



UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Documentatie arhitectura bazata pe microservicii

Proiectare Software

Autori: Mirisan Octavian

Grupa: 30231

FACULTATEA DE AUTOMATICA
SI CALCULATOARE

20 Mai 2025

Cuprins

1	Enuntul problemei	3
2	Instrumente utilizate	4
2.1	Backend	4
2.2	Frontend	4
3	Justificarea limbajului de programare ales	5
4	Justificarea tipului de comunicare între microservicii ales	5
5	Diagrama de Use Case	6
6	Diagrame de activitate	7
6.1	Vizitator	7
6.2	Angajat	10
6.3	Manager	13
6.4	Administrator	13
7	Diagrame de pachete	16
7.1	Artist Service	16
7.2	Artwork Service	17
7.3	User Service	17
7.4	Sale Service	18
7.5	Frontend	18
8	Diagrame de clase	19
8.1	Artist Service	19
8.2	Artwork Service	20
8.3	User Service	20
8.4	Sale Service	21
8.5	Frontend	22
9	Diagrame entitate-relatie	23
9.1	Artist Service	23
9.2	Artwork Service	24
9.3	User Service	24
9.4	Sale Service	25
10	Diagrame de secventa	26
10.1	Vizitator	26
10.2	Angajat	32
10.3	Manager	40
10.4	Administrator	41
11	Diagrama de componente	47
12	Diagrama de dezvoltare	48
13	Descrierea Aplicatiei	49
13.1	Vizitator	50

13.2 Angajat	52
13.3 Manager	54
13.4 Administrator	56
13.5 Shared	57

1 Enuntul problemei

Problema 5

Dezvoltați o aplicație software care poate fi utilizată într-un **lanț de galerii de artă**. Aplicația va avea 4 tipuri de utilizatori: vizitator, angajat, manager și administrator.

Utilizatorii de tip **vizitator** pot efectua următoarele operații fără autentificare:

- ❖ Vizualizarea listei tuturor operelor de artă expuse în galeriile de artă sortată după anul realizării (vizualizarea include și redarea unor imagini cu operele de artă; între 1 și 3 imagini pentru fiecare operă de artă);

- ❖ Vizualizarea listei tuturor artiștilor care au expuse opere de artă în galeriile de artă (pentru fiecare artist se va afișa numele, data nașterii, locul nașterii, naționalitatea, o fotografie și lista tuturor operelor de artă realizate de artist și expuse în galeriile lanțului de galerii de artă);

- ❖ Filtrarea listei operelor de artă plastică după următoarele criterii: artist, tipul operei de artă;

- ❖ Căutarea unei opere de artă după titlu;

- ❖ Căutarea unui artist după nume.

Utilizatorii de tip **angajat** al unei galerii de artă pot efectua următoarele operații după autentificare:

- ❖ Toate operațiile permise utilizatorilor de tip vizitator;

- ❖ Operații CRUD în ceea ce privește persistența operelor de artă expuse și a artiștilor;

- ❖ Vânzarea unei opere de artă din galeria de artă la care lucrează acel angajat;

- ❖ Salvare liste cu situația operelor de artă în mai multe formate: csv, json, xml, doc.

Utilizatorii de tip **manager** al unei galerii de artă pot efectua următoarele operații după autentificare:

- ❖ Toate operațiile permise utilizatorilor de tip angajat;

- ❖ Vizualizarea unor statistici legate de operele de artă utilizând grafice (structură radială, structură inelară, de tip coloană, etc.).

Utilizatorii de tip **administrator** pot efectua următoarele operații după autentificare:

- ❖ Toate operațiile permise utilizatorilor de tip vizitator;

- ❖ Operații CRUD pentru informațiile legate de utilizatorii care necesită autentificare;

- ❖ Vizualizarea listei utilizatorilor care necesită autentificare și salvarea acestora în format csv;

- ❖ Filtrarea listei utilizatorilor după tipul utilizatorilor;

- ❖ Notificarea fiecărui utilizator care necesită autentificare prin cel puțin 2 variante (email, SMS, WhatsApp, Skype, etc.) la orice modificare a informațiilor de autentificare aferente aceluui utilizator.

Interfața grafică a aplicației client va fi disponibilă în cel puțin 3 limbi de circulație internațională.

Figura 1: Enunt

2 Instrumente utilizate

2.1 Backend

Pentru partea de backend, am folosit **ASP.NET Core**, un framework modern, open-source, dezvoltat de Microsoft, care permite construirea de aplicații web robuste, performante și ușor de întreținut. Alegerea acestuia s-a bazat în principal pe suportul foarte bun pentru arhitectura pe microservicii și pe integrarea facilă cu alte componente Microsoft (ex: SQL Server, Visual Studio).

Arhitectura backend-ului este organizată în mai multe microservicii independente: *ArtistService*, *ArtworkService*, *SaleService* și *UserService*. Fiecare microserviciu este o aplicație ASP.NET Core separată, care are propriul său context de bază de date, model de domeniu și interfață API REST. Aceste microservicii comunică între ele printr-un API Gateway configurat cu ajutorul bibliotecii **Ocelot**, care redirecționează automat cererile HTTP către serviciul corespunzător, pe baza configurației din fișierul `ocelot.json`.

Pentru persistența datelor, am folosit **Microsoft SQL Server**, gestionat cu ajutorul **SQL Server Management Studio (SSMS)**, o unealtă familiară și ușor de folosit. În cadrul fiecărui microserviciu am utilizat **Entity Framework Core** pentru maparea între obiectele C și tabelele din baza de date (ORM).

De asemenea, pentru notificări am folosit integrare cu **Twilio** (pentru SMS) și **SMTP** (pentru email), aceste servicii fiind folosite în cadrul *UserService* pentru a notifica utilizatorii în cazul modificării contului.

2.2 Frontend

Frontend-ul aplicației a fost realizat folosind **ASP.NET Web MVC**, care se bazează pe un model de tip *Model-View-Controller*. Am ales această tehnologie deoarece se potrivește foarte bine cu restul ecosistemului .NET, oferind o separare clară între logică, date și interfață grafică.

Aplicația MVC este separată în trei părți mari:

- **Models** – aici am definit modelele care reflectă structura datelor primite de la API-uri;
- **Controllers** – conțin logica principală pentru fiecare tip de utilizator și funcționalitate (ex: *ArtistsController*, *UsersController*, *AuthController*);
- **Views** – fișierele `.cshtml` care definesc interfața grafică. Ele sunt organizate pe funcționalități și pe tipuri de utilizatori (ex: *Visitor/Index.cshtml*, *Employee/Index.cshtml*).

Pentru comunicarea cu microserviciile backend, am implementat o serie de clase numite **ApiClient**, fiecare responsabilă cu apelarea unui anumit serviciu REST. Acest lucru păstrează controller-ele curate și ușor de întreținut.

Interfața a fost stilizată folosind frameworkul **Bootstrap**, care mi-a permis să creez rapid un design responsive, prietenos și ușor de utilizat pentru toate tipurile de utilizatori.

3 Justificarea limbajului de programare ales

Pentru implementarea acestui proiect am ales limbajul de programare **C#**, împreună cu platforma **.NET** (mai exact, **ASP.NET Core** pentru backend și **ASP.NET MVC** pentru frontend). Această alegere a fost motivată de mai multe aspecte practice și tehnice.

În primul rând, C# este un limbaj modern, puternic tipizat, orientat pe obiecte, care oferă o sintaxă clară și concisă, fiind ideal pentru dezvoltarea de aplicații structurate și scalabile. În plus, suportul foarte bun pentru programarea asincronă, prin cuvintele cheie `async` și `await`, facilitează dezvoltarea unor aplicații web eficiente și responsive, mai ales în cazul apelurilor către microservicii și baze de date.

Platforma .NET vine cu un ecosistem bine pus la punct, care include framework-uri robuste pentru crearea de API-uri REST (**ASP.NET Core**), accesarea bazelor de date (**Entity Framework Core**), dar și integrarea cu servicii externe (ex: **Twilio**, **SMTP**, etc.).

De asemenea, .NET oferă suport foarte bun pentru aplicarea principiilor din Domain-Driven Design (DDD), precum și pentru organizarea clară a codului în layere (Domain, Infrastructure, Services, Controllers), ceea ce este esențial într-o aplicație bazată pe microservicii.

În concluzie, C# alături de ASP.NET Core a fost o alegere logică și eficientă, care a permis realizarea unei aplicații modulare, ușor de testat și extins, conform cerințelor proiectului.

4 Justificarea tipului de comunicare între microservicii ales

În cadrul acestui proiect, am ales ca tip de comunicare între microservicii varianta de tip **coregrafie**. Asta înseamnă că microserviciile funcționează independent și comunică între ele direct, fără un orchestrator central care să le coordoneze explicit. Fiecare serviciu își cunoaște responsabilitățile și interacționează cu alte servicii doar atunci când este necesar, prin apeluri REST HTTP.

Alegerea coregrafiei s-a bazat pe mai multe argumente. În primul rând, acest model oferă o separare mai bună a responsabilităților. Fiecare microserviciu își gestionează propriile reguli de afaceri și baze de date, ceea ce respectă principiul de *loose coupling* și asigură o mai bună scalabilitate.

De asemenea, deoarece aplicația este de dimensiune medie și nu implică procese complexe sau tranzacții distribuite, nu a fost necesar un mecanism complicat de orchestrare. Comunicarea prin HTTP (sau REST) între microservicii, realizată printr-un API Gateway (**Ocelot**), s-a dovedit a fi suficientă pentru nevoile proiectului.

În concluzie, comunicarea de tip coregrafie a fost aleasă pentru că este mai simplă de implementat, oferă independentă fiecărui microserviciu și permite o arhitectură clară și flexibilă, care răspunde cerințelor funcționale ale proiectului fără a introduce o dependență inutilă între componente.

5 Diagrama de Use Case

Diagrama de *use case* reprezintă interacțiunile dintre utilizatori (*actori*) și funcționalitățile oferite de sistem. Este utilizată în faza de analiză pentru a evidenția cerințele funcționale, fără a intra în detaliu de implementare. Actorii sunt reprezentați grafic și sunt conectați la cazurile de utilizare relevante.

- **Vizitator** – poate vizualiza opere și artiști, fără autentificare.
- **Employee** – poate gestiona opere și artiști, și înregistra vânzări.
- **Manager** – are acces la funcționalitățile employee-ului, plus statistici.
- **Admin** – gestionează conturile de utilizator.

