AdvanDEB Platforma

Potpuno obrazloženje arhitekture
Sve komponente platforme i integracija

Verzija 12.24.2 - Revizija arhitekture prosinac 2024.

AdvanDEB Razvojni tim

17. prosinca 2024.

Abstract

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

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

Contents

1 REVIZIJA ARHITEKTURE - Prosinac 2024. 3
- 1.1 Revidirane uloge komponenti . . . 3
- 1.2 Ključne promjene u odnosu na prethodni model . . . 3
- 1.3 Prednosti revidirane arhitekture . . . 4
- 1.4 Utjecaj na ovaj dokument . . . 4
- 1.5 Integracija u novom modelu . . . 4

2 Izvršni sažetak 5
- 2.1 Svrha i opseg . . . 5
- 2.2 Znanstveni kontekst: tDEB projekt . . . 5
- 2.3 Misija i zahtjevi platforme . . . 6
- 2.4 Arhitektonska filozofija . . . 7
- 2.5 Svrha dokumenta . . . 7
- 2.6 Struktura dokumenta . . . 8

3 Razmatranja skale i projektna ograničenja 8
- 3.1 Beta platforma: 10-20 korisnika, maksimalno 100 korisnika . . . 8

4 Sistemska arhitektura: Fundamentalni izbori 9
- 4.1 Arhitektura tri komponente (Revizirano prosinac 2024.) . . . 9
- 4.2 Arhitektura dijeljene baze podataka . . . 10
- 4.3 Paket advandeb-shared-utils . . . 11

5 Arhitektura autentifikacije 12
- 5.1 Google OAuth 2.0 kao primarna autentifikacija . . . 12

6 Model uloga: Dozvole temeljene na mogućnostima 13
- 6.1 Tri osnovne uloge: Administrator, Kurator, Explorator . . . 13
- 6.2 Tri mogućnosti . . . 14
- 6.3 Zašto je Knowledge Explorator važan . . . 15

7 Dizajn baze podataka: MongoDB kao temelj 16
- 7.1 Zašto MongoDB umjesto relacionih baza podataka . . . 16
- 7.2 Dizajn kolekcija i modeliranje podataka . . . 18

8 Sigurnosna arhitektura 20
- 8.1 Obrana u dubinu kroz višestruke metode autentifikacije . . . 20
- 8.2 Ograničavanje stope za prevenciju zloupotrebe . . . 20
- 8.3 Sveobuhvatno revizijsko zapisivanje . . . 21

9 Skalabilnost i performanse 22
- 9.1 Horizontalno skaliranje s MongoDB sharding-om . . . 22
- 9.2 Cachiranje za učestale upite . . . 23
- 9.3 Asinkrona obrada za dugotrajne operacije . . . 24

10 Deployment i Operations 25
- 10.1 Containerizacija s Docker-om . . . 25
- 10.2 Monitoring i observability . . . 26

10.3 Backup i disaster recovery 27

11 Arhitektura Knowledge Buildera (nastavak) 27
11.1 Tijek rada modeliranja temeljen na scenarijima 28
11.2 Sastavljanje i predstavljanje modela 29
11.3 Integracija s Knowledge Builderom 30

12 Arhitektura integracije i ugovori o podacima 30
12.1 API verzioniranje i ugovori 30
12.2 Dijeljenje sheme podataka 31
12.3 Strategija ID reference 31

13 Model podataka i odluke o pohrani 32
13.1 Struktura MongoDB kolekcija 32
13.2 Indeksna strategija 33

14 Buduća evolucija 34
14.1 Što inicijalna implementacija namjerno isključuje 34

15 Zaključak 35
15.1 Sveobuhvatna arhitektura platforme 35
15.2 Sintetizirani arhitektonski principi 35
15.3 Kriteriji uspjeha za arhitekturu 36
15.4 Arhitektura kao živi dizajn 36
15.5 Buduća arhitektonska poboljšanja 37

1 REVIZIJA ARHITEKTURE - Prosinac 2024.

Važno: Model arhitekture ažuriran

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

1.1 Revidirane uloge komponenti

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

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

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

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

- Python paket s robusnim operacijama znanja
- Ekstrakciju činjenica, stilizirane činjenice, ingestiju dokumenata, izgradnju grafa
- Nema korisničko sučelje - čista backend logika
- Uvezen i upotrijebljen od backend-a Modeling Assistanta
- Može se koristiti neovisno za batch procesiranje

advandeb-MCP je **Tool Server** za LLM agent tijekove rada. Pruža:

- Model Context Protocol (MCP) server u Rust-u
- Izlaže alate platforme (i KB i MA operacije) putem MCP-a
- Interni servis (bez autentifikacije) za značajke pokretane agentima
- Koristi ga Modeling Assistant za razgovor i inteligentnu ekstrakciju
- Omotava operacije Knowledge Buildera kao MCP alate

1.2 Ključne promjene u odnosu na prethodni model

Prethodni model:

- Knowledge Builder: Samostalna FastAPI + Vue aplikacija
- Modeling Assistant: Odvojena FastAPI + Vue aplikacija
- Obje su imale vlastita korisnička sučelja i backend-e

Novi model:

- Modeling Assistant: Glavni GUI s jednim frontend-om i backend-om
- Knowledge Builder: Toolkit paket (bez UI)
- MCP: Tool server koji izlaže operacije za LLM agente

1.3 Prednosti revidirane arhitekture

Jednostavnije korisničko iskustvo:

- Jedna aplikacija za učenje i korištenje
- Jedinstvena prijava s dosljednim sučeljem
- Značajke prikazane/skrivene na temelju uloge korisnika
- Nema zabune o kojoj komponenti pristupiti

Čistija arhitektura:

- Jasna razdvojenost: GUI vs toolkit vs servisi agenata
- Knowledge Builder postaje više ponovnoupotrebljiv
- Lakše razumijevanje odgovornosti komponenti
- Bolje usklađivanje sa stvarnom implementacijom

Lakši razvoj i postavljanje:

- Jedan frontend codebase za održavanje
- KB toolkit može biti verzioniran i testiran neovisno
- Jedna web aplikacija za postavljanje (MA)
- MCP razvijan paralelno bez utjecaja na GUI

1.4 Utjecaj na ovaj dokument

Kroz ovaj dokument obrazloženja, reference na "Knowledge Builder" i "Modeling Assistant" trebaju se razumjeti u kontekstu ove revidirane arhitekture:

- Kada se raspravlja o korisničkim sučeljima, autentifikaciji i web značajkama: odnosi se na **Modeling Assistant GUI**
- Kada se raspravlja o operacijama znanja, ekstrakciji činjenica i izgradnji grafa: odnosi se na **Knowledge Builder toolkit**
- Kada se raspravlja o tijekovima rada agenata i LLM integraciji: odnosi se na **MCP server**
- Kada se raspravlja o integraciji: MA uvozi KB kao paket, MA poziva MCP za agente

Fundamentalne arhitektonske odluke (autentifikacija, model uloga, dizajn baze podataka, sigurnost) ostaju valjane i sada su prvenstveno implementirane u Modeling Assistantu, koji služi kao glavno sučelje platforme.

1.5 Integracija u novom modelu

Odnosi komponenti:

- **Korisnici** → autentificiraju se s **Modeling Assistant GUI**
- **MA Backend** → uvozi **Knowledge Builder** kao Python paket
- **MA Backend** → poziva **MCP Server** za značajke agenata
- **MCP Server** → omotava **KB operacije** kao MCP alate
- Sve komponente dijele **MongoDB** bazu podataka

Primjer toka podataka - Upload dokumenta:

1. Korisnik se prijavljuje u Modeling Assistant GUI
2. Korisnik uploada dokument putem web sučelja
3. MA backend: `from advandeb_kb import ingest_document`
4. KB paket procesira i pohranjuje u MongoDB
5. MA vraća uspjeh frontend-u

Primjer toka podataka - AI Chat:

1. Korisnik upisuje pitanje u MA chat sučelje
2. MA šalje zahtjev MCP serveru
3. MCP izvršava alate (koristeći KB funkcije) i poziva Ollamu
4. MCP vraća formatirani odgovor MA-u
5. MA prikazuje odgovor korisniku

Za potpune detalje o revidiranoj arhitekturi, pogledajte `ARCHITECTURE-REVISION.md` u dokumentacijskom repozitoriju.

2 Izvršni sažetak

2.1 Svrha i opseg

Ovaj dokument pruža arhitektonsko obrazloženje iza izbora implementacije. Svaki važan arhitektonski izbor se ispituje kroz više leća: problem koji rješava, alternative koje su razmatrane, kompromise koji su napravljeni i implikacije za evoluciju platforme.

Sveobuhvatno pokrivanje: Ovaj dokument pokriva sve glavne komponente platforme:

- **Autentifikacija i upravljanje korisnicima**: Google OAuth, JWT tokeni, model mogućnosti, API ključevi
- **Knowledge Builder**: Asinhrona obrada dokumenata, ekstrakcija znanja AI agentima, RAG integracija
- **Modeling Assistant**: Konverzacijsko sučelje za razvoj modela, organizacija temeljena na scenarijima
- **Integracija arhitektura**: Spajanje komponenti, ugovori o podacima, verzioniranje API-ja
- **Model podataka**: MongoDB dizajn, struktura kolekcija, indeksne strategije

2.2 Znanstveni kontekst: tDEB projekt

Osnovni izazov u bioenergetici: Teorije metaboličke organizacije (kao što je Dynamic Energy Budget teorija - DEB) pružaju moćan okvir za razumijevanje kako organizam koriste energiju. Međutim, primjena ovih teorija preko raznolikih životnih oblika otkriva fundamentalne znanstvene izazove: energetski budžeti su raznoliki—vrste se razlikuju dramatično u načinima kako stječu, pohranjuju i koriste energiju. Trenutne teorije često ne uspijevaju obuhvatiti ovu raznolikost. Reproduktivni procesi su slabo shvaćeni—pravila energetske alokacije tijekom reprodukcije ostaju teorijski nejasna. Domene primjenjivosti su nedefinirane—nejasno je kada je DEB pristup valjan i kada zahtijeva modifikacije.

Teorijske poveznice nedostaju—odnosi između DEB-a i drugih metaboličkih teorija su nepotpuno istraženi.

tDEB pristup: Transport-based Dynamic Energy Budget (tDEB) teorija adresira ove izazove modelirajući metabolizam putem transportnih mreža—eksplicitno predstavljajući kako energija i materijali teku kroz organizam. Ovaj pristup prirodnije obuhvaća raznolikost života jer transportne mreže mogu biti konfigurirane da predstavljaju različite organizacijske planove. Međutim, tDEB generira vlastite istraživačke pitanje: je li tDEB konzistentan sa znanim biološkim principima? Što je točno pravila energetske uporabe tijekom reprodukcije? Koliko je tDEB općenita—vrijedi li preko svih vrsta ili ima ograničene domene? Kako se tDEB odnosi na alternativne teorije?

