Universitatea Tehnică a Moldovei,
Facultatea Calculatoare Electronică și Microelectronică

ANALIZA SISTEMELOR SOFTWARE

SISTEM DE MONITORIZARE ȘI SUPORT PENTRU ASISTENȚII SOCIALI
MONOLIT VS. MICROSERVICII

Student: **gr. TI-251M,**
Perevoznic Vladislav

Coordonator: **asis. univ., Antohi Ionel**

Chişinău, 2025

CUPRINS

ABREVIERI

MVP – minimum viable product (produs minim viabil), versiunea cea mai simplă a unui sistem care conține doar funcționalitățile esențiale pentru a putea fi folosită de primii utilizatori și pentru a obține feedback real;
API – interfață de programare a aplicațiilor, un set de reguli și instrumente care permite diferitelor aplicații sau sisteme să comunice între ele, schimbând date și funcționalități într-un mod structurat;
DevOps – development + operations (dezvoltare + operațiuni), o metodologie și o cultură de lucru în care echipele de dezvoltare (cei care scriu codul) și echipele de operațiuni (cei care pun sistemul în producție și îl mențin stabil) lucrează împreună, automatizează procesele și livrează actualizări rapid și sigur, rezultatul, mai puține erori, deploy-uri mai dese și sisteme mai stabile.
CI/CD – continuous integration / continuous delivery (integrare continuă / livrare continuă), un set de practici automate;
CI – integrare continua, de fiecare dată când un programator adaugă cod nou, acesta este testat automat și integrat în proiectul principal;
CD – livrare sau deployment continuu, codul testat și validat este pus automat în producție (pe serverele reale), uneori de mai multe ori pe zi, rezultatul, actualizări rapide, fără intervenție manuală și cu risc minim de erori;
DDD – domain driven design, (proiectare bazată pe domeniu), o abordare de dezvoltare software în care accentul principal se pune pe înțelegerea profundă a domeniului de business (în cazul dat, asistența socială, tichete de suport, fluxuri atas/stas etc.);
ATAS – agenție teritorială de asistență social;
STAS – structură teritorială de asistență socială;
CNDDCM – centrul național pentru protecția datelor cu caracter personal al republicii moldova;
AGSSSI – agenția guvernamentală pentru servicii sociale și suport institutional.

INTRODUCERE

În ultimii ani, domeniul protecției sociale din Republica Moldova a intrat într-un proces amplu de modernizare și transformare digitală, având drept obiectiv eficientizarea modului în care sunt gestionate informațiile despre beneficiarii de servicii sociale. Într-un context în care instituțiile publice trebuie să răspundă rapid și transparent nevoilor cetățenilor, sistemele informatice devin instrumente esențiale pentru administrarea coerentă a datelor, pentru monitorizarea cazurilor și pentru facilitarea colaborării între structurile teritoriale de asistență socială. Aceste sisteme oferă nu doar o modalitate de centralizare a informațiilor, ci și o bază solidă pentru luarea deciziilor bazate pe date, sprijinind politicile publice în domeniul incluziunii și al protecției categoriilor vulnerabile.

În Republica Moldova, activitatea celor aproximativ 1500 de asistenți sociali comunitari presupune gestionarea unui volum considerabil de informații privind beneficiarii (persoane cu diverse forme de vulnerabilitate economică, socială sau medical). Până recent, aceste informații erau păstrate în dosare fizice, arhivate manual, fapt care genera întârzieri, erori și dificultăți în accesarea și actualizarea datelor. Ca răspuns la aceste provocări, a fost dezvoltat sistemul informațional „eSocial”, menit să digitalizeze complet procesele de înregistrare și evidență a beneficiarilor. Prin instruirea celor 1500 de asistenți sociali în utilizarea platformei, s-a făcut un pas semnificativ spre eliminarea birocrației și eficientizarea fluxului informațional. Totuși, odată cu utilizarea sistemului, au apărut și noi necesități legate de suportul tehnic și de mentenanța aplicației.

În acest context s-a conturat ideea de creare a unui sistem de monitorizare și suport pentru asistenți sociali, conceput ca o platformă de raportare a problemelor tehnice și funcționale întâlnite în lucrul cu eSocial. Scopul acestui sistem este de a asigura o interfață prietenoasă prin care asistenții sociali pot transmite rapid informații despre erorile sau dificultățile apărute în utilizarea aplicației, oferind totodată operatorilor de suport posibilitatea de a urmări, gestiona și rezolva sesizările în timp real. Astfel, proiectul de față propune o analiză comparativă între două modele de arhitectură software „monolitică” și bazată pe „microservicii” din perspectiva managementului de proiect, urmărind modul în care fiecare dintre acestea poate susține dezvoltarea, mentenanța și extinderea unui astfel de sistem.

Obiectivul lucrării este de a identifica avantajele și dezavantajele fiecărui tip de arhitectură în raport cu nevoile unui sistem informatic destinat unui număr mare de utilizatori, distribuiți teritorial, precum și impactul pe care aceste decizii arhitecturale îl au asupra planificării proiectului, structurii echipei și proceselor de management. Lucrarea se delimitează de aspectele strict tehnice de programare, concentrându-se pe analiza managerială și organizațională a dezvoltării unui sistem de monitorizare, cu accent pe resursele umane, buget și sustenabilitate.

1 STUDIU COMPARATIV

Transformarea digitală a serviciilor sociale nu înseamnă doar înlocuirea hârtiei cu formulare electronice, ci și crearea unor sisteme informatice robuste, capabile să susțină procese complexe, cu mii de utilizatori și volume mari de date. În Republica Moldova, digitalizarea domeniului social s-a accelerat prin introducerea sistemului eSocial și instruirea celor aproximativ 1500 de asistenți sociali comunitari, care au trecut de la gestionarea dosarelor fizice la lucrul cu o platformă informatică centralizată. Această schimbare a scos la iveală atât beneficiile digitalizării, cât și nevoia unor mecanisme eficiente de suport tehnic, care să asigure continuitatea activității în teren și să reducă blocajele în utilizarea sistemului.

În acest context, apare necesitatea unui „Sistem de monitorizare și suport pentru asistenți sociali”, conceput ca un instrument dedicat raportării, urmăririi și soluționării problemelor întâmpinate în lucrul cu eSocial. Un astfel de sistem trebuie să fie suficient de simplu pentru a fi utilizat ușor de personalul din teritoriu, dar în același timp suficient de flexibil și scalabil pentru a răspunde cerințelor în continuă schimbare ale ministerului, ale structurilor teritoriale și ale altor actori implicați. Alegerea arhitecturii software devine, astfel, o decizie critică, cu impact direct asupra performanței tehnice, costurilor, modului de organizare a echipei și a riscurilor pe termen lung.

Monolitul și microserviciile reprezintă două paradigme arhitecturale foarte diferite, fiecare cu logica, avantajele și limitările sale. Un monolit promite simplitate, un singur cod sursă, o aplicație unitară, ușor de înțeles de o echipă mică și relativ rapid de pus în funcțiune. Acest model este tentant mai ales în fazele inițiale ale unui proiect, când obiectivul este obținerea unui produs minim viabil și validarea rapidă a conceptului. Pe de altă parte, arhitectura bazată pe microservicii propune împărțirea sistemului în componente independente, fiecare responsabilă de o zonă clar definită de business, cu cicluri proprii de dezvoltare și scalare. Acest model devine atractiv atunci când se dorește susținerea unui număr mare de utilizatori, integrarea cu alte sisteme și posibilitatea de a evolua rapid, fără a „bloca” întregul sistem la fiecare schimbare.

Pentru un Project Manager, discuția nu este doar una tehnică, ci și una legată de planificare, buget, resurse umane și organizare. Arhitectura aleasă influențează direct dimensiunea și structura echipei, modul de colaborare între dezvoltatori, testeri și DevOps, precum și ritmul de livrare a noilor funcționalități. În plus, criterii precum scalabilitatea, mentenabilitatea, securitatea sau performanța trebuie analizate în paralel cu cele economice, costuri de implementare, cheltuieli de infrastructură, efort de mentenanță și cu cele organizaționale, legate de managementul riscurilor și cultura de lucru în echipă.

„Sistemul de monitorizare și suport pentru asistenți sociali” se află exact la intersecția acestor preocupări, este legat de un domeniu critic, cu responsabilitate socială ridicată, presupune cooperarea dintre nivelul central și structurile teritoriale și trebuie să funcționeze fiabil, inclusiv în perioade de vârf. De aceea, analiza arhitecturilor monolitică și bazată pe microservicii, din punct de vedere tehnic, economic și

organizațional, devine esențială pentru a identifica soluția care asigură, pe termen lung, echilibrul optim între simplitatea inițială și capacitatea de a evolua și de a se adapta cerințelor viitoare.

1.1 Definirea arhitecturilor software

În dezvoltarea sistemelor informatice moderne, alegerea unei arhitecturi software reprezintă una dintre cele mai importante decizii strategice, deoarece influențează direct modul în care aplicația va fi construită, întreținută și extinsă în timp. Arhitectura software stabilește structura logică a sistemului, relațiile dintre componente și principiile care guvernează interacțiunea acestora. În cazul proiectului „Sistem de monitorizare și suport pentru asistenți sociali", alegerea între o arhitectură monolitică și una bazată pe microservicii determină nu doar abordarea tehnică, ci și modul de organizare a echipei, gestionarea resurselor și complexitatea procesului de implementare.

