Ko uporabnik ni več samo človek: od priporočil do avtonomnih agentov v poslovnih aplikacijah

Boštjan Grašič,[1] Gal Badrov,[1] Hanan Mešić,[1] Saša Grašič[2]

[1] Alloxy, informacijske rešitve, d.o.o., Maribor, Slovenija
bostjan.grasic@alloxy.io, gal.badrov@alloxy.io, hanan.mesic@alloxy.io
[2] Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, Slovenija
sasa.grasic@um.si

Umetna inteligenca (UI) postaja nepogrešljiv sestavni del poslovnih aplikacij, vendar njena vgradnja v poslovno-kritične sisteme zahteva bistveno drugačen pristop od preprostega posredovanja podatkov velikim jezikovnim modelom. Prispevek na podlagi praktičnih izkušenj z razvojem platforme za upravljanje virov v projektno usmerjenih storitvenih podjetjih predstavi sistematičen pristop k vgradnji UI. Najprej opredelimo spekter vključevanja UI v poslovne procese — od priporočil, ki jih uporabnik potrjuje, prek pol-avtonomnih agentov do avtonomnih agentov — in za vsako stopnjo avtonomije določimo ustrezne mehanizme upravljanja ter pokažemo, kje se ti pokrivajo z zahtevami Akta EU o umetni inteligenci (EU AI Act). Nadalje naslovimo tezo, da bomo informacijske sisteme morali graditi tudi za agente UI, ki bodo do njih dostopali programsko, na primer prek protokola MCP, in da morajo biti varovalke vgrajene v sama orodja, ne le v agente. Sledi praktična primerjava sedmih ogrodij za gradnjo agentov in utemeljitev naše izbire, nato pa arhitektura agentne plasti, kot smo jo zgradili v platformi Alloxy: dvoplastna avtorizacija orodij, strukturiran dolgoročni spomin, vzorec človeške potrditve pred dejanji s posledicami, deterministično usmerjanje modelov, opazljivost v lastni infrastrukturi in preverljivo kaskadno brisanje. Zaključimo s tremi konkretnimi agenti in prenosljivimi spoznanji za razvijalce in arhitekte poslovnih rešitev.

Ključne besede:
agenti umetne inteligence
poslovne aplikacije
EU AI Act
Model Context Protocol
upravljanje virov

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-299-188-3
DOI https://doi.org/10.18690/um.feri.7.2026.6

1 Uvod

Umetna inteligenca v poslovnih aplikacijah ni več zgolj funkcija, ki jo dodamo obstoječemu sistemu — kot so bili to pred nekaj leti pametno iskanje, klasifikacija dokumentov ali priporočilni algoritem. Postaja akter, ki v imenu uporabnika ali samostojno izvaja korake v poslovnem procesu: pridobiva podatke, jih povezuje, pripravlja predloge in — če mu to dovolimo — izvaja spremembe. Ta premik od »UI kot funkcije« k »UI kot akterju« odpira nov razred vprašanj, ki jih klasična arhitektura poslovnih aplikacij ne naslavlja: kdo je odgovoren za akcijo, ki jo je sprožil agent, kako jo razložimo uporabniku, kako jo omejimo, kako jo revidiramo — in kako vse to uskladimo z regulatornim okvirom, ki je medtem stopil v veljavo.

Vgradnja UI v poslovno-kritične sisteme se zato bistveno razlikuje od preprostega posredovanja podatkov velikemu jezikovnemu modelu (LLM). Potreben je sistematičen arhitekturni pristop, ne le »prompt in izpis«. Prav tako se je izkazalo, da vgradnje ni mogoče obravnavati kot enoten projekt: različni uporabniški scenariji znotraj iste aplikacije imajo bistveno različno tveganje in različne posledice napačne odločitve.

Prispevek je zavestno praktičen, ne le konceptualen: poleg ogrodja za odločanje o stopnji avtonomije predstavimo tudi konkretno primerjavo ogrodij za gradnjo agentov UI, utemeljitev naše izbire in arhitekturo, ki smo jo na tej podlagi zgradili. Izkušnje izhajajo iz razvoja platforme Alloxy — rešitve za upravljanje in načrtovanje virov v projektno usmerjenih storitvenih podjetjih z 30 do 250 zaposlenimi. Tehnološko gre za večnajemniško (multi-tenant) rešitev SaaS na skladu TypeScript in Node.js s podatkovno bazo PostgreSQL, gostovano v EU; prav večnajemniškost, EU rezidenčnost podatkov in narava domene so najbolj oblikovali naše arhitekturne odločitve.

Prispevek je členjen takole. Drugo poglavje predstavi spekter avtonomije, mehanizme upravljanja za vsako stopnjo in njihovo prekrivanje z zahtevami Akta EU o umetni inteligenci, iz česar izpeljemo tehnične zahteve. Tretje poglavje obravnava agenta kot nov tip uporabnika sistema, četrto prinaša primerjavo ogrodij in utemeljitev naše izbire, peto arhitekturo agentne plasti v platformi Alloxy, šesto tri konkretne agente, sedmo pa strne prenosljiva spoznanja.

2 Spekter vključevanja UI v poslovne procese

2.1. Zakaj iterativni pristop po scenarijih

Vgradnje UI se ni mogoče lotiti kot enega samega projekta s fiksnim obsegom. Priporočen pristop je, da najprej identificiramo posamezne uporabniške scenarije — v platformi Alloxy na primer dodelitev vira na projekt, poizvedovanje prostih kapacitet in potrjevanje zahtevkov — in za vsakega posebej opredelimo stopnjo avtonomije, varovalke ter regulatorno klasifikacijo. Prednost je hitrejše učenje na scenarijih z nizkim tveganjem, preden UI vgradimo v procese, ki vplivajo na ljudi ali na denar; kot pokažemo v razdelku 2.3, se prav ob tej ločnici odloča tudi regulatorna klasifikacija.

2.2. Tri stopnje avtonomije

Spekter vključevanja UI v poslovne procese smo razdelili na tri stopnje.

UI kot razširitev človeških sposobnosti. Sistem analizira podatke in poda priporočilo, uporabnik pa ga eksplicitno potrdi, preden se karkoli zgodi — sistem na primer predlaga najprimernejšega sodelavca za projekt, vodja oddelka pa predlog potrdi ali zavrne. Tveganje napake je omejeno, ker človek ostane zadnja instanca odločanja.