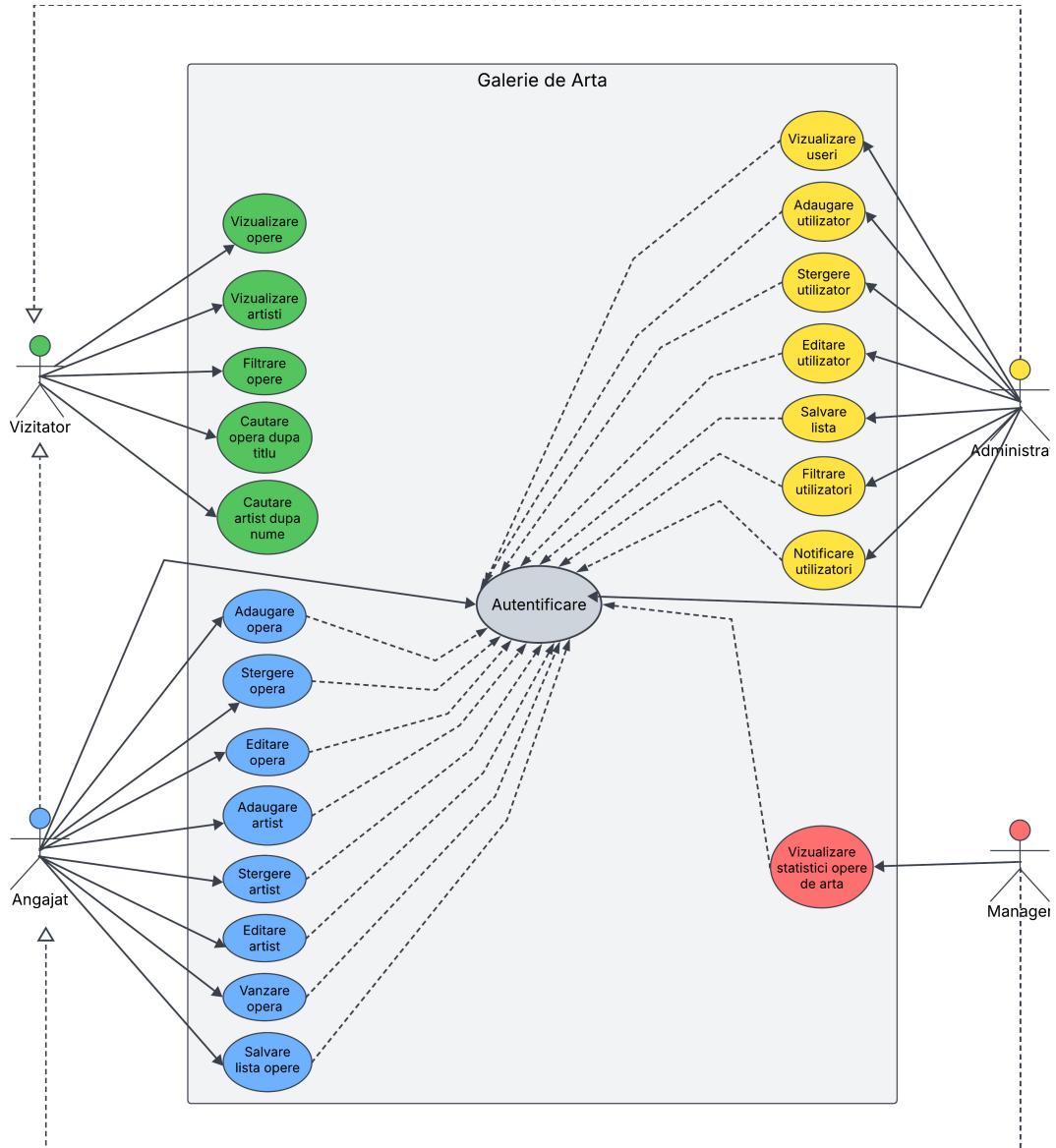


Figura 2: Diagrama de Use Case

6 Diagrame de activitate

Diagramele de activitate au fost folosite pentru a reprezenta în mod vizual pașii logici care se execută în cadrul diferitelor funcționalități din aplicație. Ele ilustrează cum curge execuția, începând de la un eveniment inițial până la finalizarea unui proces.

Acstea diagrame m-au ajutat să înțeleg mai bine succesiunea pașilor necesari într-un caz de utilizare și să identific unde ar putea apărea decizii, bifurcații sau ramuri paralele.

Fiecare diagramă de activitate este compusă din:

- **Acțiuni (activități)** – pașii efectivi realizați în proces, cum ar fi afișarea unei pagini, validarea unui formular sau salvarea unui obiect.
- **Decizii (ramuri)** – puncte în care execuția se poate bifurca în funcție de o condiție.
- **Start / Final** – marcaje care indică începutul și sfârșitul fluxului.
- **Săgeți de control** – indică ordinea în care sunt executate acțiunile.

Am realizat câte o diagramă de activitate pentru fiecare rol important din aplicație (vizitator, angajat, manager, administrator), în funcție de ce acțiuni pot realiza.

