohranjeni su u HTTP-only cookiejima (nedostupni JavaScriptu, čime se smanjuje XSS rizik) i koriste se samo kada pristupni tokeni isteknu. Korisnik koji u satu pošalje 100 zahtjeva koristi svoj pristupni token 100 puta, a osvježavajući token samo jednom. Ta smanjena izloženost opravdava dulje vrijeme trajanja — 30 dana znači da korisnici ostaju prijavljeni oko mjesec dana, što odgovara uobičajenoj funkciji "zapamti me".

Vrijeme trajanja od 30 dana za osvježavajuće tokene proizašlo je iz analize ponašanja korisnika. Istraživači koji svakodnevno rade koriste platformu kontinuirano i trebaju ostati prijavljeni. Istraživači koji odu na godišnji odmor ili rade na drugim projektima možda neko vrijeme ne koriste platformu. Automatsko odjavljivanje jednom mjesečno balansira praktičnost (ne prekidati aktivne korisnike) sa sigurnošću (ne ostavljati napuštene račune trajno prijavljenima). Korisnici kojima trebaju kraće sesije iz sigurnosnih razloga mogu se ručno odjaviti.

Osvježavajući tokeni ostaju opozivi unatoč duljem trajanju. Pohranjeni su u bazi, pa se suspendiranjem korisnika njegovi osvježavajući tokeni brišu. Sljedeći pokušaj osvježavanja pristupnog tokena tada ne uspijeva i korisnik se mora ponovno autentificirati. To administratorima daje mogućnost trenutačne odjave u slučaju sigurnosnih incidenata, a da se ne odustaje od performansnih prednosti bezdržavnih pristupnih tokena.

Dvo-token obrazac slijedi industrijski standard uspostavljen OAuth 2.0 specifikacijom i prihvaćen na velikim platformama. GitHub, Google i AWS koriste slične obrasce. Ovo nije inovacija, nego primjena dobro shvaćenih rješenja na standardne probleme. Konkretne vrijednosti trajanja (1 sat i 30 dana) parametri su koje možemo prilagoditi na temelju opaženih obrazaca korištenja i analize sigurnosnih incidenata.

9 Deployment i operacije

9.1 Strategija s tri okruženja

Dizajnerska odluka: Odvojena razvojna, staging i produkcijska okruženja

Platforma se isporučuje u tri različita okruženja s različitim konfiguracijama, umjesto u jedno okruženje ili ad-hoc deploy.

Strategija s tri okruženja odražava lekcije naučene kroz desetljeća iskustva s isporukom softvera. Razvojna okruženja rade na računalima developera s minimalnom infrastrukturom — lokalni MongoDB, mock email, bez ograničavanja brzine. Ta konfiguracija optimizira brzu iteraciju. Developeri mogu testirati promjene u sekundama, debugirati s punim pristupom i eksperimentirati s rizičnim promjenama koje bi u produkciji bile neprihvatljive. Izolacija okruženja znači da pogreške developera utječu samo na njihovo vlastito računalo.

Staging okruženja što je moguće vjernije repliciraju produkcijsku konfiguraciju. Koriste hostani MongoDB, stvarne servise za e-poštu, produkcijske limite brzine i stvarni Google OAuth (konfiguriran za testnu domenu). Ovo okruženje hvata integracijske probleme koje lokalni razvoj propušta — mrežnu latenciju, timeoutove vanjskih servisa, pogreške u konfiguraciji, ponašanje pod opterećenjem. Prihvatno testiranje od strane korisnika odvija se u stagingu, omogućujući dionicima da pregledaju značajke prije produkcijskog deploya. Načelo "testiraj u uvjetima sličnim produkciji" postiže se bez rizika za stvarne korisnike.

Produkacijska okruženja dodaju redundanciju i nadzor iznad onoga što postoji u stagingu. MongoDB radi kao replica set za visoku dostupnost — pri padu primarnog čvora jedna se sekundarna instanca automatski promovira. Redis radi kao klaster kako bi ograničavanje brzine ostalo dostupno. Više backend poslužitelja stoji iza load balancera. SSL certifikati omogućuju HTTPS. Nadzorne ploče prate stope pogrešaka, vremena odziva i upotrebu resursa. Upozorenja obavještavaju operatere o anomalijama. Automatizirane sigurnosne kopije kontinuirano se izvršavaju, uz redovito testirane procedure povrata.

Ovakav slijed okruženja pruža zaštitne mreže. Bug koji prođe jedinične testove može biti uhvaćen u integracijskom testiranju u razvoju. Bug koji prođe razvoj može biti otkriven u stagingu tijekom prihvatnog testiranja. Bug koji prođe staging i dalje može biti detektiran nadzorom produkcije prije nego što utječe na velik broj korisnika. Nijedna pojedina provjera ne sprječava sve probleme, ali slojevi provjera dramatično smanjuju vjerojatnost kvara.

Alternativa s direktnim deployem u produkciju ili s dvama okruženjima (dev/prod bez staginga) povećava rizik na neprihvatljivu razinu. Razvojna okruženja previše se razlikuju od produkcije (localhost nasuprot hostanom, mock servisi nasuprot stvarnim) da bi uhvatila integracijske probleme. Produkcijski deploy bez staging pregleda stvara scenarije "sve ili ništa" u kojima bugovi pogađaju korisnike prije nego što ih se otkrije. Staging okruženje konkretno postoji kako bi premostilo jaz između razvojne idealizacije i produkcijske realnosti.

Upravljanje konfiguracijom preko okruženja koristi varijable okoline i konfiguracijske datoteke, nikada hardkodirane vrijednosti. Isti artefakt koda (Docker container) isporučuje se u sva tri okruženja s različitim konfiguracijama. To osigurava da se u stagingu testira upravo onaj kod koji će ići u produkciju, a ne neka slična, ali različita verzija. Prakse "configuration-as-code" (Terraform za infrastrukturu, Ansible za konfiguraciju) omogućuju ponovljivo stvaranje okruženja i oporavak od katastrofe.

10 Arhitektura Knowledge Buildera

Knowledge Builder je polu-nadzirani sustav za izgradnju baze znanja platforme kroz automatizirane i ručne procese. U svojoj srži omogućuje interakciju između kuratora, dokumenata i postojećeg znanja putem chatbota. Umjesto da zahtijeva isključivo ručno izdvajanje ili se oslanja na potpuno "black-box" automatizaciju, sustav kombinira AI pomoć s ljudskim nadzorom: kuratori mogu "razgovarati" s dokumentima kroz sučelje chatbota koje predlaže činjenice za izdvajanje, referencira postojeće znanje radi održavanja konzistentnosti i omogućuje ručno dorađivanje prije objave. Ovaj hibridni pristup tretira izgradnju znanja kao interaktivan dijalog — kurator pita "Što ovaj rad kaže o metaboličkom skaliranju?", a sustav izdvaja kandidatske činjenice i pokazuje kako se one odnose na postojeće znanje. Kurator ih zatim pregledava, dorađuje i odobrava, osiguravajući kvalitetu, a istodobno iskorištava prednosti automatizacije u pogledu brzine. Rezultat je nadzirani automatizirani tijek rada u kojem ljudi usmjeravaju i validiraju, dok AI rukuje izdvajanje i inicijalno strukturiranje.

10.1 Reprezentacija znanja: činjenice, stilizirane činjenice i grafovi

Dizajnerska odluka: Trodijelni model reprezentacije znanja

Znanje se predstavlja na tri razine apstrakcije: činjenice (siroka opažanja), stilizirane činjenice (sintetizirani obrasci) i grafovi znanja (strukturirani odnosi).

Trodijelni model odražava način na koji se znanstveno znanje prirodno razvija od specifičnih opažanja prema općim principima. Činjenice predstavljaju atomska opažanja izdvojena iz literature ili podataka — "Vrsta X alocira 40% energije u reprodukciju pri temperaturi T" ili "Organizam Y pokazuje maksimalnu brzinu ingestije R pri tjelesnoj masi M". Te su činjenice brojne, specifične i izravno vezane uz izvorne materijale radi sljedivosti.

Stilizirane činjenice agregiraju više povezanih činjenica u iskaze više razine — "Vrsta X pokazuje temperaturno ovisne obrasce alokacije energije" ili "Filogenetska skupina Y pokazuje alometrijsko skaliranje metaboličke stope s eksponentom 0,75". Jedna stilizirana činjenica može sintetizirati dokaze iz desetaka činjenica, pružajući kompresiju i jasnoću uz zadržavanje sljedivosti putem referenci na fact_ids. Ovaj srednji sloj služi potrebama modeliranja: modelarima trebaju obrasci i generalizacije, a ne sirova opažanja, ali istodobno moraju imati mogućnost uvida u dokaze kada se pretpostavke dovedu u pitanje.