Arhitectura monolitică este considerată forma tradițională de construcție a aplicațiilor software. Într-un sistem monolitic, toate componentele aplicației, interfața utilizator, logica de business și accesul la bazele de date, sunt integrate într-o singură unitate compactă, situate într-o singură soluție și funcționează împreună ca un întreg. Toate funcționalitățile rulează în același spațiu de execuție, iar comunicarea dintre module se realizează intern, fără interfețe externe. Comunicarea dintre componente se face prin apeluri directe de funcții. Acest stil de dezvoltare a aplicațiilor simplifică foarte mult lucrurile, în special în primele faze de dezvoltare. Principiul de bază al arhitecturii monolitice constă în unitatea structurală și logică, aplicația este construită, testată și implementată ca un întreg. Acest model este ideal pentru proiecte mici sau medii, în care complexitatea sistemului este limitată, iar modificările sunt rare [1].

Avantajele arhitecturii monolitice includ [2]:

- simplitatea în dezvoltare inițială, o singură bază de cod, ușor de gestionat de o echipă mică (2–5 dezvoltatori);
- lansarea rapidă a unei versiuni minime funcționale (MVP);
- costuri inițiale reduse și mentenanță facilă în fazele incipiente;
- testare unitară și implementare directă fără integrare multiplă.

Totuși, pe măsură ce aplicația crește, arhitectura monolitică devine greu de întreținut, așa cum s-ar putea întâmpla cu sistemul nostru, odată cu extinderea de la 1500 de asistenți sociali, dezavantajele ies la iveală, ca scalabilitate limitată, deoarece întregul sistem trebuie scalat uniform, chiar dacă doar o parte are nevoie. Mentenanță dificilă, unde o modificare minoră poate cere redeploy total și riscuri de downtime și un potențial "spaghetti code" care complică colaborarea în echipe mari, ducând uneori la decizii radicale de refactorizare completă.

Printre dezavantaje, se remarcă:

- scalabilitate limitată, aplicația poate fi extinsă doar în ansamblu, nu pe module individuale;
- dificultate în identificarea și izolarea erorilor;

- orice modificare impune recompilarea și redeploy a întregului sistem;
- risc ridicat de instabilitate, o eroare într-o componentă poate afecta întregul sistem;
- dificultate în munca paralelă a mai multor echipe pe același cod sursă.

Arhitectura bazată pe microservicii reprezintă o abordare modernă și flexibilă de dezvoltare a aplicațiilor complexe. În acest model, sistemul este împărțit în componente independente, numite **microservicii**, fiecare responsabilă pentru o funcționalitate clar definită (de exemplu: autentificare, gestionarea sesizărilor, raportare, administrarea utilizatorilor etc.). Fiecare microserviciu rulează autonom, are propria bază de date și comunică cu celelalte componente prin **interfețe API** sau mesaje standardizate [1].

Principiul fundamental al acestei arhitecturi este **independența componentelor**, fiecare microserviciu poate fi dezvoltat, testat, scalat și implementat separat, fără a afecta funcționarea celorlalte. Această abordare oferă o agilitate ridicată în dezvoltare și întreținere, fiind adoptată de companii mari precum Google, Netflix, Amazon și Uber.

Avantajele arhitecturii cu microservicii includ:

- scalabilitate excelentă, fiecare componentă poate fi extinsă separat în funcție de nevoile de performanță;
- mentenanță facilă, erorile pot fi izolate și corectate fără impact asupra întregului sistem;
- flexibilitate tehnologică, diferite microservicii pot fi dezvoltate în limbaje și medii diferite;
- organizare eficientă a echipelor, fiecare echipă gestionează un modul clar definit;
- continuitate operațională, actualizările pot fi făcute fără a opri întregul sistem.

Pe de altă parte, dezavantajele acestui model sunt asociate cu complexitatea sa crescută:

- necesitatea unei infrastructuri mai avansate (orchestrare, DevOps, CI/CD);
- dificultăți în sincronizarea versiunilor între microservicii;
- costuri mai mari de implementare inițială;
- nevoia unei comunicări excelente între echipele tehnice.

Pentru sistemul de monitorizare și suport destinat asistenților sociali, o arhitectură bazată pe microservicii ar permite o gestionare mai eficientă a volumului mare de sesizări, integrarea cu alte sisteme (eSocial, bazele de date teritoriale) și adaptarea ușoară la noi cerințe funcționale. De exemplu, un microserviciu ar putea fi dedicat recepționării solicitărilor, altul ar gestiona comunicarea cu operatorii, iar un al treilea ar analiza indicatorii de performanță în timp real.

Implementarea unei arhitecturi pe microservicii nu este doar o alegere tehnică, ci și o strategie organizațională care influențează modul în care lucrează echipele și cum evoluează un produs software. De-a lungul timpului, s-au conturat câteva practici esențiale pentru ca microserviciile să funcționeze eficient într-un mediu real.

Una dintre cele mai importante abordări este **Domain-Driven Design (DDD)**. Prin împărțirea sistemului în domenii bine definite, fiecare cu propriile limite de responsabilitate, echipele pot crea

microservicii independente, clare și ușor de întreținut. Această delimitare reduce interdependențele inutile și permite echipelor să lucreze simultan, fiecare pe partea sa de business, fără să blocheze dezvoltarea celorlalți.

O altă componentă critică o reprezintă **service discovery** și **managementul API-urilor**. Pentru ca microserviciile să comunice între ele într-un mod dinamic și sigur, sistemele moderne folosesc gateway-uri API și mecanisme automate de descoperire a serviciilor. Aceste instrumente gestionează accesul, versiunile API-urilor și securitatea canalelor de comunicare, oferind un control centralizat asupra întregii interacțiuni dintre servicii.

Într-o arhitectură distribuită, apar inevitabil probleme locale. De aceea, este esențială aplicarea unor **pattern-uri de reziliență**. Tehnici precum „circuit breaker", „retry", „fallback" sau „bulkhead" ajută la izolarea erorilor și previn propagarea lor în restul sistemului. Chiar dacă un microserviciu întâmpină un incident, aplicația poate continua să funcționeze, menținând o experiență stabilă pentru utilizatori.

Pentru a gestiona corect acest ecosistem complex, este necesară o observabilitate avansată. Monitorizarea nu se limitează doar la loguri și grafice de performanță. Microserviciile necesită vizibilitate completă, obținută prin instrumente precum „distributed tracing", centralizarea logurilor, dashboard-uri de metrici și sisteme automate de alertare. Aceste mecanisme permit echipelor să identifice rapid problemele și să mențină stabilitatea operațională.

În final, toate aceste strategii devin cu adevărat eficiente doar atunci când sunt susținute de **procese solide de automatizare și DevOps**. Fără pipeline-uri automate pentru testare, integrare și deploy, complexitatea microserviciilor poate ajunge să depășească beneficiile lor. Automatizarea este cea care menține un ritm constant de livrare și reduce riscurile umane într-un sistem cu multe componente distribuite [2]. Un pipeline obișnuit poate avea:

- **build**, codul e compilat / împachetat;
- **test**, se execută automat testele;
- **code quality check**, se analizează stilul, vulnerabilitățile (linting, SAST);
- **deploy pe staging**, se trimite automat aplicația pe un server de test;
- **deploy pe producție**, dacă totul e OK, aplicația se publică pentru utilizatori.

1.2 Analiza comparativă generală

Alegerea arhitecturii software potrivite reprezintă una dintre deciziile esențiale în planificarea unui sistem informatic de anvergură, precum „Sistemul de monitorizare și suport pentru asistenți sociali". Întrucât obiectivele acestui proiect presupun fiabilitate, scalabilitate și un flux continuu de asistență tehnică pentru un număr mare de utilizatori, este necesară o analiză comparativă detaliată între cele două paradigme arhitecturale „*monolitică* și *bazată pe microservicii*" prin prisma **criteriilor tehnice**, economice și organizaționale.

Scalabilitate, arhitectura monolitică oferă o scalabilitate limitată, întrucât aplicația trebuie extinsă în ansamblu. Orice creștere a volumului de date sau a numărului de utilizatori implică resurse suplimentare la nivelul întregii aplicații. În schimb, arhitectura cu microservicii permite scalarea individuală a componentelor, de exemplu, modulul de raportare sau cel de gestionare a sesizărilor poate fi extins fără a afecta restul sistemului De exemplu, Amazon a făcut această trecere (de la monolit, la microservicii), deoarece baza sa de cod monolitică, aflată în rapidă expansiune, nu putea fi scalată sau actualizată eficient. Structura monolitică le-a împiedicat capacitatea de a inova și de a ține pasul cu cerințele în creștere [3].

Mentenabilitate. Într-un sistem monolitic, mentenanța devine tot mai dificilă odată cu creșterea volumului de cod. Modificarea unei funcționalități poate avea efecte nedorite asupra altor părți ale aplicației. În microservicii, fiecare modul este izolat, ceea ce permite intervenții și actualizări locale, fără riscul de a compromite sistemul global.

Securitate. Arhitectura monolitică oferă o securitate mai simplu de gestionat datorită controlului centralizat, însă orice vulnerabilitate afectează întregul sistem. În microservicii, securitatea este granulară, fiecare serviciu poate avea propriile politici de autentificare și criptare, reducând riscul de propagare a breșelor, dar necesitând o coordonare mai complexă.

Performanță. În monolit, comunicarea internă este mai rapidă, deoarece toate funcțiile se află în același spațiu de memorie. Totuși, odată cu creșterea dimensiunii aplicației, performanța scade. În microservicii, comunicarea prin API-uri poate adăuga un mic cost de latență, însă avantajul major este stabilitatea și performanța predictibilă pe termen lung, datorită scalării selective.

Criterii economice. Costuri de implementare. Sistemele monolitice sunt mai ieftine la început, având nevoie de mai puțini dezvoltatori și o infrastructură simplă. Microserviciile implică costuri mai mari inițial (echipe multiple, orchestrare, servere distribuite), dar reduc semnificativ costurile de mentenanță pe termen lung [4].