6.1 Vizitator

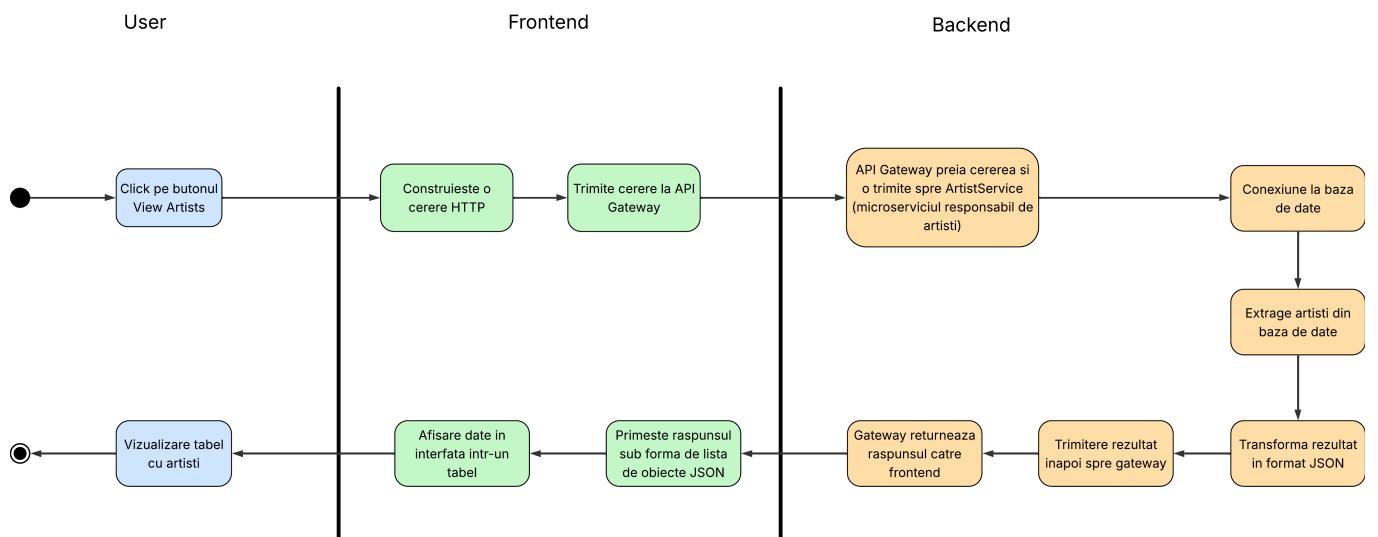


Figura 3: Vizualizare artiști

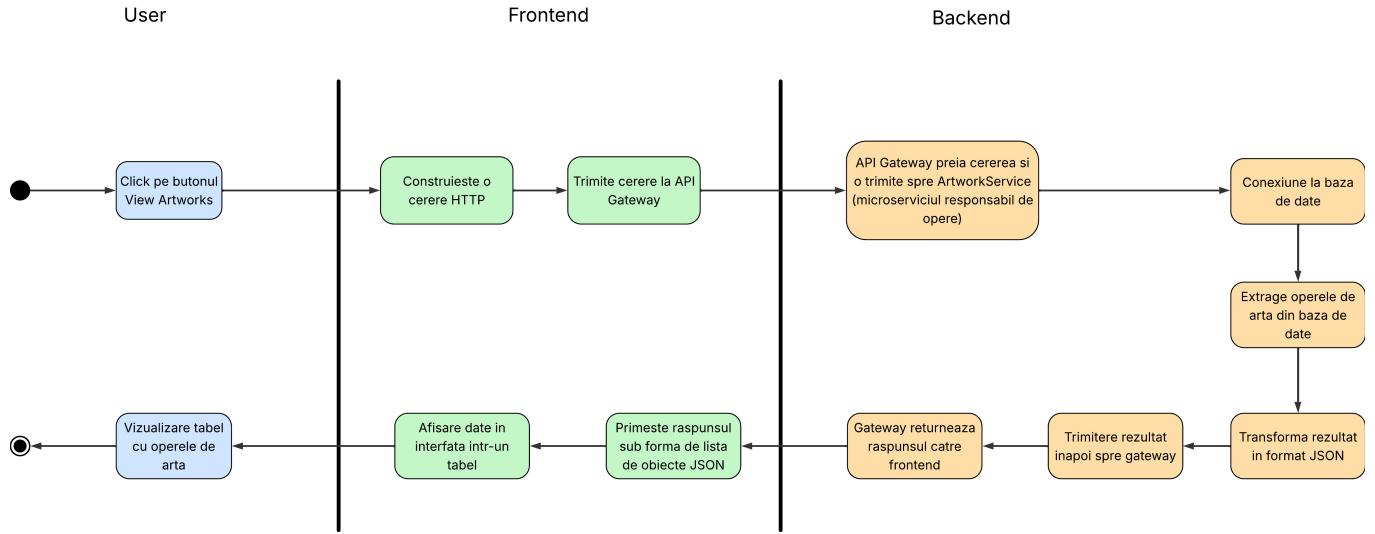


Figura 4: Vizualizare opere de arta

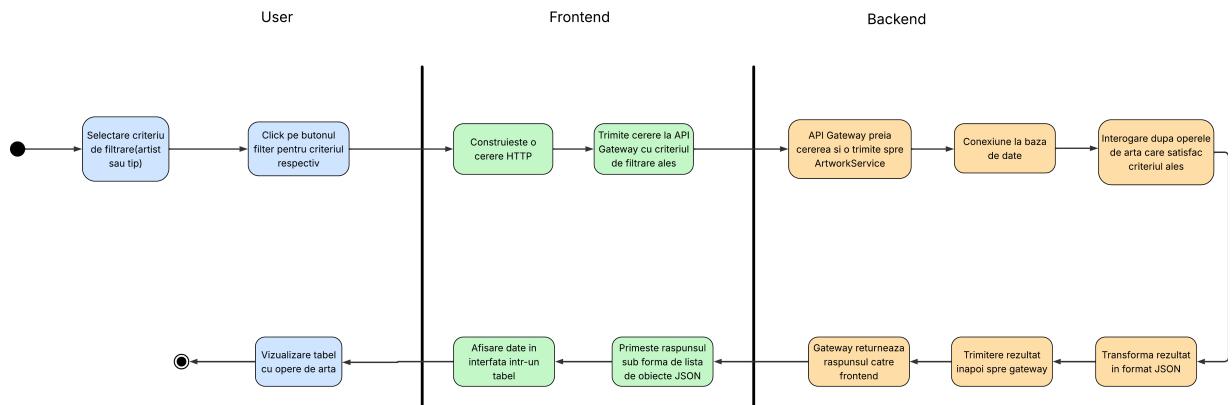


Figura 5: Filtrare opere de arta

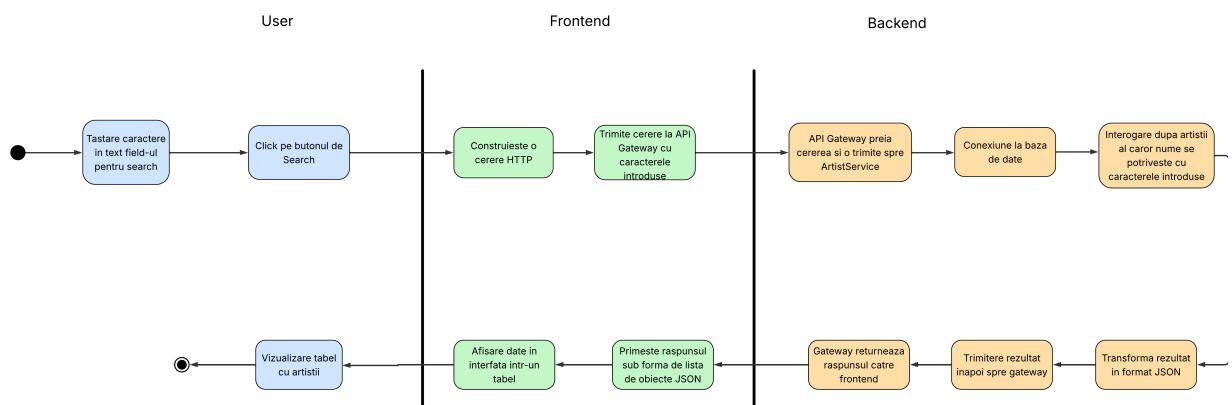


Figura 6: Cautare artisti dupa nume

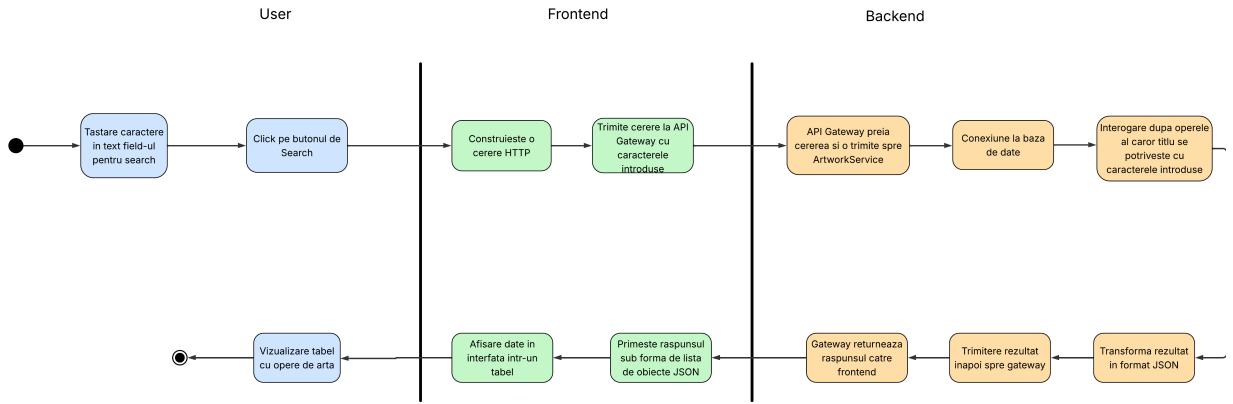


Figura 7: Cautare opere de arta dupa titlu

6.2 Angajat

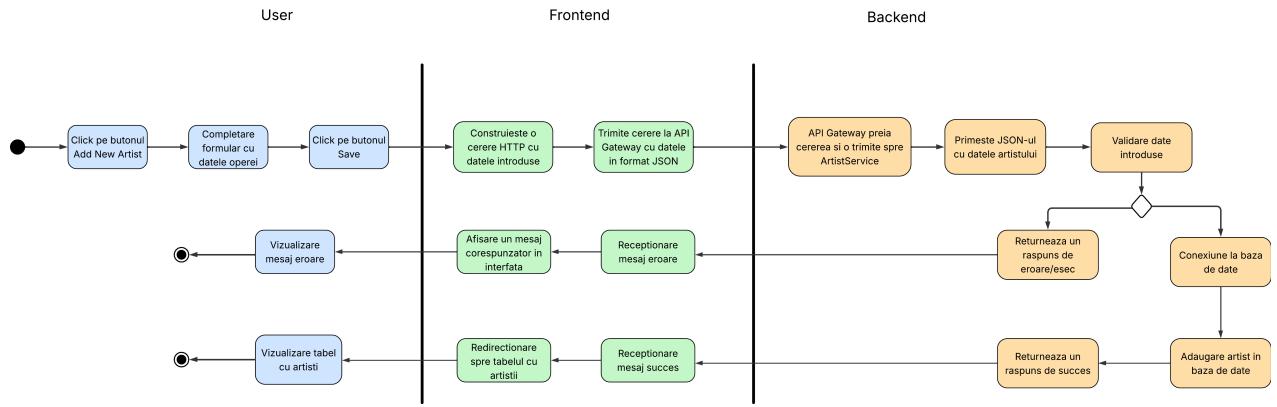


Figura 8: Adaugare artist

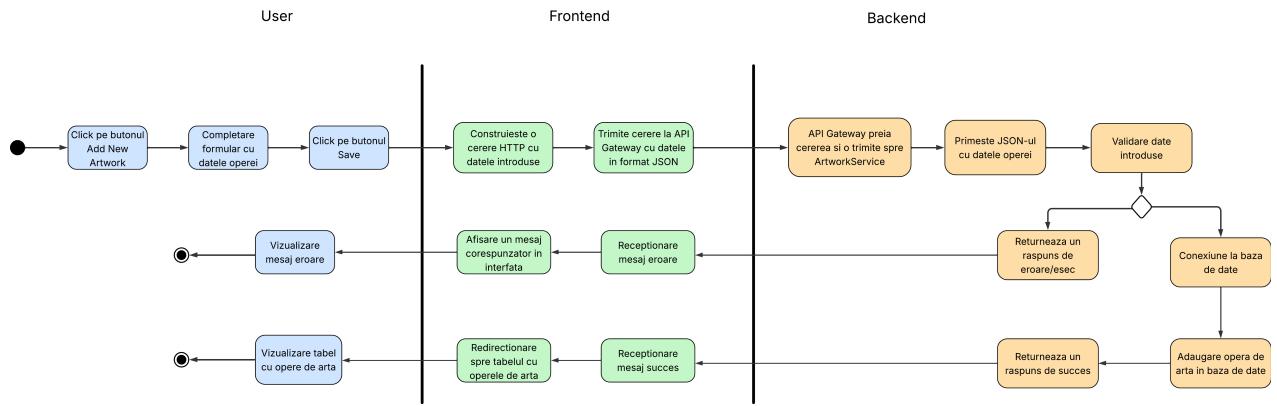


Figura 9: Adaugare opera de artă

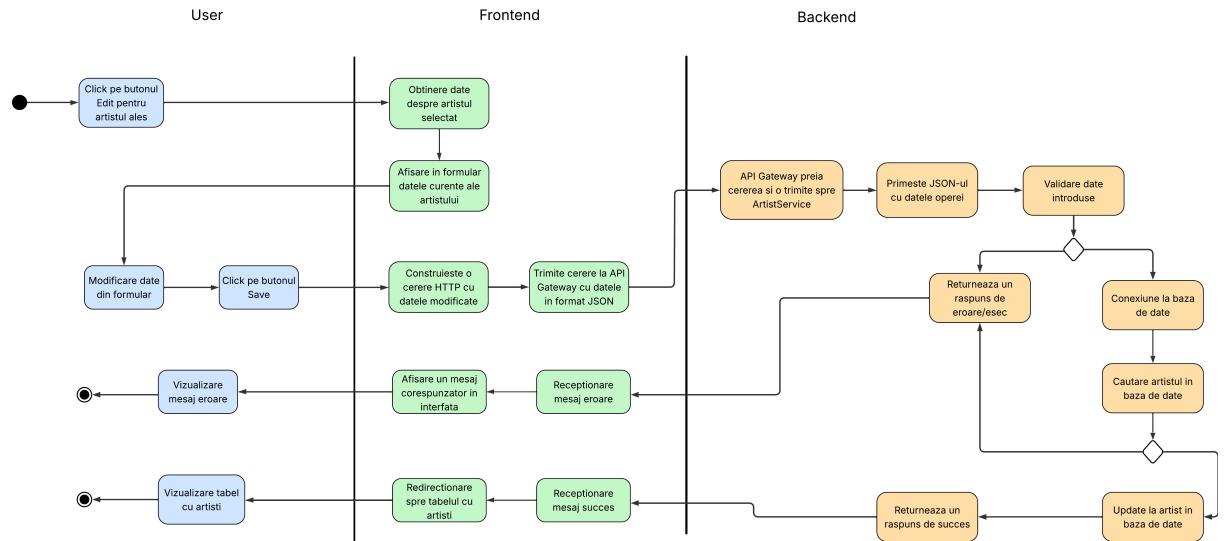


Figura 10: Editare artiști

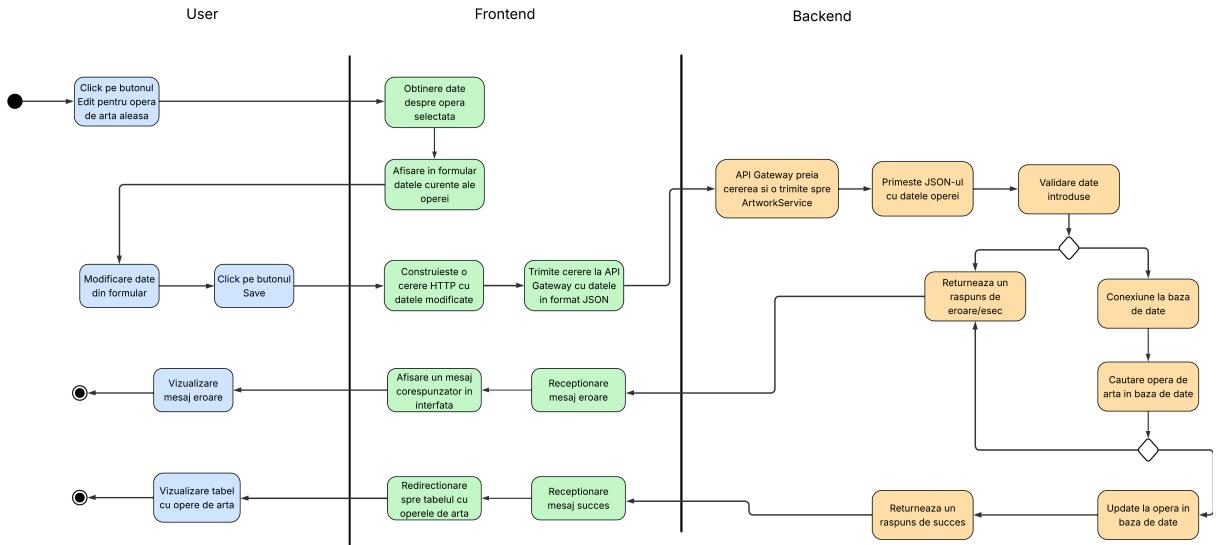


Figura 11: Editare opera de artă

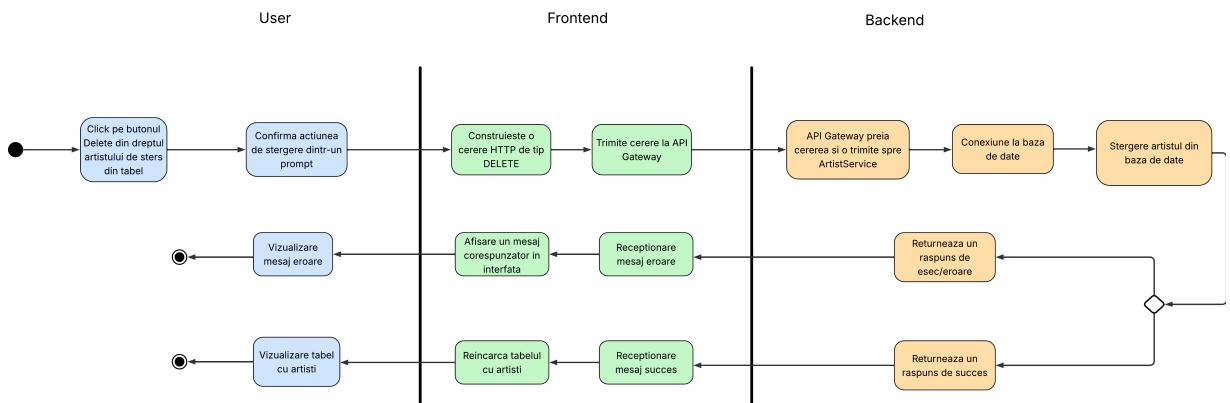


Figura 12: Stergere artist

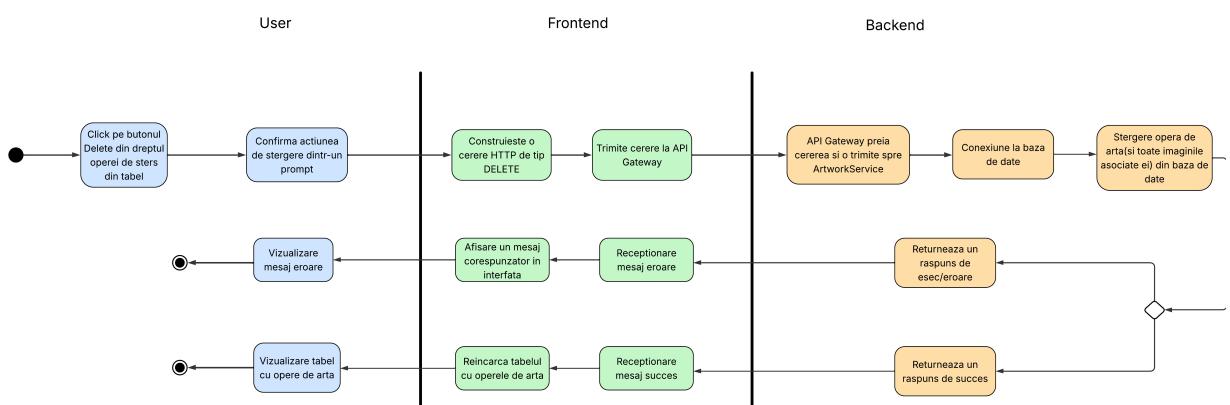


Figura 13: Stergere opera de artă