Pol-avtonomni agenti. Agent samostojno pridobiva podatke iz več virov, jih poveže in pripravi konkreten predlog za akcijo, ne le informacije. Uporabnik odločitve ne gradi od začetka, temveč pregleda in odobri, zavrne ali korigira že pripravljen predlog — agent na primer zazna prosto kapaciteto in konflikt v razporedu ter predlaga prerazporeditev.

Avtonomni agenti. Agent v ozadju izvaja optimizacijo brez neposredne udeležbe uporabnika pri vsakem koraku in mu predstavi le končni, že izračunan predlog. Vključenost človeka (angl. human-in-the-loop) se premakne z »vsak korak« na »končna odobritev«, pri manj kritičnih odločitvah pa lahko celo na naknadni pregled.

Stopnje niso strogo ločene kategorije, temveč kontinuum: isti scenarij lahko sčasoma premikamo po spektru, ko zaupanje v sistem raste. Pomembno razlikovanje, ki ga razvijemo v nadaljevanju, pa je, da stopnja avtonomije in regulatorna teža scenarija nista isto — nizka avtonomija ne pomeni samodejno nizkega regulatornega tveganja.

2.3. Mehanizmi upravljanja in njihovo prekrivanje z EU aktom o umetni inteligenci

Za vsako stopnjo avtonomije je treba določiti ustrezne mehanizme upravljanja. Ti niso dodatek k arhitekturi, temveč njen sestavni del — in v veliki meri hkrati tudi pravna zahteva.

Sledljivost in revizijska sled. Beležiti je treba, kateri podatki so bili uporabljeni, kateri model ali pravilo je generiralo predlog, kdo (človek ali agent) je akcijo sprožil in kdo jo je potrdil. Akt o umetni inteligenci [1] to za visoko tvegane sisteme zahteva kot samodejno beleženje skozi celoten življenjski cikel sistema (člen 12).

Nadzor nad akcijami in vključevanje človeka. Sem sodijo eksplicitne meje delovanja, mehanizmi razveljavitve ter časovna in obsegovna omejitev avtonomnega delovanja. Na prvi stopnji človek odloča o vsaki akciji, na drugi o vsakem predlogu, na tretji periodično ali izjemoma; ključno vprašanje torej ni, ali vključiti človeka, temveč na kateri točki procesa in s kakšno pogostostjo. Akt o umetni inteligenci to v členu 14 formalizira kot zahtevo, da je človeški nadzor sorazmeren s tveganjem, stopnjo avtonomije in kontekstom uporabe.

Kdaj scenarij postane pravno »visoko tvegan«. Za našo domeno je najbolj neposredno relevantna Priloga III, točka 4(a) — sistemi, namenjeni zaposlovanju, dodeljevanju nalog in upravljanju delavcev. Scenarij, ki se nam zdi »le priporočilo za razporeditev«, lahko pade v to kategorijo, ker vpliva na to, kdo dobi katero delo; za take sisteme veljajo dodatne obveznosti od sistema obvladovanja tveganj in kakovosti podatkov do tehnične dokumentacije, beleženja, človeškega nadzora in kibernetske varnosti (členi 9 do 15) [3].

Preglednost. Člen 50 zahteva, da uporabnik razume, da komunicira z UI. Pri agentih to ni omejeno na neposrednega uporabnika: če agent v imenu uporabnika spremeni razpored, ki zadeva druge osebe, so te »prizadete osebe«, čeprav z agentom niso nikoli neposredno komunicirale [2]. To je pomembno prav za scenarije, kjer agent posega v razpored več ljudi hkrati.

Časovnica uveljavljanja se je zaradi zakonodajnega paketa »AI Omnibus« večkrat premikala. V času pisanja velja, da so prepovedane prakse v veljavi od februarja 2025, obveznosti ponudnikov modelov za splošne namene od avgusta 2025, obveznosti preglednosti iz člena 50 od avgusta 2026, obveznosti za visoko tvegane sisteme pa so premaknjene na december 2027 oziroma avgust 2028 za sisteme, vgrajene v regulirane izdelke [3]. Odloga pri tem ni mogoče razumeti kot razloga za odlašanje: zahteve po sledljivosti in nadzoru je bistveno ceneje vgraditi v arhitekturo takoj kot naknadno v delujoč produkcijski sistem.

Tabela 1: Stopnje avtonomije, pripadajoči mehanizmi upravljanja in regulatorne zahteve.

Stopnja avtonomije	Vključenost človeka	Zahtevana sledljivost	Regulatorna teža
Razširitev človeka	Potrditev vsake akcije	Predlog, razlaga, odločitev	Odvisna od namena, ne od stopnje
Pol-avtonomni agent	Potrditev vsakega predloga	Zgoraj + viri podatkov in klici orodij	Praviloma člen 12 in 14
Avtonomni agent	Končna odobritev ali naknadni pregled	Popolna sled izvajanja in odločitev	Člen 12, 14 in praviloma Priloga III

2.4. Iz ogrodja izpeljane tehnične zahteve

Iz zgornjega ogrodja in iz našega poslovnega konteksta smo izpeljali konkreten nabor zahtev za agentno plast. Te zahteve so hkrati merila, po katerih smo v četrtem poglavju primerjali ogrodja.

Tabela 2: Zahteve za agentno plast in njihov izvor.