Resurse umane. Într-un proiect monolitic este suficientă o echipă restrânsă (3–5 dezvoltatori). Într-un proiect bazat pe microservicii sunt necesare echipe specializate pe module (backend, frontend, DevOps, QA), ajungând la 15–20 de persoane.

Timp de dezvoltare. Arhitectura monolitică permite o lansare rapidă a versiunii inițiale, fiind potrivită pentru prototipuri. În microservicii, timpul inițial crește din cauza configurării infrastructurii, însă ulterior dezvoltarea devine mai rapidă datorită lucrului în paralel al echipelor [5].

Infrastructură. Monolitul poate rula pe un singur server sau instanță cloud. Microserviciile necesită containere (Docker, Kubernetes), monitorizare și orchestrare, ceea ce implică un cost suplimentar, dar oferă o fiabilitate superioară.

Criterii organizaționale. Managementul echipei. În arhitectura monolitică, managementul este simplu, o singură echipă, un singur flux de lucru. În microservicii, PM-ul trebuie să coordoneze mai multe echipe autonome, fiecare cu propriile priorități și cicluri de livrare.

Riscuri. Într-un monolit, riscul principal este colapsul total al sistemului în caz de eroare critică. În microservicii, riscurile sunt fragmentate, o eroare într-un serviciu nu afectează întregul sistem, însă crește riscul de nealiniere între componente.

Colaborare. Modelul monolitic favorizează colaborarea directă, dar poate genera blocaje de cod. În microservicii, colaborarea este bazată pe interfețe API și documentație, necesitând o cultură organizațională matură și o comunicare constantă între echipe [1].

În tabelul 1.1, este reprezentată diferența arhitecturilor prin comparația criteriilor, asocciind o pondere și scoruri pentru fiecare criteriu, începând de la scorul 1 la 5, unde 1 este foarte slab, iar 5 este excelent. Ponderea fiecărui criteriu exprimă importanța relativă în contextul proiectului (0-100%).

Tabelul 1.1 – Compararea arhitecturilor cu scoruri ponderate

Categorie	Criteriu	Pondere	Monolitic	Microserviciii
Tehnice (40%)	Scalabilitate	0.10	2	5
	Mentenabilitate	0.10	2	5
	Securitate	0.10	3	4
	Performanță	0.10	4	4
Economice (35%)	Costuri de implimentare	0.10	5	3
	Resurse umane	0.10	4	3
	Timp de dezvoltare	0.10	4	3
	Infrastructură	0.05	3	4
Organizaționale (25%)	Managementul echipei	0.10	4	3
	Riscuri	0.10	2	4
	Colaborare	0.05	3	5
Scor total ponderat		**1**	**3.2**	**4.3**

Analiza comparativă arată că, deși arhitectura monolitică are avantaje în fazele inițiale, costuri reduse, echipă mică, implementare rapidă, aceasta devine dificil de întreținut și extins odată cu creșterea complexității sistemului. În schimb, arhitectura bazată pe microservicii oferă o scalabilitate superioară, reziliență operațională și adaptabilitate la schimbări frecvente, elemente esențiale pentru un sistem național de suport tehnic precum cel destinat asistenților sociali.

Din perspectiva unui Project Manager, microserviciile presupun un efort mai mare de coordonare și planificare, dar oferă beneficii clare pe termen lung, reducerea riscurilor de blocaj, mentenanță distribuită, actualizări independente și posibilitatea de integrare cu alte platforme guvernamentale (eSocial, CNDDCM, AGSSSI). Prin urmare, pentru un sistem aflat la început, un prototip monolitic poate fi o alegere pragmatică. Totuși, pentru implementarea finală și extinderea la scară națională, arhitectura bazată pe microservicii reprezintă soluția recomandată, oferind echilibrul optim între performanță, flexibilitate și sustenabilitate.

2 CONCEPTUL ȘI DETALIEREA ARHITECTURILOR ANALIZATE

Proiectarea unui sistem informatic modern necesită o înțelegere profundă a modului în care arhitectura software poate influența performanța, scalabilitatea și evoluția unei soluții digitale. În contextul dezvoltării unor platforme complexe, precum „Sistemul de monitorizare și suport pentru asistenții sociali”, alegerea arhitecturii devine un factor determinant pentru întreaga durată de viață a aplicației. Sistemele de acest tip implică gestionarea unui volum considerabil de informații, coordonarea unui număr mare de utilizatori cu roluri diferite și asigurarea unei funcționări stabile în condiții de încărcare variabilă. De aceea, analiza arhitecturală nu se rezumă doar la aspecte tehnice, ci include și considerente organizaționale, bugetare, operaționale și de mentenanță.

Arhitectura monolitică reprezintă unul dintre modelele fundamentale de construcție a aplicațiilor software, fiind utilizată pe scară largă în proiectele în care se urmărește simplitate, viteză de dezvoltare și un control unitar asupra codului. Structura sa compactă, organizată în jurul a trei paliere principale interfața utilizator (front-end), logica de business și nivelul de acces la date, permite o implementare rapidă și o coordonare eficientă a echipei. În astfel de sisteme, fluxurile interne, precum gestionarea solicitărilor, actualizarea statusurilor, gestionarea bazelor de date și interacțiunea între module, sunt realizate prin conexiuni directe între componente. Această abordare este potrivită mai ales în fazele inițiale ale unui proiect, atunci când este necesară construirea unui prototip sau lansarea rapidă a unei versiuni minime funcționale. Totuși, odată cu creșterea numărului de utilizatori și extinderea funcționalităților, arhitectura monolitică își arată limitele în ceea ce privește scalabilitatea, mentenanța și adaptarea rapidă la noi cerințe. Evaluarea resurselor necesare, împreună cu tabelele de costuri și estimările de risc, evidențiază faptul că monolitul devine tot mai dificil de extins și întreținut pe termen lung, mai ales într-un sistem destinat să funcționeze la nivel național.

Pe de altă parte, arhitectura bazată pe microservicii propune o abordare modernă, modulară și distribuită a dezvoltării software, potrivită pentru sisteme dinamice și cu cerințe de scalare ridicate. Într-un astfel de model, aplicația este fragmentată în componente independente,microservicii, fiecare responsabil pentru o funcționalitate clară. Pentru sistemul de monitorizare a asistenților sociali, microservicii precum „Asistent Social”, „Cerere de Suport”, „Operator și Management Suport”, „Autentificare și Roluri”, „Notificări” sau „Rapoarte și Statistici” permit o distribuție clară a logicii de business și o autonomie a dezvoltării. Comunicarea dintre aceste servicii se realizează prin API-uri, intermediari de mesaje și un API gateway centralizat, creând un ecosistem flexibil, ușor de extins și rezistent la erori. Diagrama logică descrie nu doar fluxul datelor, ci și mecanismele de interacțiune dintre microservicii, evidențiind avantajele acestei arhitecturi în medii complexe.

2.1 Arhitectura monolitică

Arhitectura monolitică reprezintă un model tradițional de dezvoltare a aplicațiilor software, în care toate componentele majore ale sistemului sunt integrate într-o singură aplicație unitară. În cadrul proiectului „Sistemul de monitorizare și suport pentru asistenți sociali”, structura monolitică este organizată în trei niveluri principale: front-end, business logic și data layer.

2.1.1 Descrierea structurii

Front-end-ul este componenta vizibilă utilizatorilor, formată din paginile și formularele prin care asistenții sociali, operatorii și administratorii interacționează cu sistemul. Acest strat gestionează prezentarea informației și validarea inițială a datelor introduse. Acest nivel este responsabil de:

- afișarea formularelor de raportare a problemelor;
- vizualizarea listelor de solicitări;
- afișarea statusurilor și a notificărilor;
- interacțiunea cu utilizatorul (validarea datelor la nivel de formular, mesaje de eroare, feedback vizual).

Business logic-ul reprezintă centrul funcțional al aplicației, unde sunt implementate regulile de procesare, deciziile automatizate, clasificarea problemelor, actualizarea statusurilor tichetelor și gestionarea fluxurilor interne de lucru. Business logic-ul conține regulile de business și procesele interne ale sistemului:

- înregistrarea și validarea solicitaților (tichete de suport);
- clasificarea automată pe categorii, prioritate și tip de problemă;
- gestionarea fluxurilor de lucru („nou” , „în lucru” , „soluționat / escaladat”);
- logica de notificare a asistenților sociali și a operatorilor;
- generarea rapoartelor de activitate și statistici.

Data layer-ul este responsabil de comunicarea cu baza de date, incluzând operațiunile de stocare, modificare și interogare a informațiilor legate de utilizatori, tichete, liste ATAS/STAS și istoricul intervențiilor. Cele trei niveluri sunt coezive și funcționează împreună într-un singur proces logic, instalat și rulat ca o aplicație compactă [6]. Aces nivel include:

- module de citire/scriere a tichetelor;
- gestiunea listelor ATAS/STAS;
- stocarea datelor despre utilizatori (asistenți, operatori, administratori);
- arhivarea istoricului de sesizări.

2.1.2 Modul de implimentare pentru sistemul de monotorizare

În cadrul unei arhitecturi monolitice, toate procesele interne ale sistemului sunt gestionate centralizat, în aceeași aplicație. Implementarea fluxurilor pornește de la înregistrarea solicitărilor raportate

de asistenții sociali. Utilizatorul introduce datele obligatorii ca ATAS, STAS, nume, telefon, email instituțional, ID-ul cazului din platforma „eSocial” și descrierea problemei, iar aplicația validează aceste informații și creează tichetul corespunzător, setând automat statusul inițial și o prioritate implicită.