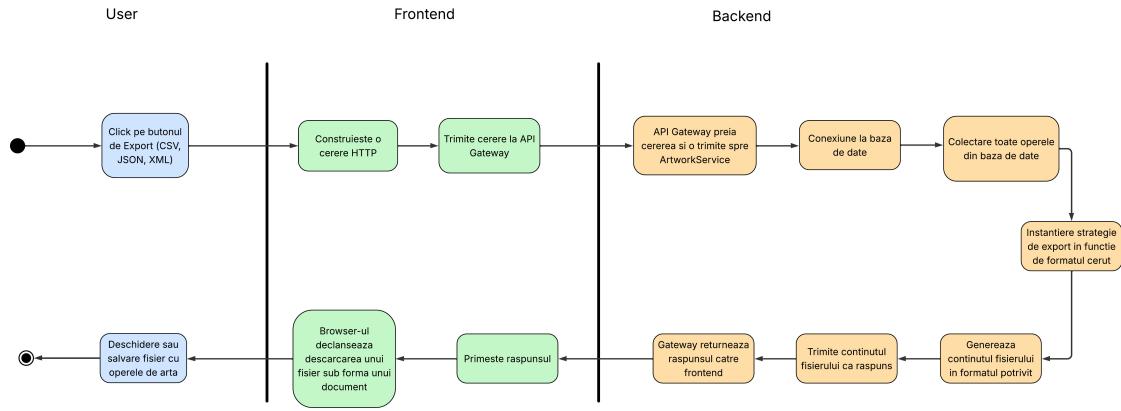


Figura 14: Salvare fisier cu operele de arta

6.3 Manager

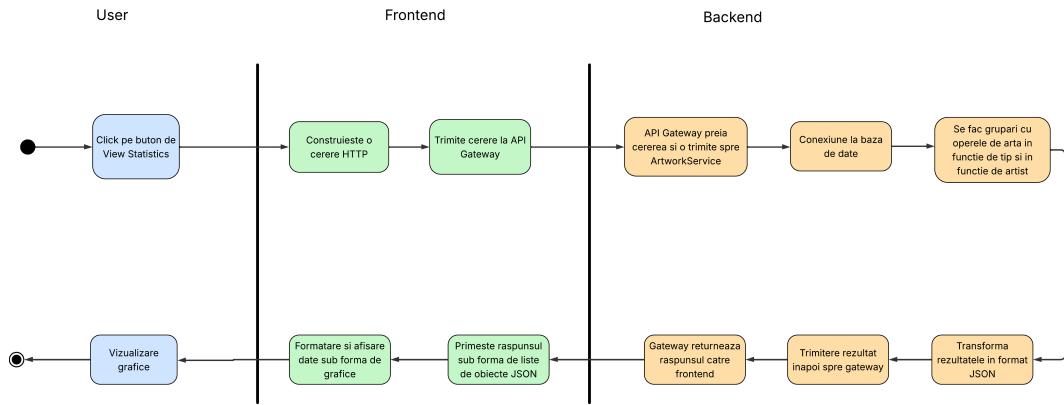


Figura 15: Vizualizare statistici legate de operele de artă

6.4 Administrator

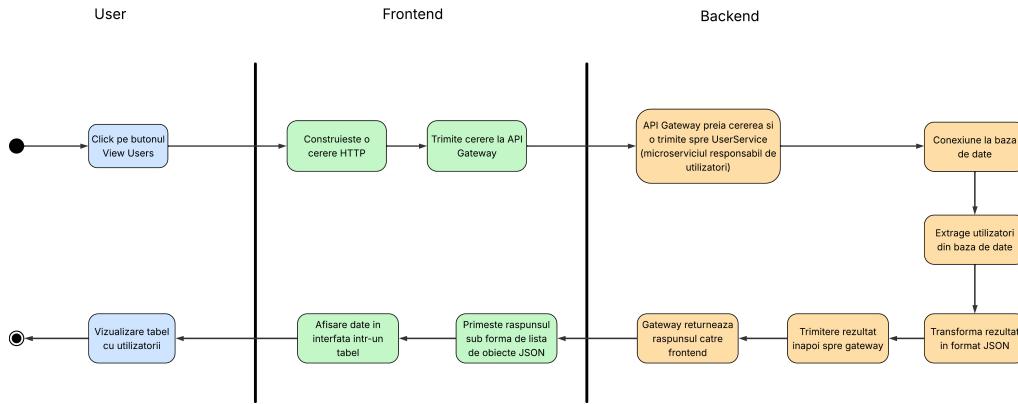


Figura 16: Cautare opere de artă după titlu

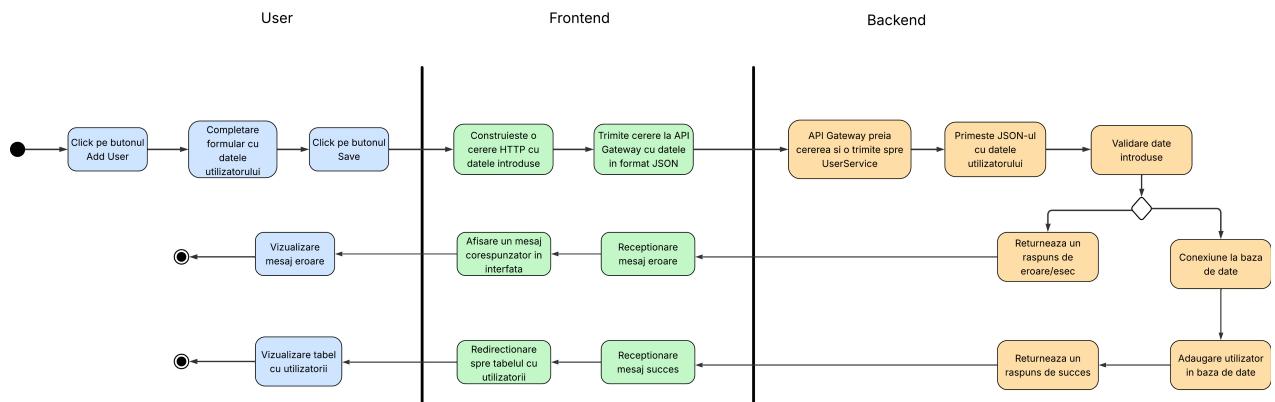


Figura 17: Cautare opere de artă după titlu

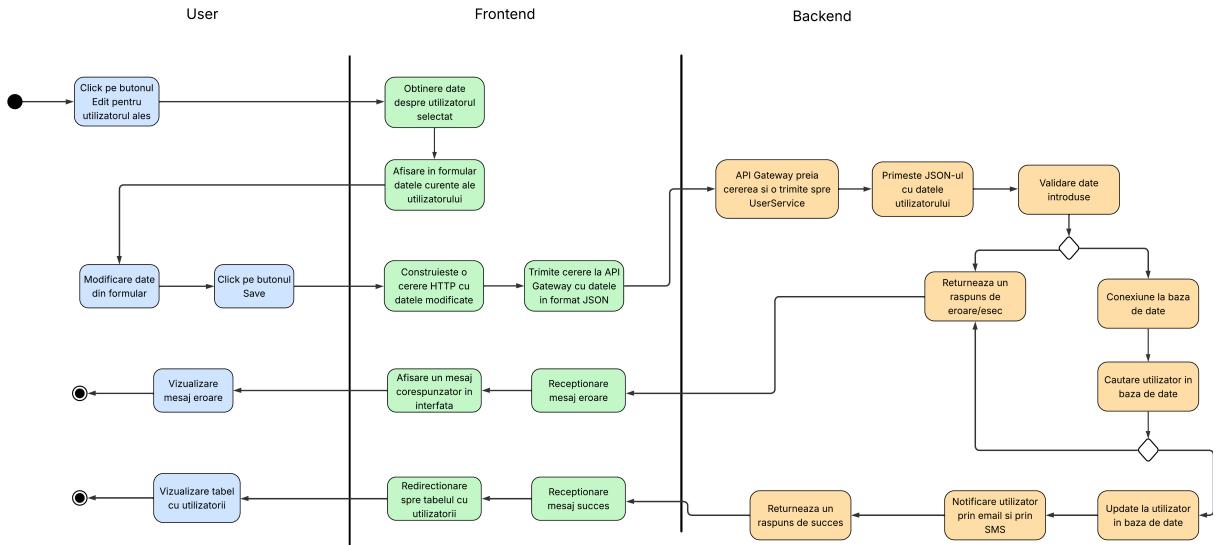


Figura 18: Cautare opere de arta dupa titlu

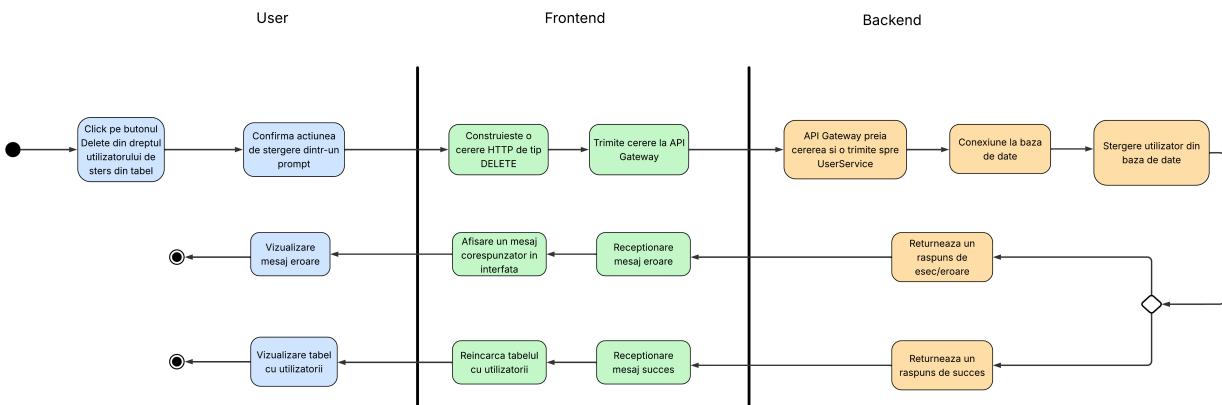


Figura 19: Cautare opere de arta dupa titlu

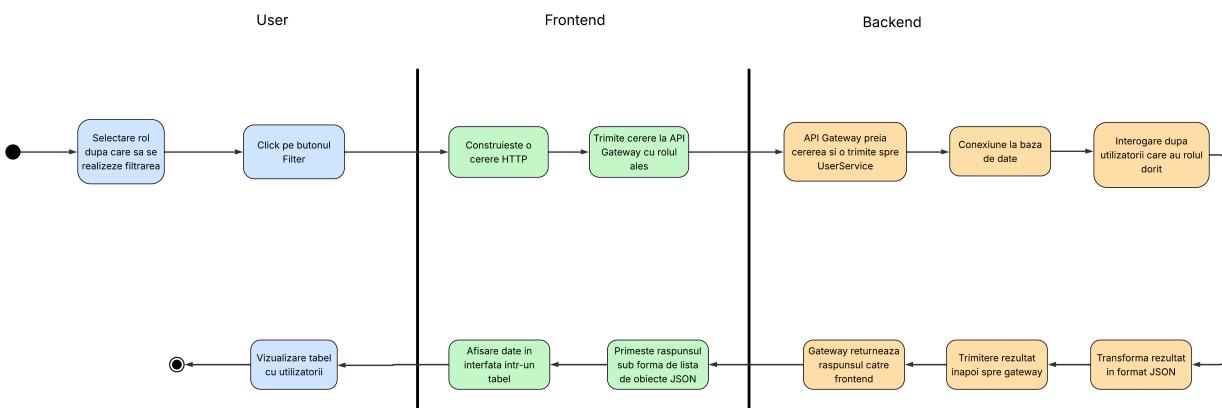


Figura 20: Cautare opere de arta dupa titlu

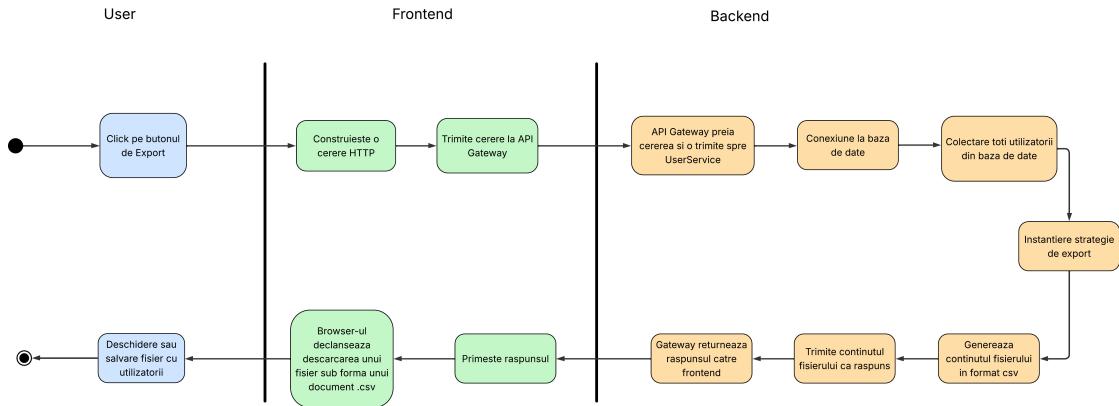


Figura 21: Cautare opere de artă după titlu

7 Diagrame de pachete

În cadrul acestui proiect am folosit o structurare a codului în mai multe pachete, pentru a separa logic funcționalitățile aplicației. Diagrama de pachete are rolul de a evidenția aceste module și relațiile dintre ele.

Fiecare pachet are o responsabilitate clară:

- **Controllers** – aici se află clasele care răspund la cererile primite de la utilizator. Acestea primesc datele din formulare sau URL, apeleză serviciile și trimit rezultatul către View sau către clientul frontend.
- **Services** – acest pachet conține logica principală a aplicației (business logic). Aici am implementat funcționalitățile care prelucră datele sau care fac legătura între controller și bază de date.
- **Domain** – am folosit acest pachet pentru a defini clasele de bază din proiect (entități, DTO-uri, interfețe, mappere). Practic, aici este descris modelul aplicației.
- **Infrastructure** – aici am pus tot ce ține de accesul la baza de date: entitățile pentru persistare, DAO-urile și configurațiile specifice.

