PROJEKT

PRENOVE AJPES IT ARHITEKTURE

Kazalo

Kazalo vsebine

1. Uvod ... 3
1.1. Namen dokumenta ... 3
1.2. Vsebina Dokumenta ... 3
2. Opis predvidenih tehnologij za prenovo informacijskega sistema ... 4
2.1 Preverjanje identitete in pravic ... 5
2.2 Sistem za verzioniranje in CI/CD postopek ... 6
2.3. Dogodkovno sporočilna platofrma ... 7
2.4. Skladiščenje znanja in dokumentacija ... 8
2.5. Bpmn 2.0 in orkestracija procesov ... 9
2.6. Kubernetes ... 10
2.7. Enoten nadzor in telemetrija ... 10
2.8. API prehod (API GATEWAY) ... 11
2.9. Mikrostoritvena arhitektura ... 12
2.10. Transformacije ... 13
2.11. Podatkovno skladišče ... 13
2.12. Strežniki ... 14

1. Uvod

1.1. Namen dokumenta

Poslovni razlog prenove je zagotoviti dolgoročno vzdržnost digitalnega poslovanja AJPES. Obstoječi informacijski sistem temelji na zastareli arhitekturi, ki je rezultat zaporednih razširitev in prilagoditev brez celovitega načrtovanja. To povzroča prekomerne odvisnosti od posameznikov, visoke stroške vzdrževanja in povečuje tveganje za napake ter nezanesljivo delovanje.

Glavni cilji prenove so:

- postavitev sodobne, razširljive in modularne arhitekture,
- razbremenitev poslovnih skrbnikov in zmanjšanje ročnih postopkov,
- vzpostavitev enotnega podatkovnega sloja za učinkovito upravljanje podatkov,
- skrajšanje časa odzivnosti na zakonske in vsebinske spremembe,
- povečanje varnosti, sledljivosti in zanesljivosti sistema.

Dokument je namenjen razvijalcem programske opreme in postavljalcem tehničnih rešitev prenove IT arhitekturne zasnove AJPES in predstavlja sestavni del razpisne dokumentacije za izvedbo javnega naročila za informacijsko rešitev prenove arhitekturnega sistema AJPES.

1.2. Vsebina Dokumenta

V dokumentu so opisane vse pomembnejše tehnične zahteve, ki jih je potrebno upoštevati pri nadgradnji oziroma prenovi informacijskega sistema AJPES. Opisane so tehnične rešitve, ki jih mora izvajalec postaviti in usposobiti, ter izobraziti zaposlene na AJPES.

Izvajalec naj koncept postavitve zasnuje tako, da bo možno celotno rešitev brez večjih prilagoditev v prihodnje razširiti tudi na rezervno lokacijo.

2. Opis predvidenih tehnologij za prenovo informacijskega sistema

Pričakovana arhitektura novega okolja bo temeljila na porazdeljenem modelu, ki vključuje uporabo storitveno usmerjene arhitekture (SOA) in dogodkovno vodene komunikacije. Takšen pristop omogoča razdelitev informacijskega sistema na sklope funkcionalnosti, ki se lahko razvijajo, testirajo, uvajajo in vzdržujejo neodvisno. Rezultat je večja odpornost sistema na napake, hitrejša uvedba sprememb in lažje prilagajanje zakonodajnim spremembam ter novim uporabniškim zahtevam.

Porazdeljena arhitektura pomeni, da posamezne funkcionalnosti niso več vezane na enoten monolitni sistem, temveč so razdeljene na medsebojno povezane, a samostojne komponente oziroma storitve. Te komponente delujejo neodvisno, se lahko po potrebi posodabljajo, širijo ali nadomeščajo, ne da bi to vplivalo na druge dele sistema.

Za AJPES tak arhitekturni pristop pomeni naslednje ključne prednosti:

- boljše razmejevanje odgovornosti med deležniki (notranjimi in zunanjimi),
- večja razpoložljivost in odpornost sistema,
- višja prilagodljivost pri uvajanju sprememb,
- zmanjšanje tveganj, povezanih z vzdrževalnimi posegi,
- podlaga za dolgoročno vzdržno upravljanje kompleksnih poslovnih procesov.