Grafovi znanja pružaju strukturalnu reprezentaciju entiteta i odnosa. Dok činjenice i stilizirane činjenice koriste prirodni jezik (fleksibilan, ali dvosmislen), grafovi koriste formalne strukture čvorova i bridova (rigidne, ali lako upitljive). Ista biološka spoznaja može postojati kao činjenica ("Rad izvještava da se vrsta A hrani vrstom B"), stilizirana činjenica ("Odnos predator–plijen između A i B pokazuje funkcijski odgovor tipa II") i brid u grafu (A –[preys_on]–> B). Ta je redundancija namjerna — različiti slučajevi uporabe preferiraju različite reprezentacije.

Tipizirani odnosi između entiteta znanja: Arhitektura grafa znanja podržava bogate tipove odnosa između činjenica, stiliziranih činjenica i drugih entiteta znanja, omogućujući eksplicitnu reprezentaciju epistemoloških odnosa:

- **Potporni dokaz** (supports): činjenica B pruža dokaz koji podupire činjenicu A ili

stiliziranu činjenicu X. Primjer: više empirijskih opažanja skaliranja metaboličke stope podupire stiliziranu činjenicu o alometrijskim odnosima.

- **Kontradiktorni dokaz** (contradicts/opposes): činjenica C proturječi činjenici D ili dovodi u pitanje pretpostavke stilizirane činjenice Y. Primjer: opažanja tropskih vrsta koja proturječe obrascima skaliranja metaboličke stope zabilježenima u umjerenim područjima.
- **Referencija/citiranje** (references/cites): činjenica ili stilizirana činjenica referencira drugu kao kontekst ili prethodni rad. Primjer: novo opažanje koje se poziva na već uspostavljene bazne mjere.
- **Profinjavanje** (refines): stilizirana činjenica Y profinjuje ili daje specifičniju verziju stilizirane činjenice X. Primjer: temperaturno ovisna alokacija energije koja profinjuje opće obrasce alokacije energije.
- **Sinteza** (synthesizes): stilizirana činjenica sintetizira više činjenica (već uhvaćeno preko fact_ids niza, ali može biti eksplicitan brid za potrebe graf upita).
- **Metodološki kontekst** (measured_by/estimated_by): povezuje činjenice s metodama mjerenja ili tehnikama procjene, omogućujući upite temeljene na metodologiji.
- **Vremenski odnosi** (precedes/follows): vremenski poredani odnosi između opažanja ili razvojnih faza.
- **Filogenetski/taksonomski** (taxonomically_related): odnosi temeljeni na srodnosti vrsta, što omogućuje komparativnu analizu između kladova.

Ovi tipizirani odnosi omogućuju sofisticirane upite: "pronađi sve činjenice koje podupiru ovu hipotezu o alokaciji energije", "identificiraj kontradiktorne dokaze za metaboličko skaliranje", "prati lanac profinjavanja od sirovih opažanja do općih principa" ili "usporedi snagu dokaza između filogenetskih skupina". Tipovi odnosa hvataju ne samo da su entiteti povezani, već i *kako* i *zašto* su povezani, što je ključno za sintezu i validaciju znanstvenog znanja.

Metapodaci o odnosima uključuju ocjene pouzdanosti, anotacije kuratora i vremenske oznake, što istraživačima omogućuje procjenu kvalitete dokaza i praćenje kako se znanstveni konsenzus razvija. Potporni odnos može imati visoku pouzdanost (0,9) temeljenu na višestrukim neovisnim replikacijama, dok kontradiktorni odnos može imati nižu pouzdanost (0,6) ako se temelji na jednoj studiji koja zahtijeva daljnje istraživanje.

Alternativni pristupi razmatrani su tijekom dizajna. Čisto grafovski model (sve su čvorovi i bridovi) prisilio bi neprirodna uklapanja narativnog znanja. Čisto tekstualni model (sve su dokumenti) žrtvovao bi mogućnost strukturiranih upita. Jedna jedinstvena reprezentacija pokušava zadovoljiti sve potrebe, a ne zadovoljava nijednu. Trodijelni pristup prihvaća trošak redundancije radi fleksibilnosti, što je primjereno beta platformi u kojoj uzorci reprezentacije znanja još nastaju.

Polje status na svakom entitetu znanja (draft, pending_review, published, rejected) omogućuje tijekove kontrole kvalitete bez kompleksnih dozvola. Privatni nacrti omogućuju eksperimentiranje. Objavljeno znanje je globalno čitljivo. Taj jednostavan životni ciklus odgovara istraživačkim tijekovima rada, u kojima ideje neko vrijeme sazrijevaju privatno prije objave.

10.2 Cjevovod za unos dokumenata

Dizajnerska odluka: Asinkroni višestupanjski unos s AI-potpomognutim izdvajanjem

Učitavanje dokumenata pokreće pozadinske cjevovode obrade koji izdvajaju tekst, pozivaju AI agente za izdvajanje činjenica i asinkrono pohranjuju rezultate.

Asinkroni cjevovod odražava realnost da je obrada dokumenata spora (minute, a ne milisekunde) i sklona greškama (PDF-ovi su neuredni, AI izdvajanje nije savršeno). Sinkrona obrada bi blokirala korisnike — učitavanje 50 radova zaključalo bi sučelje na sat vremena. Pozadinska obrada sa status pollingom pruža responzivno korisničko iskustvo, a pritom izbjegava složenost punokrvnih distribuiranih sustava.

Višestupanjski pristup (upload → izdvajanje teksta → AI analiza → pohrana činjenica) omogućuje neovisno rukovanje pogreškama. Izdvajanje teksta može uspjeti dok AI analiza ne uspije zbog iscrpljenih API kvota. Sustav pohranjuje parcijalne rezultate i omogućuje ponovni pokušaj neuspjelih faza bez ponovne obrade uspješnih. Svaka faza zapisuje napredak u dokument IngestionJob, što korisnicima pruža vidljivost, a administratorima informacije za debugiranje.

AI-potpomognuto izdvajanje putem agenata predstavlja svjesnu odluku da se prioritet da automatizaciji nad savršenom točnošću. Ručno izdvajanje činjenica je sporo, ali precizno; AI izdvajanje je brzo, ali sklono pogreškama. Platforma podržava oba pristupa: korisnici sa sposobnošću agent_access mogu pokretati masovno izdvajanje za inicijalno popunjavanje baze znanja, dok kuratorski tijek rada omogućuje ručno kreiranje i uređivanje činjenica. Sustav recenzije hvata AI pogreške prije objave, pružajući gateove kvalitete bez odricanja od koristi automatizacije.

Apstrakcija IngestionBatch omogućuje masovne operacije uobičajene u istraživanju: "učitaj sve radove s ove konferencije" ili "unesi korpus za pregled literature". Batch objekti grupiraju povezane dokumente, omogućuju praćenje statusa na razini batcha i podržavaju "Day Zero" inicijalno punjenje znanja (sadržaji koje administrator označi kao temeljne, is_day_zero=true). Taj batch model bolje odgovara istraživačkim tijekovima rada od strogo dokument-centričnih modela.

Odluke o pohrani datoteka uključuju kompromise. Pohranjivanje sadržaja dokumenata u MongoDB (polje content u kolekciji Document) pojednostavljuje sigurnosne kopije i deploy, ali ograničava veličinu dokumenata i neefikasno troši bazni storage. Pohranjivanje datoteka u object storage (S3, MinIO) poboljšava skalabilnost, ali dodaje operativnu složenost i nove načine kvara u distribuiranom sustavu. Za beta razmjer (tisuće dokumenata, ukupno u razini gigabajta) pohrana u MongoDB je jednostavnija. Migracija na object storage postaje nužna tek kod desetaka tisuća dokumenata.

10.3 Arhitektura agentnog okvira

Dizajnerska odluka: Agentni okvir inspiriran LangChainom s RAG-om i lokalnom LLM integracijom putem Ollame

AI mogućnosti pruža agentni okvir koji orkestrira LLM pozive, Retrieval-Augmented Generation (RAG), pozive alata i parsiranje strukturiranog izlaza.

Agentni okvir apstrahira interakcije s LLM-ovima iza konzistentnog sučelja: prihvati

ulaz na prirodnom jeziku, dohvatí relevantan kontekst znanja, pozovi potrebne alate, vrati strukturirani izlaz. Ta apstrakcija omogućuje više tipova agenata (izdvajanje znanja, pomoć pri upitu, prijedlozi modela) da dijele infrastrukturu, a razlikuju se ponašanjem. Klasa AgentFramework pruža orkestraciju, LocalModelClient rukuje komunikacijom s Ollamom, a ToolRegistry omogućuje dinamičko proširenje sposobnosti.