7.1 Artist Service

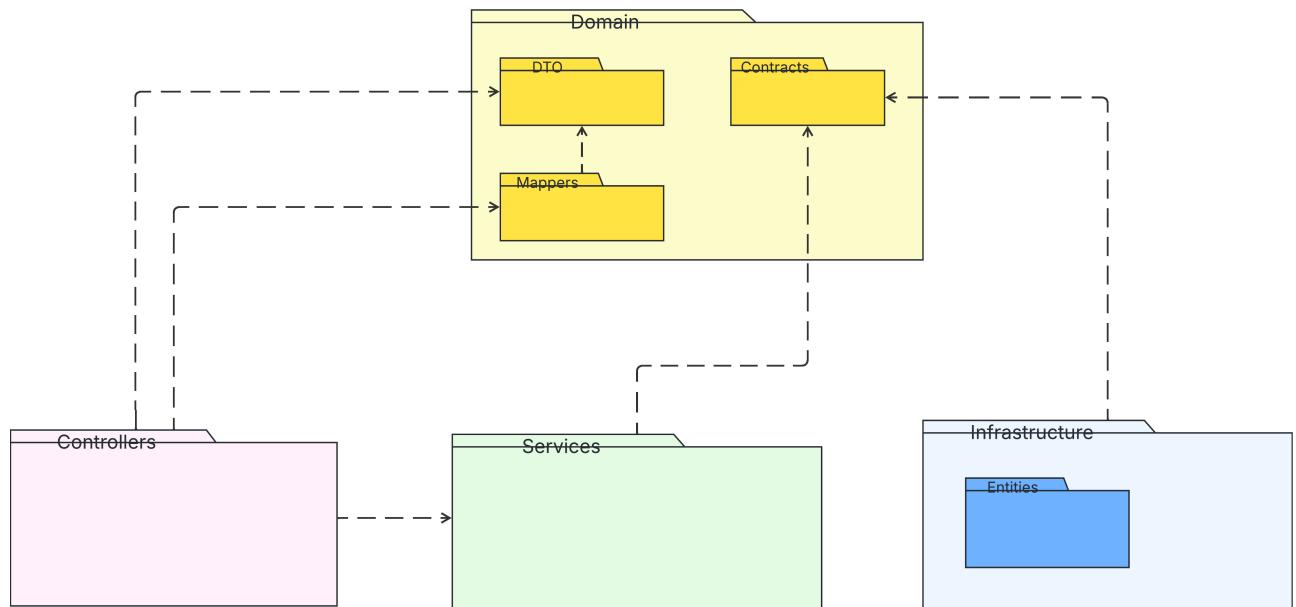


Figura 22: Diagrama de pachete pentru Artist Service

7.2 Artwork Service

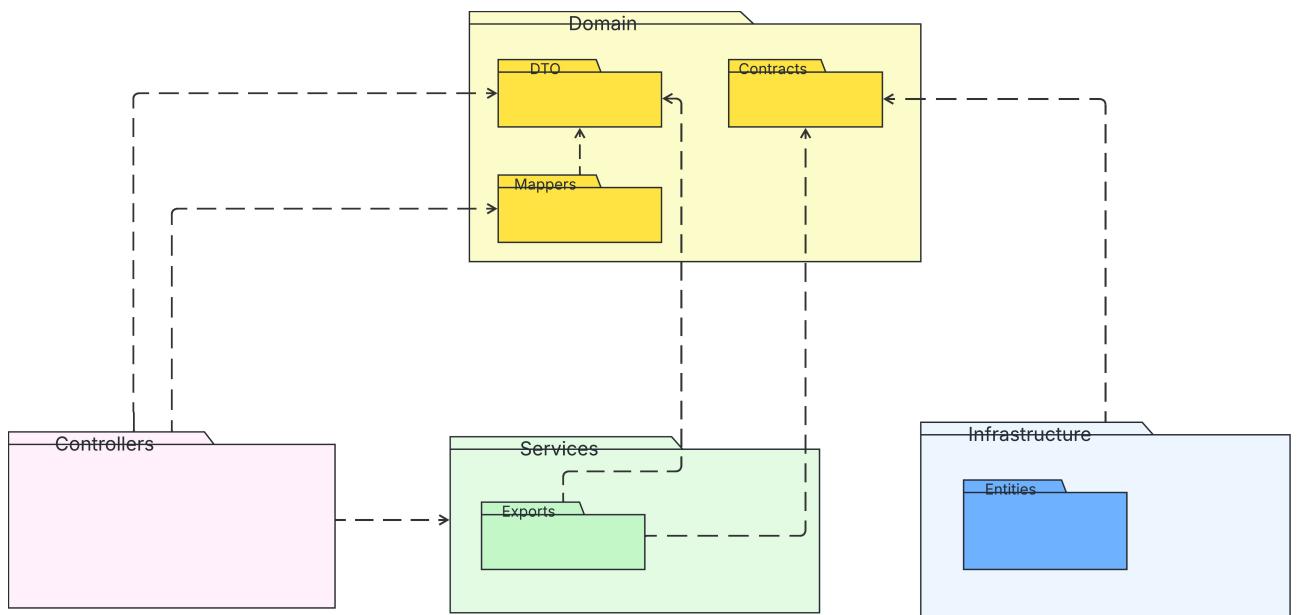


Figura 23: Diagrama de pachete pentru Artwork Service

7.3 User Service

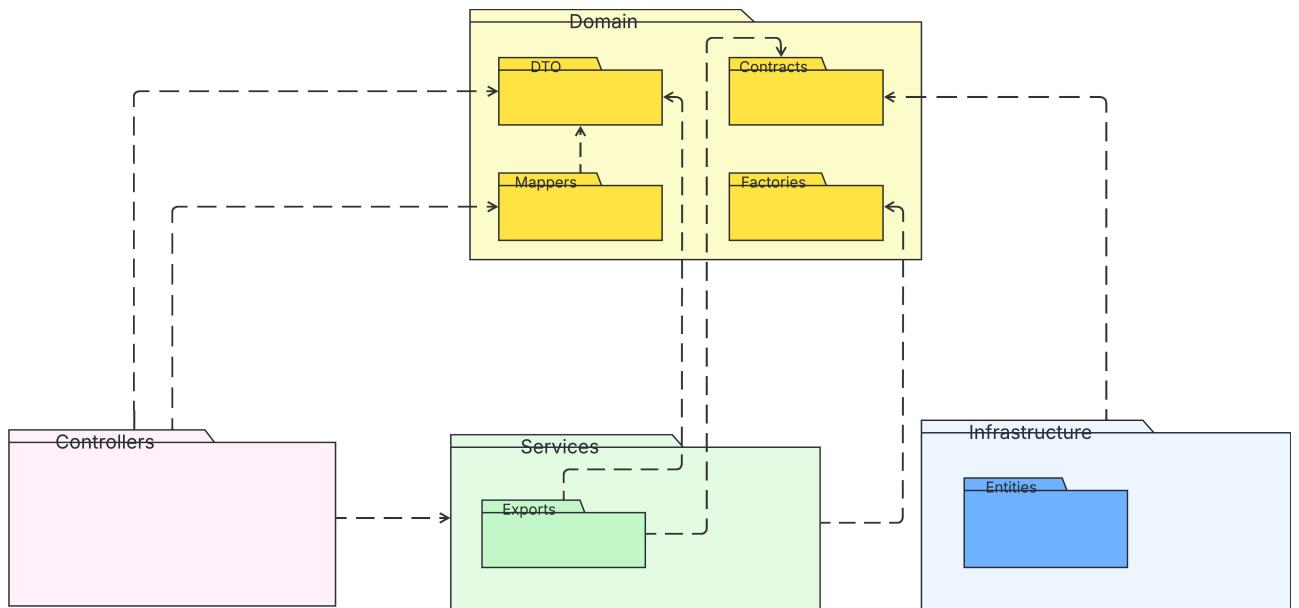


Figura 24: Diagrama de pachete pentru Sale Service

7.4 Sale Service

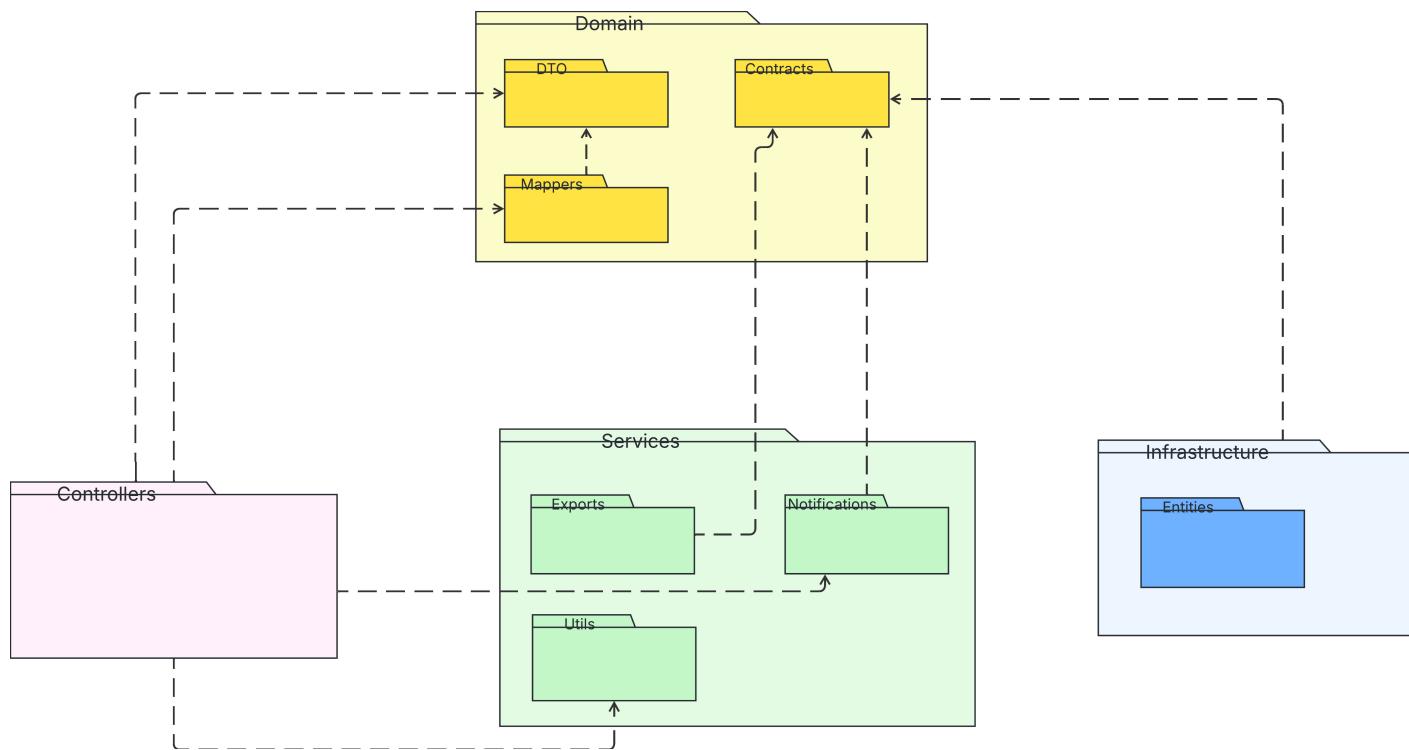


Figura 25: Diagrama de pachete pentru User Service

7.5 Frontend

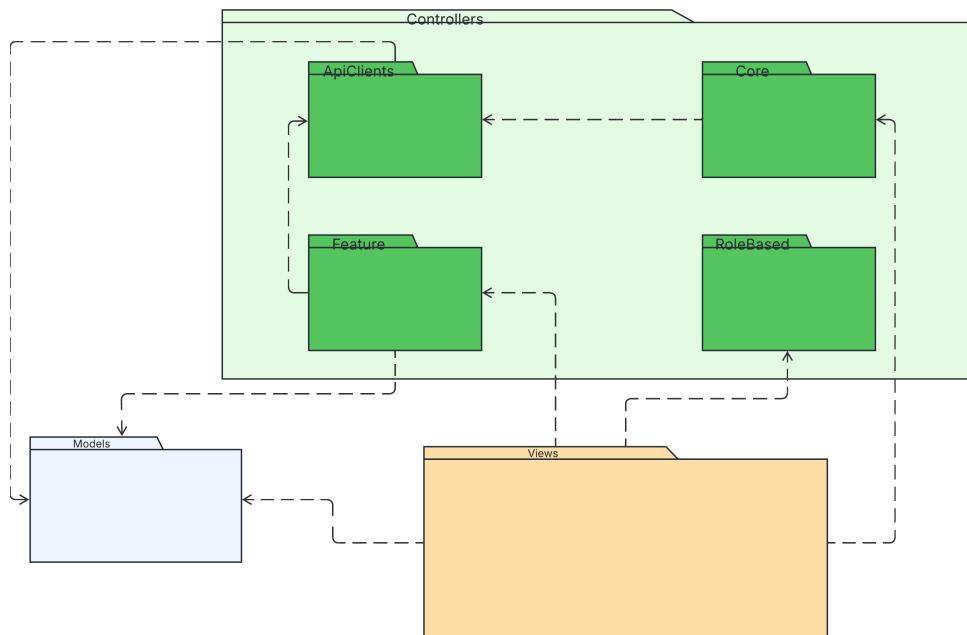


Figura 26: Diagrama de pachete pentru Frontend

8 Diagramme de classe

8.1 Artist Service

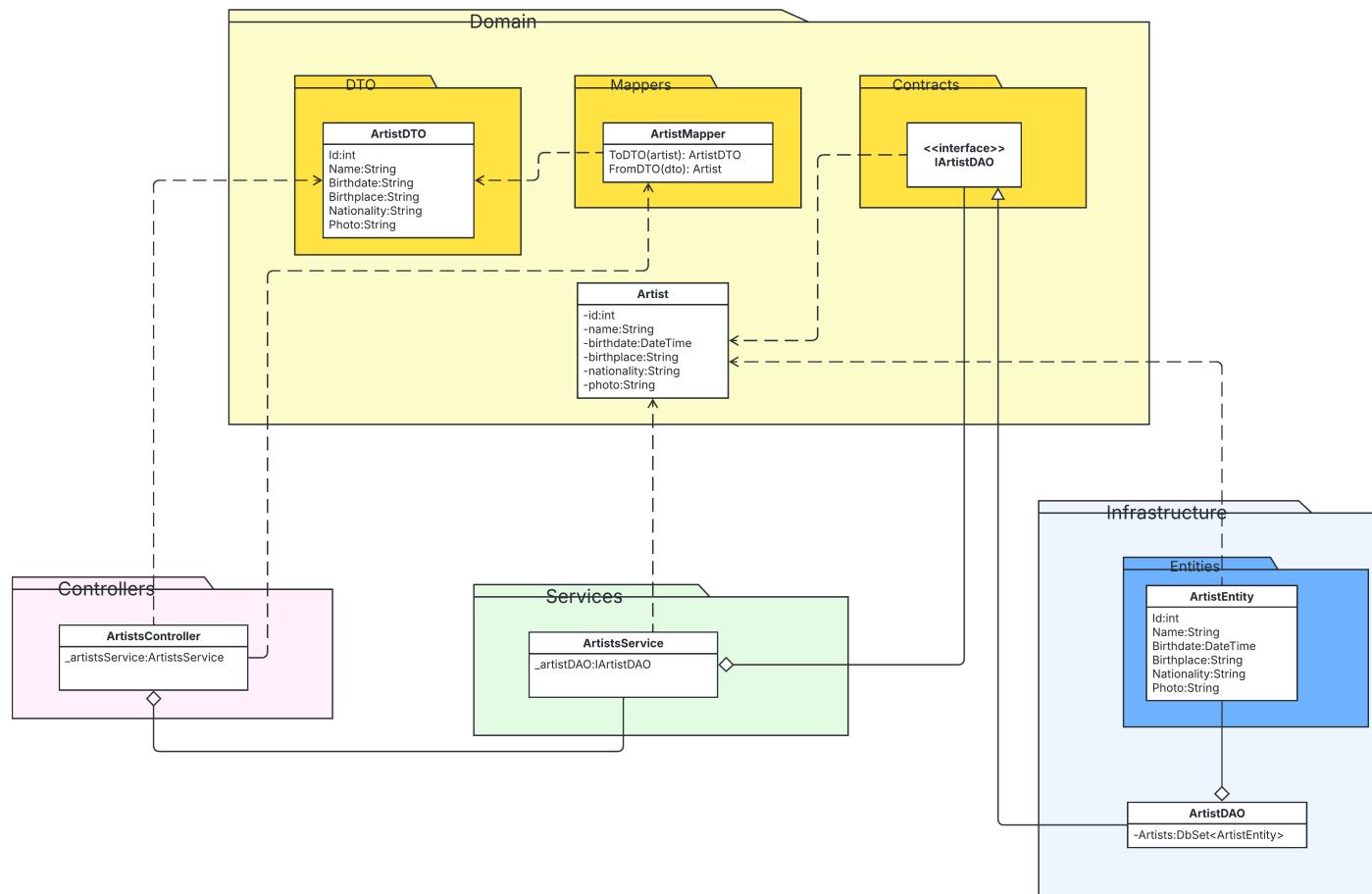


Figura 27: Diagrama de clase pentru Artist Service

8.2 Artwork Service

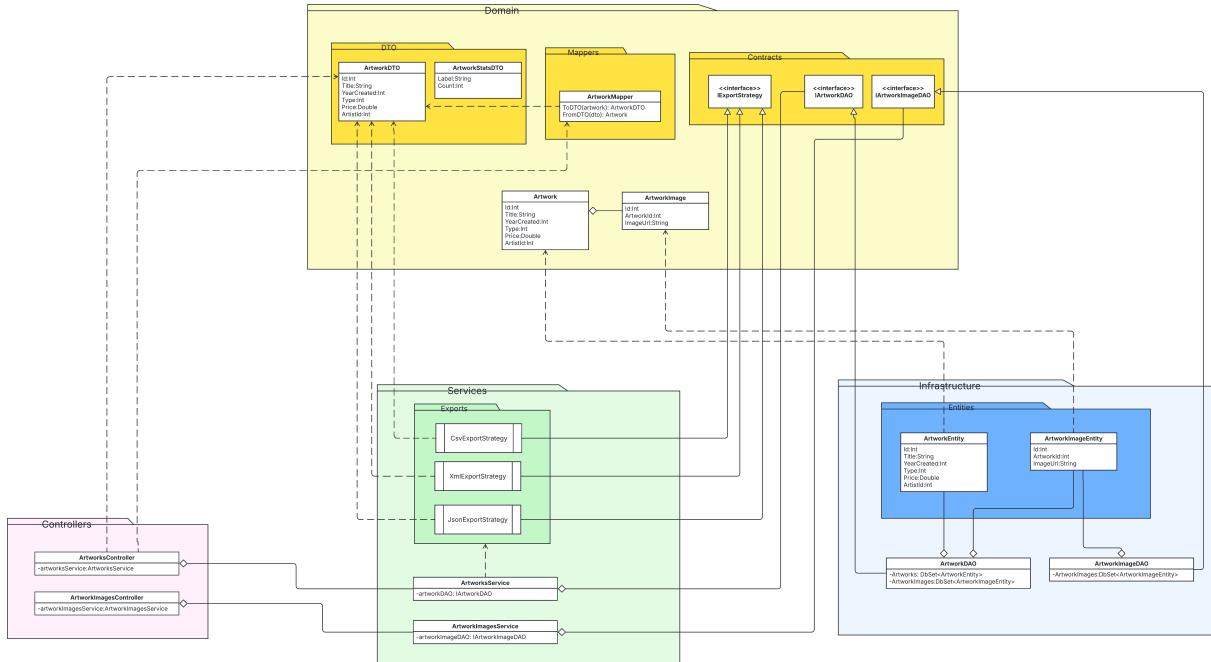


Figura 28: Diagrama de clase pentru Artwork Service

8.3 User Service

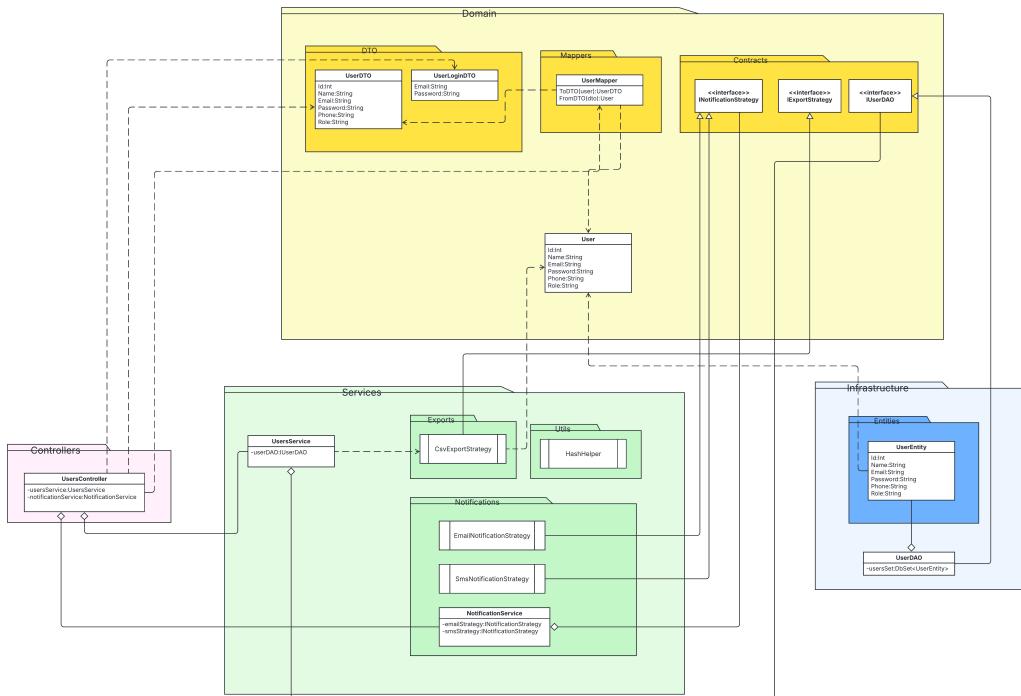


Figura 29: Diagrama de clase pentru User Service

8.4 Sale Service

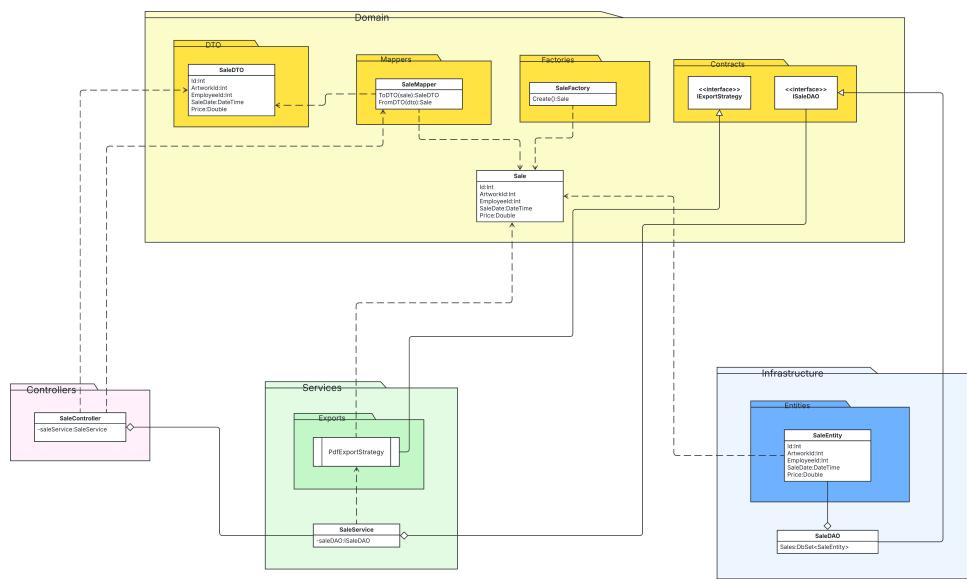