Oznaka	Zahteva	Izvor
Z1	TypeScript in Node.js, vgradljivost v obstoječi zaledni sistem	obstoječi tehnološki sklad
Z2	Lastna orodja nad internimi vmesniki in bazo	domenska narava rešitve
Z3	Dostop do orodij glede na vlogo in naročniški paket	nadzor nad dejanji (2.3)
Z4	Stroga izolacija med najemniki	zahteva B2B, GDPR
Z5	Usmerjanje na modele glede na zahtevnost naloge	stroški in kakovost
Z6	Kratkoročna zgodovina in dolgoročni spomin pogovorov	uporabniška izkušnja
Z7	Človeška potrditev pred dejanji s posledicami	razdelek 2.2 in 2.3, člen 14
Z8	Revizijsko beleženje vsakega klica LLM in orodja	razdelek 2.3, člen 12
Z9	Rezidenčnost podatkov v EU	GDPR in Akt o umetni inteligenci
Z10	Samogostljivost brez obveznega oblačnega ponudnika	Z9 in neodvisnost
Z11	Odprta koda brez licenčnine na uporabnika ali sled	ekonomika rešitve SaaS
Z12	Produkcijska zrelost in obvladljiva vezanost na ogrodje	poslovno tveganje

3 Poslovne aplikacije za dva tipa uporabnikov

3.1. Agent kot nov tip uporabnika sistema

Osrednja teza tega poglavja je, da bomo informacijske sisteme v prihodnosti morali graditi ne zgolj za človeške uporabnike, temveč tudi za agente UI. To pomeni, da bodo poslovne aplikacije morale ponuditi dva vzporedna dostopa: uporabniški vmesnik za ljudi in programski vmesnik za agente, na primer prek protokola MCP (Model Context Protocol) [4]. Ta ponuja standardiziran način, kako zunanji agent odkrije in kliče funkcionalnosti aplikacije, ne da bi poznal njeno notranjo implementacijo.

Posledica je, da aplikacija dobi drugega »uporabnika«, ki ne klika po zaslonih, temveč neposredno kliče funkcije in orodja. S tem odpade celotna vrsta implicitnih omejitev grafičnega vmesnika: predpisan vrstni red korakov, potrditveni dialogi in vizualni kontekst, ki pove, da je predlagana sprememba v konfliktu z drugo.

3.2. Zakaj agentom ne moremo izpostaviti enakih zmogljivosti kot ljudem

Človek v grafičnem vmesniku dobi implicitne varovalke — vidi kontekst, sledi predpisanemu toku in naleti na potrditvena okna. Agent, ki kliče orodje neposredno, teh omejitev nima: naredi natanko to, kar orodje dopušča, ne glede na to, ali je to v danem trenutku smiselno. Za agente je zato treba eksplicitno opredeliti, katera orodja so na voljo, s kakšnimi parametri in v kakšnih mejah ter katera dejanja zahtevajo potrditev.

Iz tega sledi ključno arhitekturno načelo tega prispevka: varovalke morajo biti vgrajene v sama orodja, ne le v logiko agenta. To ni zgolj dobra inženirska praksa. Strokovna literatura o skladnosti agentnih sistemov z Aktom o umetni inteligenci [5] izrecno opozarja, da navodilo modelu v pozivu — na primer »ne izvedi rezervacije brez potrditve« — ni varnostna kontrola, ker ga lahko model prezre zaradi napačne interpretacije, vrivanja navodil (angl. prompt injection) ali preprosto nepredvidenega vedenja. Zahtevana je uveljavitev omejitev na ravni orodja oziroma vmesnika. Isto načelo neodvisno izpostavlja tudi nabor tveganj za agentne aplikacije organizacije OWASP [6], ki med najresnejšimi tveganji navaja prav zlorabo orodij in postopno razširitev pravic prek zaporednih klicev.

3.3. Interni in zunanji agenti

Pristop k upravljanju se bistveno razlikuje glede na to, ali agenta razvijamo sami ali do naših orodij dostopa tuj agent. Pri internih agentih poznamo njihovo logiko, pozive in omejitve ter jih lahko vnaprej testiramo na znanih scenarijih. Pri zunanjih agentih, ki dostopajo prek protokola MCP, ne poznamo niti njihove notranje logike niti namena klicatelja niti konteksta, v katerem delujejo, in ne moremo vnaprej testirati vseh možnih načinov klicanja naših orodij. Upravljanje se mora zato preseliti iz »nadzora nad agentom« v »nadzor nad orodjem«: orodje mora biti varno ne glede na to, kdo in zakaj ga kliče. Tveganja, ki jih je treba predvideti, so napačna interpretacija namena uporabnika, verižno klicanje več orodij, ki posamezno delujejo smiselno, skupaj pa povzročijo neželen učinek, poskusi razširitve obsega delovanja prek zaporednih klicev ter preobremenitev sistema z avtomatiziranimi klici.

Praktični protiukrepi, ki jih konkretiziramo v šestem poglavju, so omejitve obsega, potrditveni koraki za kritična dejanja, način poskusne izvedbe (angl. dry-run) pred dejansko spremembo, omejitve hitrosti klicanja in dosledno načelo minimalnih potrebnih pravic.

4 Primerjava ogrodij za gradnjo agentov UI

4.1. Namen in metodologija primerjave

Ko smo se v Alloxyju odločili, da bomo agentne zmogljivosti razvijali sistematično, smo se znašli pred odločitvijo, ki je bila hkrati tehnološka in arhitekturna: ali agentno plast zgraditi sami neposredno nad klici LLM ali prevzeti eno od obstoječih ogrodij. Lastna implementacija je privlačna, ker daje popoln nadzor in nič vezanosti, a pomeni, da je treba samostojno rešiti zanko klicanja orodij, upravljanje zgodovine, perzistenco, sledenje in obravnavo napak — kar je prvih nekaj sto vrstic vsakega agentnega projekta, ki jih je nato treba tudi vzdrževati. Zato smo pred odločitvijo sistematično pregledali obstoječa ogrodja.

Merila so izpeljana neposredno iz zahtev Z1 do Z12 iz tabele 2. Poudariti velja, da so regulatorne zahteve (Z8 in Z9) v tej primerjavi enakovredna tehnična merila, ne naknaden dodatek — kar je ena od praktičnih posledic ogrodja iz drugega poglavja. Ocenili smo sedem ogrodij; izbor odraža stanje ekosistema v času odločitve, ki se hitro spreminja [14].