Strežniška infrastruktura bo vzpostavljena na način, ki omogoča:

- visoko razpoložljivost sistemov,
- horizontalno skalabilnost komponent,
- jasno ločevanje razvojnega, testnega in produkcijskega okolja.

Podatkovni sloj mora podpirati asinhrono obdelavo in vključevanje več podatkovnih virov. Predvidena je tudi možnost centralnega nadzora nad podatki in njihovo kakovostjo.

Vmesniki med komponentami in drugimi sistemi bodo temeljili na standardih, kot so:

- REST, OpenAPI, JSON Schema za sinhrono komunikacijo,
- gRPC za bolj učinkovite binarne prenose,
- podpora različnim načinom integracije: polling, push in event-driven.

Vse komponente bodo razvite z mislijo na t. i. observability – kar vključuje:

- dnevniške zapise,
- sistemske metrike,

- sledenje dogodkom (traces), kar bo omogočalo boljše spremljanje in odpravljanje napak ter splošno večjo zanesljivost delovanja.

Za potrebe sodobnega upravljanja in uvedbe v različna okolja mora biti vsaka komponenta:

- pripravljena za izvajanje v zabojniškem okolju,
- opremljena z jasnim receptom (build script) za izgradnjo končne slike, ki je lahko dostopna v registrih (javno ali zasebno) ali kot del končnega produkta.

Predvidena je tudi avtomatizirana izdelava tehnične dokumentacije vmesnikov (npr. s pomočjo Swagger/OpenAPI generatorjev) ter podpora avtomatiziranemu testiranju komponent že v fazi razvoja.

Končna rešitev mora biti povezljiva z obstoječimi aplikacijami AJPES, kjer je to smiselno in ekonomsko upravičeno, pri čemer se bodo upoštevale obstoječe integracijske poti, standardi in poslovna pravila.

Skladnost z zakonodajo (GDPR, ZInfP, ZVOP ipd.)

Pri snovanju arhitekture, podatkovnega modela in logike delovanja sistema mora izvajalec upoštevati vse veljavne zakonske podlage, zlasti:

- Uredbo GDPR (EU) 2016/679,
- Zakon o varstvu osebnih podatkov (ZVOP-2),
- Zakon o informacijski družbi (ZInfP),
- Zakonske podlage za delovanje AJPES kot nosilca uradnih evidenc.

Sistem mora omogočati nadzorovan dostop, sledljivost obdelav, uveljavljanje pravic posameznikov (vpogled, popravek, izbris) in jasno razmejitev namenov obdelave podatkov. Vključena mora biti revizijska sled obdelav ter časovna in pravna ločenost med podatkovnimi sklopi, kadar gre za različne pravne podlage.

2.1 Preverjanje identitete in pravic

Sistem za preverjanje identitete in pravic omogoča centralizirano, varno in učinkovito upravljanje uporabniških identitet ter dostopnih pravic, s čimer ščiti podatke in vire organizacije pred zlorabami in napadi.

Vsi moduli in aplikacije, postavljene na platformi, naj uporabljajo enoten sistem za dodeljevanje in upravljanje identitet, kot je npr. Keycloak, MS Entra ali enakovredno rešitev.

Aplikacije se neposredno ne ukvarjajo z avtentikacijo uporabnikov, temveč delegirajo to funkcionalnost na npr. Keycloaku. Ko se uporabnik uspešno identificira, ga Keycloak preusmeri nazaj v aplikacijo s kodo, katero aplikacija pri Keycloaku izmenja za dostopni žeton. Ta žeton aplikacija nato pošilja z vsako zahtevo na zaledne storitve.

Ključne (željene) funkcije:

- Preverjanje identitete
- Dodeljevanje pravic (avtorizacija)
- Upravljanje življenjskega cikla identitet
- Centralizirana evidenca in nadzor
- Napredne varnostne funkcije (podpora MFA, SSO)
- Podpora prijavi s kvalificiranim digitalnim potrdilom
- Podpora SI-PASS
- Podpora/simbioza z AJPES-ovim sistemom prijave in pravic

Poleg integracije z obstoječim sistemom UUM mora vsaka komponenta nove arhitekture omogočati vključitev v skupno rešitev za identiteto, avtorizacijo in sledljivost. To pomeni:

- da je vsaka storitev sposobna sprejemati in preverjati žetone ali drugo obliko varnostnega identifikatorja (npr. OAuth2, OpenID Connect),
- da je zagotovljena revizijska sled vseh dodeljenih in odvzetih pravic,
- da se za upravljanje pravic uporablja enotna shema vlog, ki jo je mogoče centralno upravljati,
- da so vmesniki za preverjanje identitete in pravic dokumentirani in skladni z odprtimi standardi (npr. REST + OpenAPI specifikacije).