Figura 30: Diagrama de clase pentru Sale Service

8.5 Frontend

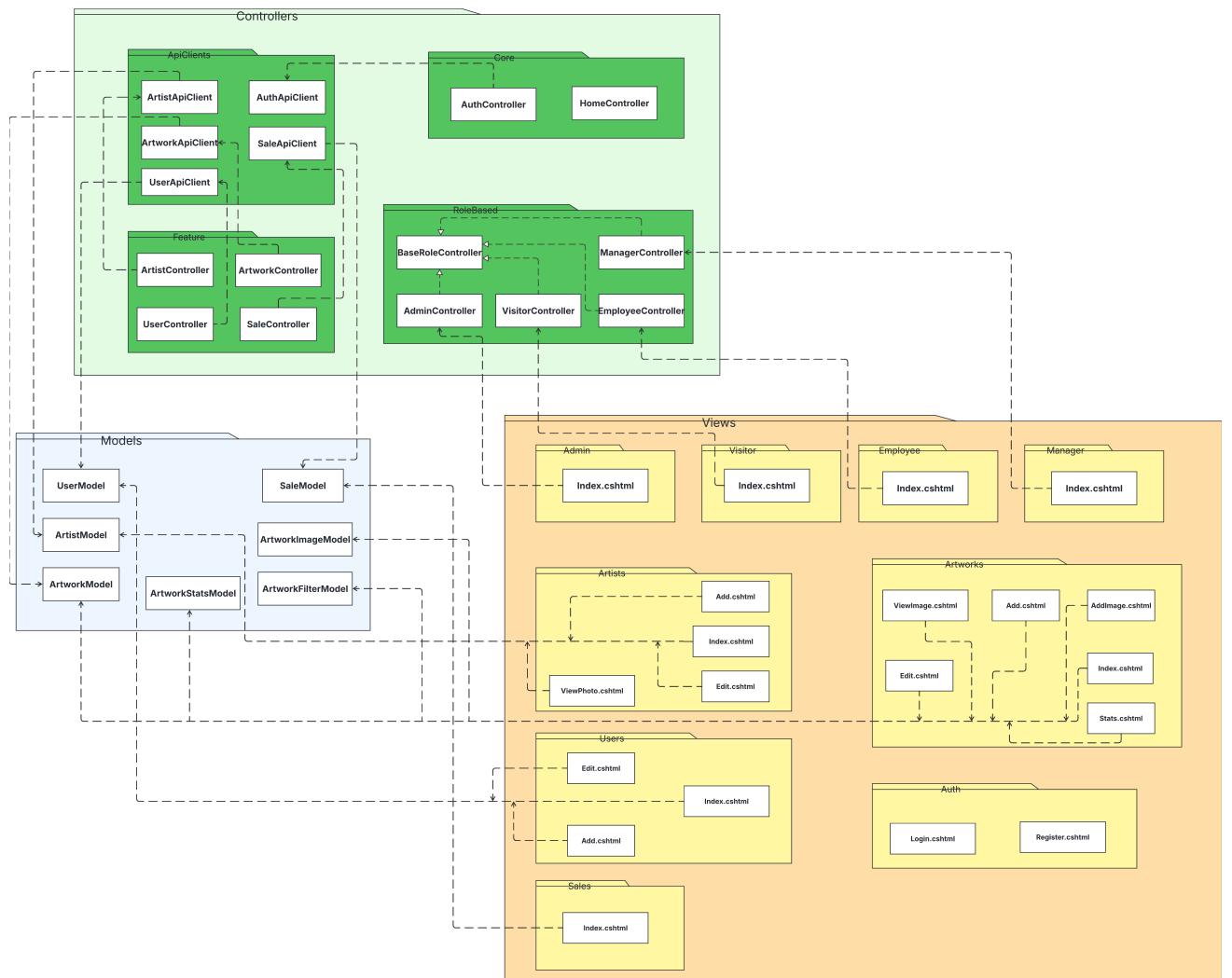


Figura 31: Diagrama de clase pentru Frontend

9 Diagrame entitate-relatie

Diagramale de entitate-relație (ER) au fost folosite pentru a reprezenta structura bazei de date, într-un mod vizual și ușor de înțeles. Ele arată ce tabele (entități) există în sistem, ce atribute are fiecare entitate și cum sunt legate între ele.

Pentru acest proiect, am realizat câte o diagramă ER separată pentru fiecare microserviciu, deoarece fiecare dintre ele are propria bază de date, conform arhitecturii de tip *Database per Service*. Astfel, fiecare diagramă reflectă strict modelul de date local al unui microserviciu, fără legături directe cu celelalte.

O diagramă de entitate-relație este compusă din:

- **Entități** – care corespund tabelelor din baza de date.
- **Atribute** – coloanele fiecărei entități (ex: Id, Nume, Email).
- **Relații** – legăturile dintre entități (ex: o operă are mai multe imagini).