Retrieval-Augmented Generation (RAG): Okvir implementira RAG metodologiju kako bi LLM odgovore uzemljio u znanju platforme. Pri obradi upita sustav dohvaća relevantne činjenice, stilizirane činjenice i entitete grafa znanja iz baze, te ih uključuje u kontekst LLM-a. Ovaj pristup smanjuje halucinacije jer model generira odgovore na temelju stvarnog kuriranog znanja, a ne na temelju vlastite parametarske memorije. Pri unosu dokumenata, RAG omogućuje agentima da se pri izdvajanju novih činjenica referenciraju na postojeće znanje, čime se osigurava konzistentnost i otkrivaju potencijalne kontradikcije. Kod upita na znanje, RAG omogućuje da se pitanja prirodnim jezikom ("Koji je odnos metaboličkog skaliranja za ribe?") odgovore konkretnim sadržajem iz platforme, a ne generičkim znanjem LLM-a.

RAG cjevovod sastoji se od: (1) analize upita — izdvajanje ključnih pojmova i entiteta iz korisničkog ulaza, (2) dohvaćanja znanja — pretraživanje baze znanja korištenjem tekstualne pretrage i buduće vektorske sličnosti za relevantan sadržaj, (3) sastavljanja konteksta — formatiranje dohvaćenih činjenica i stiliziranih činjenica kao konteksta za LLM, (4) generiranja odgovora — LLM generira odgovor uzemljen u dohvaćenom znanju, (5) citiranja izvora — odgovori uključuju reference na konkretne entitete baze znanja radi sljedivosti. Takva arhitektura osigurava znanstvenu odgovornost — svaka AI-generirana tvrdnja može se povezati s kuriranim izvorima.

Integracija lokalnih LLM-ova putem Ollame odražava razmatranja troška i privatnosti. Cloud LLM API-ji (OpenAI, Anthropic) nude veću kvalitetu, ali uvode trošak po tokenu koji skalira s upotrebom — problematično u istraživanjima gdje su budžeti tokena nepredvidljivi. Ollama pokreće modele lokalno (Llama 2, Mistral, i dr.) s nultim graničnim troškom nakon što je hardver osiguran. Za beta razmjer dovoljan je jedan GPU poslužitelj s Ollamom. Cloud API-ji mogu nadopuniti lokalne modele za slučajeve visoke vrijednosti (npr. izdvajanje za konačne publikacije), dok lokalni modeli pokrivaju istraživački rad.

Sustav alata omogućuje agentima da rade više od samog generiranja teksta: pretražuju postojeće znanje, upućuju upite prema bazama, pozivaju vanjske API-je. Svaki je alat Python funkcija registrirana u ToolRegistry s JSON shemom koja opisuje parametre. LLM generira pozive alatu kao strukturirani JSON, okvir izvršava alate, a rezultati se vraćaju natrag u LLM za sljedeće korake. Ovaj obrazac generiranja uz pomoć alata odgovara metafori istraživačkog asistenta: agenti koji mogu "provjeriti informacije" i "izvesti analize", a ne samo generirati tekst.

Upravljanje sesijama prati povijest razgovora u AgentSession dokumentima. Višekratni razgovori omogućuju iterativno dorađivanje: "izdvoji činjenice iz ovog rada" → "fokusiraj se na odjeljak metodologije" → "stvori stiliziranu činjenicu koja sažima rezultate". Sesije pohranjuju povijest poruka, tragove poziva alata i kontekst (referencirane fact_ids, document_ids). Ta povijest omogućuje debugiranje (zašto je agent proizveo ovaj izlaz?) i učenje (analiza uspješnih i neuspješnih sesija izdvajanja).

Alternativne arhitekture uključuju isključivo cloud rješenja (jednostavna, ali skupa), fino podešavanje modela (točno, ali sporo za iteraciju) i potpuni izostanak AI-ja (točno, ali potpuno ručno). Kombinacija lokalnog LLM-a i agentnog okvira balansira koristi automatizacije, kontrolu troškova i brzinu iteracije primjereno za razvoj istraživačke platforme.

10.4 Upiti na znanje i pretraživanje

Dizajnerska odluka: MongoDB tekstualna pretraga s opcionalnom vektorskom sličnosti

Pretraživanje znanja koristi ugrađeni tekstualni indeks MongoDB-a za free-text upite, uz arhitekturu koja predviđa buduće pretraživanje pomoću vektorskih ugradnji.

MongoDB tekstualna pretraga pruža osnovne, ali brze full-text upite preko sadržaja činjenica, teksta dokumenata i oznaka čvorova u grafu. Tekstualni indeks na facts.content, stylized_facts.summary i poljima nodes unutar knowledge_graphs omogućuje upite poput "pronađi činjenice o lososovim nametnicima" s bodovanjem relevantnosti i rangiranjem rezultata. Ovaj pristup koristi postojeću MongoDB infrastrukturu bez dodatnih servisa.

Ograničenje je semantičko razumijevanje — tekstualna pretraga podudara ključne riječi, ali propušta konceptualnu sličnost. "sea lice" i "parasitic copepods" neće se nužno podudarati iako označavaju povezane pojmove. Vektorske ugradnje (kodiranje teksta u numeričke vektore i pretraživanje prema vektorskoj sličnosti) hvataju semantičke odnose, ali zahtijevaju modele za ugradnje, pohranu vektora i infrastrukturu za izračun sličnosti.

Arhitektura omogućuje buduću vektorsku pretragu dizajniranjem trenutačne tekstualne pretrage iza apstraktnog sloja. Endpoint /api/knowledge/search vraća bodovane rezultate neovisno o podlozi implementacije. Kada zatreba vektorska pretraga, dodajemo generiranje ugradnji (poziv modela za ugradnje pri unosu znanja), pohranu vektora (dodavanje polja embedding u dokumente) i implementaciju pretraživanja sličnosti (kosinusna sličnost prema ugradnji upita). API ugovor se ne mijenja, što omogućuje nenarušavajuće nadogradnje.

"Opcionalna" priroda odražava prioritete u beta fazi: tekstualna pretraga zadovoljava početne slučajeve uporabe, a dodavanje kompleksnosti vektorske pretrage prije nego što se pokaže nužnom rizik je preuranjene optimizacije. Povratne informacije korisnika pokazat će je li semantička pretraga nužna ili samo poželjna. Izgradnja jednostavnijeg rješenja prvo omogućuje bržu iteraciju, dok arhitektura ostaje otvorena za kasnija poboljšanja.

Filtriranje temeljeno na oznakama (primjerice činjenice s tags=["salmon", "parasite"]) nadopunjuje tekstualnu pretragu omogućujući precizne upite nad podskupovima podataka. Oznake ručno dodjeljuju kuratori ili ih automatski izdvajaju agenti, pružajući strukturirane metapodatke uz nestrukturirani tekst. Kombinacija tekstualne pretrage (pronalaženje relevantnog sadržaja) i filtriranja po oznakama (sužavanje na specifične domene) odgovara istraživačkim tijekovima rada u kojima domenska stručnost vodi istraživanje.

11 Arhitektura Modeling Assistant-a

11.1 Razvoj modela putem chatbota s RAG-om

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

Modeling Assistant pruža sučelje chatbota u kojem istraživači komuniciraju s LLM-om koji uz pomoć RAG-a prolazi kroz bazu znanja i gradi bioenergetske modele.

Arhitektura Modeling Assistant-a usredotočena je na suradnju čovjeka i AI-ja pri razvoju novih paradigmatskih modela u bioenergetici. Umjesto da istraživači moraju ručno pretraživati bazu znanja i sastavljati modele, sučelje chatbota omogućuje prirodnojezično

istraživanje: "Što znamo o alokaciji energije kod lososa tijekom reprodukcije?" ili "Pokaži obrasce metaboličkog skaliranja kod koštunjača (teleost)". LLM, proširen RAG-om, dohvaća relevantne činjenice i stilizirane činjenice iz baze znanja, sintetizira informacije iz više izvora i predlaže modelne pristupe temeljem dostupnih dokaza.

Konverzacijska izgradnja modela: Tijek rada odražava iterativno znanstveno razmišljanje. Istraživač može postavljati istraživačka pitanja o obrascima alokacije energije, tražiti usporedbe između vrsta ili životnih stadija, pregledavati potporne i kontradiktorne dokaze, predlagati strukture modela te prilagođavati pretpostavke na temelju sadržaja iz baze znanja. Chatbot održava kontekst razgovora kroz više koraka, što omogućuje postupno preciziranje. Svaka interakcija koristi RAG kako bi odgovori bili utemeljeni u kuriranim činjenicama – sprječavaju se halucinacije i čuva se znanstvena strogoća.