Vse komponente morajo omogočati enostavno vključitev v bodoči skupni državni sistem (npr. SI-CeGIN ali naslednika), ko bo ta na voljo, in podpirati večnivojsko avtorizacijo v zahtevnejših okoljih.

2.2 Sistem za verzioniranje in CI/CD postopek

Sistem za verzioniranje je orodje ali programska rešitev, ki omogoča beleženje in upravljanje sprememb datotek oziroma dokumentov skozi čas. Najpogosteje se uporablja za izvorno kodo, a je uporaben tudi za druge vrste vsebin, kot so dokumenti ali konfiguracije.

Ključne lastnosti sistema za verzioniranje:

- Sledenje spremembam: Vsaka sprememba v datoteki se zabeleži kot nova različica (verzija), kar omogoča vpogled, kdo je kaj in kdaj spremenil.
- Obnavljanje prejšnjih verzij: Možnost vrnitve na katerokoli prejšnjo različico datoteke, kar je ključnega pomena za odpravljanje napak ali primerjavo sprememb.
- Sodelovanje več uporabnikov: Več uporabnikov lahko hkrati ureja datoteke, sistem pa upravlja združevanje sprememb in reševanje konfliktov.
- Centralizirano ali distribuirano hranjenje: Sistemi lahko hranijo podatke na enem centralnem strežniku ali pa distribuirano, kjer ima vsak uporabnik lokalno kopijo celotnega repozitorija.

CI/CD postopek je avtomatiziran proces, ki omogoča hitro, zanesljivo in ponovljivo uvajanje programske opreme, vključno z API prehodom in mikroservisi. Primer orodij: Jenkins, GitLab CI/CD, Azure Devops.

Gre za dve glavni fazi:

- Continious integration (CI): Vsakič ko razvijalec spremeni kodo in jo potisne v repozitorij se sproži avtomatizirana gradnja in testiranje aplikacije. Namen je zgodnje odkrivanje napak in zagotavljanje, da nova koda ne povzroči težav
- Continious Deploymet(CD): Po uspešnem prehodi CI faze se spremembe avtomatsko ali polavtomatsko namestijo v razvojno, testno ali produkcijsko okolje. To omogoča hitre in zanesljive izdaje novih verzij.

Podprt mora biti celovit CI/CD pristop, ki vključuje:

- avtomatizirano gradnjo, testiranje in varnostne preglede kode (npr. SAST/DAST),
- avtomatizirano odlaganje kontejnerskih slik (npr. Docker) v registrske repozitorije,
- avtomatizirano nameščanje v ločena okolja (DEV, TEST, PROD),
- spremljanje stanja sistemov po vsakem nameščanju (t.i. post-deploy health checks),
- možnost rollbacka na prejšnje stabilne verzije,
- integracijo z orodji za spremljanje kvalitete kode in testne pokritosti.

2.3. Dogodkovno sporočilna platofrma

Dogodkovno sporočilna platforma je sistem, ki omogoča prenos, obdelavo in usmerjanje dogodkov, oziroma sporočil med različnimi aplikacijami, storitvami ali sistemi v realnem času. Temelji na arhitekturi, kjer so dogodki (eventi) glavni način komunikacije, kar omogoča reaktivno, skalabilno in razpršeno obdelavo podatkov.

Primer dogodkovno – sporočilne platforme: Apache Kafka, Redpanda

Glavne značilnosti dogodkovno sporočilne platforme:

- Asinhrono sporočanje: komponente ne komunicirajo neposredno, ampak preko dogodkov, kar povečuje odpornost in prilagodljivost sistema
- Visoka razpoložljivost in skalabilnost: omogočajo obdelavo velikega števila dogodkov v realnem času, tudi v porazdeljenih okoljih
- Podpora za različne vzorce: Publish-subscribe, message queue, event streaming…
- Trajnost sporočil: zagotavljanje, da dogodki niso izgubljeni, ampak ob napakah
- Integracija z različnimi sistemi: omogočajo povezovanje heterogenih aplikacij in mikroservisov

Za standardizirano izmenjavo dogodkov je priporočljivo uporabljati odprti standard CloudEvents, ki omogoča dosledno strukturiranje metapodatkov dogodkov ne glede na uporabljeno tehnologijo ali programski jezik. CloudEvents prispeva k boljši interoperabilnosti med sistemi, olajša spremljanje in omogoča lažje upravljanje dogodkov v raznolikih okoljih.