4.2. Pregled posameznih ogrodij

OpenAI Agents SDK [7] je bil naše izhodišče. Je dobro dokumentiran in podpira klicanje orodij, človeško potrditev in sledenje, a je tesno vezan na enega ponudnika modelov, nima večnajemniškega ločevanja spomina in zgodovine, opazljivost pa je na voljo le prek platforme ponudnika, kar je z vidika zahteve Z9 problematično. LangChain [8] je bil v času odločanja najbolj razširjen, a njegova visoka cena abstrakcije se redko lepo preslika na domensko specifično logiko agenta, Python pa ostaja primarno okolje; upravičen je na začetku projekta, ne kot temelj zrele domenske rešitve. LangGraph [11] iz istega ekosistema je najzrelejša možnost za kompleksne grafovsko modelirane tokove, a za agenta z veliko orodji in preprosto zanko dodaja kompleksnost brez koristi. Google ADK [10] je zmogljiv in odprtokoden, vendar v praksi optimiziran za en oblak, kar je pri gostovanju v EU zunaj njega pomemben zadržek. Letta [12] postavlja spomin v ospredje, a je celovit izvajalnik in ne vstavljiva plast — njegov prevzem pomeni prevzem celotnega modela agenta. Vercel AI SDK [9] smo že uporabljali in ga odlikuje odlična podpora za TypeScript ter neodvisnost od ponudnika, a gre primarno za plast za klice modelov: manjkajo delovni tokovi, upravljanje spomina, večnajemniško ločevanje in evalvacije.

Mastra [13] je ogrodje, zasnovano primarno za TypeScript, z različico 1.0 iz januarja 2026 in temelji na ogrodju Vercel AI SDK. Eno samo ogrodje pokriva agente, spomin, orodja, delovne tokove, človeško potrditev in opazljivost, ključna zmogljivost za naš kontekst pa je dinamično razreševanje orodij glede na kontekst zahteve: orodja, ki niso v vrnjenem naboru, so v celoti odsotna iz konteksta modela. Strežniški adapterji omogočajo vgradnjo v obstoječi zaledni sistem brez ločene storitve, kot hramba zadostuje že obstoječa baza PostgreSQL, licenca Apache 2.0 pa pomeni samogostljivost in odsotnost licenčnine. Med slabostmi je treba izpostaviti mladost ogrodja ter dejstvo, da izolacija na ravni najemnika ni samodejna, prav tako niso vgrajeni proračuni stroškov po najemniku, razkritja po členu 50 pa so stvar uporabniškega vmesnika.

Tabela 3: Primerjava ogrodij glede na zahteve Z1 do Z12.

Merilo	OpenAI SDK	LangChain	Vercel AI SDK	Google ADK	LangGraph	Mastra
Domoroden za TypeScript	delno	ne	da	delno	delno	da
Register lastnih orodij	da	da	da	da	da	da
Omejevanje orodij (vloga, paket)	ročno	ročno	ročno	ročno	ročno	domorodno
Večnajemniška izolacija	ročno	ročno	ročno	ročno	ročno	delno
Usmerjanje modelov	omejeno	da	da	da	da	da
Upravljanje pogovorov	ročno	da	delno	da	da	da
Dolgoročni spomin	ne	prek vtičnikov	ne	delno	delno	da
Stiskanje zgodovine	ne	delno	ne	delno	delno	da
Človeška potrditev	da	delno	delno	da	da	da
Vgrajena opazljivost	delno	prek orodja	delno	delno	prek orodja	da
Kaskadno brisanje po GDPR	ne	ne	ne	ne	ne	da
Samogostljivost	da	da	da	delno	da	da
Rezidenčnost v EU	da	da	da	tvegano	da	da
Vgradnja v obstoječi sistem	da	da	da	delno	da	da
Odprta koda brez licenčnine	ne	da	da	da	da	da
Produkcijska zrelost	visoka	visoka	visoka	srednja	visoka	srednja
Tveganje vezanosti	visoko	visoko	nizko	srednje	visoko	srednje

4.3. Zakaj smo izbrali Mastro

Odločili smo se za Mastro kot orkestracijsko plast, pri čemer smo Vercel AI SDK ohranili kot plast za klice LLM pod njo. To nam omogoča prilagodljivost pri izbiri ponudnika modelov LLM, ki ga lahko enostavno zamenjamo brez spremembe izvorne kode, hkrati pa nam omogoča, da našim strankam ponudimo, da uporabijo lastne modele LLM. OpenAI Agents SDK je bil v celoti nadomeščen, LangChain, Google ADK in LangGraph pa nismo sprejeli. Odločilni argumenti so bili, po vrsti pomembnosti za naš kontekst, naslednji.

Prvič, Mastra je edino ogrodje z domorodnim omejevanjem orodij glede na vlogo in naročniški paket (Z3). Pri vseh ostalih bi ta mehanizem morali zgraditi ročno, pri nas pa je nosilec ključne zahteve iz razdelka 3.2. Konkretno to pomeni, da agent dostopa izključno do podatkov in dejanj, do katerih ima dostop trenutno prijavljeni uporabnik glede na svojo vlogo, in izključno do zmogljivosti, ki jih ima njegova organizacija glede na svoj naročniški paket. Agent torej ni privilegiran akter s širšim dostopom kot človek, v imenu katerega deluje. Vodja oddelka, ki v uporabniškem vmesniku ne vidi finančnih podatkov projekta, jih ne bo dobil niti prek asistenta — ne zato, ker bi model to zavrnil, temveč zato, ker orodje, ki bi jih vrnilo, sploh ne pride v njegov kontekst.

Drugič, vgrajena opazljivost s hrambo v lastni bazi (Z8, Z9). Alternativa bi bila zunanja storitev SaaS za sledenje, kar je zaradi rezidenčnosti v EU izključeno, ali lastnoročno pisanje revizijskih sledi. Licenca Apache 2.0 pri tem pomeni tudi, da opazljivost ni metrirana po revizijskih sledeh (Z11) — pri metriranem modelu bi se strošek skaliral skupaj z uporabo.

Tretjič, vgradnja v obstoječi zaledni sistem brez ločene storitve (Z1). To pomeni manj operativne kompleksnosti, eno samo avtentikacijsko površino in eno samo bazo. Kaskadno brisanje spomina ob izbrisu uporabnika je pri tem del ogrodja in ne naknadno opravilo.

Kljub izbiri ogrodja smo morali sami zgraditi izolacijo na ravni najemnika, proračune stroškov po najemniku, razkritja po členu 50 v uporabniškem vmesniku ter — kot se je izkazalo med načrtovanjem — tudi lastno plast dolgoročnega spomina in lasten mehanizem človeške potrditve; zakaj smo se pri zadnjih dveh odločili proti vgrajenim rešitvam, utemeljujemo v razdelkih 5.4 in 5.5.