Înregistrarea, pașii consecutivi a problemelor (solicitărilor) asistenților sociali:

- asistentul social accesează interfața web și se autentifică;
- alege ATAS și STAS din liste predefinite;
- introduce manual în câmpurile corespunzătoare (nume, prenume, telefon, email instituțional, ID-ul cazului din eSocial, descrierea problemei);
- la trimiterea formularului, front-end-ul apelează direct logica de business.

Logica de business gestionează ulterior clasificarea și reorganizarea ticheteor în funcție de tipul problemei, data raportării și nivelul de prioritate (imediată, normală sau pe termen lung).

Business logic:

- validează câmpurile (obligatorii, format, lungime);
- înregistrează solicitarea în baza de date;
- setează statusul inițial „Nou” și o prioritate implicită (de ex. „Normal”);
- aplică reguli de business pentru prioritate (ex.: probleme critice care blochează mulți utilizatori pot fi marcate ca „Imediat”).

Managementul solicitărilor (prioritate, tip, data):

- operatorii de suport accesează o secțiune dedicată (dashboard);
- monitorizeaza lista de tichete filtrabile:
- filtrare după prioritate (imediată, medie, de lungă durată);
- tip problemă (tehnică, funcțională, eroare utilizator);
- status (nou, în lucru, escaladat, soluționat).

Operatorul poate:

- actualiza statusul;
- modifica prioritatea;
- adăuga note interne;
- marca tichetul ca „escaladat” către echipa tehnică.

Logica de business actualizează automat câmpurile de dată și urmărește timpul de răspuns și soluționare.

Managementul bazei de date se realizează printr-un set de module interne care controlează direct tabelele principale ale sistemului. Toate operațiunile ca inserare, actualizare și arhivare, sunt executate în același context aplicațional, fără separarea datelor pe module individuale, ceea ce simplifică accesul, dar poate complica mentenanța ulterioară. Interfața utilizator este uniformă și integrată în monolit, oferind atât

formulare simple pentru raportarea incidentelor, cât și un panel de management pentru operatori, cu posibilitatea de filtrare după ATAS, STAS, prioritate sau tipul problemei. Baza de date conține tabele precum:

- AsistentiSociali (nume, ATAS, STAS, telefon, email);
- TicheteSuport (ID tichet, ID asistent, ID caz eSocial, descriere, tip problemă, prioritate, status, data creării, data rezolvării);
- Utilizatori (conturi de operatori și administratori);
- ATAS, STAS (liste codificate pentru clasificare).

Toate operațiunile de insert, update, delete sunt realizate prin metode centralizate în data layer.

Interfața utilizator. Pentru asistenți sociali:

- formular simplu și intuitiv de raportare a problemelor;
- pagină „Sesizările mele” pentru urmărirea statusului.

Pentru operatori:

- dashboard cu listă de tichete;
- filtre și sortări (după prioritate, dată, ATAS/STAS);
- acces la istoricul de comunicare pe fiecare tichet.

2.1.3 Diagrama logică și descrierea componentelor

Din punct de vedere logic, un monolit include toate componentele funcționale într-o singură structură. Principalele componente sunt (modulul de interfață web, modulul de gestionare a tichetelor, modulul de administrare a utilizatorilor, modulul de generare a rapoartelor și modulul de notificări). Acestea nu sunt independente, ci comunică direct între ele prin apeluri interne, fără interfețe API separate. De exemplu, atunci când este creat un tichet, modulul UI apelează direct logica de business, care actualizează baza de date și transmite informația necesară modulului de notificări, figura 2.1. Această abordare simplifică implementarea inițială, dar limitează flexibilitatea pe termen lung.

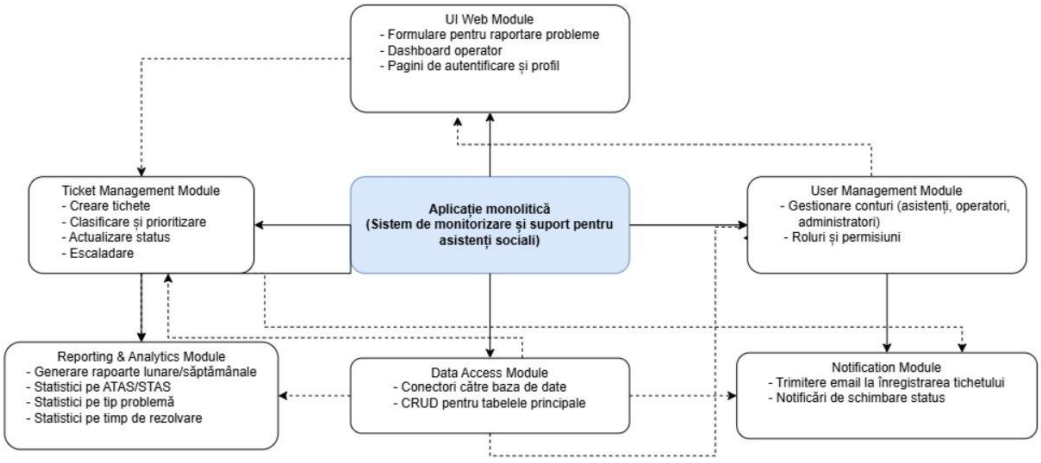

Figura 2.1 – Diagrama componentelor

2.1.4 Evaluarea necesarului de resurse

Din perspectiva unui Project Manager, arhitectura monolitică necesită o echipă redusă ca dimensiune. Efortul de dezvoltare poate fi acoperit de aproximativ cinci până la șase persoane, un Project Manager, un analist de business, doi dezvoltatori backend, un dezvoltator frontend și un inginer QA, cu un specialist DevOps doar parțial implicat. Durata estimată a proiectului până la un MVP funcțional este de aproximativ cinci până la șapte luni, incluzând analiza, dezvoltarea, testarea și pilotarea sistemului.

Din punct de vedere al infrastructurii, un monolit poate fi găzduit pe un singur server de aplicație și o singură bază de date relațională. Backup-ul se realizează prin strategii simple, copii zilnice ale bazei de date și arhivarea săptămânală a aplicației. Această infrastructură minimală permite costuri reduse la început, dar reprezintă în același timp un punct unic de eșec.

Echipa de dezvoltare (număr de persoane, roluri) [5]. Pentru un sistem monolitic de complexitate medie, necesarul estimativ ar putea fi:

- 1 Project Manager (coordonare, planificare, relația cu beneficiarii);
- 1 Business Analyst / Product Owner, detalierea cerințelor (fluxuri, câmpuri obligatorii, reguli de business);
- 2 Backend Developers, implementarea logicii de business și a accesului la bază de date;
- 1 Frontend Developer, implementarea interfețelor web pentru asistenți și operatori;
- 1 QA Engineer (Tester), testare funcțională, regresie, testare de acceptanță;
- 1 DevOps / Sysadmin (parțial), configurarea serverului, deploy, backup.

Total 6 persoane, cu un DevOps eventual part-time.

Durata estimată a proiectului. Pentru un MVP funcțional:

- analiză și specificații de 4–6 săptămâni;
- dezvoltare și testare inițială de 3–4 luni;
- pilot și ajustări de 1–2 luni.

Total estimativ între 5–7 luni pentru o versiune utilizabilă la scară națională în mod pilot. Infrastructura necesară. În arhitectura monolitică, infrastructura poate fi relativ simplă:

a) 1 server de aplicație unic (fizic sau virtual / cloud) care găzduiește:
   1) aplicația web monolitică;
   2) serverul web (ex.: Nginx/Apache/tomcat etc.).
b) 1 bază de date relațională (ex.: PostgreSQL, MySQL, SQL Server), poate fi pe același server sau pe o instanță separată, în funcție de încărcare.

Mecanisme de backup:

- backup zilnic al bazei de date;
- backup săptămânal al aplicației;
- plan de restaurare în caz de defecțiune.

2.1.5 Estimarea costurilor și a riscurilor operaționale

Costurile unei arhitecturi monolitice sunt reduse în etapa inițială, atât la nivel de resurse umane, cât și de infrastructură. Echipa este mai mică, serverele sunt mai puține, iar procesele DevOps sunt mai simple. Însă riscurile operaționale cresc odată cu extinderea sistemului. Monolitul poate deveni dificil de întreținut și modernizat, deoarece orice modificare impune actualizarea întregii aplicații. În același timp, dependența de un singur server înseamnă că orice defecțiune afectează întregul sistem, iar costurile de recuperare în caz de avarie pot fi ridicate. Un alt risc major este dependența de echipa inițială (codul monolitic, odată extins, devine greu de înțeles de către noi dezvoltatori), crescând costurile de mentenanță.

Tabelul 2.1 privind *costurile de dezvoltare pentru o perioadă de șase luni*, prezintă defalcarea detaliată a bugetului necesar pentru construirea versiunii inițiale a sistemului (MVP) [7]. În acesta sunt enumerate rolurile implicate în proiect ca Project Manager, Business Analyst, dezvoltatori backend și frontend, specialist QA și DevOps, împreună cu nivelul salarial lunar estimat, perioada de implicare și costul total pentru fiecare resursă. Tabelul oferă o imagine clară asupra investiției inițiale necesare pentru lansarea funcționalităților de bază ale sistemului.