Ob tem naj se uporablja register shem dogodkov, v katerem so centralno shranjene in dokumentirane sheme vseh vrst dogodkov (npr. v obliki JSON Schema). To omogoča boljše razumevanje, validacijo in razvoj odjemalcev, ki te dogodke obravnavajo.

Nekaj primerov dogodkov:

- -naročilo oddano
- -obrazec oddan (form submitted),
- -neuspešna prijava(login failed)

Aplikativni sloj mora biti zgrajen modularno in omogočati:

- neodvisno delovanje in razvoj posameznih komponent (npr. uporabniški vmesnik, poslovna logika, zaledne storitve),
- njihovo samostojno nadgradnjo,
- povezljivost preko asinhrone komunikacije (dogodki, sporočilni sistemi kot RabbitMQ ali Kafka) ter sinhrone integracije (REST/gRPC z OpenAPI in JSON Schema),
- delovanje v zabojniškem okolju (Docker/Kubernetes), s predvidenimi recepti za izgradnjo slik in avtomatizirano CI/CD verigo.

Vsaka komponenta mora omogočati vključitev v arhitekturo za opazljivost (observability), vključno z:

- strukturiranimi dnevniškimi zapisi,
- metrikami (npr. Prometheus),
- sledenjem zahtevkov (tracing, npr. OpenTelemetry ali Jaeger).

Prav tako naj vključuje mehanizme za odpornost sistema: *retry logiko*, *circuit breakerje*, *timeout nastavitve* in druge dobre prakse za odpornost mikrostoritev.

Cilj je vzpostaviti okolje, ki omogoča visoko razpoložljivost, hitro nadgradljivost in obvladljivost sistema skozi celoten življenjski cikel.

2.4. Skladiščenje znanja in dokumentacija

Sistem organiziranega zbiranja, hranjenja in upravljanja znanja v sistemu, ki omogoča njihovo učinkovito uporabo, iskanje in prenos. Pri tem je važno, da je prisotna tudi možnost dodeljevanja dostopa zunanjim izvajalcem, ki morajo do določene dokumentacije in znanja tudi dostopati. S tem pristopom povečamo produktivnost, podporo uporabnikom in sodelovanje.

Dokumentacija mora zajemati celovit spekter znanja, ki je potrebno za upravljanje, razvoj, uporabo in integracijo sistemov. To vključuje najmanj:

- sistemsko dokumentacijo (arhitektura, podatkovni model, odvisnosti),
- tehnično dokumentacijo (CI/CD procesi, recepti za gradnjo, konfiguracije),
- API dokumentacijo z uporabo standardov (npr. OpenAPI/Swagger),
- vsebinsko dokumentacijo (poslovna pravila, uporabniški priročniki),

- spremno dokumentacijo za vse objavljene komponente in posodobitve,
- repozitorij znanja, ki vključuje tudi dnevniške vpoglede, metrike, kode napak in zgodovino odločitev.

Dokumentacija mora biti del CI/CD procesa, versionirana, dostopna v skupnem skladišču (npr. Git), in jasno ločena po ciljnih občinstvih (razvoj, vzdrževanje, poslovni uporabniki).

2.5. Bpmn 2.0 in orkestracija procesov

BPMN 2.0 (Business Process Model and Notation) je standard za modeliranje poslovnih procesov, ki omogoča vizualno predstavitev poteka dela (workflow). Namenjen je tako poslovnim analitikom kot razvijalcem, saj omogoča enostavno razumevanje in avtomatizacijo procesov. Prispeva tudi k širšemu razumevanju procesa v podjetju vsem deležnikom, ki se s procesom ukvarjajo.