Ciljevi AdvanDEB projekta: Projekt adresira ova kritična pitanja: (i) verificiranje i osiguravanje konzistentnosti s poznatim biološkim principima, (ii) definiranje jasnih pravila za korištenje energije tijekom reprodukcije, (iii) testiranje općenitosti i definiranje tDEB primjenjivosti, (iv) analiziranje odnosa između tDEB-a i drugih teorija, i (v) razvijanje formalnih skupova pretpostavki i ujedinjene terminologije.

Novi istraživački pristup: AdvanDEB će koristiti postojeće resurse poput AmP baze podataka DEB zajednice koja sadrži 4,500 vrsta, koristeći i konvencionalne metode i AI-asitirane tehnike za identificiranje domena primjenjivosti za tDEB, istraživanje općenitosti unutar i preko filogenetskih grupa, i formalizaciju tDEB teorije. Dodatno, AdvanDEB može ujediniti DEB teoriju s Metaboličkom teorijom ekologije, stvarajući potencijal za paradigmske promjene u bioenergetici i otvarajući nove istraživačke puteve.

Uloga platforme u istraživanju: AdvanDEB Platforma omogućava ovu znanstvenu misiju kroz nove pristupe pregledu literature i izgradnji baze znanja. Platforma mora podržavati: sistematsku ekstrakciju bioloških činjenica iz znanstvene literature, organizaciju heterogenog znanja o organizmima preko filogenetskih grupa, povezivanje znanja s parametrima i pretpostavkama modeliranja, AI-asistiranu identifikaciju obrazaca i općenitosti preko vrsta, i suradničko kuriranje od strane domenskih stručnjaka s rigoroznom sljedivošću.

Arhitektura platforme—osobito njezin tri-slojni model znanja (Činjenice → Stilizirane činjenice → Grafovi znanja), AI agent framework za obradu dokumenata, i MongoDB-bazirana pohrana grafa—direktno podržava ove istraživačke zahtjeve. Arhitektonske odluke prioritiziraju znanstvenu strogost, sljedivost znanja i podršku za istraživačku analizu nad konvencionalnim metrikama softverskog inženjerstva.

2.3 Misija i zahtjevi platforme

AdvanDEB Platforma postoji na sjecištu znanstvenog upravljanja znanjem i računalnog modeliranja. Njena arhitektura mora služiti tri fundamentalna zahtjeva: skalabilnu pohranu i upitivanje umreženih grafova znanja koji obuhvaćaju tisuće vrsta i bioloških odnosa, rigoroznu odgovornost i sljedivost koju zahtijeva znanstveno istraživanje gdje svaka činjenica mora biti pripisiva izvorima i kuratorima, i fleksibilnu integraciju između upravljanja znanjem i tijekova rada modeliranja koja dopušta istraživačima da se fluidno kreću između pregleda literature, ekstrakcije znanja i razvoja modela.

Izbor MongoDB-a kao temeljne baze podataka odražava ove zahtjeve skalabilnosti grafa znanja, dok arhitektura autentifikacije, modeliranja i integracije grade na ovom temelju kako bi podržale suradnički znanstveni proces.

2.4 Arhitektonska filozofija

Arhitektura je vođena četiri temeljna principa koji su proizašli iz pažljive analize osnovne misije platforme. Prvo, znanstveni integritet ima prednost nad prikladnošću—svaki dio znanja mora biti sljediv do svog doprinositelja, i svaka akcija mora ostaviti revizijski trag. Ovaj princip se očituje u zahtjevima autentifikacije, sustavima atribucije i nepromjenjivom zapisivanju.

Drugo, sustav favorizira suradnju nad kontrolom. Umjesto stvaranja rigidnih hijerarhija koje prisiljavaju korisnike u unaprijed definirane kutije, model dozvola temeljen na mogućnostima dopušta korisnicima da rastu svoj pristup kako njihove potrebe evoluiraju. Istraživač može početi kao osnovni kurator, kasnije zatražiti pristup agentima za masovnu ekstrakciju, i na kraju zaraditi status revizora kroz dokazanu ekspertizu. Ovaj organski obrazac rasta odražava kako se stvarne istraživačke suradnje razvijaju.

Treće, jednostavnost proizlazi kroz integraciju umjesto razdvajanja. Dok se platforma sastoji od više komponenti (Knowledge Builder i Modeling Assistant), one dijele infrastrukturu autentifikacije, korisničke modele i konvencije podataka. Korisnici doživljavaju jedinstvenu platformu, ne skup odvojenih alata. Ova integracija smanjuje trenje dok održava koristi neovisnosti komponenti.

Četvrto, sigurnost je implementirana po defaultu, ne konfiguracijom. Korisnici ne mogu oslabiti zahtjeve autentifikacije, administratori ne mogu onemogućiti revizijsko zapisivanje, i dozvole su eksplicitne umjesto implicitne. Sustav je dizajniran da bude siguran čak i kada ga upravljaju ne-sigurnosni-stručnjaci, jer većina znanstvenih institucija nedostaje posvećenih sigurnosnih timova.

Napomena o stabilnosti arhitekture: Dok ovi principi vode sve projektne odluke, specifične implementacije dokumentirane ovdje nisu fiksne. Kako AdvanDEB projekt napreduje kroz razvoj i postavljanje, praktično iskustvo može otkriti bolje pristupe, razotkriti nepredviđena ograničenja, ili demonstrirati da su određene pretpostavke bile netočne. Ovaj dokument bilježi naše trenutno arhitektonsko razmišljanje i bit će ažuriran kako bi odražavao značajne promjene kako platforma evoluira.

2.5 Svrha dokumenta

Ovaj dokument odgovara na "zašto ovi arhitektonski obrasci" i "koje alternative su odbačene." Podržava arhitektonske preglede, evaluacije tehnologija i buduće projektne odluke pružanjem detaljnog obrazloženja za svaki važan izbor.

Ključne arhitektonske odluke dokumentirane uključuju: model tri uloge s mogućnostima, arhitekturu dijeljene baze podataka, pristup JWT tokena, izbor MongoDB-a za skalabilnost grafa znanja, i principe stateless dizajna. Svaki izbor uključuje kompromise prikladne za beta skalu (10-20 korisnika) s kapacitetom rasta na otprilike 100 korisnika.

Evolucija arhitekture: Odluke dokumentirane ovdje odražavaju naše trenutno razumijevanje zahtjeva i ograničenja. Kako AdvanDEB projekt napreduje kroz faze implementacije, anticipiramo da će praktično iskustvo otkriti prilike za optimizaciju, razotkriti skrivena ograničenja ili sugerirati alternativne pristupe. Ovaj dokument će biti ažuriran kako bi odražavao značajne arhitektonske promjene, s poviješću verzija koja prati evoluciju projektnog razmišljanja kako platforma sazrijeva.

2.6 Struktura dokumenta

Svaki odjeljak slijedi dosljedan analitički obrazac. Prvo jasno navodimo projektnu odluku, zatim objašnjavamo obrazloženje iza nje u narativnom obliku. Alternative pristupe raspravljamo s iskrenom procjenom njihovih prednosti i nedostataka—odbačene alternative često su imale istinsku vrijednost ali nisu odgovarale našim specifičnim ograničenjima. Kompromisi se ispituju eksplicitno jer svaka arhitektonska odluka uključuje žrtvovanje nečega da bi se dobilo nešto drugo. Konačno, raspravljamo o zavisnostima i implikacijama, pokazujući kako svaka odluka utječe na druge dijelove sustava.

3 Razmatranja skale i projektna ograničenja

3.1 Beta platforma: 10-20 korisnika, maksimalno 100 korisnika

Fundamentalni pokretač arhitektonskih odluka je očekivana skala korisnika. AdvanDEB Platforma cilja 10-20 korisnika tijekom inicijalnog beta postavljanja, s maksimalnim očekivanim rastom na otprilike 100 korisnika. Ova skala duboko utječe na projektne izbore na načine koji se razlikuju od platformi koje ciljaju tisuće ili milijune korisnika.

Implikacije za arhitekturu:

Dizajn baze podataka - Pri 100 korisnika, cijela baza korisnika ugodno stane u memoriju. MongoDB brige o performansama koje dominiraju na skali (strategije sharding-a, topologije replica set-a, optimizacija indeksa za milijune dokumenata) su u biti irelevantne. Jednostavni MongoDB s jednom instancom i osnovnim indeksiranjem pruža vremena upita ispod milisekunde. Ovo nam dopušta prioritiziranje fleksibilnosti sheme nad optimizacijom upita.

Performanse autentifikacije - S 100 korisnika koji čine možda 10-50 zahtjeva po minuti svaki (1,000-5,000 zahtjeva/minuta ukupno), overhead autentifikacije je zanemariv. Pristup JWT validacije pruža izvrsne performanse, ali iskreno, autentifikacija temeljena na sesiji bi također radila dobro na ovoj skali. Biramo JWT-e prvenstveno za arhitektonsku čistoću i buduću sigurnost, ne performansnu nužnost.

Ograničavanje stope - Ograničenja stope postoje prvenstveno kao sigurnosne zaštite protiv buggynih skripti, ne zlonamjernih zloupotreba. S 10-20 povjerljivih akademskih korisnika, profil rizika je slučajne pogreške umjesto namjernih napada. Velikodušna ograničenja (50-500 zahtjeva/minuta po korisniku) vjerojatno neće biti pogođena tijekom normalne operacije.

Operativna jednostavnost - Male baze korisnika dopuštaju jednostavnije operacije. Ručno odobravanje zahtjeva za mogućnostima je izvedivo—administratori koji pregledavaju 1-2 zahtjeva tjedno je upravljivio, dok bi 100 zahtjeva dnevno zahtijevali automatizaciju. Analiza dnevnika revizije može koristiti osnovne MongoDB upite umjesto zahtijevanja specijalizirane analitičke infrastrukture. Procedure sigurnosnog kopiranja i oporavka mogu biti jednostavne bez složene orkestracije oporavka od katastrofe.

Fokus na suradnju - Na ovoj skali, svaki korisnik može potencijalno poznavati svakog drugog korisnika. Platforma može poticati istinsku istraživačku suradnju umjesto anonimnog masovnog doprinosa. Značajke poput dodjeljivanja revizora mogu razmatrati individualnu ekspertizu umjesto potpuno oslanjanja na algoritemsko sparivanje.

Ove karakteristike skale dopuštaju nam odgađanje određenih arhitektonskih složenosti koje veće platforme zahtijevaju odmah: balansiranje opterećenja (jedan server je dovoljan inicijalno), replikacija baze podataka (sigurnosna kopija je dovoljna inicijalno), slojevi

cacheiranja (performanse baze podataka su adekvatne), distribuirano praćenje (jednostavno zapisivanje radi), i napredno praćenje (osnovne provjere zdravlja su dovoljne). Gradimo ove mogućnosti kao modularne dodatke kada rast to zahtijeva, umjesto preinženjeringa od početka.