Prolazak kroz bazu znanja: RAG omogućuje inteligentnu navigaciju kroz umreženo znanje. Kada se raspravlja o alokaciji energije, sustav dohvaća povezane činjenice o metaboličkim stopama, reprodukcijskim parametrima i dinamici transportnih mreža. Slijedi tipizirane odnose (potporni dokazi, kontradiktorni dokazi, lanci profinjavanja) kako bi prikazao cjelovitu sliku. Chatbot može objasniti: "Ova stilizirana činjenica o metaboličkom skaliranju podržana je s 15 empirijskih opažanja preko 8 vrsta" te istovremeno pružiti izravne poveznice na temeljne činjenice. Ta sposobnost prolaska kroz znanje pomaže istraživačima u otkrivanju veza i obrazaca koji bi pri ručnom pretraživanju lako promakli.

Razvoj paradigmatskih modela: Primarni fokus platforme u početnoj fazi jest izgradnja osnovnih alata i utilitarnih funkcija za stvaranje sveobuhvatne baze znanja za bioenergetsko modeliranje organizama. Arhitektura chatbota s RAG-om podupire tu misiju čineći znanje pristupačnim i primjenjivim. Umjesto da istraživači tjednima ručno pregledavaju literaturu, sustav kroz razgovor izdvaja relevantno znanje, što ubrzava formuliranje hipoteza i konceptualizaciju modela. Naglasak je na izgradnji baze znanja i alata; detaljni API-ji i sposobnosti izvršavanja modela su u aktivnom razvoju i evoluirat će kako budu jasniji istraživački zahtjevi.

Trenutni status razvoja: Arhitektura predviđa buduće sposobnosti (simulaciju modela, procjenu parametara, vizualizaciju rezultata), ali daje prednost neposrednim potrebama: unosu znanja, kuratorskim tijekovima rada, upravljanju odnosima između činjenica te istraživačkom pristupu znanju kroz razgovorno sučelje. API krajnje točke za izvršavanje modela i naprednu analitiku su u razvoju, a dizajni su dovoljno fleksibilni da se prilagode zahtjevima koji će se pojaviti kroz stvarnu uporabu.

Ovakav pristup chatbota s RAG-om usklađen je s ciljem platforme da unaprijedi DEB modele temeljene na transportnim mrežama (tDEB). Omogućujući istraživačima da prirodnim jezikom istražuju AmP bazu s oko 4.500 vrsta, identificiraju domene primjenjivosti i otkrivaju općenitosti unutar i između filogenetskih skupina, sustav ubrzava proces znanstvenog otkrića. Konverzacijsko sučelje snižava prag pristupa znanju, čineći vrijednost platforme – sveobuhvatno znanje o bioenergetici – izravno primjenjivom za razvoj modela.

11.2 Scenarijima vođen tijek modeliranja

Dizajnerska odluka: Entiteti scenarija kao spremnici modelskih projekata

Modeliranje je organizirano u dokumente tipa Scenario koji objedinjavanju ciljeve, reference na znanje, predložene modele i rezultate.

Arhitektura u kojoj je scenarij središnji entitet odražava način na koji modelari u stvarnosti rade. Istraživački projekt ne počinje s "izgradi model", već s "razumij fenomen X u uvjetima Y". Scenariji hvataju upravo to: naziv ("Dinamika lososovih nametnika u norveškim fjordovima"), opis (istraživački kontekst), ciljeve (konkretna pitanja na koja treba odgovoriti) i `knowledge_queries` (upite prema KB-u koji su relevantni za scenarij). Scenarij je, dakle, spremnik za projekt modeliranja.

Takav obrazac spremnika donosi više koristi. Za jedan scenarij može se predložiti više modela, što omogućuje usporedbu: "model A koristi ODE formulaciju, model B individual-based pristup – koji se bolje slaže s empirijskim podacima?". Rezultati se povezuju natrag na scenarije i modele, čuvajući potpunu sljedivost. Scenariji postaju osnovna jedinica suradnje: istraživači dijele scenarije, a ne samo izvorni kod modela. UUID scenarija pojavljuje se u audit logovima, što omogućuje analizu koji istraživački problemi troše najviše resursa platforme.

Alternativni obrasci organizacije uključuju model-centrični pristup (model je primaran, scenariji su implicitni) ili potpuno "plosnati" pristup bez grupiranja. Model-centrični pristupi loše se nose sa situacijama u kojima se za jedno pitanje istražuje više modelnih strategija. Plosnate strukture gube kontekst projekta koji je istraživačima nužan za interpretaciju rezultata. Spremnik scenarija prirodno odgovara mentalnom modelu istraživača, a istodobno pruža jasnu tehničku strukturu.

Polje `knowledge_queries` povezuje scenarije s KB sadržajem. Umjesto kopiranja znanja u MA (što bi vodilo duplikaciji), scenariji pohranjuju upite prema KB-u koji se izvršavaju pri učitavanju scenarija. Ova indirekcija održava prikaz znanja u MA ažurnim kako KB evoluira, sprječava zastarjelost podataka i održava jedan izvor istine. Kompromis je ovisnost: ako KB nije dostupan, MA ne može razriješiti reference na znanje. Za beta razmjer i su-hostane servise taj je kompromis prihvatljiv.

11.3 Sastavljanje i reprezentacija modela

Dizajnerska odluka: Deklarativne specifikacije modela s vezama na KB

Modeli se predstavljaju kao JSON specifikacije koje opisuju entitete, stanja, procese i parametre, pri čemu je svaki element opravdan referencama na KB.

Deklarativni pristup specifikacijama odvaja strukturu modela (koji entiteti postoje, koja stanja i procesi) od izvršavanja modela (simulacijski kod). Dokument specifikacije modela sadrži entitete (npr. ["domaćin", "parazit"]), stanja ("infekcijski status", "dob", "lokacija"), procese ("prijenos", "oporavak", "kretanje") i parametre ("beta: stopa prijenosa", "gamma: stopa oporavka"). Ova strukturirana reprezentacija omogućuje programsku analizu, vizualizaciju, a kasnije i generiranje koda.

Veze za opravdanje povezuju elemente modela s entitetima u KB-u. Proces "prijenos" referencira `fact_ids` koji opisuju opažene mehanizme prijenosa. Parametar "beta" referencira stilizirane činjenice o izmjerenim stopama prijenosa. Te veze služe višestrukim svrhama: čine pretpostavke modela eksplicitnima i sljedivima, omogućuju automatizirani pregled literature (koji dokazi podupiru ovaj model?) i olakšavaju usporedbu modela (modeli A i B se razlikuju u pretpostavkama o prijenosu; koji KB zapisi podupiru svaki od njih?).

Ovakva arhitektura odražava zahtjeve znanstvene odgovornosti. Modeli bez jasnih pretpostavki su "crne kutije"; pretpostavke bez dokaza su spekulacije. Veze na KB

pretvaraju modele iz neprozirnog koda u transparentne lance zaključivanja. Recenzenti mogu kritički procjenjivati pretpostavke pregledom pripadajućih dokaza. Ova sljedivost nije besplatna, ali je opravdana u znanstvenom modeliranju, gdje vjerodostojnost modela ovisi o jasnoći argumentacije.

Specifikacija je neovisna o načinu izvršavanja: opisuje što model predstavlja, a ne kako se simulira. Budući sustavi za izvršavanje moći će iz iste specifikacije generirati, primjerice, NetLogo kod, Python simulacijske petlje ili sustave diferencijalnih jednadžbi. Takvo razdvajanje omogućuje evoluciju: strategije izvršavanja mogu se poboljšavati bez mijenjanja semantike modela. U beta fazi specifikacije prvenstveno služe dokumentiranju i komunikaciji; kasnije će postati podloga za automatizirano izvršavanje.

Alternativni pristupi uključuju "kod kao model" (model je Python kod simulacije) ili isključivo grafičke prikaze (model je crtež). Kod kao model je precizan, ali neproziran – razumijevanje zahtijeva čitanje implementacije. Čisto grafički prikazi su intuitivni, ali neformalni – okviri i strelice često nemaju strogo definiranu semantiku. Deklarativna specifikacija s vezama prema KB-u pruža preciznost, sljedivost i semantiku prikladnu za znanstveni dijalog.

11.4 Integracija s Knowledge Builderom

Dizajnerska odluka: HTTP API integracija s prosljeđivanjem JWT tokena

Modeling Assistant se integrira s Knowledge Builderom putem HTTP API-ja, pri čemu prosljeđuje JWT tokene korisnika kako bi se zadržala jedinstvena autentikacija i atribucija.

HTTP integracija održava MA i KB labavo spojenima. MA šalje standardne HTTP GET/POST zahtjeve prema KB krajnjim točkama (`/api/knowledge/search`, `/api/knowledge/facts/{id}`, `/api/agents/run`). Ovakav stil integracije omogućuje neovisno isporučivanje: KB se može nadograditi bez redeploya MA, dokle god su API ugovori očuvani. Razvoj može napredovati paralelno: timovi za KB i MA rade neovisno, a integracijske točke su jasno definirani ugovori.