Orkestrator procesov, npr. Camunda /CIB seven, podpira večino elementov, predpisanih v standardu BPMN, ter omogoča modeliranje in izvajanje kompleksnih procesnih diagramov. Za modeliranje se lahko uporablja npr. Camunda Modeler (odprtokodna brezplačna namizna aplikacija), iz katerega potem lahko proces tudi izvozimo v orkestrator oziroma nam služi za analizo procesov.

Orkestrator procesov za posamezno instanco procesa hrani spremenljivke, katere lahko skozi življenjski tok procesa dodajamo in spreminjamo. Vrednosti spremenljivk orkestrator uporabi pri odločanju v vejitvah (pogojnih stavkih), uporabljamo pa jih lahko pri implementaciji opravil.

Orkestrator objavi na voljo opravila, ki jih je potrebno obdelati (podobno kot v primeru Apache Kafka, kjer se objavijo na temah), zunanje storitve pa ta opravila prevzemajo, obdelajo in orkestratorju vrnejo rezultat v obliki nabora novih/posodobljenih spremenljivk.

Do funkcionalnosti orkestratorja se dostopa preko REST API-ja, ki ponuja možnosti za prevzem in zaključevanje nalog, pridobivanje informacij in statistike o procesih, migracije instanc procesov iz starih v nove verzije, ...

Za potrebe povezave BPMN 2.0 modelov s tehnično izvedbo je predvidena uporaba izvajalnega okolja, ki omogoča:

- izvajanje orkestriranih procesov neposredno iz BPMN modelov (npr. Camunda, Zeebe ali podobno),
- integracijo posameznih storitev kot opravil (service tasks) v procesih,
- podporo asinhronemu izvajanju in sledenju dogodkom.

Modeli, izdelani v BPMN 2.0, bodo služili kot podlaga za specifikacijo izvedbenih tokov, s podporo za avtomatsko generiranje delov konfiguracij za orkestracijsko okolje. Pomemben cilj je, da bodo poslovni procesi lahko sledljivi (auditable), ponovljivi in dokumentirani na način, da se v vsakem trenutku vidi, kateri koraki procesa so bili opravljeni in s kakšnim rezultatom.

Sistemi morajo omogočati arhiviranje in revizijsko sledenje izvedb procesov (npr. prek logiranja v skupni monitoring sistem).

2.6. Kubernetes

Kubernetes je odprtokodni sistem za orkestracijo kontejnerjev, ki avtomatizira uvajanje, upravljanje, skaliranje in mreženje aplikacij v kontejnerjih.

Kubernetes omogoča, da:

- Se aplikacije zaganjajo v kontejnerjih (npr. Docker) na več strežnikih,
- samodejno razporejanje aplikacije glede na obremenitev,
- samodejni ponovni zagon aplikacije, če odpovejo,
- skaliranje aplikacije gor ali dol glede na promet,
- posodabljanje aplikacije brez izpadov (t.i. rolling updates).

Vse nove komponente morajo biti zasnovane tako, da jih je mogoče poganjati v zabojniškem okolju in jih orkestrirati preko Kubernetes ali primerljive platforme. To vključuje tudi:

- pripravo »receptov« za gradnjo zabojniških slik (Dockerfile),
- podporo za horizontalno skaliranje,
- podporo za nadzorovanje stanja (health checks, readiness/liveness probes),
- vključitev v centralni sistem spremljanja (observability), ki vključuje:
    - logiranje (strukturirani logi),
    - metrike (npr. Prometheus),
    - sledenje (tracing, npr. OpenTelemetry).

Pričakuje se, da bo izvajalec zagotovil možnost avtomatske namestitve in posodobitve komponent (CI/CD integracija), skladno s standardi, ki jih uporablja AJPES.

2.7. Enoten nadzor in telemetrija

Aplikacije omogočijo, da je njihovo interno delovanje možno spremljati z orodji za analizo in varnost, kot so Grafana, Kibana/OpenDash, Prometheus. Tehnično to pomeni, da aplikacije uporabijo standard Open Telemetry in pripadajoče knjižnice.

Aplikacije prav tako hkrati s standardnimi operativnimi metrikami (npr. poraba pomnilnika in procesorja) po potrebi izpostavijo tudi vsebinske metrike, ki so lahko uporabne na analitičnih nadzornih

ploščah (npr. število oddanih poročil). Pomembno je vzpostaviti enoten sistemski monitoring, na katerem se bo jasno videlo delovanje sistema, kje prihaja do napak, da bo vzpostavljen sistem obveščanja in s tem tudi najkrajši možni čas do odprave napak.