4 Sistemska arhitektura: Fundamentalni izbori

4.1 Arhitektura tri komponente (Revizirano prosinac 2024.)

Projektna odluka: Modeling Assistant kao glavni GUI, Knowledge Builder kao Toolkit, MCP kao Tool Server

Arhitektura platforme je revidirana kako bi pojasnila uloge komponenti: Modeling Assistant služi kao glavni GUI platforme i jedinstvena korisnička ulazna točka, Knowledge Builder pruža toolkit operacija bez UI-ja, i MCP izlaže alate platforme za LLM agent tijekove rada.

Arhitektonska revizija proizašla je iz prepoznavanja da korisnici trebaju **jedno jedinstveno sučelje** za sve značajke platforme, ne odvojene aplikacije za izgradnju znanja naspram modeliranja. Revidirani model pozicionira:

Modeling Assistant kao glavni GUI platforme: Svi korisnici autentificiraju se kroz MA i pristupaju svim značajkama—chat, istraživanje znanja, ingestiju dokumenata, scenarije modeliranja—kroz jedno sučelje. Kontrola pristupa temeljena na ulogama određuje koje značajke svaki korisnik vidi. Korisnici s ulogama kuratora pristupaju značajkama izgradnje znanja; korisnici s mogućnostima modeliranja pristupaju stvaranju scenarija. Ovo eliminira zabunu "koju aplikaciju koristim" i pruža besprijekorne tijekove rada gdje korisnici se fluidno kreću između istraživanja znanja i modeliranja.

Knowledge Builder kao Toolkit: Robusne operacije znanja—algoritmi ekstrakcije činjenica, generiranje stiliziranih činjenica, cjevovodi ingestije dokumenata, izgradnja grafa znanja—su dovoljno složeni da opravdavaju odvojeni repozitorij, ali ne trebaju vlastito korisničko sučelje. Strukturiranje KB-a kao Python paketa dopušta MA da uvozi i koristi njegove funkcije dok također omogućava samostalno batch procesiranje. Toolkit može biti verzioniran, testiran i evoluiran neovisno dok je integriran u MA korisničko sučelje.

MCP kao Tool Server: Izlaganje operacija platforme kao MCP alata omogućava LLM agent tijekove rada—inteligentnu ekstrakciju činjenica, konverzacijske upite znanja, analizu scenarija. MCP server temeljen na Rust-u operira kao interni servis (bez autentifikacije) koji MA poziva za značajke pokretane agentima. Omotava KB operacije kao alate i pruža jedinstvenu Ollama integraciju za LLM inferiranje. Ova razdvojenost dopušta računalno intenzivnim operacijama agenata da skaliraju neovisno od web sloja.

Prethodni model—s odvojenim KB i MA aplikacijama, svaka sa vlastitim UI-jem—stvarao je nepotrebnu složenost. Korisnici su morali razumjeti kojoj komponenti pristupiti za koji zadatak. Dijeljene značajke (autentifikacija, pregledavanje znanja) postojale su u oba UI-ja s potencijalnim nekonzistentnostima. Razvojni timovi duplicirali su frontend kod i obrasce dizajna. Revidirani model eliminira ovu duplikaciju: jedan GUI (MA), jedan toolkit (KB), jedan servis agenata (MCP).

Obrazac integracije: Revidirana arhitektura koristi jednostavnu, direktnu integraciju:

- MA backend uvozi KB paket: `from advandeb_kb import ingest_document`

- MA backend poziva MCP putem HTTP: `POST localhost:3000/mcp/tools/execute`
- MCP koristi KB funkcije: `from advandeb_kb import search_knowledge`
- Sve komponente dijele MongoDB: jednostavna povezivost, nema sinkronizacije

MA backend uvozi KB paket: `from advandeb_kb import ingest_document`

MA backend poziva MCP server putem internog HTTP/WebSocket-a za značajke agenata

MCP omotava KB operacije kao alate za LLM agente

Sve komponente dijele MongoDB bazu podataka radi konzistentnosti podataka

Ovaj pristup održava prednosti razdvajanja komponenti (KB toolkit je ponovnoupotrebljiv, MCP skalira neovisno, jasne granice modula) dok pruža jedinstveno korisničko iskustvo koje znanstveni istraživači trebaju. Overhead integracije je minimalan: uvozi Python paketa su native operacije, a pozivi internim servisima imaju zanemariv latenciju.

Revidirana arhitektura eksplicitno optimizira za stvarne korisničke tijekove rada: istraživači ne misle "sada radim izgradnju znanja, sada radim modeliranje"—oni fluidno istražuju znanje, ekstrahiraju činjenice, grade modele i iteriraju. Arhitektura platforme sada odgovara ovom mentalnom modelu.

4.2 Arhitektura dijeljene baze podataka

Projektna odluka: Jedna MongoDB instanca 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 dnevnike revizije.

Arhitektura dijeljene baze podataka odražava izbor MongoDB-a kao temeljne pohrane podataka platforme, odabrane prvenstveno za zahtjeve skalabilnosti umreženog grafa znanja i baze znanja. Jednom kada je MongoDB baza podataka za upravljanje znanjem (osnovna funkcija platforme), korištenje iste za upravljanje korisnicima, podatke modeliranja i brige koje prelaze komponente pruža snažnu konzistentnost i operativnu jednostavnost.

Dijeljena baza podataka pruža trenutnu konzistentnost preko komponenti. Kada korisnik stvori činjenicu u KB-u, ona je odmah upitiva od strane MA za izgradnju scenarija. Kada scenarij referencira ID činjenice, MA upituje istu bazu podataka koju koristi KB, eliminirajući kašnjenja sinkronizacije. Kada se uloga korisnika promijeni, obje komponente vide nove dozvole odmah. Ova snažna konzistentnost je moguća jer MongoDB ACID transakcije obuhvaćaju više kolekcija unutar baze podataka—ažuriranje korisničke uloge, unos dnevnika revizije i poništavanje cache-a dozvola događaju se atomski.

Alternative arhitekture s odvojenim bazama podataka po komponenti zahtijevale bi složenu sinkronizaciju. Ako bi KB i MA svaki imali MongoDB instance, stvaranje činjenice u KB-u bi trebalo event streaming za replikaciju u MA. Tijekom kašnjenja replikacije, MA korisnici ne bi vidjeli nove činjenice. Potrebno bi bilo rješavanje konflikata ako obje komponente modificiraju isti entitet znanja. Promjene korisničkih dozvola zahtijevale bi obrasce eventualne konzistentnosti s njihovom inherentnom složenošću.

Centralizirani servis autentifikacije s odvojenim bazama znanja predstavlja drugu alternativu. Svaka komponenta bi imala vlastiti MongoDB za domenske podatke, pozivajući dijeljeni auth API za validaciju korisnika. Ovo održava neovisnost komponenti ali fragmentira graf znanja—MA ne može direktno upitati KB-ove kolekcije znanja; mora koristiti HTTP API-je s overhead-om latencije. Problematičnije, upiti koji prelaze granice i obuhvaćaju

podatke znanja i modeliranja (nađi scenarije koji koriste činjenice koje je stvorio korisnik X) postaju skupi višestruki API pozivi.

Pristup dijeljene baze podataka optimizira za primarni slučaj uporabe platforme: integrirano upravljanje znanjem i modeliranje. Upiti koji obuhvaćaju komponente ("pokaži mi scenarije izgrađene s ovim činjenicama," "koje činjenice podržavaju ovaj parametar modela") izvršavaju se efikasno unutar jedne baze podataka. Granice transakcija uključuju i operacije znanja i korisnika—stvaranje činjenice i zapisivanje unosa revizije događaju se atomski. Sigurnosno kopiranje i oporavak su jednostavniji s jednom bazom podataka koja sadrži cijelo stanje platforme.

Ova odluka eksplicitno spaja KB i MA na sloju podataka. Dijele evoluciju sheme, moraju koordinirati migracije baze podataka i dijele modove neuspjeha (nedostupnost baze podataka utječe na oboje). Međutim, ovo spajanje već postoji logički—MA ovisi fundamentalno o KB-ovom znanju za modeliranje. Činjenje spajanja eksplicitnim kroz dijeljenu bazu podataka pojednostavljuje umjesto da komplicira: komponente su usko integrirane po dizajnu, ne neovisno postavljivi mikroservisi.

Odvojene kolekcije unutar dijeljene baze podataka pružaju logičku razdvojenost: KB prvenstveno radi s facts, stylized_facts, documents, knowledge_graphs; MA prvenstveno radi sa scenarios, models, results. Kolekcije users, api_keys i audit_logs služe objema. Ova organizacija na razini kolekcija omogućava buduće sharding baze podataka ako je potrebno—kolekcije znanja mogle bi se shardati po taksonomskoj grupi (kralježnjaci vs. beskralježnjaci) ili tipu organizma (vodeni vs. kopneni) dok kolekcije korisnika ostaju neshardane.

4.3 Paket advandeb-shared-utils

Projektna odluka: Python paket koji sadrži dijeljenu logiku autentifikacije

Stvoriti verzionirani Python paket koji oba KB i MA backend servisa uvoze za svu autentifikaciju, autorizaciju i funkcionalnost revizije.

Paket dijeljenih uslužnih programa utjelovljuje DRY (Don't Repeat Yourself) princip primijenjen na razini arhitekture. Logika generiranja JWT tokena nije trivijalna—uključuje kriptografsko potpisivanje, formatiranje zahtjeva, rukovanje isticanjem i sigurnosna razmatranja. Pisanje ovog koda dvaput (jednom za KB, jednom za MA) udvostručuje teret testiranja, udvostručuje priliku za bugove i udvostručuje trošak održavanja kada se otkriju sigurnosne ranjivosti. Podmuklije, duplirani kod ima tendenciju divergiranja tijekom vremena kako developeri prave promjene na jednoj lokaciji ali zaborave drugu, dovodeći do suptilnih bihevioralnih razlika između komponenti.

Paket sadrži sve što mora biti identično preko komponenti: generiranje i validaciju JWT tokena, Google OAuth integraciju klijenta, Pydantic modele za korisnike i uloge, funkcije provjere dozvola (has_base_role, has_capability), hashiranje i validaciju API ključeva, formattere revizijskog zapisivanja i MongoDB uslužne programe za povezivanje. Kod koji se može legitimno razlikovati između komponenti—poslovna logika, handleri API ruta, modeli specifični za komponentu—namjerno ostaje izvan paketa.

Ova granica je ključna. Dijeljeni kod stvara spajanje, što je prihvatljivo za brige koje prelaze granice poput autentifikacije ali štetno za poslovnu logiku. Ako KB-ova obrada dokumenata treba specijalne mogućnosti, one pripadaju u KB, ne u shared-utils gdje bi stvarale nepotrebne ovisnosti za MA. Paket pruža primitive (provjeri ima li korisnik

mogućnost X) ne politike (kurator može uploadati dokumente). Politike su sastavljene od primitiva u kodu specifičnom za komponentu.

Verzioniranje paketa dopušta kontroliranu evoluciju. KB i MA mogu privremeno koristiti različite verzije paketa tijekom migracija, izbjegavajući problem big-bang nadogradnje. Semantičko verzioniranje komunicira kompatibilnost—patch verzije popravljaju bugove, minor verzije dodaju značajke unatrag-kompatibilno, major verzije signaliziraju breaking promjene. CI/CD cjevovodi mogu automatski testirati obje komponente protiv ažuriranja paketa prije puštanja.