Prosljeđivanje JWT tokena čuva sigurnost i odgovornost preko granice komponenti. Kada se korisnik autentificira u MA, dobiva JWT token. Kada MA u njegovo ime poziva KB, token uključuje u HTTP zaglavlja. KB validira token, provjerava dozvole i radnju pripisuje izvornom korisniku. Time se čuva audit trag: upiti na znanje koje pokreće kreiranje scenarija u MA-u u audit logovima se pripisuju korisniku koji je pokrenuo scenarij, a ne generičkom MA servisnom računu.

Alternativa s autentikacijom na razini servisa (MA ima vlastite vjerodajnice za pozive prema KB-u) bila bi implementacijski jednostavnija, ali bi izgubila atribuciju korisnika. KB bi sve zahtjeve iz MA-a vidio kao da dolaze od "MA servisa", a ne od pojedinih korisnika. Time bi se narušila odgovornost i onemogućilo provođenje provjera temeljnih na sposobnostima (MA ne bi smio pokretati agente ako korisnik nema `agent_access`). Prosljeđivanje tokena zadržava korisnički kontekst od kraja do kraja, uz cijenu dodatnog rukovanja istekom i osvježavanjem tokena u MA→KB pozivima.

HTTP integracija uvodi latenciju u odnosu na izravan pristup bazi (MA koji bi čitao KB-ove kolekcije izravno iz MongoDB-a). Svaki poziv prema KB-u dodaje mrežno vrijeme i trošak parsiranja HTTP-a. Za beta razmjer (korisnici rade "ljudskom brzinom", ne radi se o viskofrekventnom trgovanju) ta je latencija zanemariva (reda desetaka milisekundi).

Dobivena arhitektonska čistoća odvajanja komponenti nadmašuje troškove performansi.

Buduće optimizacije mogu uključivati cacheiranje često korištenih KB odgovora u MA-u, grupiranje više KB upita u jedan poziv ili korištenje GraphQL-a za fleksibilno dohvaćanje podataka. Te nadogradnje su moguće jer je HTTP integracija standardna i dobro shvaćena. Preuranjena optimizacija putem prilagođenih RPC sustava ili zajedničkog pristupa bazi u ovoj fazi bi dodala kompleksnost bez mjerljive koristi.

12 Integracijska arhitektura i podatkovni ugovori

12.1 Model dijeljenja znanja: API ugovori nasuprot zajedničkoj bazi

Dizajnerska odluka: MA pristupa znanju iz KB-a preko HTTP API-ja, a ne izravno preko baze

Modeling Assistant dohvaća znanje putem dokumentiranih HTTP krajnjih točaka, umjesto da čita MongoDB kolekcije koje koristi Knowledge Builder.

Integracija temeljena na API-ju nameće jasne sučeljske ugovore između komponenti. KB objavljuje dokumentirane krajnje točke (`/api/knowledge/search`, `/api/knowledge/facts/{id}`) sa specificiranim formatima zahtjeva i odgovora. MA ovisi o tim ugovorima, a ne o internim strukturama podataka KB-a. Ta apstrakcija omogućuje KB-u da refaktorira sheme baza, optimizira upite ili čak migrira na druge tehnologije pohrane bez utjecaja na MA – pod uvjetom da su API ugovori očuvani.

Zajedničko korištenje baze (komponente koje čitaju/pišu iste MongoDB kolekcije) uklonilo bi mrežnu latenciju i pojednostavilo upite koji presijecaju znanje i podatke o modeliranju. Međutim, takva integracija stvara čvrsto spajanje koje ograničava evoluciju. KB ne bi mogao mijenjati shemu `facts` bez koordinacije s MA. Migracije baze morale bi se sinkronizirati preko komponenti. Istovremeni upisi bi zahtijevali složene mehanizme zaključavanja kako bi se izbjegla nekonzistentnost. Ti troškovi spajanja usporili bi razvoj.

Pristup temeljen na API-ju prihvaća trošak latencije kako bi dobio slobodu evolucije. Svaki KB poziv iz MA-a nosi HTTP overhead od nekoliko milisekundi, ali razvojni timovi rade neovisno. KB može mijenjati implementaciju upita bez izmjene MA. MA može lokalno cacheirati KB odgovore. Verzioniranje API-ja (`/api/v1/...`, `/api/v2/...`) omogućuje postupnu migraciju kada se ugovori mijenjaju. Na beta razmjeru, gdje korisnici generiraju zahtjeve ljudskom brzinom, dobivena fleksibilnost znatno je vrjednija od izgubljenih milisekundi.

Dokumentirani integracijski ugovori služe kao ključne točke koordinacije. Obje se strane dogovaraju o oblicima krajnjih točaka, značenju polja i kodovima pogrešaka. Ti eksplicitni ugovori sprječavaju implicitne ovisnosti i čine integraciju vidljivom. Ugovori su verzionirani i pregledavaju se kao i ostatak koda, čime se osigurava da su promjene namjerne i dobro komunicirane.

12.2 Sinkronizacija podataka i konzistentnost

Dizajnerska odluka: Bez sinkronizacije podataka – MA pohranjuje reference, a ne kopije

MA u dokumentima scenarija i modela pohranjuje ID-jeve entiteta iz KB-a (`fact_ids`, `document_ids`) umjesto da kopira sadržaj znanja.

Pristup temeljen na referencama održava jedan izvor istine za znanje. Kada scenarij referencira činjenicu s ID-jem `f123`, MA pohranjuje samo `"f123"` a ne kopiju sadržaja činjenice. Pri prikazu scenarija, MA poziva KB API i dohvaća trenutačni sadržaj činjenice. Tako MA uvijek prikazuje najnoviju verziju znanja – ako kuratori izmijene `f123`, svi scenariji koji je referenciraju automatski odražavaju promjene, bez potrebe za sinkronizacijom.

Alternativa s kopiranjem znanja u MA (denormalizacija) uklonila bi potrebu za KB pozivima tijekom rada u MA-u. Prikaz scenarija tada bi zahtijevao samo upite u lokalnu bazu MA-a, bez mrežnih skokova. No, takav pristup uvodi problem zastarijevanja: činjenica `f123` kopirana u 50 scenarija zahtijeva 50 zasebnih ažuriranja kad se činjenica promijeni. Logika sinkronizacije postaje složena, sklona greškama i povećava operativni teret.

Pristup referencama svjesno prihvaća ovisnost u izvođenju kako bi očuvao integritet podataka. MA ovisi o dostupnosti KB-a – ako je KB nedostupan, MA ne može razriješiti reference i može prikazati samo ID-jeve. Za beta razmjer s ko-lociranim servisima i ciljanom dostupnošću od barem 99,9 %, taj je kompromis prihvatljiv. Jednostavnost izbjegavanja sinkronizacije nadmašuje rizike povezane s dostupnošću.

Cacheiranje može ublažiti troškove ovisnosti bez potpune denormalizacije. MA može cacheirati KB odgovore (npr. sadržaj činjenica) s kratkim TTL-om (npr. 5 minuta). Pogoci u cacheu izbjegavaju pozive prema KB-u i poboljšavaju performanse, dok istjecanje cachea osigurava barem eventualnu svježinu. Takav hibridni pristup pruža praktične performanse, a istodobno čuva semantiku pristupa temeljenog na referencama. Cacheiranje je optimizacija koju ima smisla uvesti tek nakon što mjerenja pokažu da latencija KB API-ja predstavlja stvaran problem.

Problem distribuiranih transakcija između komponenti izbjegava se samim dizajnom: MA ne piše podatke u KB. Scenariji i modeli su entiteti MA-a; činjenice i dokumenti su entiteti KB-a. Jednosmjerni tok podataka (KB proizvodi znanje, MA ga konzumira) uklanja potrebu za koordinacijom transakcija u distribuiranom sustavu. Ako se u budućnosti pojavi potreba da MA stvara znanje (npr. "spremi uvid iz scenarija kao novu činjenicu u KB-u"), te će operacije koristiti KB-ove API-je za pisanje, uz autentikaciju korisnika, čime se granice komponenti i dalje zadržavaju.

12.3 Kompatibilnost verzija i evolucija API-ja

Dizajnerska odluka: Verzije API-ja u putanji URL-a uz očekivanje unatrag kompatibilnih promjena

KB krajnje točke uključuju prefiks verzije (npr. `/api/v1/knowledge/search`) uz obvezu održavanja v1 ugovora dok se razvija v2.