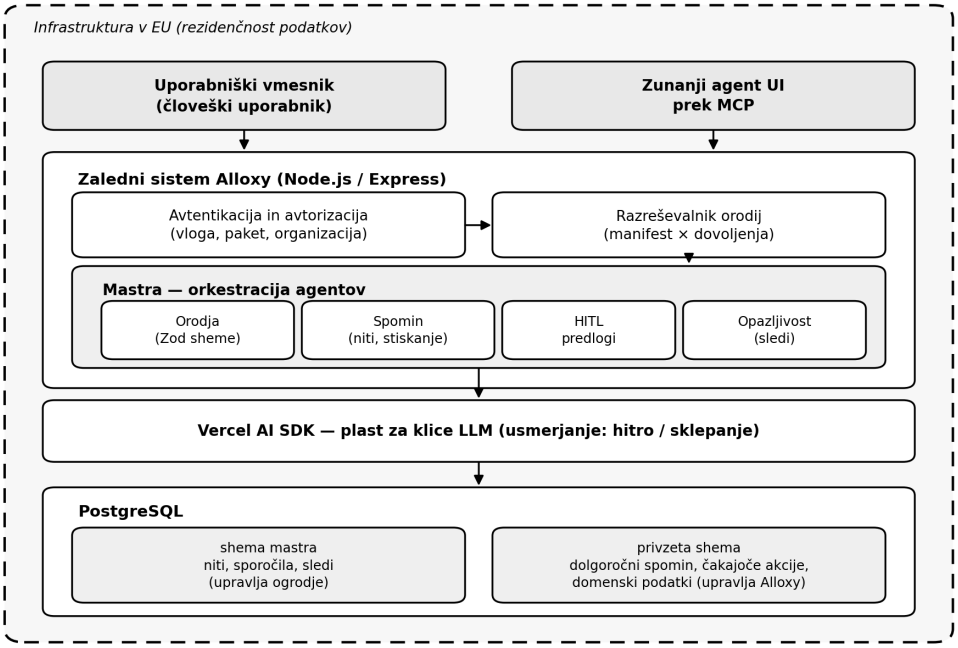
5 Arhitektura agentne plasti v platformi Alloxy

5.1. Izhodišče in cilj migracije

Prvotna implementacija asistenta je tekla na OpenAI Agents SDK: en usmerjevalni agent s trdo zapisanim modelom, pet orodij, stanje pogovora pri ponudniku modela, brez lokalne perzistence in brez sledenja. Cilj migracije ni bil zgolj »zamenjati ogrodje«, temveč prenesti avtorizacijo, spomin, usmerjanje modelov, opazljivost in brisanje po GDPR iz ročno pisanega lepila v konfigurabilne platformne zmogljivosti.

5.2. Namestitveni model in hramba

Mastra teče v procesu obstoječega zalednega sistema, ne kot ločena storitev, saj bi samostojni strežnik ogrodja pomenil drugo namestljivo enoto in drugo avtentikacijsko površino. Vsa hramba, ki jo upravlja ogrodje — niti pogovorov, sporočila, viri spomina in revizijske sledi — živi v ločeni shemi v obstoječi bazi PostgreSQL, medtem ko privzeta shema ostane v domeni naše plasti za objektno-relacijsko preslikavo; migracijski tok te plasti namreč ne sme nikoli povleči vase tabel, ki jih upravlja ogrodje. Nove tabele pod našim upravljanjem uvajamo le tam, kjer podatkovni model in podatke upravlja Alloxy: konfiguracija usmerjanja, dolgoročni spomin in zapis čakajočih akcij. Arhitektura je prikazana na sliki 1.



Slika 1: Arhitektura agentne plasti v platformi Alloxy.

5.3. Avtorizacija orodij: dve plasti brez zaupanja modelu

To je arhitekturno najpomembnejši del sistema in neposredna izvedba načela iz razdelka 3.2. Kontekst zahteve nosi v vsak klic agenta naročniški paket, učinkovita dovoljenja ter identiteto organizacije in zaposlenega. Vsako orodje v manifestu deklarira zahtevana dovoljenja kot polje polj, pri čemer zunanja raven pomeni konjunkcijo, notranja pa disjunkcijo — branje razporeditev na primer zahteva dovoljenje za pregled razporeditev vseh zaposlenih ali zgolj za pregled lastnih razporeditev. Ključna odločitev je bila, da ponovno uporabimo obstoječe domenske ključe dovoljenj in ne uvedemo vzporednega »dovoljenjskega modela za UI«; za celotno migracijo ni bil uveden noben nov ključ, saj vsak vzporeden model sčasoma neizogibno zdrsne od izvirnega.

Razreševalnik orodij preseje manifest skozi isti evalvator dovoljenj, ki ga uporablja preostala aplikacija, in to sveže ob vsaki zahtevi, tako da se sprememba pravic sredi seje odrazi že pri naslednji. Neavtorizirano orodje ni nikoli posredovano agentu in zato nikoli ne pride v kontekst modela; model zanj ne ve, zato ga ne more ponuditi, se zanj

opravičiti ali tiho spodleteti. Alternativo, pri kateri bi orodje ob klicu vrnilo sporočilo »tega ne smete«, smo zavrnili, ker razkrije obstoj orodja in posredno strukturo pravic v organizaciji.

Druga plast je ponovna avtorizacija v vsaki funkciji: vsaka izvedbena funkcija orodja prejme organizacijo in obseg ter vedno filtrira po njiju, tudi če je razreševalnik klic že spustil skozi, saj bi bilo zanašanje zgolj nanj enojna točka odpovedi. Kot dodatni varovalki delujeta (a) projekcija polj: občutljiva finančna polja so v naboru dovoljenih le, če klicatelj hkrati drži ustrezni finančni ključ, ter (b) filtriranje podatkov na nivoju orodja: agent dobi le tiste podatke, do katerih ima dostop uporabnik. Stranski učinek je manjša poraba žetonov.

5.4. Spomin: kaj prepustimo ogrodju in kaj ne

Ogrodju prepuščamo perzistenco niti pogovora in stiskanje delovnega konteksta, tako da dolgi pogovori ne prelijejo kontekstnega okna modela; ročno rezanje zgodovine je iz kode odstranjeno. Dolgoročni spomin pa smo zgradili sami: namesto da bi se zanesli na nestrukturirane spominske zapise ogrodja, ob koncu seje zaženemo lasten ekstrakcijski cevovod — klic modela s strukturiranim izhodom — ki zapiše tipizirane vrstice v našo tabelo. Taksonomija dolgoročnega spomina je namreč produktno sredstvo, ki napaja diferenciator asistiranega razporejanja: mora biti strukturirana, poizvedljiva po kategoriji in preverljivo izbrisljiva, česar neprozoren zapis, ki ga upravlja model, ne omogoča.