Tabelul 2.1 – Costuri de dezvoltare pentru 6 luni

Rol	Nr.pers	Sal. lunar(lei)	Durata (luni)	Cost total (lei)
Project Manager	1	30000	6	180000
Business Analyst	1	25000	2	50000
Backend Developer	2	32000	6	384000
Frontend Developer	1	30000	6	180000
QA Engineer	1	22000	4	88000
DevOps (part-time)	0.5	35000	2	35000
Suma totală dezvoltare(6 luni)	**-**	-	-	**917000**

Tabelul 2.2, dedicat *costurilor de infrastructură anuală,* evidențiază cheltuielile anuale asociate funcționării tehnice a sistemului. Sunt prezentate costurile pentru găzduirea serverelor (cloud), baza de date dedicată, spațiul de stocare pentru backup-uri și serviciile auxiliare precum SSL, DNS și monitorizare. Prin acest tabel se conturează necesarul financiar pentru menținerea operațională a sistemului în condiții stabile și sigure, fără a lua în calcul dezvoltarea sau modificările funcționale.

Tabelul 2.2 – Costuri infrastructură anuală

Resursă	Preț lunar (lei)	Preț anual (lei)
Server cloud (AWS/Azure/ DigitalOcean - 8GB RAM, 4CPU)	1600	19200
Bază de date dedicată (PostgreSQL)	1200	14400
Back-up automat & storage	500	6000
DNS, SSL, monitorizare	200	2400
Total infrastructură	**3500**	**42000**

În tabelul 2.3 sunt *costurile de mentenanță anuale* sintetizează resursele necesare pentru întreținerea aplicației după lansare. Acesta include salariile personalului tehnic implicat continuu, dezvoltatori backend

și frontend, QA parțial și Project Manager cu implicare redusă, prezentând cheltuiala anuală totală asociată activităților de corectare a erorilor, actualizări minore, suport tehnic și optimizări de stabilitate. Tabelul clarifică nivelul de finanțare necesar pentru asigurarea funcționării sistemului pe termen lung.

Tabelul 2.3 – Costuri de mentenanță anuală

Rol	Nr.pers	Sal. lunar(lei)	Durata (luni)	Cost anual (lei)
Backend Developer	1	32000	12	384000
Frontend Developer	1	28000	12	336000
QA Engineer (part-time)	0.5	20000	12	120000
Project Manager (10% timp)	0.1	30000	12	36000
Total mentenanță		**110000**		**876000**

Tabelul 2.4, remarcă *costurilor inițiale,* combină cheltuielile majore din primul an de viață al proiectului, dezvoltarea MVP și infrastructura tehnică necesară pentru operare. Prin agregarea acestor elemente, este prezintă valoarea totală a investiției inițiale, oferind o imagine completă asupra bugetului necesar pentru implementarea și lansarea sistemului la nivel național. Aceasta este esențială pentru estimarea competitivă a resurselor financiare în planificarea bugetară.

Tabelul 2.4 – Costuri totale pentru primul an

Categoria	Cost (lei)
Dezvoltare MVP (6 luni)	917000
Infrastructură 12 luni	42000
Mentenanță anul 1	876000
Total costuri pentru primul an	**1835000**

Tabelul 2.5, privind *costurile din anii următori* descrie cheltuielile operaționale recurente pentru funcționarea sistemului după primul an, excluzând dezvoltarea inițială. În tabel se regăsesc costurile anuale de mentenanță și infrastructură, arătând nivelul de finanțare necesar pentru menținerea stabilității platformei, suportul utilizatorilor și actualizările esențiale. Această analiză permite anticiparea bugetelor multianuale și evaluarea sustenabilității financiare a proiectului.

Tabelul 2.5 – Costuri anuale pentru anii urmatori

Categoria	Cost anual (lei)
Infrastructură anuală	42000
Mentenanță anuală	876000
Total costuri anuale după lansare	**918000**

Arhitectura monolitică este mai ieftină la început, dar mentenanța devine rapid un cost dominant. Costurile mari anuale vin din faptul că întreaga aplicație e greoaie, orice schimbare necesită modificări în bloc. Un monolit este bun pentru MVP, dar pe termen lung devine scump (aproape 1 milion lei/an mentenanță). Dacă proiectul evoluează și apar cereri de integrare sau încărcare mai mare, costurile de refactorizare pot depăși 2–3 milioane lei.

2.2 Arhitectura bazată pe microservicii

Principiul modularității și independenței componentelor. Arhitectura bazată pe microservicii reprezintă un model modern de proiectare a aplicațiilor, în care sistemul este împărțit în componente mici, independente și specializate, fiecare responsabilă pentru o funcționalitate bine delimitată. Principiul fundamental al acestei arhitecturi este modularitatea, care permite ca fiecare microserviciu să fie dezvoltat, testat, scalat și implementat separat, fără a afecta funcționarea altor părți ale sistemului. Fiecare serviciu are propriul său ciclu de viață, o logică de business proprie și, în multe cazuri, chiar propria bază de date. Această independență structurală asigură o flexibilitate ridicată, o scalabilitate granulară și o mentenabilitate superioară în comparație cu o arhitectură monolitică [8].

În contextul proiectului „Sistemul de monitorizare și suport pentru asistenți sociali", adoptarea unei arhitecturi bazate pe microservicii permite gestionarea eficientă a volumului mare de sesizări, precum și adaptarea sistemului la necesități viitoare, cum ar fi integrarea cu eSocial sau extinderea spre alte servicii publice. Această abordare facilitează dezvoltarea paralelă în echipe separate, reducând riscul de blocaje tehnice și permițând o distribuire clară a responsabilităților.

Descrierea componentelor pentru sistemul de evidență. Într-o arhitectură cu microservicii, aplicația este împărțită în module autonome, fiecare microserviciu acoperind o funcționalitate distinctă. Pentru sistemul de monitorizare și suport destinat asistenților sociali, structura recomandată include următoarele componente de mai jos.

Microserviciul „Asistent Social", gestionează informațiile despre utilizatorii din teritoriu, date personale, ATAS/STAS, date de contact, istoricul solicitărilor. Acest serviciu stochează profilurile asistenților și permite identificarea rapidă a acestora atunci când trimit o cerere de suport.

Microserviciul „Cerere de Suport", este centrul sistemului, responsabil de recepționarea, înregistrarea și clasificarea problemelor raportate. Microserviciul înregistrează detaliile ticheteor, atribuie automat o prioritate, gestionează statusul (nou, în lucru, escaladat, soluționat) și asigură logica principală a fluxului operațional.

Microserviciul „Operator și Management Suport", administrează activitatea operatorilor: preluarea ticheteor, actualizarea statusurilor, escaladarea către echipa tehnică, adăugarea notelor interne și gestionarea indicatorilor de performanță operațională. Acest microserviciu permite distribuirea echitabilă a ticheteor între operatori și monitorizează încărcarea fiecăruia.

Microserviciul „Autentificare și Roluri", asigură autentificarea utilizatorilor (asistenți sociali, operatori, administratori) și gestionează permisiunile. Implementarea pe un microserviciu distinct permite securitate granulară și posibilitatea integrării ulterioare cu sisteme externe de identity management [9].

Microserviciul „Notificări" trimite notificări automate prin email, SMS sau în platformă, pentru a informa utilizatorii despre schimbările de status ale solicitărilor, alocarea unui operator sau rezolvarea unui tichet. Acest serviciu funcționează independent și se activează automat la evenimentele generate de alte

microservicii.

Microserviciul „Rapoarte și Statistici”, generează rapoarte periodice privind activitatea sistemului, timp, mediu de rezolvare, volum de tichete per ATAS/STAS, categorii frecvente de probleme. Permite vizualizarea indicatorilor pentru management și oferă baza pentru luarea deciziilor administrative.

2.2.1 Diagrama logică a descrierea comunicațiilor

Diagrama logică a comunicațiilor între servicii (API gateway, REST, message broker). Într-o arhitectură cu microservicii, comunicarea dintre componente este un element central. Interacțiunea serviciilor se realizează printr-un API Gateway, care funcționează ca un punct unic de acces pentru utilizatori și pentru alte sisteme externe. API Gateway distribuie cererile către microserviciul corespunzător, gestionând autentificarea, rutarea, monitorizarea și rate limiting-ul.

Microserviciile comunică între ele folosind protocoale precum REST, pentru operațiuni sincrone (ex. cereri de date), sau prin intermediul unui message broker (precum RabbitMQ sau Kafka), pentru evenimente asincrone, cum ar fi trimiterea notificărilor sau generarea rapoartelor. Această abordare reduce dependența directă dintre servicii și crește reziliența întregului sistem, un microserviciu poate cădea temporar fără a afecta funcționarea întregii platforme.

Din perspectiva unui Project Manager, diagrama logică pentru arhitectura pe microservicii poate fi descrisă ca un set de componente principale care colaborează prin API-uri și mesagerie, figura 2.2.

API Gateway. Punct unic de intrare pentru toți utilizatorii (asistenți sociali, operatori, administratori). Primește toate cererile HTTP din exterior. Verifică autentificarea și drepturile de acces (în colaborare cu microserviciul „Autentificare și Roluri”). Rutează cererile către microserviciile corespunzătoare („Asistent Social”, „Cerere de Suport”, „Operator și Management Suport”, „Rapoarte și Statistici”, „Notificări”). Aplică politici de securitate, limitare de trafic (rate limiting) și logare la nivel de interfață publică.

Layer REST / Comunicație sincronă între microservicii. Expune endpoint-uri REST pentru fiecare microserviciu (ex.: /asistent, /cerere-suport, /operator, /rapoarte). Permite microserviciilor să solicite date unul de la celălalt (ex.: „Cerere de Suport” cere date despre asistent de la „Asistent Social”). Asigură schimbul de informații în timp real acolo unde este nevoie de răspuns imediat. Folosește formate standardizate (JSON) și convenții clare de versionare a API-urilor.