Verzioniranje API-ja omogućuje evoluciju ugovora bez razbijanja postojećih MA implementacija. Kada KB treba uvesti nekompatibilne promjene (preimenovanje polja

u odgovoru, drugačija semantika), objavljuje v2 krajnje točke paralelno s v1. MA nastavlja koristiti v1 dok postupno ne migrira na v2. Zastarjele verzije ostaju dostupne tijekom prijelaznog razdoblja (npr. šest mjeseci uz najavu uklanjanja), čime se omogućuje nadogradnja bez prisilne sinkronizacije.

Verzioniranje putem putanje URL-a (`/v1/`, `/v2/`) čini verzije eksplicitnima i lako uočljivima. Alternativa s verzijama u zaglavljima (`Accept: application/vnd.kb.v1+json`) manje je vidljiva. Verzije kao query parametri (`?version=1`) miješaju semantiku verzije i filtriranja. Putanje s prefiksom verzije slijede uobičajene REST konvencije i ne zahtijevaju posebnu podršku klijenata.

Unutar pojedine verzije primjenjuju se načela semantičkog verzioniranja: dodavanje opcionalnih polja je sigurno, promjena tipova postojećih polja je lomljiva, uklanjanje polja je lomljivo. KB može u v1 odgovorima dodavati nova polja (MA ih jednostavno ignorira), ali ne smije uklanjati obavezna polja bez uvođenja v2. Ta disciplina sprječava slučajna razbijanja postojećih klijenata, a ipak dopušta organski rast API-ja.

Obveza unatrag kompatibilnih promjena utječe na razvoj: lomljive promjene moraju opravdati uvođenje nove verzije sa svim pripadajućim troškovima (dokumentacija, testiranje, podrška pri migraciji). Ta "frikcija" potiče pažljiv dizajn API-ja – vrijedi se potruditi oko dobrog prvog dizajna jer je kasnije skupo mijenjati ugovore. U beta fazi prihvaća se određeni stupanj nestabilnosti (brze tranzicije v1→v2) kako bi se zadržala fleksibilnost, ali i dalje se teži minimaliziranju prekida kompatibilnosti.

Dokumentacija uključuje primjere poziva, sheme odgovora (JSON Schema) i kodove pogrešaka. Dokumentacija se automatski testira (provjerava se da dani primjeri zaista rade) kako bi se spriječilo razilaženje između dokumentacije i koda. Jasni ugovori omogućuju razvoj MA-a bez stalnog kopanja po KB kodu, čime se smanjuje potreba za koordinacijom.

13 Podatkovni model i odluke o pohrani

13.1 MongoDB za sve podatke: temelj grafa znanja

Dizajnerska odluka: Koristiti MongoDB kao objedinjeno spremište za grafove znanja, činjenice, dokumente, scenarije, modele i korisničke podatke

Platforma koristi MongoDB za svu trajnu pohranu, prvenstveno zbog zahtjeva grafa znanja, pri čemu ostali tipovi podataka koriste istu infrastrukturu.

Odabir MongoDB-a u srži je usmjeren na skalabilnost grafa znanja. Vrijednost platforme leži u upravljanju umreženim bioenergetskim znanjem: vrste pokazuju obrasce alokacije energije, organizmi dijele odnose metaboličkog skaliranja, fiziološki procesi međudjeluju kroz različite životne stadije. Ti odnosi čine graf znanja koji mora podržavati upite nad tipiziranim odnosima ("pronađi sve činjenice koje podupiru ovu hipotezu o alokaciji energije", "identificiraj kontradiktorne dokaze za metaboličko skaliranje"), prolazak kroz graf uz filtriranje po tipu brida ("koji bioenergetski parametri su dokumentirani za lososе s visokim stupnjem pouzdanosti?"), traženje putanja ("prati lanac profinjavanja od sirovih opažanja do općih principa") i izdvajanje podgrafova ("izvezi podgraf znanja za alokaciju energije tijekom reprodukcije, uključujući sve potporne i kontradiktorne dokaze").

MongoDB-ov dokumentni model s ugniježđenim nizovima i referencama prirodno predstavlja graf: dokument čvora može sadržavati {id, label, type, properties,

outgoing_edges: [{target_id, edge_type, weight}]}. Polje `edge_type` omogućuje tipizirane odnose koji su ključni za znanstvenu reprezentaciju znanja: "supports" (dokaz koji podupire tvrdnju), "contradicts" (kontradiktoran dokaz), "refines" (specijalizirana verzija), "synthesizes" (sinteza više izvora), "references" (kontekst citiranja) ili domenski specifične odnose poput "measured_by" (metodološke veze) i "taxonomically_related" (filogenetske veze). Prolazak kroz graf postaje rekurzivno dohvaćanje dokumenata s filtriranjem nizova bridova po tipu. Agregacijski pipelineovi implementiraju graf-algoritme: `$graphLookup` za pretragu u širinu s filtriranjem po tipu brida, lanci `$lookup` operacija za višekorakne putanje koje slijede specifične obrasce odnosa.

Sposobnosti skaliranja razlikuju MongoDB od alternativa. Kako graf znanja raste (cilj: 100.000+ entiteta, milijun+ odnosa), shardanje u MongoDB-u omogućuje raspodjelu podataka preko više servera. Često korišteni podgrafovi ostaju u memoriji, dok rjeđe korišteni podaci ostaju na disku. Performanse upita ostaju zadovoljavajuće jer graf upiti tipično istražuju lokalna susjedstva (nekoliko skokova), a ne cijelu strukturu. PostgreSQL s rekurzivnim CTE-ovima može reprezentirati grafove, ali ne skalira horizontalno tako prirodno. Neo4j je izvrstan za graf upite, ali bi zahtijevao održavanje dvije baze (Neo4j za graf, nešto drugo za dokumente i korisnike).

Entiteti znanja (činjenice, stilizirane činjenice) profitiraju od fleksibilnosti sheme pri reprezentaciji sadržaja. Činjenica o alokaciji energije kod lososa ima drugačija polja od činjenice o metaboličkom skaliranju među sve ribama. Stilizirane činjenice sintetiziraju promjenjiv broj izvorišnih činjenica. Unos dokumenata stvara različite metapodatke ovisno o tipu izvora (PDF, HTML, strukturirani podaci). Te heterogene strukture prirodno se mapiraju na MongoDB dokumente – svaki tip entiteta ima jezgru obaveznih polja plus opcionalna domenska polja.

JSON-native pohrana omogućuje neprekinuti tok podataka kroz cijeli sustav: LLM agenti proizvode JSON strukture znanja, FastAPI prima JSON od frontenda, MongoDB pohranjuje BSON (binarni JSON), a HTTP API-ji vraćaju JSON. Taj kraj-do-kraja JSON tok uklanja impedance-mismatch probleme karakteristične za ORM sustave u kojima se objekti teško preslikavaju na relacijske retke.

Podaci o korisnicima i autentikaciji žive u MongoDB-u kao sekundarni aspekt – baza je ionako potrebna za graf znanja, pa je racionalno koristiti je i za korisničke podatke radi operativne jednostavnosti. U usporedbi s entitetima znanja, strukture korisnika su jednostavne, zbog čega je MongoDB za njih "prejak" alat, ali uvođenje PostgreSQL-a samo zbog upravljanja korisnicima povećalo bi kompleksnost (dvije baze za backup, nadzor i skaliranje) bez rješavanja glavnog izazova: grafa znanja.

Kompromisi postoje. Kompleksni analitički upiti koji spajaju korisnike, doprinose, recenzije i entitete znanja nisu elegantni u MongoDB-u; u PostgreSQL-u bi takvi upiti s više JOIN-ova bili jednostavniji. Ipak, ti su upiti rijetki – platforma optimizira za obrasce pristupa znanju (tekstualno pretraživanje, prolazak kroz graf, dohvat entiteta), a ne za administrativnu analitiku.

Podaci vremenskih serija (slijedovi poruka agenata, audit logovi, metričke uporabe) mogli bi profitirati od specijaliziranih spremišta (TimescaleDB, InfluxDB). Za beta volumen (tisuće poruka dnevno, 10–100 aktivnih korisnika), MongoDB-ovi vremenski indeksi i TTL kolekcije su dostatni. Ako analiza telemetrije postane središnja funkcija, baze vremenskih serija mogu nadopuniti, ali ne zamijeniti MongoDB – graf znanja ga ionako zahtijeva.

Odluka o jednoj tehnologiji pruža fokus u operacijama. Mali istraživački tim razvija duboku ekspertizu u MongoDB-u (strategije indeksiranja, optimizacija agregacijskih pipelineova, konfiguracija shardinga), umjesto plitkog poznavanja više baza. Backup,

nadzor, optimizacija upita i rješavanje problema razvijeni za graf znanja koriste se svugdje. Ta operativna učinkovitost važnija je na beta razmjeru od marginalnih dobitaka performansi koje bi donijele specijalizirane baze.