Alternativa dupliciranog koda inicijalno se čini jednostavnijom—nema paketa za objavljivanje, nema koordinacije verzija, svaki tim kontrolira vlastiti kod. Ova jednostavnost je iluzorna. Kada sigurnosna revizija otkrije ranjivost u rukovanje JWT-a, popravci moraju biti pažljivo replicirani preko codebasa. Kada se dodaju nove mogućnosti, logika provjere dozvola mora biti ažurirana u lockstep-u. Noćna mora debugginga kada se komponente ponašaju malo drugačije zbog drifta koda daleko nadmašuje overhead održavanja dijeljenog paketa.

5 Arhitektura autentifikacije

5.1 Google OAuth 2.0 kao primarna autentifikacija

Projektna odluka: Koristiti Google OAuth 2.0 za autentifikaciju web UI-ja

Platforma koristi Google OAuth 2.0 za autentifikaciju korisnika umjesto implementacije vlastitog sustava korisničko ime/lozinka ili korištenja drugih OAuth pružatelja (GitHub, Microsoft).

Izbor Google OAuth-a odražava karakteristike ciljane populacije korisnika i institucionalna ograničenja akademskog istraživanja. Virtualno svako sveučilište i istraživačka institucija standardizira na Google Workspace (ranije G Suite) za email i suradničke servise. Istraživači već imaju Google račune kroz svoje institucije, često s višefaktorskom autentifikacijom koju institucija provodi. Korištenje Google OAuth-a eliminira potrebu za registracijom novog računa—korisnici kliknu "Prijavi se s Googleom" i autentificirani su odmah.

Pristup autentifikacije delegiranom Googleu prebacuje sigurnosni teret na tim s resursima. Google ulaže značajne inženjerske resurse u detekciju neobičnih prijava, zaštitu od automatiziranih napada, upravljanje oporavkom računa i implementaciju naprednih autentifikacijskih metoda poput Security Keys. Mali razvojni tim koji gradi AdvanDEB ne bi mogao odgovoriti kompetentno u svakom od ovih područja. Korištenje Google OAuth-a daje AdvanDEB korisnicima koristi od Googleovih sigurnosnih ulaganja bez izravnih troškova.

Alternativa implementacije vlastitog sustava lozinki zahtijeva netrivijalnu sigurnosnu infrastrukturu. Hashiranje lozinki mora koristiti moderne algoritme (bcrypt, scrypt ili Argon2) s prikladnim cost faktorima. Resetiranje lozinki zahtijeva sigurno generiranje tokena i email isporuku. Sigurnosne provjere—prevencija brute force-a, blokiranje računa nakon neuspjelih pokušaja, validacija snage lozinke—moraju biti implementirane pažljivo. Svaka od ovih značajki predstavlja potencijalni sigurnosni rizik ako je pogrešno implementirana. OAuth delegiranje eliminira cijelu ovu klasu ranjivosti iz AdvanDEB codebasa.

Izbor Google specifično nad drugim OAuth pružateljima (GitHub, Microsoft Azure AD, institucionalnim SAML) odražava penetraciju Google Workspace-a u akademiji. Dok mnogi istraživači također imaju GitHub račune, GitHub je prvenstveno usluga developera;

neki znanstvenici ne održavaju aktivne GitHub račune. Microsoft Azure AD postoji na nekim institucijama ali penetracija je manja od Googlea u europskoj akademiji. Institucionalni SAML pružatelji variraju po instituciji, zahtijevajući AdvanDEB da održava konfiguracije za svaku partnersku instituciju—operativni teret nesrazmeran beta skali.

Razmatranje privatnosti utječe na ovaj izbor. Korištenje Google OAuth-a znači da Google može vidjeti kada korisnici pristupaju AdvanDEB-u (kroz OAuth redirecte) ali ne mogu vidjeti što korisnici rade unutar platforme. Za znanstvenu platformu koja rukuje neklasificiranim istraživačkim podacima, ovaj profil privatnosti je prihvatljiv. Institucije koje rukuju klasificiranim ili HIPAA-reguliranim podacima zahtijevale bi drugačija rješenja autentifikacije, ali to nisu AdvanDEB ciljani korisnici.

Tehnička implementacija slijedi OAuth 2.0 authorization code flow s PKCE (Proof Key for Code Exchange) dodano za dodatnu sigurnost. Kada neautentificirani korisnik pristupi AdvanDEB-u, frontend redirecta na Google OAuth krajnju točku s client_id-om registriranim za AdvanDEB. Korisnik se autentificira s Googleom (ako već nisu prijavljeni) i odobravaju AdvanDEB pristup njihovom osnovnom profilu (email, ime). Google redirecta natrag na AdvanDEB s authorization kodom. AdvanDEB backend razmjenjuje ovaj kod za access token direktno komunicirajući s Google tokenskim krajnjim točkama (nikad kroz frontend, sprječavajući token presretanje). Backend validira token, ekstrahira korisnički email i profile informacije, i stvara ili ažurira korisnikov zapis u MongoDB-u.

JWT token se zatim generira za korisnika i vraća frontend-u, gdje je pohranjen u localStorage. Naknadni API zahtjevi uključuju ovaj JWT token u Authorization headeru. Backend validira JWT potpis i ekstrahira user_id i mogućnosti bez pozivanja Googleu—JWT omogućava stateless autentifikaciju nakon inicijalne OAuth prijave.

Ovo dvostruko-tokensko rješenje (Google OAuth za inicijalnu autentifikaciju, JWT za naknadne zahtjeve) balansira sigurnost s performansom. OAuth tijek angažira Googleu samo tijekom prijave, ne kod svakog API zahtjeva. JWT provajdira brzu validaciju bez vanjskih poziva. Kratki rokovi isteka JWT-a (tipično 1 sat) ograničavaju prozor za zlouporabu ako je token kompromitiran.

6 Model uloga: Dozvole temeljene na mogućnostima

6.1 Tri osnovne uloge: Administrator, Kurator, Explorator

Projektna odluka: Model tri osnovne uloge umjesto višestrukih granularnih uloga

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

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

Istraživačev identitet kao kuratora je stabilan, ali njihova potreba za pristupom agentima ili mogućnostima izvoza podataka varira s vremenom. Novi korisnik ne treba pristup

agentima odmah—prvo trebaju razumjeti platformu i kurirati nešto znanja. Nakon stjecanja iskustva, mogu zatražiti pristup agentima za skaliranje svog rada. Još kasnije, ako pokažu ekspertizu, mogu zatražiti status revizora. Ova postupna progresija odražava prirodnu evoluciju tijeka rada umjesto prisiljavanja korisnika u rigidne kategorije.

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

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

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

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

6.2 Tri mogućnosti

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

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

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

legitimnost zahtjeva.

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

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

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

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

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

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

6.3 Zašto je Knowledge Explorator važan

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

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

Uloga Knowledge Exploratora odražava prepoznavanje da nisu svi vrijedni korisnici platforme doprinositelji sadržaja. Studenti koji uče o domeni, novinari koji istražuju članke, kreatori politika koji prikupljaju informacije i zainteresirani javni članovi koji

istražuju znanost možda nikad neće uploadati rad ili stvoriti činjenicu. Ako je kurator minimalna potrebna uloga, tim korisnicima je odbijen pristup ili su primorani zatražiti dozvole koje ne trebaju i neće koristiti.

Pružanje uloge samo-za-čitanje snizuje barijeru angažmana platforme. Odobrenje je jednostavno jer exploratori ne mogu oštetiti podatke ili potrošiti značajne resurse. Administrator može odobriti zahtjeve exploratora brzo, možda automatski, znajući da je rizik minimalan. Neki exploratori će na kraju postati kuratori kako stječu povjerenje i identificiraju praznine koje bi mogli ispuniti. Ova prirodna progresija od potrošača ka doprinositelju je uobičajena u online zajednicama; blokiranje toga s nepotrebnim vratima dozvola bi oštetilo rast zajednice.

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

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

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

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

7 Dizajn baze podataka: MongoDB kao temelj

7.1 Zašto MongoDB umjesto relacionih baza podataka

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

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

Izbor MongoDB-a je fundamentalno vođen osnovnom misijom platforme: upravljanje umreženim grafom znanja i skalabilnom bazom znanja za istraživanje bioenergetike. Grafovi znanja sastoje se od heterogenih čvorova (vrste, fiziološki procesi, metabolički putevi, bioenergetički parametri) s varijabilnim svojstvima i odnosima. Činjenice izvučene iz

literature imaju različite strukture ovisno o sadržaju. Stilizirane činjenice sintetiziraju varijabilan broj izvornih činjenica. Ova inherentna heterogenost čini rigidne relacione sheme problematičnima—svaki novi tip entiteta ili obrazac odnosa zahtijevao bi migracije sheme.

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

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

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

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

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

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

Operativna jednostavnost arhitekture jedne baze podataka postaje važnija kako volumeni podataka rastu. Primarni podaci platforme—graf znanja—će skalirati na gigabajte ili terabajte (tisuće radova, milijuni činjenica, složeni međusobno povezani grafovi). MongoDB stručnost, procedure sigurnosnog kopiranja, optimizacija upita i infrastruktura praćenja

razvijena za pohranu znanja automatski se primjenjuju na korisničke podatke. Dodavanje PostgreSQL-a bi podijelilo operativni fokus bez rješavanja osnovnog izazova: skalabilna pohrana umreženog grafa znanja.

Garancije konzistentnosti u MongoDB-u su se dramatično poboljšale s transakcijama više dokumenata (uvedenim u MongoDB 4.0, zrelim u 4.4). Stvaranje činjenice i povezivanje s čvorom grafa znanja može se dogoditi atomski. Dok su PostgreSQL-ove transakcije zrelije, MongoDB-ove sposobnosti su dovoljne za operacije upravljanja znanjem. Obrasci transakcija platforme (stvaranje entiteta znanja + ažuriranje odnosa) dobro pristaju MongoDB-ovom modelu transakcija.

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

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

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

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

7.2 Dizajn kolekcija i modeliranje podataka

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

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

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

Odluka pohranjivanja mogućnosti kao niza u korisničkom dokumentu zaslužuje posebno objašnjenje jer je u sukobu s relacionom normalizacijom. U trećoj normalnoj formi, odnos

mnogo-prema-mnogo (korisnici imaju mnogo mogućnosti, mogućnosti pripadaju mnogim korisnicima) bi koristio spojnu tablicu. Ovaj normalizirani pristup optimizira integritet podataka—dodavanje novog tipa mogućnosti ne zahtijeva doticanje postojećih korisničkih zapisa. Međutim, pesimizira uobičajeni slučaj provjera dozvola tijekom obrade zahtjeva.