Vse komponente sistema morajo zagotavljati opazljivost (observability), kar vključuje podporo za zbiranje metrik, dnevniških zapisov (logs) in distribuciranih sledi (traces). Tehnične zahteve vključujejo standardizirano vključitev v centraliziran sistem nadzora, ki podpira diagnostična orodja ter spremljanje zmogljivosti in razpoložljivosti. Predvidena je skladnost z orodji, kot so Prometheus, Grafana, OpenTelemetry in podobna odprtokodna orodja.

2.8. API prehod (API GATEWAY)

API Gateway je programska komponenta, ki deluje kot enotna vstopna točka za vse zahteve odjemalcev (npr spletnih in mobilnih aplikacij) do različnih zalednih storitev oziroma mikroservisov. Njegova glavna naloga je sprejemanje, preverjanje (avtentikacija in avtorizacija) usmerjanje in združevanje zahtevkov ter vračanje odgovorov odjemalcem. Dodatno lahko izvaja dodatne funkcije, kot so omejevanje hitrosti, analiza prometa in zaščita pred grožnjami.

API Gateway poenostavi komunikacijo med odjemalci in zalednimi storitvami, izboljša varnost, omogoča centralizirano upravljanje in optimizira delovanje celotne arhitekture.

Poenostavljen primer delovanja API prehoda:

Odjemalec-->|Zahteva| API_Gateway(API Gateway)

API_Gateway -->|Avtentikacija/Avtorizacija|

API_Gateway -->|Usmerjanje| Servis1

API_Gateway -->|Usmerjanje| Servis2

API_Gateway -->|Usmerjanje| Servis3

Servis1 -->|Odgovor| API_Gateway

Servis2 -->|Odgovor| API_Gateway

Servis3 -->|Odgovor| API_Gateway

API_Gateway -->|Odgovor| Odjemalec

Poleg osnovne izpostavitve vmesnikov mora biti vzpostavljen sistem za celovito upravljanje API-jev (API Management), ki vključuje: nadzor nad dostopom, beleženje klicev (rate limiting, throttling), verzioniranje, analitiko uporabe, varnostne politike (npr. OAuth2/OpenID Connect), dokumentacijo (npr. Swagger/OpenAPI) in podporo za testno okolje. Cilj je omogočiti varen in nadzorovan dostop do storitev ter zagotoviti sledljivost in kakovost uporabniške izkušnje.

2.9. Mikrostoritvena arhitektura

Mikrostoritev predstavlja sklop sorodnih funkcionalnosti, ki rešujejo določen vsebinski problem. Obseg storitve je lahko ozek na nivoju posamezne funkcije (npr. pretvorba valut po trenutnem tečaju) ali pa širši na nivoju sistema (npr. sistem za upravljanje matičnih podatkov organizacije).

Funkcionalnosti mikrostoritev so izpostavljene preko HTTPS ali GRPC protokolov (po potrebi lahko tudi drugih), uporabniški vmesniki pa so implementirani ločeno ter služijo kot prožilci poslovnih funkcij s strani uporabnikov ali drugih sistemov.

Posamezne storitve primarno komunicirajo s širšim sistemom tako, da objavijo dogodek na dogodkovnem vodilu (npr. Apache Kafka / Redpanda). S tem znižujemo stopnjo sklopljenosti sistema. V primeru dileme, ko se je težko odločiti ali je klic res potreben, je potrebno razmisliti o ustreznosti postavljenih mej storitev. Včasih je potrebno dve storitvi združiti, po navadi takrat, ko se naše razumevanje problematike poglobi.

Storitve v grobem ločimo v dve kategoriji:

- nivo poslovne logike, procesov in storitev,
- nivo tehnično-vsebinskih podpornih storitev.

Storitve poslovne logike izvajajo primarne (jedrne) poslovne funkcije, ki predstavljajo delovanje organizacije (npr. upravljanje letnih finančnih ali premoženjskih poročil). Te storitve dalje delimo na vsebinske in skupne storitve s ciljem optimizacije razvojnih sredstev in poenostavitve arhitekture. Razvijalci so dolžni tekom razvojnega procesa refaktorirati sistem skladno z arhitekturo (npr. prepoznati priložnost za uporabo ali uvedbo skupne storitve).