Buduća specijalizacija ostaje moguća: ako graf upiti budu dominirali, Neo4j može dopuniti MongoDB; ako tekstualno pretraživanje postane nedostatno, Elasticsearch može ga proširiti; ako analitički upiti postanu brojni, PostgreSQL može preuzeti izvještavanje. No takva se proširenja uvode kasnije, na temelju stvarnih obrazaca uporabe. Početak s MongoDB-om kao jedinom bazom omogućuje da prvo naučimo što platformi zaista treba, umjesto da rano optimiziramo za pretpostavljene zahtjeve.

13.2 Kolekcije dokumenata i odnosi

Dizajnerska odluka: Odvojene kolekcije za svaki tip entiteta s referencama temeljenima na ID-jevima

Znanje koristi odvojene MongoDB kolekcije (facts, stylized_facts, documents, knowledge_graphs) s referencama preko ObjectId vrijednosti, umjesto ugniježđenih dokumenata ili jedne kolekcije s tipnim oznakama.

Odvojene kolekcije odražavaju prirodne granice entiteta: činjenice su konceptualno i po obrascima pristupa različite od dokumenata. Činjenice se često pretražuju (tekstualno, po oznakama), dok se dokumenti rjeđe dohvaćaju (kao izvor). Posebne kolekcije omogućuju neovisno indeksiranje: tekstualni indeks na `facts.content`, složeni indeks na `documents.uploaded_by` + `created_at` itd. Jedna kolekcija s poljem tipa zahtijevala bi rijetke indekse i kompleksne upite koji stalno filtriraju po tipu.

Reference temeljene na ID-jevima (npr. `fact.document_id` referencira `documents._id`) nalikuju relacijskim vanjskim ključevima, ali bez nametnute integritetske kontrole. MongoDB ne osigurava referencijalni integritet – brisanje dokumenta ne uzrokuje automatsko brisanje njegovih činjenica. Ta "slabost" je ovdje namjerna: istraživački tijekovi mogu zahtijevati da činjenice prežive i nakon uklanjanja dokumenta (npr. dokument je tehnički neupotrebljiv, ali su podaci već provjereni i korišteni). Eksplicitno rukovanje brisanjima na razini aplikacije daje fleksibilnost uz cijenu ručnog održavanja integriteta.

Alternativa s ugniježđivanjem (dokumenti s ugniježđenim nizom činjenica) uklonila bi neke reference, ali se ne uklapa u obrasce pristupa. Činjenice se često dohvaćaju neovisno putem pretraživanja, a ne uvijek preko nadređenog dokumenta. Ugniježđivanje činjenica unutar dokumenata prisililo bi na upite nad kolekcijom `documents` s pretragom po elementima niza, što bi bilo neučinkovito za upite usmjerene na činjenice. Ugniježđivanje ima smisla za istinski ugniježđene podatke koji se gotovo uvijek dohvaćaju zajedno (npr. poruke unutar jedne sesije agenta).

Modeliranje odnosa slijedi uobičajene MongoDB obrasce: odnosi jedan-prema-više koriste reference (dokument ima mnogo činjenica: `facts.document_id`), odnosi više-prema-više koriste nizove ID-jeva (graf znanja povezan s više stiliziranih činjenica: `knowledge_graphs.stylized_fact_ids = [id1, id2, ...]`). Ovaj hibridni pristup (reference za 1:N, nizovi za N:M) balansira učinkovitost upita i jednostavnost ažuriranja.

Obrazac odnosa s korisnicima (polje `created_by: ObjectId` koje referencira kolekciju `users`) povezuje znanje s autorima radi atribucije. Svaki entitet znanja nosi `created_by` dodan tijekom autentikacijskog sloja, što omogućuje upite poput "činjenice koje je stvorio

kurator X" za praćenje doprinosa i analizu kvalitete. Ovaj univerzalni obrazac atribucije omogućuje buduće značajke (reputacijski sustavi, metričke kvalitete znanja, preporuke temeljene na stručnosti autora) bez izmjena sheme.

14 Buduća evolucija

14.1 Što početna implementacija namjerno isključuje

Dizajnerska odluka: Svjesno izostavljanje određenih značajki iz početnog opsega

Nekoliko značajki koje su uobičajene u autentikacijskim sustavima namjerno je izostavljeno iz početne implementacije – nisu zaboravljene niti previđene.

Višefaktorska autentikacija iznad one koju pruža Google nije implementirana jer bi bila suvišna. Korisnici koji na svom Google računu uključe dvofaktorsku autentikaciju automatski dobivaju istu razinu zaštite i u AdvanDEB-u, zahvaljujući OAuth-u. Samostalno uvođenje TOTP (Time-based One-Time Password) rješenja dodalo bi složenost i natjeralo korisnike da upravljaju još jednim uređajem/aplikacijom. Ako buduće potrebe zahtijevaju platform-specifični MFA (npr. za visokorizične operacije), postojeća arhitektura autentikacije omogućuje njegovo dodavanje bez potpune rekonstrukcije sustava.

IP whitelisting za API ključeve nije implementiran jer većina korisnika nema statične IP adrese. Akademske institucije često koriste dinamičko adresiranje, a istraživači rade od kuće, iz kafića ili s konferencija – njihove se IP adrese stalno mijenjaju. Implementacija IP whitelisting-a bi stvorila ozbiljan teret podrške (legitimni korisnici zaključani izvan sustava), uz ograničen sigurnosni dobitak (sofisticirani napadači ionako koriste proxyje). Rijetki slučajevi koji doista zahtijevaju IP ograničenja (server-to-server integracije) mogu se po potrebi rješavati ručnom konfiguracijom.

Finozrnasti ACL-ovi (liste kontrole pristupa) na razini pojedinih entiteta znanja nisu implementirani jer ih trenutačni zahtjevi ne opravdavaju. Svo objavljeno znanje je javno – svaki kurator ga može čitati. Nacrti su privatni za autora do slanja na recenziju. Ovaj binarni model (javno ili privatno autorsko) zadovoljava trenutne potrebe bez operativnog i konceptualnog tereta provjeravanja dozvola za svaki pojedini dokument. Ako se u budućnosti pojavi zahtjev za timskim prostorima (workspaces) ili privatnim kolekcijama znanja, model sposobnosti se može prirodno proširiti – nova sposobnost tipa `workspace_member` mogla bi kontrolirati pristup dokumentima unutar određenog prostora.

Hijerarhije uloga s nasljeđivanjem nisu uvedene jer kompleksnost koju donose nije potrebna – fleksibilna kompozicija sposobnosti postiže isti cilj. U hijerarhijskim modelima "viša" uloga (npr. senior_curator) nasljeđuje dozvole osnovne uloge i dodaje nove. Time promjene u osnovnoj ulozi automatski utječu na sve izvedene uloge, što stvara krute ovisnosti. Model sa sposobnostima postiže sličan rezultat kombiniranjem (kurator + analytics + reviewer) bez logike nasljeđivanja. Sposobnosti se kombiniraju neovisno, bez grafova nasljeđivanja koje je teško pratiti i razumjeti.

Ovi izostanci slijede YAGNI (You Aren't Gonna Need It) načelo agilnog razvoja. Izgradnja značajke prije nego što postane potrebna često znači izgradnju pogrešne značajke, jer se razumijevanje zahtjeva formira tek kroz stvarnu uporabu. Arhitektura autentikacije je proširiva – niti jedna od ovih odluka nas ne zatvara u "slijepu ulicu". Ako se pokaže da su IP whitelisting ili finozrnasti ACL-ovi potrebni, mogu se dodati bez radikalnih

promjena. Do tada, jednostavnost je prednost.

15 Zaključak

15.1 Sveobuhvatna arhitektura platforme

Ovaj dokument sada pruža cjelovito arhitektonsko obrazloženje AdvanDEB platforme preko svih glavnih komponenti:

- **Upravljanje korisnicima i autentikacija**: Integracija s Google OAuth-om, JWT tokeni, uloge temeljene na sposobnostima, API ključevi, audit logiranje i jedinstvena prijava između komponenti
- **Knowledge Builder**: Polu-nadzirana izgradnja baze znanja putem chatbota i obrade dokumenata, trodijelna reprezentacija znanja (činjenice, stilizirane činjenice, grafovi), asinkroni cjevovod unosa dokumenata, agentski okvir s RAG-om i lokalnim LLM-om te MongoDB tekstualno pretraživanje
- **Modeling Assistant**: Sučelje chatbota s LLM-om i RAG-om za konverzacijsko istraživanje znanja i razvoj modela, scenarijima vođen tijek rada, deklarativne specifikacije modela s vezama na KB
- **Integracijska arhitektura**: Povezivanje komponenti preko API-ja, model podataka temeljen na referencama bez sinkronizacije, verzionirani HTTP ugovori
- **Podatkovni model**: MongoDB za sve tipove podataka, odvojene kolekcije po entitetu, reference putem ID-jeva i univerzalna atribucija autora

Arhitektonske odluke u svim komponentama slijede konzistentna načela definirana u ranijim poglavljima: prednost jednostavnosti nad preuranjenom optimizacijom, odabir dokazanih tehnologija umjesto egzotičnih rješenja, labavo spajanje uz eksplicitne ugovore te mogućnost evolucije kroz pažljivo definirane apstrakcijske slojeve. Svaka se komponenta može razvijati neovisno, a koherencija platforme održava se zajedničkim autentikacijskim slojem, konzistentnom atribucijom podataka i jasno dokumentiranim integracijskim ugovorima.