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

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

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

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

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

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

8 Sigurnosna arhitektura

8.1 Obrana u dubinu kroz višestruke metode autentifikacije

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

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

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

Ova raznolikost pruža obranu u dubinu. Ako je OAuth privremeno nedostupan zbog Google prekida, korisnici s API ključevima mogu nastaviti programski pristup. Ako je korisnikov JWT refresh token ukraden, opozivanje ga ne poništava njihove zasebno izdane API ključeve (pod pretpostavkom da ključevi također nisu kompromititani). Ako je otkrivena ranjivost u JWT biblioteci, API ključevi pružaju fallback dok se primjenjuju zakrpe. Ne postoji jedinstvena točka neuspjeha jer višestruki neovisni putevi autentifikacije služe platformi.

Svaka metoda ima ograničenja koja druge adresiraju. OAuth ovisi o Googleovoj dostupnosti i politikama. JWT-ovi ne mogu biti odmah opozvani prije isteka. API ključevi zahtijevaju sigurnu pohranu koju mnogi korisnici loše rukuju. Kombiniranje metoda dopušta korisnicima da biraju pristup koji odgovara njihovoj sigurnosnoj poziciji i slučaju uporabe dok pruža platformi opcije fallbacka.

Implementacija dijeli logiku autorizacije preko metoda kroz advandeb-shared-utils. Bilo da se korisnik autentificira s OAuth+JWT ili API ključem, iste funkcije provjere dozvola određuju njihov pristup. Ova konzistentnost sprječava sigurnosne praznine gdje jedan put slučajno zaobilazi provjere koje drugi put provodi. Metoda autentifikacije se bilježi u dnevnicima revizije, omogućavajući analizu pokazuju li različite metode različite stope sigurnosnih incidenata.

8.2 Ograničavanje stope za prevenciju zloupotrebe

Projektna odluka: Ograničenja stope po korisniku temeljena na mogućnostima, implementirana s Redis-om

Zahtjevi su ograničeni na temelju toga tko ih je napravio i koje mogućnosti imaju, koristeći Redis za praćenje.

Ograničavanje stope služi višestrukim svrhama koje opravdavaju njegovu složenost implementacije. Prvo, sprječava slučajno uskraćivanje usluge od buggynih skripti. Python skripta istraživača s beskonačnom petljom ne bi trebala moći iscrpiti veze baze podataka ili CPU vrijeme. Drugo, ograničava namjernu zlouporabu od zlonamjernih korisnika koji pokušavaju scrapiati podatke ili preopteretiti infrastrukturu. Treće, pruža pravednu

alokaciju resursa—korisnici s višim mogućnostima (i tipično većom potrebom) dobivaju viša ograničenja dok sprječava bilo kojeg pojedinog korisnika da monopolizira resurse.

Ograničenja temeljena na mogućnostima odražavaju realnost da različiti tipovi korisnika imaju legitimno različite potrebe. Knowledge Explorator koji pregledava platformu može praviti 50 zahtjeva po minuti klikajući kroz činjenice i radove. Knowledge Curator koji aktivno doprinosi može praviti 100 zahtjeva po minuti dok uploadaju dokumente i stvaraju unose. Kurator s Analytics Access-om koji pokreće skripte izvoza podataka može legitimno trebati 500 zahtjeva po minuti. Za našu beta bazu korisnika (10-20 korisnika inicijalno), ova ograničenja su velikodušna i vjerojatno neće biti pogođena tijekom normalne operacije. Prvenstveno služe kao sigurnosne zaštite protiv slučajnih beskonačnih petlji u skriptama ili buggy automatizacije, umjesto obrane protiv zlonamjerne zloupotrebe.

Redis implementira ograničavanje stope jer provjera mora dogoditi na svakom zahtjevu s minimalnom latencijom. Pohranjivanje brojača ograničenja stope u MongoDB-u bi dodalo opterećenje baze podataka i latenciju koja poništava svrhu. Redis drži brojače u memoriji za vremena pristupa u mikrosekundama i podržava atomske operacije povećanja koje sprječavaju race uvjete u distribuiranim sustavima. Ugrađena TTL (time-to-live) značajka automatski istječe stare brojače bez ručnog čišćenja, implementirajući ograničenja stope kliznog prozora efikasno.

Specifični korišteni algoritam je token bucket, gdje svaki korisnik ima bucket koji drži tokene i svaki zahtjev konzumira token. Tokeni se ponovno pune konstantnom stopom (njihova stopa ograničenja). Ovo dopušta kratke burste iznad prosječne stope dok sprječava trajne visoke stope zahtjeva. Korisnik s 200 req/min ograničenjem može napraviti 300 zahtjeva u jednoj minuti ako su pravili malo zahtjeva u prethodnim minutama, ali ne mogu održati 300 req/min kontinuirano. Ova fleksibilnost prilagođava realistične obrasce uporabe (periodi intenzivne aktivnosti nakon kojih slijede mirni periodi) dok sprječava zlouporabu.

Alternativni pristupi poput ograničavanja temeljenog na bazi podataka (presporo), ograničavanja na razini proxy-a (ne zna o mogućnostima korisnika), ili bez ograničavanja (poziva zlouporabu) odbijeni su jer ne uspijevaju balansirati konkurentne brige performansi, pravednosti i fleksibilnosti.

8.3 Sveobuhvatno revizijsko zapisivanje

Projektna odluka: Zapisivati sva pisanja, događaje autentifikacije i promjene dozvola

Platforma zapisuje svaku operaciju koja modificira stanje ili mijenja sigurnosni položaj, pohranjujući dnevnike u kolekciju audit_logs.

Revizijsko zapisivanje služi znanstvenom integritetu, praćenju sigurnosti, regulatornoj usklađenosti i korisničkoj podršci što kolektivno opravdava njegov operativni trošak. Za znanstveni integritet, svaki doprinos znanju mora biti pripisiv njegovom kreatoru. Ako se upitna činjenica pojavi u bazi podataka, dnevnici revizije pokazuju tko ju je stvorio, kada i s koje IP adrese. Ova sljedivost gradi povjerenje—korisnici koji doprinose znanje dobre namjere nisu okaljani lošim akterima jer su doprinosi individualno sljedivi.

Praćenje sigurnosti zahtijeva zapisivanje za detekciju kompromitiranih računa koji pokazuju neobično ponašanje. Kurator koji tipično stvara 10 činjenica dnevno koji iznenada stvara 1000 po satu sugerira kompromitiranje računa. Korisnik koji pristupa platformi iz

nove zemlje zaslužuje verifikaciju. Ovi obrasci jedino proizlaze iz analiziranja zabilježenog ponašanja tijekom vremena. Praćenje u stvarnom vremenu može pokrenuti upozorenja za administratore da istražuju, dok povijesni dnevnici podržavaju forenzičku analizu nakon otkrivanja incidenta.

Zahtjevi usklađenosti od agencija za financiranje, institucionalnih odbora za pregled i regulacija zaštite podataka često nalažu revizijske tragove. GDPR zahtijeva praćenje tko je pristupio osobnim podacima i kada. Financijeri grantova žele dokaze da su istraživački podaci obrađeni prema protokolima. Buduće certifikacije (HIPAA ako se dodaju zdravstveni podaci, FedRAMP za vladiniu upotrebu) zahtijevaju sveobuhvatno zapisivanje. Izgradnja mogućnosti revizije od početka je daleko lakša nego retrofitting ih kasnije kada je certifikacija potrebna.

Korisnička podrška ima koristi od dnevnika revizije pri odgovaranju na pitanja "što se dogodilo s mojim podacima?". Korisnik prijavljuje da je njihova skica činjenice nestala—dnevnici pokazuju da su je slučajno izbrisali jučer. Korisnik ne može pristupiti značajci za koju su sigurni da su imali dozvolu—dnevnici pokazuju da su njihove mogućnosti opozvane prošli tjedan s dokumentiranim obrazloženjem. Ovi svakodnevni scenariji podrške rješavaju se brzo s dobrim dnevnicima ali postaju frustrirajuće misterije bez njih.

Odluka da se zapisuju pisanja ali ne čitanja odražava analizu troškova/koristi. Operacije pisanja su relativno rijetke i visokog utjecaja. Činjenica stvorena netočno može zavarati istraživače. API ključ generiran neoprezno može omogućiti eksfiltraciju podataka. Ove operacije opravdavaju trošak pohrane i performansi zapisivanja. Operacije čitanja su daleko češće (možda 100x više od pisanja) i nižeg utjecaja. Zapisivanje svakog čitanja bi generiralo masivne volumene dnevnika s ograničenom sigurnosnom vrijednošću—čitanje javnih podataka tipično nije sigurnosni događaj. Za našu beta implementaciju s 10-20 korisnika, volumen dnevnika ostaje upravljiv čak i sa sveobuhvatnim zapisivanjem pisanja, dopuštajući detaljnu forenzičku analizu bez briga o pohrani.

Izuzetak od zapisivanja čitanja su neuspjeli pokušaji autorizacije. Ako korisnik pokuša pristupiti nečemu za što nedostaje dozvola, taj pokušaj se zapisuje. Ovo detektira zabunu dozvola (korisnik očekuje pristup koji nema, sugerirajući UX probleme) i potencijalne napadače koji sondiraju za ranjivosti. Uspješna čitanja javnih podataka se ne zapisuju, ali neautorizirani pokušaji jesu, pružajući vidljivost sigurnosti bez utapanja u šumu.

9 Skalabilnost i performanse

9.1 Horizontalno skaliranje s MongoDB sharding-om

Projektna odluka: Dizajn za buduće sharding temeljeno na taksonomskim grupama

Dok beta implementacija koristi MongoDB s jednom instancom, arhitektura omogućava buduće sharding gdje kolekcije znanja particioniraju po taksonomskim grupama ili filogenetskim linijama.

Trenutna beta skala (10-20 korisnika, 10,000 dokumenata, 100,000 činjenica) ne zahtijeva sharding—jedna MongoDB instanca rukuje ovim opterećenjem ugodno. Međutim, arhitektonske odluke danas omogućavaju ili sprječavaju buduće skaliranje. Dizajniranje shema podataka s budućim sharding-om u vidu izbjegava kasnije bolno prepravljanje.

Ključ sharding dizajna je razumijevanje prirodnih granica particioniranja u podacima.

Za AdvanDEB, znanje prirodno grupirajte po filogenetskim linijama—kralježnjaci nasuprot beskralježnjaka, vodeni nasuprot kopneni organizmi, biljke nasuprot životinjama. Upiti rijetko prelaze ove granice: istraživač koji modelira dinamiku populacije riba tipično ne treba podatke biljaka u istoj transakciji. Ova lokalnost upita čini taksonomsko particioniranje učinkovitim.

Strategija implementiranja bi particionirala kolekcije facts, stylized_facts i knowledge_graphs po taxonomy_group polju. Shard ključ {taxonomy_group: 1, _id: 1} osigurava da sve činjenice za "Teleost fish" žive na istom shardu dok omogućava ujednačenu distribuciju unutar grupa. Upiti filtriraju po taxonomy_group koji MongoDB usmjerava na odgovarajući shard, izbjegavajući broadcast upite koji bi upitali sve shardove.