Taksonomija ima štiri kategorije: preference (izražene omejitve pri razporejanju, na primer da se na določeno stranko ne dodeljujejo pripravniki ali da je nekdo ob petkih zaposlen polovično), cilji (ponavljajoči se nameni, na primer redno prizadevanje za popolnitev ekipe na projektu), leče (privzete ekipe, projekti ali filtri, h katerim se uporabnik vrača) in utemeljitve (razlog za sprejeto odločitev, pri čemer transakcijski zapis že živi v domenski bazi). Pomembno je tudi, česa namenoma ne shranjujemo: prehodnih vprašanj in odgovorov ter podatkov, ki že obstajajo v domenskih tabelah. Spomin ni druga kopija baze — če bi to postal, bi se neizogibno razšel z njo.

Prožilci konca seje so začetek novega pogovora, zaprtje okna asistenta, konec seje aplikacije in odjava, kar nastopi prej. Ker nobenega od teh dogodkov ni mogoče zanesljivo ujeti v vseh primerih, imamo kot varovalo periodično opravilo, ki obdela nedejavne niti; ekstrakcija je idempotentna na nit, tako da prekrivanje prožilcev ne povzroči podvojenih zapisov.

5.5. Človeška potrditev: predlagaj, shrani, potrdi, izvedi

Ogrodje ponuja domoroden mehanizem zaustavitve in nadaljevanja izvajanja pred dejanjem s posledicami. Kljub temu smo izbrali lasten vzorec: orodje pripravi predlog, ta se shrani kot vrstica v namenski tabeli, uporabnik ga potrdi ali zavrne prek ločene končne točke, in šele nato se sprememba izvede. Razloga sta bila dva. Prvi je manjša kompleksnost: asistent je navaden agent brez delovnih tokov, domorodna zaustavitev pa bi zahtevala, da zapisovalne akcije postanejo delovni tokovi in da se žetoni za nadaljevanje pretakajo skozi transport klepeta. Drugi je regulatorno ujemanje: pričakovanja glede sledljivosti iz člena 12 ter odgovornosti in izbrisa po GDPR nagrajujejo prvorazreden, poizvedljiv in pripisljiv zapis z nadzorovano hrambo, kar namenska tabela daje, hramba stanja izvajanja ogrodja pa ne. To ni varnostna trditev — tudi pri domorodni zaustavitvi je mutacija po nadaljevanju deterministična koda; razlika je izključno v kakovosti revizijske sledi.

Vsebina potrditvene kartice ni formalnost. Vsak predlog mora nositi prizadeto osebo ali entiteto, človeku berljiv prikaz stanja pred spremembo in po njej ter nastali učinek; golo vprašanje »Potrdite?« je v našem sistemu napaka in ne funkcionalnost, saj je prav to vsebina človeškega nadzora, ki jo pričakujeta člen 14 Akta o umetni inteligenci in člen 22 Splošne uredbe o varstvu podatkov. Isti tok velja za vsako zapisovalno orodje, vključno z orodji za odobritve — potrditev odobritve je namreč sama po sebi mutacija. Vrstica čakajočega dejanja je hkrati revizijski zapis, ki je kljub trajni hrambi vključen v kaskado brisanja.

5.6. Usmerjanje modelov

Vsi trdo zapisani modeli so odstranjeni iz agentne poti; namesto njih tipizirana in ob zagonu validirana konfiguracija preslika razred zahteve — hitro ali sklepanje — v ponudnika in model. Neveljavna konfiguracija poruši zagon in ne posamezne zahteve, tako da napako odkrijemo ob namestitvi. Konfiguracija živi v kodi in ne v bazi, verzioniranje in vračanje nazaj pa pridejo iz zgodovine različic.

Razvrščanje zahteve v razred je deterministično predklasifikator in ne dodaten klic modela. Deluje po prednostnem vrstnem redu: ročni globalni pribitek kot operativno stikalo, oznaka na posameznem orodju (razvrščanje kandidatov na primer vedno teče na močnejšem modelu), jezikovno občutljivo ujemanje ključnih besed in nazadnje privzeta vrednost. Tak pristop je poceni, razložljiv in nastavljiv, medtem ko bi klic modela za odločanje o modelu dodal strošek in latenco vsakemu sporočilu. Konfiguracija nosi tudi nabor odobrenih regij, tako da usmerjanje ne more izbrati ponudnika zunaj odobrenega nabora.

5.7. Opazljivost in izpolnjevanje zahtev GDPR

Vsak klic modela in vsak klic orodja oddaja revizijsko sled z modelom, porabo žetonov, latenco in stroškom; sledi se hranijo v naši bazi, brez pošiljanja v zunanjo storitev za sledenje, saj je rezidenčnost v EU trda omejitev. Vsaka sled nosi identiteto uporabnika in najemnika, agenta, razred usmerjanja in dejansko uporabljen model, kar omogoča razčlembo porabe po vseh teh dimenzijah. Neuspel klic prav tako proizvede sled z zabeleženim neuspehom in delno porabo.

Kaskada brisanja po GDPR zajema vse domene hrambe, povezane z UI: niti in sporočila, vire spomina, dolgoročni spomin, čakajoče akcije in revizijske sledi. Vsak korak vrne število izbrisanih zapisov, preverjena poizvedba pa mora vrniti nič, preden se brisanje označi za dokončano — brisanje torej ni domnevano, temveč dokazano. Kaskada je vpeta v brisanje uporabnika in organizacije, pri čemer mehko brisanje osebe še vedno trdo počisti njene podatke, povezane z UI.

6 Trije agenti, s katerimi smo podprli poslovne procese

6.1. Pametno priporočanje virov

Prvi scenarij je dodeljevanje sodelavcev na projekte ob upoštevanju razpoložljivosti, kompetenc, preferenc in poslovnih omejitev. Najpomembnejša ugotovitev je bila, da mora biti sam predlog deterministično: vodja, ki dvakrat postavi isto vprašanje, mora dobiti primerljiv odgovor, predlog, ki ga ni mogoče razložiti, pa v postopku, ki vpliva na razporeditev dela med ljudi, ni uporaben.