15.2 Sažeti arhitektonski principi

Arhitektura proizlazi iz primjene nekoliko ključnih principa na sve dizajnerske odluke. Jednostavnost ima prednost pred "pametnim" rješenjima – preferiraju se tehnologije poput MongoDB-a i JWT-ova, koje su široko provjerene u praksi, umjesto novotarija. Sigurnost po defaultu zamjenjuje sigurnost kroz konfiguraciju – korisnici ne mogu slučajno oslabiti zaštitu pogrešnim postavkama. Osnaživanje korisnika kroz model sposobnosti omogućuje organski rast dozvola, umjesto krutih hijerarhija. Evolvabilnost znači da se budući zahtjevi rješavaju proširenjima, a ne ponovnim osmišljavanjem sustava.

Ti se principi ponekad sukobljavaju, što zahtijeva svjesne kompromise. Zajednička baza korisnika žrtvuje dio neovisnosti komponenti radi konzistentnosti i jednostavnosti. Model sposobnosti žrtvuje krajnju granularnost (nema ACL-ova po dokumentu) radi jednostavnijeg procesa odobravanja. JWT-ovi žrtvuju trenutačni opoziv sesije radi

bezdržavne performanse. Svaki je kompromis napravljen svjesno, nakon razmatranja alternativa i razumijevanja implikacija.

Arhitektura nije "optimalna" u univerzalnom smislu – nijedna nije. Ona je optimirana za specifičan kontekst AdvanDEB-a: znanstvenu platformu koja zahtijeva sljedivost i odgovornost, služi akademskim korisnicima s umjerenom tehničkom ekspertizom, razvija je mali tim koji prioritet daje funkcionalnostima pred infrastrukturnom složenošću, te cilja na 10–20 korisnika u beta fazi, s kapacitetom rasta na približno 100 korisnika.

15.3 Kriteriji uspjeha arhitekture autentikacije

Arhitektura autentikacije uspješna je ako se korisnici jednostavno autentificiraju i nesmetano rade preko svih komponenti, administratori učinkovito upravljaju dozvolama bez gušenja u zahtjevima za odobrenje, developeri mogu dodavati značajke bez lomljenja autentikacije, a platforma se glatko skalira s 10–20 beta korisnika na oko 100 korisnika uz održavanje barem 99,9 % dostupnosti i vremena odziva autentikacije ispod 100 ms. Sigurnosni uspjeh znači odsutnost incidenata povezanih s autentikacijom – nema ukradenih lozinki (jer ih ne pohranjujemo), nema eskalacija privilegija, nema rupa u audit logovima koje bi prikrile zlonamjerne radnje.

Ti su kriteriji mjerljivi i vremenski ograničeni. Kvaliteta korisničkog iskustva vidi se u broju tiketa prema podršci (manje prijava problema s autentikacijom znači bolji dizajn). Učinkovitost administratora ogleda se u vremenu potrebnom za obradu zahtjeva za sposobnostima. Produktivnost developera prepoznaje se u brzini isporuke značajki i učestalosti bugova. Performanse sustava vide se na nadzornim pločama. Sigurnosni uspjeh očituje se u rezultatima sigurnosnih revizija i zapisu incidenata (ili njihovom izostanku).

Arhitektura je sredstvo za postizanje tih ciljeva, a ne cilj sama po sebi. Elegantna arhitektura koja frustrira korisnike ili usporava razvoj ne uspijeva bez obzira na tehničku ljepotu. Pragmatična arhitektura koja omogućuje ostvarenje projektnog cilja uspijeva čak i ako ne bi osvojila nagrade za "čistoću dizajna". Ovaj dokument postoji kako bi povezao arhitektonske odluke s uspjehom projekta i učinio pragmatizam svake odluke eksplicitnim i opravdanim.

15.4 Arhitektura kao živi dizajn

Ovo nije konačna arhitektura. Odluke dokumentirane ovdje predstavljaju naše najbolje trenutačno razumijevanje, temeljeno na analizi zahtjeva, istraživanju korisnika i tehničkoj evaluaciji. Ipak, arhitektura mora evoluirati kako implementacija otkriva neočekivane izazove, povratne informacije korisnika pokažu probleme s upotrebljivošću ili testiranja performansi otkriju uska grla.

Stvarno iskustvo često ruši teorijske pretpostavke. Upit prema bazi koji je na papiru izgledao učinkovit može se u praksi pokazati problematičnim. Tijek autentikacije koji je na dijagramima djelovao intuitivno može zbunjivati korisnike. Integracijski pristup koji je izgledao arhitektonski "čist" može u produkciji stvarati operativno trenje.

Arhitektura AdvanDEB-a dizajnirana je za evoluciju. Granice komponenti, API ugovori i apstrakcijski slojevi pružaju fleksibilnost da se implementacije mijenjaju bez potpunih rekonstrukcija. Fleksibilnost sheme u MongoDB-u omogućuje prilagodbe podatkovnog modela. Model sposobnosti omogućuje promjene u sustavu dozvola bez izmjena koda. Bezdržavni dizajn JWT-ova dopušta promjene u infrastrukturnim komponentama autentikacije bez prekidanja aktivnih sesija.

Očekujemo da će se ovaj dokument razvijati zajedno s platformom. Značajne arhitektonske promjene zahtijevat će ažuriranje dokumenta, s poviješću verzija koja prati razvoj dizajnerskog promišljanja. Neuspjeli eksperimenti dokumentirat će se uz uspješne, kako bi se očuvalo institucionalno znanje o tome što nije funkcioniralo i zašto. Cilj nije arhitektonsko savršenstvo od prvog dana, već kontinuirano učenje kroz razvoj.

15.5 Buduća arhitektonska poboljšanja

Iako ovaj dokument pokriva jezgru arhitekture platforme, nekoliko će područja zahtijevati dodatne odluke kako implementacija bude napredovala:

1. **Napredne mogućnosti pretraživanja**: Vektorske ugradnje za semantičko pretraživanje, hibridno pretraživanje (tekst + vektori), podešavanje relevantnosti za znanstvenu literaturu

2. **Suradničke značajke** (faze 3–4): Timski prostori, kolaborativno uređivanje znanja, zajednički razvoj scenarija, rješavanje konflikata u doprinosima

3. **Izvršavanje modela**: Integracija simulacijskih motora, generiranje koda iz specifikacija modela, procjena parametara, vizualizacija rezultata

4. **Optimizacija performansi**: Strategije cacheiranja, fino podešavanje indeksa baze, optimizacija upita, pozadinska obrada zadataka (npr. Celery)

5. **Napredne mogućnosti agenata**: Višeagentni tijekovi rada, kompozicija alata, verifikacija rezultata agenata, inkrementalno učenje iz korekcija

6. **Sustavi povjerenja i reputacije**: Reputacijske ocjene za doprinositelje, metričke kvalitete znanja, automatizirano provjeravanje činjenica, analiza mreže citiranja

7. **Plugin arhitektura**: Ekstenzijske točke za prilagođene alate, agentske plugine, plugine za vizualizaciju i integraciju s trećim stranama

Ta poboljšanja grade se na temeljima opisanim u ovom dokumentu. Odluka da se ona odgode odražava prioritete beta faze: prvo uspostaviti osnovnu funkcionalnost (upravljanje znanjem, osnovna podrška modeliranju, upravljanje korisnicima), a zatim dodavati napredne mogućnosti. Arhitektura već sada sadrži ekstenzijske točke u koje se te buduće funkcionalnosti mogu prirodno uklopiti.

Verzija dokumenta: 12.25.1
Datum: 12. prosinca 2025.
Status: Potpuno obrazloženje arhitekture (sve komponente platforme)
Ciljani opseg: Beta s 10–20 korisnika, maksimalno 100 korisnika
Status arhitekture: Živi dokument – podložan reviziji na temelju iskustava iz implementacije
Kontakt: AdvanDEB razvojni tim