Kolekcije korisnika ostaju neshardane jer su male (100 korisnika = 100KB čak s opsežnim metapodacima) i upituju se nasumično (svaki zahtjev provjerava jednog korisnika). Audit dnevnici bi se shardali po vremenskom rasponu (mjesec ili godina), omogućavajući arhiviranje starih dnevnika na hladno skladištenje dok drže nedavne dnevnike vruće.

Ova strategija temeljena na taksonomiji pretpostavlja da upiti ostaju unutar taksonomskih grupa što je razumna pretpostavka za trenutne istraživačke tijekove rada ali može se promijeniti. Ako budući istraživači često rade upite preko taksonomija ("usporedi metaboličke stope preko svih vrsta"), sharding strategija bi trebala preispitivanje. Arhitektura ostaje fleksibilna—MongoDB dopušta re-sharding s različitim ključevima, iako s operativnim troškom.

9.2 Cachiranje za učestale upite

Projektna odluka: Redis za cachiranje učestalo pristupanih činjenica i rezultata pretraživanja

Redis pruža in-memory cache za smanjenje opterećenja MongoDB-a za često pristupane podatke.

Odluka za cachiranje-by-Redis balansira performansne dobitke protiv operativne složenosti. Na beta skali gdje MongoDB u jednoj instanci rukuje svim upitima brzo, cachiranje je optimizacija prije nego nužnost. Međutim, cache-friendly arhitektura dizajnirana rano omogućava dodavanje cacheiranja kasnije bez preinženjeringa.

Kandidati za cachiranje su činjenice koje se često pristupaju (popularne vrste, često citirane stilizirane činjenice, često posjećivane čvorove grafa znanja) i rezultati skupih upita (kompleksni agregirani cjevovodi, upiti prelaženja grafa preko mnogo nivoa). Korisničke sesije također cacheaju—nakon autentifikacije, korisnički profili se cachaju u Redis-u tako da provjere dozvola ne pogađaju MongoDB na svakom zahtjevu.

TTL (time-to-live) politike balansiraju svježinu s hit stopama. Korisnički profili cacheaju 1 sat—dovoljno dugo da provjere dozvola izbjegavaju DB pogotke, dovoljno kratko da promjene uloga stupaju na snagu razumno brzo. Činjenice cacheaju 24 sata—rijetko se mijenjaju nakon stvaranja, tako da duže cachiranje je sigurno. Rezultati pretraživanja cacheaju 5 minuta—dosta se brzo zastarijevaju kako novi sadržaj dodaju.

Invalidacija cachea rukuje se za operacije pisanja koje utječu na cacheirane podatke. Kada se činjenica ažurira, njen cache unos se poništava. Kada korisnikove mogućnosti se mijenjaju, njihov cache profil se poništava. Invalidacija mora biti pouzdana—ne poništavanje cachea vodi do korisnika koji vide zastarjele podatke, potencijalno vide sadržaj kojem nedostaje pristup.

Alternativa bez cacheiranja pojednostavljuje operacije ali ograničava buduću skalabilnost. Alternativa cacheiranja MongoDB na razini drivera (putem wiredTiger cachea) je automatski ali manje fleksibilna—ne možete cacheirati rezultate skupih agregacija, i cache dijeljenje preko više servera zahtijeva replikaciju skupova. Redis pruža eksplicitnu kontrolu nad čime se cacheuje i koliko dugo, za račun upravljanja dodatnim servisom.

9.3 Asinkrona obrada za dugotrajne operacije

Projektna odluka: Celery za queue poslove za obradu dokumenata i izvršavanje agenata

Dugotrajne operacije (obrada dokumenata, pokretanje agenata, generiranje izvještaja) izvršavaju se asinkrono putem Celery task queue-a.

Sinkrone operacije koje traju minute ili sate su loše iskustva korisnika i operativni problemi. HTTP zahtjevi timeout nakon 30-60 sekundi. Korisnici ne znaju je li njihova operacija u tijeku ili je sustav zamrznut. Ponovno pokretanje nakon timeout-a može duplicirati rad ili ostaviti djelomično završeno stanje.

Celery rješava ove probleme offloading-om dugotrajnih poslova na background worker procese. Kada korisnik predaje dokument, frontend odmah prima "Dokument u redu za obradu" odgovor dok se stvarni rad događa asinkrono. Korisnik može nastaviti raditi na drugi sadržaj ili zatvoriti browser—obrada nastavlja neovisno. Kada završi, korisnik je obavješten putem email-a ili in-app notification-a.

Arhitektura Celery-a koristi Redis kao message broker (laki, brz, već postoji za cachiranje i rate limiting). Worker procesi povlače poslove iz Redis queue-a, izvršavaju ih i ažuriraju status u MongoDB-u. Višestruki workeri mogu raditi paralelno, skaliranje throughput-a za dokument procesiranje dok frontend serveri ostaju brzi za interaktivne zahtjeve.

Task retry logika rukuje prijelaznim neuspjesima. Ako LLM API povremeno zakazuje ili je MongoDB privremeno nedostupan, Celery automatski ponovno pokušava poslove s eksponencijalnim backoff-om. Nakon maksimalno ponovljenih pokušaja, task se označava neuspjeo i administrator je obavješten. Ova otpornost na greške izbjegava potrebu za ručnom intervencijom za prolazne probleme.

Priority queues omogućavaju hitnim poslovima da preskaču rutinske poslove. Ekstrakcija činjenica od revizienta koji čeka na rezultate ide u high-priority queue. Bulk izvoz velikog skupa podataka za arhiviranje ide u low-priority queue. Ovo osigurava da interaktivni tijekovi rada ostaju brzi čak kada je sustav opterećen batch poslovima.

Alternativa bez asynca ograničava skup operacija koje platforma može ponuditi. Bez asinc obrade, dokument procesiranje ne bi moglo trajati duže od HTTP timeout-a. Batch operacije poput "ekstrahiraj činjenice iz svih 100 dokumenata" ne bi mogle postojati. Architektura temeljena na Celery-u omogućava ove bogate tijekove rada dok održava brze odgovore za interaktivne operacije.

10 Deployment i Operations

10.1 Containerizacija s Docker-om

Projektna odluka: Svaka komponenta ima Dockerfile za kontejnerizaciju

advandeb-modeling-assistant, advandeb-knowledge-builder workeri i advandeb-MCP server deployiraju kao Docker kontejneri orkestiranu putem docker-compose za razvoj i Kubernetes za produkciju.

Kontejnerizacija rješava "radi na mom stroju" problem koji muči tradicionalne deploymente. Developeri grade u Python virtuelnim okruženjima s specifičnim verzijama paketa. Produkcijski serveri imaju različite OS verzije, biblioteke sistema, ili konfiguracije mreže. Kod koji funkcionira lokalno zakazuje u produkciji zbog suptilnih razlika okoline.

Docker kontejneri pakuju aplikaciju s svim zavisnostima—Python runtime, sistem biblioteke, aplikacijski kod, konfiguracijske datoteke—u imutabilnu sliku. Ista slika koja je testirana u developmentu deployi u produkciju, garancija konzistentnosti. Build process je ponovljiv—docker build proizvodi identične slike iz istog Dockerfile-a.

advandeb-modeling-assistant kontejner pokreće Vue.js frontend i FastAPI backend u multi-stage build-u. Prvi stage gradi frontend (npm install, npm run build), drugi stage postavi Python zavisnosti, konačni stage kopira kompiliran frontend i izvršava backend. Ova struktura optimizira veličinu slike—build alati ne dolaze u konačni kontejner.

advandeb-knowledge-builder ne deployi kao dugotrajna usluga—nema HTTP krajnjih točaka. Umjesto toga, advandeb-modeling-assistant uvozi ga kao Python paket. Za batch procesiranje, KB kontejner može pokrenuti izvršavanje Celery worker procesa koji procesiraju dokumente i izvršavaju agente.

advandeb-MCP server deployi kao lagani Rust binarni kontejner. Multi-stage build kompajlira Rust kod u jednom stage i kopira binarni u minimalni distroless sliku u konačnom stage-u. Rezultat je 10-50MB slika naspram stotina MB za kontejnere temeljene na Pythonu.

docker-compose orkestracijom za lokalni razvoj pokreće sve servise s jednom komandom: MongoDB, Redis, Ollama, MA backend, MCP server. Developeri ne moraju ručno instalirati i konfigurirati svaki servis—`docker-compose up` započinje cijeli stack. Volume mount-ovi omogućavaju live code reload-a: mijenjanje Python koda automatski restarta servis bez rebuilding kontejnera.

Za produkciju, Kubernetes orkestracijom pruža napredne sposobnosti koje docker-compose nedostaje: automatsko skaliranje (dodavanje replika kad opterećenje raste), health checks i automatsko ponovno pokretanje (restart unhealthy kontejnera), rolling updates (novi deploymenti bez downtime-a), resource limits (osiguravajući servise ne monopolizira CPU/memoriju).

Alternativa bez kontejnerizacije—deployirajući direktno na VM-ove s Python virtualenvima—radi ali zahtijeva više ručnog upravljanja zavisnosti i otežava replikaciju okruženja. Za mali tim, Docker single-command development setup dramatično smanjuje onboarding trenje za nove contributore.

10.2 Monitoring i observability

Projektna odluka: Prometheus za metrike, Grafana za vizualizaciju, strukturirani logovi

Zdravlje platforme prati se putem Prometheus metrika (latencija zahtjeva, opterećenje baze podataka, upotreba memorije), visualizirana u Grafana dashboardima. Strukturirani JSON logovi omogućavaju efikasno pretraživanje i agregaciju.

Monitoring služi dva različita ali povezana cilja: operativno upozorenje (detektirati probleme dok se događaju) i post-mortem analizu (razumijevanje što je pošlo po zlu nakon incidenta). Prometheus metrike služe operativnom upozorenju—kada latencija baze podataka prelazi 100ms ili CPU upotreba prelazi 80%, upozorenja obavještavaju administratore prije nego korisnici prijave probleme.

Ključne metrike za praćenje:

- **Latencija zahtjeva**: p50, p95, p99 latencije za API krajnje točke
- **Performanse baze podataka**: vrijeme upita, pool konekcija, aktivne transakcije
- **Upotreba resursa**: CPU, memorija, disk prostor, mrežni throughput
- **Metrike aplikacije**: ukupno korisnika, činjenice stvorene/dan, trajanje procesiranja dokumenata
- **Stope pogrešaka**: HTTP 5xx odgovori, neuspjeli Celery poslovi, MongoDB neuspjeli upiti

Grafana dashboardi vizualiziraju ove metrike s grafovima vremenskih serija, toplinskim mapama i gauge-ovima. "Zdravlje sustava" dashboard pokazuje ključne vitale za brzi status provjere. "Engagement korisnika" dashboard pokazuje aktivnost kroz vrijeme i najaktivnije korisnike. "Performanse baze podataka" dashboard razotkriva spore upite i visoke opterećenje kolekcija.