Message Broker (sistem de mesagerie asincronă). Primește evenimente generate de microservicii (ex.: „tichet creat”, „status modificat”, „notificare necesară”). Publică mesaje într-un mod decuplat, permițând mai multor servicii să asculte aceleași fluxuri de evenimente. Asigură reziliență: dacă un consumator (ex. „Notificări”) este temporar indisponibil, mesajele sunt stocate și livrate ulterior. Reduce dependențele directe între servicii și evită supraîncărcarea comunicațiilor sincrone.

Microserviciul „Autentificare și Roluri” (Auth Service). Validează credențialele utilizatorilor și emite token-uri de acces. Gestionează roluri și permisiuni (asistent social, operator, administrator, manager). Este apelat în principal de API Gateway pentru verificarea accesului. Poate fi interogat și de alte microservicii atunci când este necesară reconfirmarea identității sau a rolurilor.

Microserviciile de business (Asistent Social, Cerere de Suport, Operator, Notificări, Rapoarte). Comunică cu API Gateway pentru cererile venite din exterior. Comunică între ele prin REST atunci când au nevoie de date specifice (de ex. „Rapoarte și Statistici” cere date de la „Cerere de Suport”). Trimit evenimente către Message Broker atunci când are loc o acțiune importantă (creare tichet, schimbare status, generare raport). Consumul de mesaje din broker (în special „Notificări” și „Rapoarte și Statistici”) permite actualizări și notificări fără a încărca fluxul principal al aplicației.

Canal de integrare cu sisteme externe (ex.: eSocial). Expus prin API Gateway sau prin microservicii dedicate de integrare. Permite schimbul de date între sistemul de monitorizare și eSocial (ex.: transmiterea ID-ului de caz, sincronizarea anumitor statusuri). Folosește aceleași mecanisme REST și, la nevoie, mesaje asincrone prin broker [10].

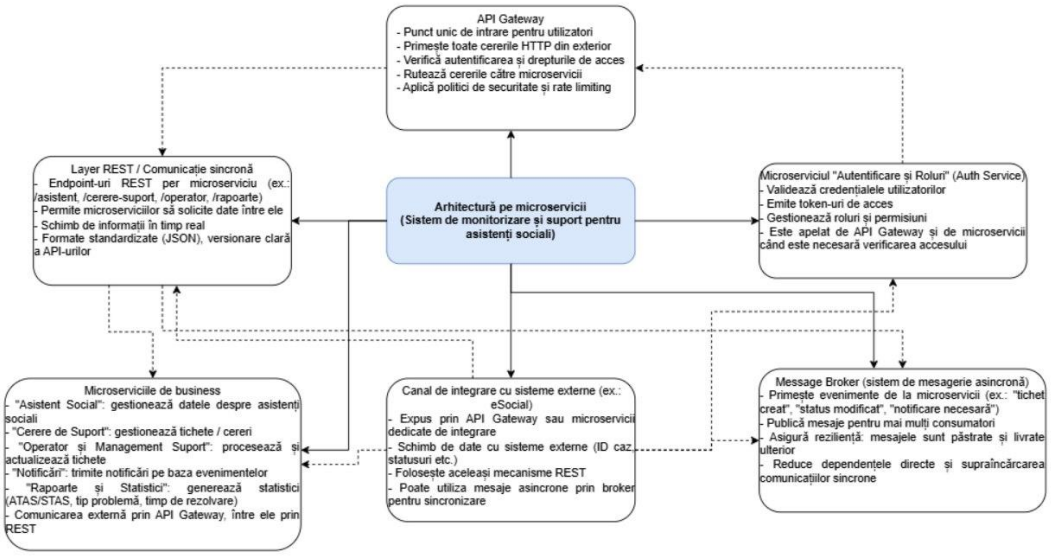

Figura 2.2 – Diagrama comunicațiilor

2.2.2 Evaluarea resurselor

Din perspectiva managementului de proiect, arhitectura cu microservicii necesită o echipă mai mare și mai specializată decât un monolit, deoarece fiecare microserviciu implică un set separat de responsabilități tehnice. Echipa include, de regulă, 2–3 dezvoltatori backend, responsabili de diferitele microservicii, 1 dezvoltator frontend, care construiește interfața aplicând principiile API-first, 1 inginer DevOps, pentru containere, orchestrare și CI/CD, 1 QA Engineer, pentru testare distribuită, 1 Project Manager și, eventual, 1 arhitect software, care coordonează interdependențele dintre servicii.

Timpul de dezvoltare este mai mare decât în cazul unui monolit pentru faza inițială, deoarece se configurează infrastructura, pipeline-urile de CI/CD, containerele și orchestrarea. Dezvoltarea unui MVP

bazat pe microservicii poate dura în medie 8–10 luni, însă ulterior extinderea devine mult mai rapidă datorită modularității. Infrastructura necesară este semnificativ mai complexă, incluzând, containere Docker pentru fiecare microserviciu, un cluster de orchestrare Kubernetes, un API Gateway, un message broker (ex. Kafka sau RabbitMQ), sisteme avansate de monitorizare și distributed tracing, pipeline-uri CI/CD pentru automatizarea testării și deploy-ului.

2.2.3 Estimarea costurilor, riscurilor și avantajelor de scalabilitate

Costurile pentru o arhitectură bazată pe microservicii sunt mai ridicate în etapa inițială. Echipele sunt mai numeroase, infrastructura este mai complexă, iar configurarea platformei necesită timp și expertiză. Un MVP bazat pe microservicii poate depăși cu 40–60% costul unei versiuni monolitice, iar infrastructura poate costa de 3–5 ori mai mult decât un singur server. Totuși, pe termen lung, mentenanța devine mai eficientă, deoarece modificările sunt locale și nu necesită redeployarea întregului sistem.

Riscurile includ complexitatea ridicată a coordonării, dependența de DevOps și necesitatea unei documentații excelente pentru API-uri. De asemenea, costurile de instruire și sincronizare a echipelor pot afecta bugetul inițial. Cu toate acestea, avantajele de scalabilitate sunt net superioare, fiecare microserviciu poate fi scalat independent în funcție de volum, permițând gestionarea a mii de cereri simultane. Sistemul devine rezilient, o eroare într-un serviciu nu afectează întregul flux operațional, iar capacitatea de integrare cu alte platforme (eSocial, CNDDCM, AGSSSI) crește semnificativ.

Tabelul 2.6 privind *costurile de dezvoltare pentru perioada de opt până la zece luni* prezintă efortul financiar necesar pentru realizarea versiunii inițiale a sistemului bazat pe microservicii. Acesta include roluri specializate precum arhitectul software, inginerul DevOps și un număr mai mare de dezvoltatori backend, reflectând complexitatea ridicată a acestei arhitecturi. Pentru fiecare rol sunt indicate salariile lunare, durata implicării și costul total, oferind o imagine completă asupra investiției necesare pentru construirea MVP-ului în arhitectura distribuită.

Tabelul 2.6 – Costuri de dezvoltare pentru 9 luni

Rol	Nr.pers	Sal. lunar(lei)	Durata (luni)	Cost total (lei)
Project Manager	1	30000	9	270000
Arhitect Software	1	40000	6	240000
Business Analyst	1	25000	3	75000
Backend Developers	3	35000	9	945000
Frontend Developer	1	32000	9	288000
QA Engineer	1	22000	8	176000
DevOps Engineer	1	40000	9	360000
Suma total dezvoltare(9 luni)				**2354000**

Tabelul 2.7, dedicat *costurilor anuale de infrastructură* detaliază bugetul necesar pentru operarea tehnică a unui sistem distribuit. Sunt incluse aici costuri precum, clusterul Kubernetes, bazele de date separate pentru microservicii, message brokerul, API Gateway-ul, sistemele de monitorizare și componenta

de securitate. Acest tabel evidențiază diferențele majore de infrastructură dintre o arhitectură monolitică și una pe microservicii, prezentând investiția anuală necesară pentru asigurarea unei platforme reziliente și scalabile.

Tabelul 2.7 – Costuri de infrastructură anuală

Resursă	Preț lunar (lei)	Preț anual (lei)
Cluster Kubernetes (3 noduri x 2GB CPU/ram mediu)	3500	42000
Baza de date multipla (PostgreSql x 3 instanțe)	2800	33600
Message Brocker (Kafka/RabbitMQ gestionat)	1600	19200
API Gateway + Load Balancer	1000	12000
Logging & Monitoring (ELK / Grafana Cloud)	2000	24000
Backup & Storage	800	9600
DNS, SSL, securitate	300	3600
Total infrastructură		**144000**

În tabelul 2.8, sunt *costurile de mentenanță anuală* reflectă resursele umane necesare pentru întreținerea sistemului după lansare. Arhitectura microservicii necesită o echipă tehnică stabilă, incluzând cel puțin doi dezvoltatori backend, un dezvoltator frontend, un QA dedicat și un inginer DevOps cu rol continuu. Costurile prezentate includ salariile acestora pe parcursul unui an și ilustrează volumul considerabil de efort necesar pentru a menține un sistem distribuit, format din multiple componente autonome.

Tabelul 2.8 – Costuri de mentenanță anuală

Rol	Nr.pers	Sal. lunar(lei)	Durata (luni)	Cost anual (lei)
Backend Developer	2	35000	12	840000
Frontend Developer	1	30000	12	360000
QA (full-time)	1	22000	12	264000
DevOps Engineer	1	40000	12	480000
Project Manager (10% timp)	0.1	30000	12	36000
Total mentenanță				**1980000**

Tabelul 2.9, oferă o imagine clară asupra *costurilor inițiale* pentru implementarea arhitecturii bazate pe microservicii. Acesta combină costurile de dezvoltare ale MVP-ului cu cheltuielile pentru infrastructură și mentenanță din primul an. Prin agregarea acestor valori, tabelul evidențiază investiția totală necesară pentru lansarea sistemului într-un mediu operațional complet funcțional, demonstrând impactul financiar semnificativ al adoptării unei arhitecturi distribuite încă din primul an de implementare.