Zato je bilo treba natančno načrtovati, kje v procesu uporabiti katero tehnologijo. Veliki jezikovni model uporabljamo na ravni šifranta, ne na ravni posameznega predloga: z njegovo pomočjo izračunamo faktor zamenljivosti med vlogami — na primer, v kolikšni meri lahko razvijalec za celoten sklad nadomesti razvijalca uporabniškega vmesnika — in podobno oceno sorodnosti med veščinami. Ti faktorji se izračunajo vnaprej in shranijo, tako da so ob predlogu na voljo kot podatek in ne kot sprotna presoja modela.

Sam predlog zaposlenega za določen projekt pa poteka povsem deterministično na podlagi več dejavnikov: primernosti vloge, ujemanja veščin, razpoložljivosti, predhodnega sodelovanja s projektno ekipo in stroška. Uteži se prilagajajo glede na preference organizacije — nekatere dajejo prednost kontinuiteti ekipe, druge stroškovni učinkovitosti. Rezultat je ponovljiv in razložljiv predlog, pri katerem je vsak dejavnik mogoče prikazati kot razlog za uvrstitev kandidata.

Veliki jezikovni model ima v tem scenariju še eno vlogo, in sicer sintezo razlage. Deterministični rezultat je nabor številskih ocen po faktorjih, kar je za sistem uporabno, za uporabnika pa slabo berljivo. Model iz teh ocen sestavi kratko utemeljitev v naravnem jeziku, ki izpostavi dejavnike, ki so prevladali, in tiste, ki so bili sporni — na primer da kandidat vlogo pokriva posredno prek zamenljivosti in je z ekipo že sodeloval, a da je njegova razpoložljivost v

drugem tednu obdobja tesna. Model pri tem ne spreminja razvrstitve in ne dodaja novih dejstev; njegov vhod so izključno izračunane ocene. Če bi model smel vplivati na razvrstitev, bi izgubili ponovljivost, ki je bila izhodiščna zahteva.

Sorodnost veščin smo prvotno računali iz vektorskih vložitev, kar je dajalo netočne zadetke: veščine, ki si le delijo besedišče, so bile označene kot sorodne, resnično zamenljive pa spregledane. Zamenjali smo jih s predizračunano tabelo parne podobnosti, torej z istim pristopom kot pri zamenljivosti vlog. Merilo uspeha je bilo povsem konkretno: veščini »React« in »ReactJS« morata biti prepoznani kot tesno ujemanje, veščini »React« in »Angular« pa ne, čeprav sta si besedilno blizu in v vektorskem prostoru pogosto sosednji.

Stopnja avtonomije tega scenarija je prva — sistem predlaga, vodja potrdi. Kljub temu se dotika dodeljevanja nalog in upravljanja delavcev, kar po Prilogi III, točki 4(a) potencialno spada med visoko tvegana področja uporabe. To je konkreten dokaz razlikovanja iz razdelka 2.2: nizka avtonomija ne pomeni nizkega regulatornega tveganja, ker je slednje odvisno predvsem od namena uporabe. Prav zato so razložljivost predloga, revizijska sled in obvezna človeška potrditev tu več kot dobra praksa.

6.2. Poizvedovanje razpoložljivosti in rezervacija virov

Drugi scenarij odgovarja na vprašanja tipa »kdo je naslednji mesec vsaj dvajset ur na teden prost in obvlada določeno tehnologijo« ter omogoča posledično rezervacijo. Pri njem smo prišli do treh prenosljivih spoznanj.

Prvo spoznanje je, da mora izračun izvesti orodje in ne asistent. Prvotno bi asistent sam ugibal odstotek razpoložljivosti iz zahtevanih ur, kar je nedeterministično in — kar je pomembneje — napačno za zaposlene s krajšim delovnim časom, ker se tedenska kapaciteta razlikuje po osebi. Orodje zato prejme zahtevane ure eksplicitno in pretvorbo izvede samo, za vsakega zaposlenega glede na njegovo lastno tedensko kapaciteto. Splošno pravilo: kjerkoli obstaja deterministična formula, naj jo izvede koda in ne model.

Drugo spoznanje je zahteva po enem samem viru resnice. Agent za razpoložljivost je sprva računal sproti na podlagi operativnih tabel. Rezultati se niso povsem skladali s podatki iz poročevalskih dejstev, zato je lahko ista oseba v odgovoru agenta veljala za prosto, dejansko pa ni bila na voljo zaradi dopusta ali skrajšanega delovnega časa. Rešitev je bila preusmeriti orodje na iste poročevalske podatke, s čimer smo brez dodatnega dela dobili tudi upoštevanje praznikov in odobrenih odsotnosti.

Tretje spoznanje je, da se negotov podatek prikaže in ne vračuna. Še neodobrene odsotnosti se ne odštejejo od razpoložljivosti, temveč se izpišejo kot opozorilo ob zaposlenem, uporabnik pa odloči, kako jih upoštevati. Vgraditi špekulativne podatke v uradno številko bi naredilo poročila nestabilna ob vsakem neodobrenem zahtevku, in to za uporabnika nevidno.

Prav ta scenarij najbolje ponazori tezo iz tretjega poglavja, saj isto orodje deluje v dveh bistveno različnih kontekstih. Znotraj aplikacije je klic pod neposrednim nadzorom: uporabnik je avtenticiran, vidi kontekst, dejanje je del znanega toka. Pri izpostavitvi prek protokola MCP pa isto orodje kliče zunanji agent, nad katerim nimamo nadzora; tveganja so konflikti v razporejanju, neželene spremembe plana, rezervacije, ki ne upoštevajo pravil, vidnih le v uporabniškem vmesniku, ter posreden vpliv na tretje osebe, katerih razpored se spremeni, ne da bi vedele, da je spremembo sprožil avtonomni agent. Varovalke, ki smo jih uvedli, so obvezen potrditveni korak po istem vzorcu kot v razdelku 5.5, omejitev časovnega okna in obsega, odgovor v načinu poskusne izvedbe ter beleženje vseh poskusov klicev, vključno z zavrnjenimi. Zadnje se je izkazalo za nepričakovano koristno: zavrnjeni klici so koristen pokazatelj napak v logiki agenta in morebitnih poskusov zlorabe.