Strukturirani logovi (JSON format) omogućavaju moćna filtriranja i agregacije koja plain text logovi ne mogu. Svaki log unos uključuje: timestamp, severity level, user_id (ako ima), request_id (korelacija preko servisa), component (MA vs KB vs MCP), action (što je pokušano), outcome (uspjeh/neuspjeh), duration (za operacije performansi). Ova struktura omogućava upite poput "prikaži sve neuspješne requests od user_123 u posljednjih 24 sata" ili "korelira sve servise request za request_id abc123" bez parsiranja free-form teksta.

Log agregacija (putem Loki ili ELK stack-a) centralizira logove iz svih kontejnera, omogućavajući search preko distribuiranih servisa. Tijekom incidenta, administratori mogu slijediti request kroz komponente pregledavajući logove od MA (zahtjev primljen) do MCP (agent invoked) do KB (činjenica ekstrahirana) do MongoDB (upit izvršen), rekonstruirajući cijeli tijek.

Trace sampling (opciono sa Jaeger-om) omogućava duboku analizu distribuiranih transakcija ali dodaje značajnu operativnu složenost. Za beta skalu, strukturirani logovi s request ID korelacijom pružaju dovoljnu trace sposobnost bez puno tracing infrastructure.

10.3 Backup i disaster recovery

Projektna odluka: Dnevni MongoDB backups na remotno skladištenje, dokumentirani postupci oporavka

MongoDB baza se backupuje dnevno na S3 ili alternativno objektno skladištenje. Recovery procedura je dokumentirana i testirana redovno.

Backup strategija balansira zaštitu podataka s operativnom jednostavnošću. Za beta skalu, dnevni backups pružaju prihvatljivu recovery točku cilj (RPO) od 24 sata—u najgorem slučaju, gubimo jedan dan činjenica i dokumenata. Dok bi produkcijski sustavi mogli zahtijevati hourly backups ili kontinuirane backups (MongoDB change streams), dnevne backups odgovaraju beta profilu rizika.

Backup process koristi mongodump da stvori logički backup MongoDB baze (dokumenti exportirani u BSON format), kompresira output (gzip smanjuje veličinu 10x), enkriptira arhivu (age ili GPG-based enkripcija), uploada na S3 bucket s enabled verzioniranjem. Kompletni proces traje 5-30 minuta ovisno o veličini baze (za beta: 10 GB, proces traje 10 minuta).

Retention policy drži 30 dnevnih backupa (pokrivajući posljednji mjesec), 12 mjesečnih backupa (pokrivajući posljednju godinu) i definitivni backup na kraju faze beta (enabling projektne post-morteme). Stariji backupi arhiviraju u Glacier za dugoročno čuvanje po niskoj cijeni.

Recovery procedura testira se kvartalno s proizvodnjom realističnih scenarija (MongoDB host zakaže, korumpirani podaci zahtijevaju vraćanje na ranije stanje, developer slučajno izbriše kolekciju). Testiranje utvrđuje procedura funkcionira i trening administratora izvode ga pod pritiskom. Netestirani planovi oporavka sistemički zakazuju kada su stvarno potrebni.

Alternativa MongoDB Atlas-basirana deployment automatizira backupe i pruža point-in-time recovery (PITR) omogućavajući vraćanje na bilo koji trenutak unutar retention prozora. Ovaj managed pristup smanjuje operativni teret ali uvodi vendor lock-in i povećava troškove. Za beta, self-managed backups pruža kontrolu i naučne mogućnosti uz prihvatljiv operativni overhead.

Dokumenti i datoteke koje su prenesene također se backupuju, ali odvojeno od MongoDB backupa. GridFS datoteke izvlače i tar/gzip prije S3 uploada. Alternativno, dokumenti mogu biti stavljeni direktno na S3 pri uploadu (izbjegavajući GridFS), slučaj u kojem su backup implicitni—S3 verzioniranje funkcionira kao backup mehanizam.

11 Arhitektura Knowledge Buildera (nastavak)

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

Konverzacijska izgradnja modela: Tijek rada odražava iterativno znanstveno razmišljanje. Istraživač može postavljati istraživačka pitanja o obrascima alokacije energije,

tražiti usporedbe preko vrsta ili životnih faza, ispitivati potporne i kontradiktorne dokaze, predlagati strukture modela i usavršavati pretpostavke temeljene na sadržaju baze znanja. Chatbot održava kontekst razgovora preko više okretaja, dopuštajući progresivno usavršavanje. Svaka interakcija koristi RAG za utemeljenje odgovora u kuriranom znanju—sprječavajući halucinacije i osiguravajući znanstvenu strogost.

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

Razvoj paradigmskog modela: Primarni fokus platforme tijekom inicijalnog razvoja je stvaranje osnovnih alata i uslužnih programa za izgradnju sveobuhvatne baze znanja za bioenergetsko modeliranje organizama. Arhitektura chatbot-RAG podržava ovu misiju čineći znanje dostupnim i djelotvornim. Umjesto da istraživači provedu tjedne ručno pregledavajući literaturu, sustav iznosi relevantno znanje kroz razgovor, omogućavajući brže formiranje hipoteza i konceptualizaciju modela. Naglasak je na konstrukciji baze znanja i tooling-u—detaljne API specifikacije i mogućnosti izvršavanja modela su u aktivnom razvoju i će evoluirati temeljeno na istraživačkim potrebama.

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

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

11.1 Tijek rada modeliranja temeljen na scenarijima

Projektna odluka: Entiteti scenarija kao spremnici projekata modeliranja

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

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

Ovaj obrazac spremnika pruža nekoliko koristi. Više modela može biti predloženo za jedan scenarij, omogućavajući usporedbu: "model A koristi ODE formulaciju, model

B koristi IBM pristup—koji bolje odgovara empirijskim obrascima?" Rezultati se povezuju natrag na scenarije i modele, održavajući provenijenciju. Scenariji postaju jedinica suradnje: istraživači dijele scenarije, ne sirovi kod modela. Scenarij UUID pojavljuje se u dnevnicima revizije, omogućavajući analizu koja istraživačka pitanja troše najviše resursa platforme.

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

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

11.2 Sastavljanje i predstavljanje modela

Projektna odluka: Deklarativne specifikacije modela s vezama obrazloženja na KB

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

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

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

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

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

Alternativni pristupi uključuju kod-kao-model (model je Python simulacijski kod) ili

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

11.3 Integracija s Knowledge Builderom

Projektna odluka: In-process integracija kroz uvoz paketa

MA se integrira s KB uvozom kao Python paket, pozivajući njegove funkcije direktno unutar istog procesa.

Revidirana arhitektura mijenja integraciju s HTTP API poziva na direktne pozive funkcija. MA backend uvozi advandeb-knowledge-builder kao Python paket: `from advandeb_kb import search_knowledge, ingest_document`. Funkcije se pozivaju direktno bez mrežnog hop-a, smanjujući latenciju i pojednostavljujući obradu pogrešaka.

Ovaj in-process pristup eliminirajući složenost HTTP integracije: nema potrebe za serializacijom zahtjeva/odgovora u JSON, nema timeout-a mreže za rukovanje, nema zasebnih API verzija za koordinaciju. Error stack trace-ovi održavaju preko poziva funkcija, čineći debugging jednostavnijim. Dijeljeno MongoDB konekcija omogućava transakcije koje obuhvaćaju MA i KB operacije atomski.

Trade-off je uža spajanje: MA i KB moraju biti u kompatibilnim Python verzijama i dijeliti zavisnosti. Međutim, za beta gdje obje komponente razvija isti tim, ovo spajanje je upravljivije nego za nezavisno razvijene servise. advandeb-shared-utils paket već dijeli autentifikacijsku logiku, tako da dijeljenje dodatnih paketa je konzistentno s tom arhitekturom.

Kontekst korisnika se prosljeđuje kroz pozive funkcija. MA ekstrahira user_id i mogućnosti iz JWT tokena, zatim prosljeđuje kao argument: `search_knowledge(query, user_context={user_id, capabilities})`. KB funkcije koriste ovaj kontekst za provjere autorizacije i atribuciju revizijskog dnevnika.

12 Arhitektura integracije i ugovori o podacima

12.1 API verzioniranje i ugovori

Projektna odluka: Semantičko verzioniranje za API ugovore

Iako su komponente sada usko integrirane, verzioniranje interfejsa omogućava evoluciju. advandeb-knowledge-builder paket koristi semantičko verzioniranje (major.minor.patch) za komuniciranje breaking promjena.

Čak u in-process arhitekturi, verzioniranje interfejsa omogućava evoluciju. Kada KB dodaje novu značajku ili mijenja ponašanje funkcije, verzija paketa označava utjecaj: patch verzije (1.2.3 → 1.2.4) popravljaju bugove bez promjene ponašanja, minor verzije (1.2.4 → 1.3.0) dodaju značajke unatrag-kompatibilno, major verzije (1.3.0 → 2.0.0) uvode breaking promjene koje zahtijevaju MA ažuriranja.

Python package struktura omogućava verzioniranje kroz modularizaciju:

```
advandeb_knowledge_builder/
  __init__.py  # verzija 2.0.0
```

```
v1/
  extraction.py  # legacy API za MA koji još nisu migrirani
v2/
  extraction.py  # novi API s poboljšanjima
```

MA može importirati specifičnu verziju: `from advandeb_kb.v2 import extract_facts`. Ovo dopušta postupnu migraciju—MA može koristiti v1 za neke značajke dok testira v2 za druge. Kada je v2 stabilna, v1 se deprecira i na kraju uklanja.

CI/CD pipeline testira MA protiv KB verzija prije puštanja. Automated testi validiraju da MA radi s najnovijom minor verzijom KB-a. Ako breaking promjena je nužna (nova major verzija), tim koordinira MA ažuriranja prije KB release-a.

12.2 Dijeljenje sheme podataka

Projektna odluka: Zajednički Pydantic modeli u advandeb-shared-utils

Strukture podataka dijeljene između komponenti (User, Fact, StylizedFact) definirane su kao Pydantic modeli u shared-utils paketu.

Dijeljeni Pydantic modeli osiguravaju konzistentnost podataka preko komponenti. Kada MA stvara User objekt, koristi istu klasu koju KB koristi za validaciju korisničkih podataka. Ova konzistentnost sprječava drift gdje MA i KB imaju različite razumijevanja što User objekt sadrži.

Pydantic validacija osigurava integritet podataka na granicama komponenti. Kada MA poziva KB funkciju s Fact objektom, Pydantic validira da su sva potrebna polja prisutna i ispravnog tipa. Greške tip-a otkrivaju se na granicama funkcija umjesto širenja kroz sistem.

Evolucija sheme rukuje se kroz verzioniranje modela. Kada se Fact shema mijenja (dodavanje novog polja), verzija modela se povećava: FactV1 → FactV2. Kod koji stvara facts koristi FactV2; kod koji čita facts prihvaća oba FactV1 i FactV2, migracija-automatizirano kod stare verzije na novu.

Alternative odvojenih shema po komponenti vode do sinkronizacijskih problema. Ako MA definira User u svom codebaseu a KB definira User odvojeno, promjene u jednom ne automatski propagiraju drugome. Duplikat testiranja nužan za osiguravanje obje verzije kompatibilne. Dijeljeni modeli eliminiraju ovu duplikaciju.