Tabelul 2.9 – Costuri totale pentru primul an

Categoria	Cost (lei)
Dezvoltare MVP (6 luni)	2354000
Infrastructură 12 luni	144000
Mentenanță anul 1	1980000
Total costuri pentru primul an	**4478000**

Tabelul 2.10, sintetizează *cheltuielile recurente anuale* ale unui sistem de tip microservicii, excluzând dezvoltarea inițială. Aceste costuri includ mentenanța completă și infrastructura necesară pentru asigurarea performanței, securității și disponibilității sistemului. Tabelul evidențiază că, deși costurile de

dezvoltare sunt eliminate în anii următori, mentenanța și infrastructura rămân considerabil mai mari decât în cazul unei arhitecturi monolitice, reflectând complexitatea unui sistem compus din multiple servicii autonome.

Tabelul 2.10 – Costuri anuale pentru anii urmatori

Categoria	Cost anual (lei)
Infrastructură anuală	144000
Mentenanță anuală	1980000
Total costuri anuale după lansare	**2124000**

Microserviciile sunt de aproape 2,5 ori mai scumpe în primul an decât monolitul. Costurile de mentenanță sunt mari deoarece fiecare microserviciu necesită testare și deploy separat. Infrastructura este costisitoare (Kubernetes, broker, monitorizare), dar oferă fiabilitate și scalabilitate. Pe termen lung, microserviciile reduc costurile de refactorizare masivă și riscurile de downtime. Pentru proiecte naționale mari (ex. eSocial), microserviciile devin opțiunea preferată.

3 DOMENIUL APLICATIV

Transformarea digitală a administrației publice și a serviciilor sociale a devenit o necesitate în majoritatea țărilor, inclusiv în Republica Moldova. Pe măsură ce instituțiile statului gestionează volume tot mai mari de date, interacționează cu mii de utilizatori și procesează solicitări în timp real, presiunea asupra sistemelor informatice crește considerabil. Domeniile precum protecția socială, sănătatea, educația, serviciile publice locale sau asistența în situații de urgență au nevoie de soluții informatice stabile, scalabile și ușor de întreținut, capabile să răspundă cerințelor operative ale personalului din teren.

Într-un astfel de context instituțional, arhitecturile software devin esențiale. Structura unui sistem informatic influențează nu doar performanța și disponibilitatea lui, ci și modul în care instituțiile publice pot acționa rapid, pot colabora între ele și pot adapta soluțiile tehnice la schimbările legislative și sociale. Pentru domeniul social, unde informațiile sunt sensibile, fluxurile de lucru sunt complexe, iar beneficiarii depind de procesări rapide și corecte, alegerea unei arhitecturi adecvate devine un element critic în succesul unui proiect digital.

Un exemplu ilustrativ este implementarea sistemului eSocial în Republica Moldova, un efort național de digitalizare a serviciilor sociale. Peste 1500 de asistenți sociali comunitari, organizați la nivelul ATAS și STAS, au trecut de la dosare fizice la formulare digitale și baze de date integrate. Acest salt tehnologic a îmbunătățit considerabil eficiența serviciilor sociale, dar a scos la iveală și noi provocări, blocaje tehnice, dificultăți în utilizarea platformei, erori de sincronizare sau lipsa unui suport uniform și centralizat. Aici se conturează nevoia unui „Sistem de monitorizare și suport pentru asistenții sociali”, care să asigure un flux continuu de asistență tehnică, gestionare a incidentelor, comunicare rapidă între operatori și utilizatori, precum și capacitatea de a genera rapoarte precise privind problemele recurente.

Sisteme de acest tip nu sunt specifice doar domeniului social. În sănătate, platforme precum MyChart sau NHS Care Records folosesc arhitecturi distribuite pentru a permite accesul simultan a milioane de pacienți și cadre medicale, în timp ce suportul tehnic și sistemele de ticketing sunt gestionate automat prin microservicii. În sectorul fiscal, platformele de administrare a taxelor utilizează microservicii pentru a gestiona independent zone precum autentificarea, plățile, raportarea fiscală și evidența utilizatorilor. În educație, platformele universitare moderne folosesc arhitecturi distribuite pentru a separa gestionarea studenților, programarea cursurilor, evaluările și rapoartele academice.

Motivele pentru care aceste domenii preferă arhitectura bazată pe microservicii sunt clare, sistemele publice sunt în continuă expansiune, numărul de utilizatori fluctuează, cerințele legislative se schimbă frecvent, iar disponibilitatea sistemului trebuie să fie aproape permanentă. Microserviciile permit scalarea independentă a componentelor, de exemplu, modulul de autentificare poate suporta un număr mare de accesări, în timp ce modulul de rapoarte poate rula procese mai complexe în fundal fără a afecta experiența

utilizatorilor. De asemenea, instituțiile pot adăuga funcționalități noi fără a întrerupe întregul sistem, lucru imposibil sau foarte dificil de realizat într-o arhitectură monolitică.

Chiar și companiile care operează la scară globală oferă un exemplu clar al aplicabilității acestor arhitecturi. **Amazon** a renunțat la monolitul său inițial din cauza dificultății de a-l scala și de a-l actualiza odată cu creșterea afacerii. Trecerea la microservicii le-a permis să ofere performanță mai mare, să reducă timpii de răspuns și să lanseze funcționalități noi în ritm accelerat. **Netflix**, în momentul în care platforma a fost extinsă în zeci de țări, a adoptat microserviciile pentru a gestiona independent streamingul video, autentificarea, recomandările și monitorizarea sesiunilor. În mod similar, **Uber** și-a restructurat întreg sistemul într-un set de microservicii autonome pentru a gestiona cursele, plățile, maparea, notificările și suportul tehnic, eliminând blocajele din arhitectura monolitică inițială [11], [12].

Aplicând aceste observații la domeniul social, devine evident că un sistem precum „Sistemul de monitorizare și suport pentru asistenți sociali” poate beneficia enorm de principiile distribuite, fie prin scalarea microserviciului de înregistrare a solicitărilor în perioade de vârf, fie prin izolarea microserviciilor critice, precum cel de autentificare sau cel de notificări. Totuși, este important de remarcat că arhitectura monolitică rămâne adecvată în fazele incipiente ale proiectelor sau în situațiile în care bugetul și timpul sunt limitate, oferind o implementare rapidă și costuri reduse. Alegerea finală dintre un monolit și o arhitectură compusă din microservicii depinde de contextul instituțional, de volumul de utilizatori, de cerințele funcționale și de perspectiva de extindere pe termen lung.

În ansamblu, domeniul aplicativ al acestei lucrări reflectă realitatea digitalizării administrației publice și nevoia tot mai mare de sisteme adaptabile, reziliente și capabile să evolueze în ritmul schimbărilor instituționale și sociale. Arhitecturile software moderne, fie ele monolitice sau bazate pe microservicii, devin elemente strategice în construcția acestor soluții, iar alegerea lor trebuie realizată cu atenție, în funcție de necesitățile specifice ale domeniului și ale utilizatorilor finali.

3.1 Context instituțional și necesități funcționale

În Republica Moldova, sistemul de asistență socială reprezintă o componentă esențială a protecției populației vulnerabile, gestionând un volum mare de cazuri, evaluări, anchete și intervenții sociale. Asistenții sociali comunitari, peste 1500 la nivel național, activează în structuri locale și raionale, efectuând monitorizări, completând dosare, gestionând documentații și menținând legătura cu alte instituții responsabile. În ultimii ani, digitalizarea proceselor a devenit o prioritate strategică, odată cu introducerea sistemului național eSocial, care înlocuiește dosarele fizice ale beneficiarilor cu evidență digitală centralizată.

În acest context, apare necesitatea unui sistem de monitorizare și suport tehnic dedicat asistenților sociali. În procesul de utilizare a platformei eSocial, aceștia întâmpină frecvent dificultăți tehnice (erori la completarea dosarelor, blocaje, probleme de acces, confuzii funcționale). Fără un instrument eficient pentru

raportarea problemelor și urmărirea lor, procesul de intervenție se prelungește, afectând atât activitatea specialiștilor, cât și serviciile oferite beneficiarilor.

Sistemul de monitorizare și suport urmărește asigurarea unei comunicări rapide între asistenți și operatorii tehnici, centralizarea cererilor de suport, clasificarea și prioritizarea problemelor, minimizarea timpului de intervenție și oferirea unei evidențe clare a sesizărilor la nivel local, raional și național.

3.2 Descrierea proceselor de bază

Sistemul de monitorizare și suport include câteva procese esențiale, care definesc fluxul operațional general. **Înregistrarea solicitărilor**, asistentul social accesează platforma, completează formularul (ATAS, STAS, nume, email instituțional, telefon, ID-ul cazului și descrierea problemei) și transmite sesizarea către sistem. **Clasificarea și prioritizarea**, platforma, manual sau automat, atribuie fiecărei cereri un nivel de prioritate (imediată, medie sau redusă) în funcție de natura problemei și impactul asupra activității asistentului. **Alocarea către operator**, cererile sunt repartizate unui operator sau unei echipe de suport. Operatorii pot vizualiza toate solicitările active și le pot marca în lucru. **Intervenție și comunicare**, operatorul analizează problema, contactează asistentul social dacă este necesar, oferă instrucțiuni sau escaladează cererea către echipa tehnică superioară. **Actualizarea statusului**, pe parcursul procesului, statusul solicitării este modificat (Nou, În lucru, În așteptare, Rezolvat), păstrând un istoric complet al acțiunilor. **Generarea rapoartelor și** statisticilor, sistemul creează automat rapoarte privind numărul total de solicitări, timpul de rezolvare, centrele cu cele mai multe probleme sau tipologiile de erori frecvente.