6.3. Avtonomni optimizacijski agent

Tretji scenarij je avtonomni optimizacijski agent, ki periodično analizira stanje virov in pripravi predloge za izboljšanje razporeditve. Gre za najvišjo stopnjo avtonomije v našem naboru, saj agent ni sprožen z uporabnikovim vprašanjem, temveč se zažene sam.

Agent v analizo zajame tri skupine vhodnih podatkov: izkoriščenost virov, torej kdo je v obravnavanem obdobju preobremenjen in kdo premalo zaseden (iz istih poročevalskih podatkov kot scenarija 6.1 in 6.2), vpliv načrtovanih odsotnosti na kapaciteto, saj odobrena odsotnost ključne osebe sredi projekta ustvari vrzel, ki jo je bolje predvideti vnaprej, ter maržo posameznih dodelitev, ker prerazporeditev, ki uravnovesi obremenitev, a znatno poslabša donosnost projekta, ni izboljšava.

Agent najprej identificira problematične vzorce — preobremenitve, nepokrite vrzeli, dodelitve z neugodnim razmerjem med stroškom in vlogo — nato pa za vsakega pripravi konkreten predlog: prerazporeditev ur z enega projekta na drugega, zamenjavo dodeljene osebe ali opozorilo, da za vrzel ni ustreznega kandidata. Vsak predlog je opremljen z izračunanim učinkom, tako da uporabnik vidi tudi, kaj se s tem izboljša in kaj morda poslabša. Bistveno je, da agent kljub najvišji stopnji avtonomije ne izvede nobene spremembe brez končne človeške potrditve — teče skozi isti vzorec kot vsa druga zapisovalna orodja. Avtonomija se torej nanaša na to, kdaj in kako agent pride do predloga, ne na to, ali ga sme izvesti.

7 Zaključek

Vgradnja UI v poslovne aplikacije ni binarna odločitev med tem, ali umetno inteligenco imamo ali ne, temveč niz odločitev o stopnji avtonomije za vsak scenarij posebej. Vsaka stopnja zahteva svoj nabor mehanizmov upravljanja, ti pa ne smejo biti zgolj dodatek, temveč sestavni del arhitekture.

Prvo prenosljivo spoznanje je, da je izbira ogrodja upravljalska odločitev in ne le tehnološka: merila, ki so pri nas prevesila tehtnico — omejevanje orodij glede na pravice uporabnika, opazljivost v lastni infrastrukturi in kaskadno brisanje — izhajajo neposredno iz regulatornih in poslovnih zahtev. Drugo je, da varovalke sodijo v orodja in ne v poziv: neavtorizirano orodje ne sme priti v kontekst modela, avtorizacija se preverja dvakrat, deterministične izračune izvaja koda in ne veliki jezikovni model, negotovi podatki pa se prikažejo in ne vračunajo.

Tretje spoznanje je, da mora biti človeški nadzor vsebinski: potrditvena kartica brez prikaza, kdo je prizadet in kakšno je stanje pred spremembo in po njej, je videz nadzora in ne nadzor. Četrto pa je, da je regulatorna klasifikacija scenarija odvisna predvsem od namena in področja uporabe, ne od stopnje avtonomije, zato je regulatorni pregled treba izvesti za vsak scenarij posebej.

Predstavljeno ogrodje je po naši oceni prenosljivo na druge domene. Med odprtimi smermi nadaljnjega dela izpostavljamo standardizacijo varovalk za dostop prek protokola MCP, avtomatizirano evalvacijo agentnih scenarijev pred produkcijsko uvedbo, metrike zaupanja za premik scenarija po spektru avtonomije ter večagentno zasnovo, pri kateri agenti kličejo druge agente. Zadnja odpira vprašanje, ki ga tu še nismo naslovili: kako se avtorizacija in sledljivost obnašata v verigi agentov, kjer vsak deluje v imenu prejšnjega.

Literatura

[1] Evropski parlament in Svet EU, Uredba (EU) 2024/1689 o določitvi harmoniziranih pravil o umetni inteligenci (Akt o umetni inteligenci), UL EU L 2024/1689, 12. 7. 2024. https://eur-lex.europa.eu/legal-content/SL/TXT/?uri=CELEX:32024R1689, obiskano 13. 7. 2026.

[2] Evropska komisija, Guidelines on transparency obligations for providers and deployers of certain AI systems (Article 50 AI Act), julij 2026. https://digital-strategy.ec.europa.eu/en/policies/guidelines-transparency-ai-generated-content, obiskano 13. 7. 2026.

[3] Evropska komisija, Guidelines for providers and deployers of AI high-risk systems. https://digital-strategy.ec.europa.eu/en/policies/guidelines-ai-high-risk-systems, obiskano 13. 7. 2026.

[4] Anthropic, Model Context Protocol, specifikacija. https://modelcontextprotocol.io, obiskano 13. 7. 2026.

[5] NANNINI Luca et al., "AI Agents Under EU Law: A Compliance Architecture for AI Providers", arXiv:2604.04604, april 2026. https://arxiv.org/abs/2604.04604, obiskano 13. 7. 2026.

[6] OWASP Foundation, Agentic Security Initiative: Top 10 for Agentic Applications, december 2025. https://owasp.org/www-project-agentic-security-initiative/, obiskano 13. 7. 2026.

[7] OpenAI, Agents SDK. https://openai.github.io/openai-agents-python/, obiskano 13. 7. 2026.

[8] LangChain. https://python.langchain.com/docs/introduction/, obiskano 13. 7. 2026.

[9] Vercel, AI SDK. https://ai-sdk.dev/docs, obiskano 13. 7. 2026.

[10] Google, Agent Development Kit. https://google.github.io/adk-docs/, obiskano 13. 7. 2026.

[11] LangChain, LangGraph. https://langchain-ai.github.io/langgraph/, obiskano 13. 7. 2026.

[12] Letta. https://docs.letta.com/, obiskano 13. 7. 2026.

[13] Mastra, TypeScript agent framework. https://mastra.ai/docs, obiskano 13. 7. 2026.

[14] Langfuse, Comparing Open-Source AI Agent Frameworks, 2026. https://langfuse.com/blog/2025-03-19-ai-agent-comparison, obiskano 13. 7. 2026.