Tehnično-vsebinske podporne storitve omogočajo izvajanje splošnih funkcionalnosti, ki se lahko uporabijo v različnih domenah (npr. upravljanje z matičnim katalogom storitev ali skupnimi šifranti). Nabor teh storitev je na voljo na platformi za vse vsebinske storitve. Nekatere od teh imajo tudi pripadajoče bogate uporabniške vmesnike in tako delujejo že kot zaključene celote. Cilj razvijalcev je skladno z optimizacijo stroškov razvoja širiti nabor teh storitev in poskušati vsebinske težave reševati z že znanimi orodji in tehnologijami (t.j. tehnično-vsebinskimi podpornimi storitvami).

Nabor nekaj tipičnih podpornih storitev:

- dinamični spletni obrazci (upravljanje obrazcev in enoten zajem podatkov),

- generiranje standardnih dokumentov (PDF, XLSX, Word, ...),
- hitro iskanje (po nestrukturirani vsebini),
- upravljanje z matičnimi podatki /šifranti (Master Data Management),
- podatkovni katalog (nabor podatkovnih produktov).

Vsaka mikrostoritev mora biti zasnovana tako, da deluje kot neodvisna, zamenljiva in testabilna enota. Upravljanje stanja, nadzor nad različicami, neodvisno skaliranje in možnost izolacije napak so ključni elementi arhitekture.

Mikrostoritve morajo izpostavljati jasne vmesnike (npr. REST/gRPC) in podpirati standarde za opazljivost (observability), vključno z beleženjem dogodkov, metrikami in sledmi (tracing), skladno z arhitekturo spremljanja celotnega sistema.

Predvidena je uvedba API prehoda (API Gateway) za centralizirano usmerjanje prometa, preverjanje avtorizacije in agregacijo podatkovnih klicev, kar dodatno prispeva k varnosti in nadzoru nad delovanjem mikrostoritev.

2.10. Transformacije

Transformacije oziroma ETL transformacije (Extract – Transform - Load) je faza v kateri se podatki iz različnih izvornih sistemov preoblikujejo v obliko, ki je primerna za analizo, poročanje ali nadaljnjo obdelavo v ciljnem sistemu. Ta korak zagotavlja kakovost, konsistentnost in uporabnost podatkov, ker s pomočjo transformacij zagotovimo prilagoditev podatkov ciljnemu modelu ali strukturi skladišča (npr. prevod iz formata XML v JSON in obratno). Zagotavlja tudi pripravo podatkov za analitiko, poročanje in nadaljnjo avtomatizacijo za različna orodja poslovne inteligence.

2.11. Podatkovno skladišče

Podatkovno skladišče mora biti učinkovito vključeno v okolje porazdeljenih sistemov.

Zato naj arhitektura omogoča:

- standardiziran in zanesljiv prenos podatkov iz operativnih sistemov v skladišče podatkov (ETL/ELT),
- jasno ločen analitični podatkovni sloj,
- podporo za priklop analitičnih orodij (BI rešitve, napredna analitika),
- sledenje spremembam podatkovnih struktur in nadzor kakovosti podatkov.

Vsebinska opredelitev skladišča in njegova logična zasnova bosta predmet ločenega razpisa.
Arhitektura mora torej predvideti povezljivost, ne pa izvajanja vsebinskih rešitev skladišča podatkov.

Vzpostavitev podatkovno skladišča služi več namenom:

- Shranjevanje nestrukturiranih podatkov (slike, video, varnostne kopije)
- Ima visoko razpoložljivost in trajnost
- Dostopno preko API vmesnikov
- Ima možnost razširitve
- Ima podporo za različne scenarije (distribucija, arhiv, analitika, …)

2.12. Strežniki

Strežniška infrastruktura bo zagotovljena s strani naročnika. Ponujena arhitektura mora biti združljiva z načrtovanim virtualizacijskim okoljem, zmožna horizontalnega skaliranja in omogočati učinkovito ločevanje razvojnega, testnega in produkcijskega okolja.

Ponudnik mora upoštevati podporo za orkestracijo zabojnikov (npr. Kubernetes) in načela visoke razpoložljivosti, z možnostjo vključitve v nadzorni sistem naročnika.