Cerințe funcționale și nefuncționale ale sistemului

Cerințe funcționale:

- posibilitatea de autentificare a utilizatorilor pe baza rolurilor (asistent social, operator, administrator);
- formulare pentru raportarea problemelor, cu câmpuri obligatorii;
- dashboard operațional pentru operatorii de suport;
- mecanism de clasificare automată/manuală;
- flux de gestionare a statusurilor ticketelor;
- sistem de notificări prin email;
- modul de raportare și analiză vizuală;
- integrare potențială cu alte sisteme guvernamentale (eSocial, AGSSSI).

Cerințe nefuncționale:

- disponibilitate ridicată, dat fiind numărul mare de utilizatori distribuiți în teritoriu;
- timp de răspuns scurt pentru operațiunile critice (ex. înregistrarea unei cereri);
- securitate ridicată, mai ales în ceea ce privește datele personale și informațiile sensibile ale beneficiarilor;
- scalabilitate pentru a suporta viitoare creșteri ale numărului de utilizatori;

- ușurință în utilizare pentru persoane cu competențe digitale diverse;
- logare și trasabilitate completă a acțiunilor efectuate în sistem.

3.3 Analiza scenariilor de implimentare

Implementare locală (on-premises). O implementare locală presupune instalarea sistemului pe servere gestionate de instituțiile naționale. Acest model oferă control total asupra datelor, ceea ce poate fi important în domeniul social. Totuși, implică costuri semnificative pentru mentenanță, administrarea hardware-ului, echipă tehnică dedicată și actualizări periodice. Scalabilitatea poate deveni o provocare, mai ales în cazul unei arhitecturi monolitice.

Implementare în cloud. Cloud-ul permite scalare elastică, costuri predictibile și o infrastructură modernă, administrată de furnizor. Pentru microservicii, cloud-ul este mediul ideal, oferind suport nativ pentru containere, orchestrare și monitorizare distribuită. De asemenea, disponibilitatea și securitatea sunt garantate prin mecanisme moderne certificate internațional.

Extindere ulterioară. Indiferent de arhitectura aleasă, sistemul trebuie să permită extinderea către:
- noi centre teritoriale,
- noi categorii de utilizatori (ex. psihologi, asistenți medicali comunitari),
- integrarea cu sisteme naționale suplimentare,
- noi module (ex. audit intern, managementul indicatorilor de performanță).

Arhitectura monolitică permite extinderea rapidă pe termen scurt, însă microserviciile oferă flexibilitatea necesară pentru extinderi continue fără întreruperi sau rescriere masivă a codului.

CONCLUZII

Analiza comparativă realizată asupra arhitecturilor monolitică și bazată pe microservicii a evidențiat diferențe fundamentale în ceea ce privește modul de proiectare, dezvoltare, mentenanță și scalare a unui sistem informatic complex, precum „Sistemul de monitorizare și suport pentru asistenții sociali”. Datele obținute arată că, deși ambele arhitecturi sunt viabile în anumite contexte, ele servesc obiective diferite și oferă avantaje distincte în funcție de maturitatea proiectului, resursele disponibile și necesitățile instituționale pe termen lung.

În cazul arhitecturii monolitice, rezultatele indică o fezabilitate ridicată în fazele timpurii ale proiectului. Simplitatea dezvoltării, numărul redus de programatori necesari, infrastructura minimă și costurile inițiale scăzute fac din monolit o opțiune potrivită pentru realizarea unui MVP sau a unui prototip funcțional. Totuși, dezavantajele devenite vizibile pe măsură ce sistemul crește, scalabilitate limitată, mentenanță dificilă, timpi mari de redeploy și vulnerabilitatea ridicată la erori, reduc atractivitatea acestui model în implementările reale de nivel național. Concluzia este că monolitul rămâne fezabil pe termen scurt, dar insuficient pentru un sistem care trebuie să deservească mii de utilizatori simultan și care necesită extindere continuă.

Arhitectura bazată pe microservicii oferă, în schimb, o flexibilitate substanțial mai mare. Rezultatele analizelor tehnice și economice arată că microserviciile permit scalarea independentă a componentelor, actualizări fără întreruperea întregului sistem, reziliență operațională și o capacitate ridicată de integrare cu alte platforme guvernamentale. Costurile inițiale sunt semnificativ mai mari, atât în ceea ce privește resursele umane, cât și infrastructura, însă investiția se amortizează în timp prin reducerea riscurilor, a blocajelor și a nevoii de refactorizări majore. Din perspectiva fezabilității pe termen lung, microserviciile sunt mai potrivite pentru proiectele care trebuie să evolueze dinamic și să suporte fluxuri de date mari, precum sistemul de suport pentru asistenții sociali.

Pe baza tuturor criteriilor analizate, tehnice, economice și organizaționale, arhitectura recomandată pentru implementarea reală este cea bazată pe microservicii. Aceasta răspunde mai bine cerințelor instituționale din domeniul social, unde volumele de date sunt ridicate, numărul de utilizatori este mare, cerințele se modifică frecvent, iar fiabilitatea sistemului este crucială. De asemenea, distribuția componentelor pe microservicii permite extinderea platformei cu noi module, precum integrarea cu eSocial, raportare avansată sau gestionarea centrelor de plasament, fără a afecta funcționalitățile existente.

În procesul de analiză au fost identificate și o serie de lecții importante. În primul rând, arhitectura software nu trebuie aleasă exclusiv după criterii tehnice, ci și în funcție de capacitatea instituțională, disponibilitatea personalului specializat, buget și orizontul de timp al proiectului. În al doilea rând, arhitectura monolitică nu trebuie exclusă complet, deoarece poate reprezenta o etapă intermediară valoroasă

pentru prototipare și validare inițială. În al treilea rând, implementarea microserviciilor necesită o cultură organizațională matură, disciplină tehnică și o infrastructură DevOps solidă.

Direcțiile viitoare de dezvoltare includ automatizarea proceselor interne, introducerea tehnologiilor de tip event-driven, extinderea integrărilor cu sistemele naționale de protecție socială, migrarea progresivă spre cloud și adoptarea unor mecanisme avansate de securitate și audit. În plus, folosirea inteligenței artificiale pentru detectarea tiparelor de erori, predicția blocajelor sau optimizarea timpilor de răspuns poate transforma sistemul într-o componentă strategică a administrației publice.

În concluzie, alegerea arhitecturii potrivite reprezintă fundamentul unui proiect informatic durabil și eficient. Pentru un domeniu atât de sensibil și dinamic precum protecția socială, o arhitectură modernă, scalabilă și modulară, precum cea bazată pe microservicii, oferă premisele necesare pentru un sistem stabil, sigur și pregătit pentru evoluții viitoare.

BIBLIOGRAFIE

[1] “Microservices vs. monolithic architecture | Atlassian.” Accessed: Nov. 12, 2025. [Online]. Available: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith

[2] “Monolithic vs Microservices Architecture: Pros and Cons for 2025 - Scalo.” Accessed: Nov. 12, 2025. [Online]. Available: https://www.scalosoft.com/blog/monolithic-vs-microservices-architecture-pros-and-cons-for-2025/

[3] “Advantages and Disadvantages of Microservices: A Guide for 2025.” Accessed: Nov. 16, 2025. [Online]. Available: https://apiko.com/blog/advantages-of-microservices/

[4] “Monolithic vs Microservices: Features, Pros & Cons, and Real-World Use Cases | HatchWorks AI.” Accessed: Nov. 16, 2025. [Online]. Available: https://hatchworks.com/blog/software-development/monolithic-vs-microservices/

[5] “Монолитная и микросервисная — разница между архитектурами разработки программного обеспечения — AWS.” Accessed: Nov. 16, 2025. [Online]. Available: https://aws.amazon.com/ru/compare/the-difference-between-monolithic-and-microservices-architecture/

[6] “Common web application architectures - .NET | Microsoft Learn.” Accessed: Nov. 18, 2025. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures

[7] “How Should You Build Products Users Actually Want? Here’s An MVP Development Strategy You Should Know.” Accessed: Nov. 18, 2025. [Online]. Available: https://fullscale.io/blog/mvp-development-strategy/

[8] “Microservices Architecture Style - Azure Architecture Center | Microsoft Learn.” Accessed: Nov. 18, 2025. [Online]. Available: https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices

[9] “Authentication and authorization in a microservice architecture: Part 2 - Authentication.” Accessed: Nov. 18, 2025. [Online]. Available: https://microservices.io/post/architecture/2025/05/28/microservices-authn-authz-part-2-authentication.html

[10] “Microservices Communication Patterns - GeeksforGeeks.” Accessed: Nov. 18, 2025. [Online]. Available: https://www.geeksforgeeks.org/system-design/microservices-communication-patterns/

[11] “Decomposing monoliths into microservices - AWS Prescriptive Guidance.” Accessed: Nov. 18, 2025. [Online]. Available: https://docs.aws.amazon.com/prescriptive-guidance/latest/modernization-decomposing-monoliths/welcome.html

[12] “Netflix Microservices Architecture Guide for Tech Professionals.” Accessed: Nov. 18, 2025. [Online]. Available: https://www.yochana.com/netflixs-evolution-from-monolith-to-microservices-a-deep-dive-into-streaming-architecture/