12.3 Strategija ID reference

Projektna odluka: MongoDB ObjectId reference s lazy loading-om

Odnosi između entiteta koriste ObjectId reference umjesto ugrađenih podataka. Povezani entiteti se učitavaju on-demand umjesto eager-loading svega.

ID-based reference optimizira za write performanse i konzistenciju podataka. Kada stilizirana činjenica referencira 10 potpornih činjenica, ona pohranjuje array ObjectId-ova: `supporting_fact_ids: [ObjectId(...), ObjectId(...), ...]`. Aktualni fact dokumenti ostaju u facts kolekciji.

Ovaj pristup sprječava dupliciranje podataka. Ako se potporna činjenica ažurira, promjena je odmah vidljiva svim stiliziranim činjenicama koje je referenciraju—nema

potrebe za propagiranjem ažuriranja. Brisanje činjenice ostavlja dangling reference (stilizirana činjenica pokazuje na nepostojeći ObjectId), koji aplikacijska logika mora rukavati gracefully—ili vraćanje null za nedostajući reference ili označavanje stiliziranih činjenica kao nevaljane.

Lazy loading znači da čitanje stilizirane činjenice ne automatski učitava sve potporne činjenice. Umjesto toga, frontend zahtijeva specifične činjenice koje korisnik želi vidjeti. Ovo smanjuje opterećenje baze podataka i veličinu odgovora—ne prenosi podatke koje korisnik ne pregledava.

MongoDB aggregation framework omogućava učinkovito joining kada je nužno. $lookup operator kombinuje stilizirane činjenice sa svojim potpornim činjenicama u jednom upitu:

```
db.stylized_facts.aggregate([
  {$match: {_id: stylized_fact_id}},
  {$lookup: {
    from: "facts",
    localField: "supporting_fact_ids",
    foreignField: "_id",
    as: "supporting_facts"
  }}
])
```

Alternative ugrađivanje (dupliciranje fact podataka u stylized_facts dokumentima) optimizira čitanje ali komplicira ažuriranja. Alternative cross-collection transakcije (atomski linking više dokumenata) dodaju overhead za običnu slučaj gdje je jedna-document write dovoljna.

13 Model podataka i odluke o pohrani

13.1 Struktura MongoDB kolekcija

Projektna odluka: Devet glavnih kolekcija organiziranih po domenskoj svrsi

Baza podataka strukturirana oko: users, capability_requests, api_keys, audit_logs (korisnik-povezane); facts, stylized_facts, knowledge_graphs, documents (znanje-povezane); scenarios, models, results (modeliranje-povezane).

Ova organizacija kolekcija balansira normalizaciju s praktičnošću. Kolekcije grupirane po domenskoj svrsi čini razumijevanje sheme intuitivnim—sve što se odnosi na korisničke račune živi u users/capability_requests/api_keys/audit_logs. Developeri znaju gdje potražiti podatke i gdje dodati nove tipove podataka.

Unutar domena, kolekcije razdvajaju po access patternima. users je hot collection—pristupana kod svakog zahtjeva. capability_requests je hladan—pristupan samo tijekom approval tijekova. audit_logs je append-only—piše često, čita rijetko. Razdvajanje im omogućava različite indeksne strategije: users heavy indexiran za brze lookupe, audit_logs particijska-friendly za arhiviranje.

Znanje-kolekcije (facts/stylized_facts/knowledge_graphs/documents) čine većinu podataka platforme. Na beta skali, svi žive u istoj MongoDB instanci. Za buduće skaliranje, ove kolekcije su kandidati za sharding po taxonomy_group ili filogenetskoj liniji. Sharding-friendly dizajn znači korištenje compound keys {taxonomy_group, _id} i izbjegavanje cross-shard transakcija kada je moguće.

Modeliranje-kolekcije (scenarios/models/results) su relativno male—mali korisnici stvara relativno malo scenarija. Opstaju unsharded duže od kolekcija znanja. Međutim, reference iz models natrag u facts uključuju taxonomy_group za omogućavanje eventualnog query routing-a.

Alternative flat schema (sve u jednoj kolekciji s type discriminatorom) ili over-normalized schema (separate kolekcije za svaku podvrstu entiteta) odbačene su. Flat schema komplicira indeksiranje; over-normalizacija fragmentira upite. Trenutna struktura pruža čistu konceptualnu organizaciju s performansnim karakteristikama prikladnim za skalu platforme.

13.2 Indeksna strategija

Projektna odluka: Compound indeksi za učestale access patterne, text indeksi za search

MongoDB indeksi osmišljeni za podržavanje primarnih patterna upita platforme: korisničke lookupe, search znanja, graph traversal, audit queries.

Svaka kolekcija ima indekse dizajnirane za svoje access patterne:
users kolekcija:

- {`google_id: 1`} - unique index za OAuth lookup
- {`email: 1`} - za admin user search
- {`status: 1, role: 1`} - za filtriranje po status/role

facts kolekcija:

- {`content: "text"`} - text index za fulltext search
- {`taxonomy_group: 1, _id: 1`} - compound za sharding-friendly queries
- {`created_by: 1, created_at: -1`} - za contributor analytics
- {`tags: 1`} - za tag-based filtering

audit_logs kolekcija:

- {`user_id: 1, timestamp: -1`} - za po-korisnik auditing
- {`action: 1, timestamp: -1`} - za filtriranje po action type
- {`timestamp: -1`} - za arhiviranje starih dnevnika

Text indeksi omogućavaju `db.facts.find({$text: {$search: "salmon lice"}})` queries s relevance scoring. Compound indeksi pokrivaju učestale filtre—traženjem svih činjenica stvorenih od korisnika X u zadnjih 30 dana koristi {`created_by, created_at`} indeks za efikasno ograničavanje.

Index selection balansirajući query performance protiv write overhead-a. Svaki indeks usporava inserte (indeks mora biti održavan). Za beta s relativno malo piše, indeks overhead je prihvatljiv. Za produkcijski sustav s millione pisanja dnevno, možda trebate aggressive index pruning.

Partial indeksi omogućavaju indeksiranje samo podskupa dokumenata. {`status: "active"`} partial indeks na users zadržava samo aktivne korisnike u indeksu, smanjujući veličinu indeksa i ubrzavajući upite koji filtriraju po aktivnih korisnika. Suspended korisnici pristupani rijetko ne zaslužuju indeks space.

14 Buduća evolucija

14.1 Što inicijalna implementacija namjerno isključuje

Projektna odluka: Svjesno isključivanje iz inicijalnog opsega

Nekoliko značajki uobičajeno pronađenih u autentifikacijskim sustavima namjerno isključeno od inicijalne implementacije, ne zaboravljeno ili previđeno.

Multi-factor autentifikacija izvan što Google pruža nije implementirana jer je redundantna. Korisnici koji omoguće 2FA na svojim Google računima donose tu zaštitu AdvanDEB-u automatski kroz OAuth. Implementiranje odvojene TOTP autentifikacije bi dodalo složenost i zahtijevalo korisnike da upravljaju još jednom device/app konfiguracijom. Ako budući zahtjevi zahtijevaju platform-specific MFA (možda za high-privilege operacije poput odobravanja financijskih transakcija), arhitektura autentifikacije omogućava dodavanje bez fundamentalnog redizajna.

IP whitelisting za API ključeve nije implementirano jer većina korisnika nema statičke IP-e. Akademske institucije često koriste dinamičko adresiranje. Istraživači rade od kuće, kafića, konferencija—njihove IP adrese se konstantno mijenjaju. Implementiranje IP whitelisting-a stvorilo bi teret podrške (legitimni korisnici zaključani vani) bez smislene sigurnosne koristi (sofisticirani napadači koriste proxyje ionako). Rijetki slučajevi uporabe zahtijevajući IP restrikciju (API ključevi za server-to-server integracije) mogu biti riješeni kroz ručnu konfiguraciju ako se pojave.

Fine-grained ACL-ovi (access control lists) na individualnim stavkama znanja nisu implementirani jer trenutni zahtjevi ne opravdavaju složenost. Sve objavljeno znanje je javno—bilo koji kurator može čitati. Draft sadržaj je privatan njegovom kreatoru dok se ne predaje za pregled. Ovaj binarni model (javno ili creator-private) rukuje trenutnim potrebama bez overhead-a provjere dozvola po dokumentu. Ako budući zahtjevi dodaju team workspace ili privatne kolekcije znanja, model mogućnosti se ekstendira prirodno—"workspace_mem mogućnost mogla bi dodijeliti pristup workspace dokumentima.

Role hierarchies sa nasljeđivanjem nisu implementirane jer dodaju složenost što kompozicija mogućnosti izbjegava. U hijerarhijskim modelima, "senior curator" bi mogao nasljeđivati od "curator" plus dodatne dozvole. Ovo nasljeđivanje stvara spajanje—mijenjanje base role dozvola utječe na sve derivirane uloge. Model mogućnosti postiže slične rezultate kroz kompoziciju (curator + analytics + reviewer) bez složenosti nasljeđivanja. Mogućnosti se kombiniraju neovisno bez stvaranja grafova zavisnosti koji postaju teško razumljivi.

Ova isključivanja odražavaju YAGNI (You Aren't Gonna Need It) princip od agilnog razvoja. Građenje značajki prije nego su potrebne često znači građenje pogrešnih značajki jer razumijevanje zahtjeva evoluira kroz upotrebu. Arhitektura autentifikacije je ekstenzibilna—nijedno od ovih isključivanja nas nije zatvorilo u kut. Ako dokaz se pojavi da su IP whitelisting ili fine-grained ACL-i potrebni, mogu biti dodani bez fundamentalnog redizajna. Dok takav dokaz se pojavi, jednostavnije je bolje.

15 Zaključak

15.1 Sveobuhvatna arhitektura platforme

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

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

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

15.2 Sintetizirani arhitektonski principi

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

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

Arhitektura nije optimalna prema bilo kojem univerzalnom mjerilu—niti jedna arhitektura nije. Ona je optimalna za AdvanDEB-ov specifični kontekst: znanstvena platforma koja

zahtijeva odgovornost, služeći akademske korisnike s umjerenom tehničkom sofisticiranošću, izgrađena od malog tima koji prioritizira razvoj značajki nad kompleksnošću infrastrukture, skaliranje na 10-20 korisnika inicijalno u beta s kapacitetom za otprilike 100 korisnika. Različiti konteksti opravdavali bi različite odluke.

15.3 Kriteriji uspjeha za arhitekturu

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

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

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

15.4 Arhitektura kao živi dizajn

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

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

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

Očekujemo da će ovaj dokument evoluirati uz platformu. Značajne arhitektonske promjene će pokrenuti ažuriranja dokumenta s poviješću verzija koja prati evoluciju dizajna. Neuspjeli eksperimenti bit će dokumentirani uz uspjehe da se sačuva institucionalno znanje o tome što nije radilo i zašto. Cilj nije arhitektonska perfekcija od početka, već arhitektonsko učenje kroz razvoj.

15.5 Buduća arhitektonska poboljšanja

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

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

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

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