9.1 Artist Service

ARTIST		
INTEGER	Id	PK
TEXT	Name	
DATE	BirthDate	
TEXT	Birthplace	
TEXT	Nationality	
TEXT	Photo	

Figura 32: Diagrama ER pentru artist service

9.2 Artwork Service

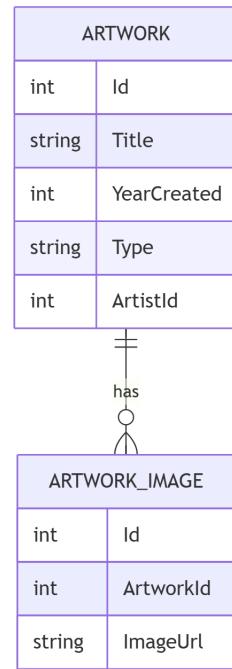


Figura 33: Diagrama ER pentru artwork service

9.3 User Service

USER	
int	Id
string	Name
string	Email
string	PasswordHash
string	Role
string	Phone

Figura 34: Diagrama ER pentru user service

9.4 Sale Service

SALE	
int	Id
int	ArtworkId
int	EmployeeId
string	SaleDate
float	Price

Figura 35: Diagrama ER pentru sale service

10 Diagrame de secvență

Diagramele de secvență au fost folosite pentru a reprezenta modul în care interacționează componentele aplicației între ele în cadrul unui anumit scenariu. Ele arată clar ordinea în care au loc apelurile și cum sunt transmise datele între utilizator, interfață, controller, serviciu și microserviciu.

Fiecare diagramă descrie un caz de utilizare concret (cum ar fi adăugarea unui artist sau autentificarea unui utilizator), și ilustrează pașii prin care informația circulă în aplicație. Aceste diagrame m-au ajutat să înțeleg mai bine logica de comunicare și cum se leagă părțile aplicației în timp real.

Elementele principale ale unei diagrame de secvență sunt:

- **Autorul** – de obicei un utilizator care inițiază o acțiune (ex: administratorul).
- **Componentele aplicației** – cum ar fi controllerul, clientul API, microserviciul și baza de date.
- **Mesajele** – săgețile care indică apeluri între componente (ex: cereri HTTP, răspunsuri, redirecționări).
- **Ordinea apelurilor** – este reprezentată vertical, de sus în jos.

10.1 Vizitator

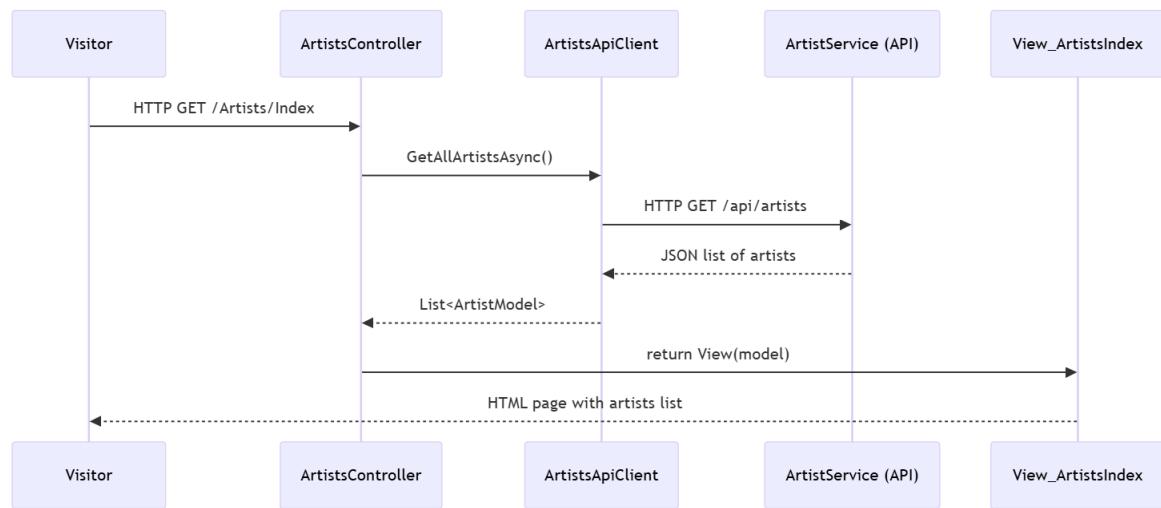


Figura 36: Vizualizare artiști

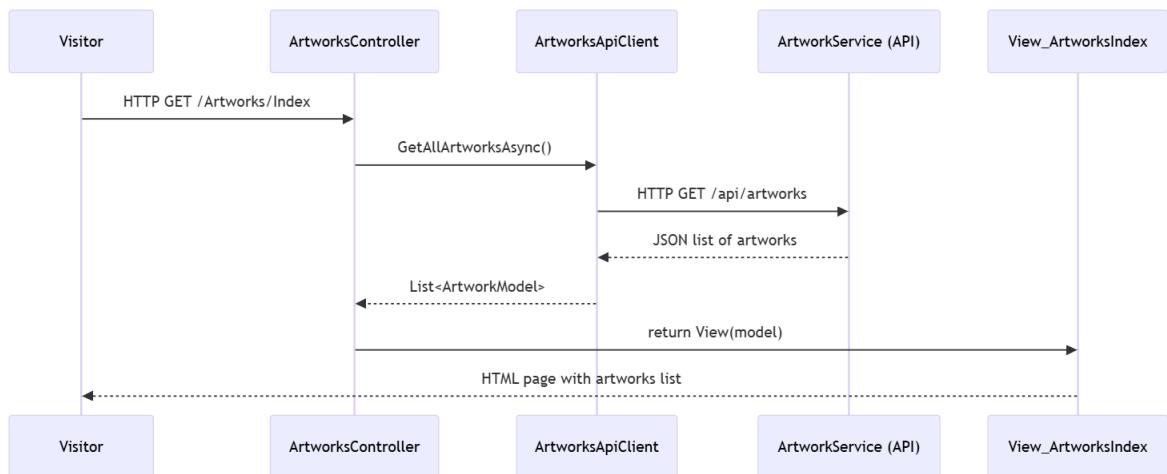


Figura 37: Vizualizare opere de artă

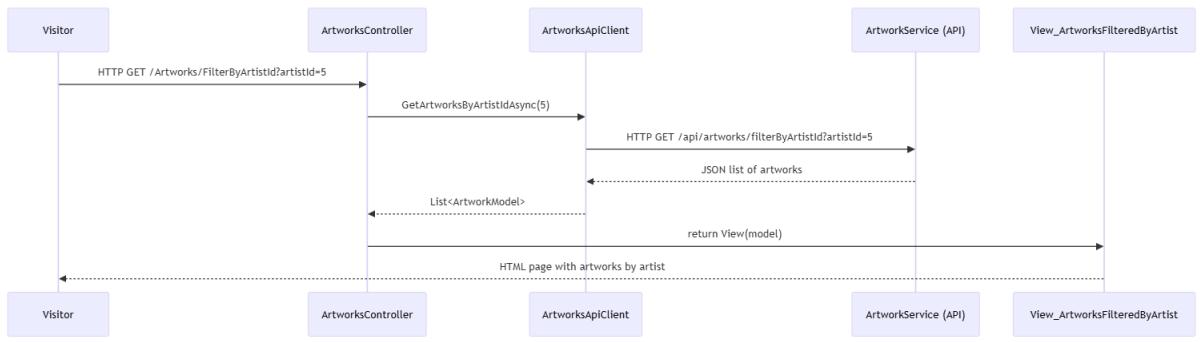


Figura 38: Filtrare opere de artă după artist

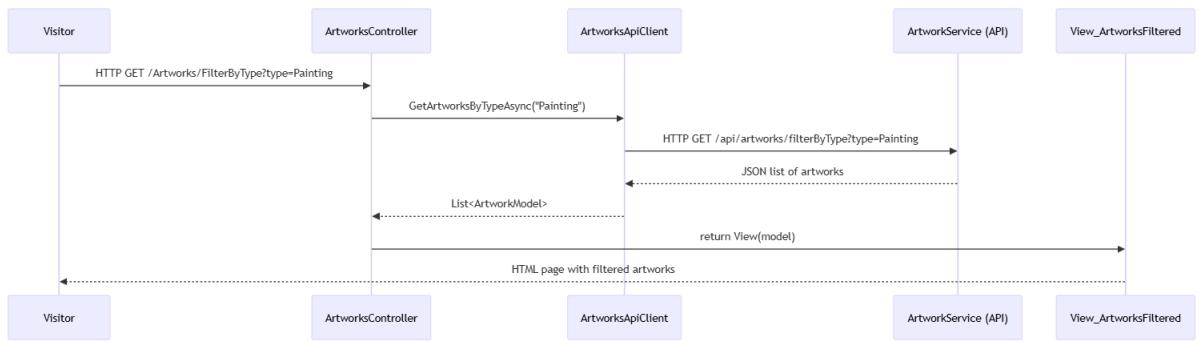


Figura 39: Filtrare opere de artă după tip

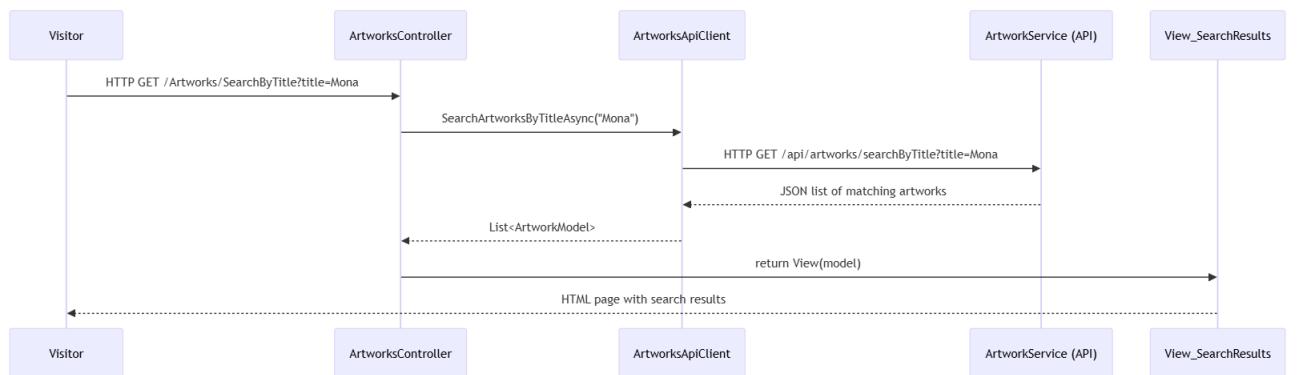


Figura 40: Cautare opere de artă după titlu

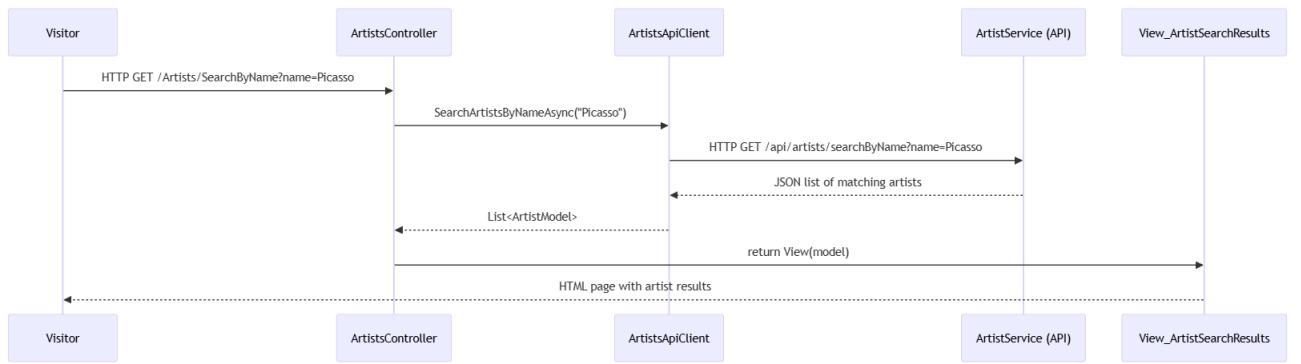


Figura 41: Cautare artiști după nume

10.2 Angajat

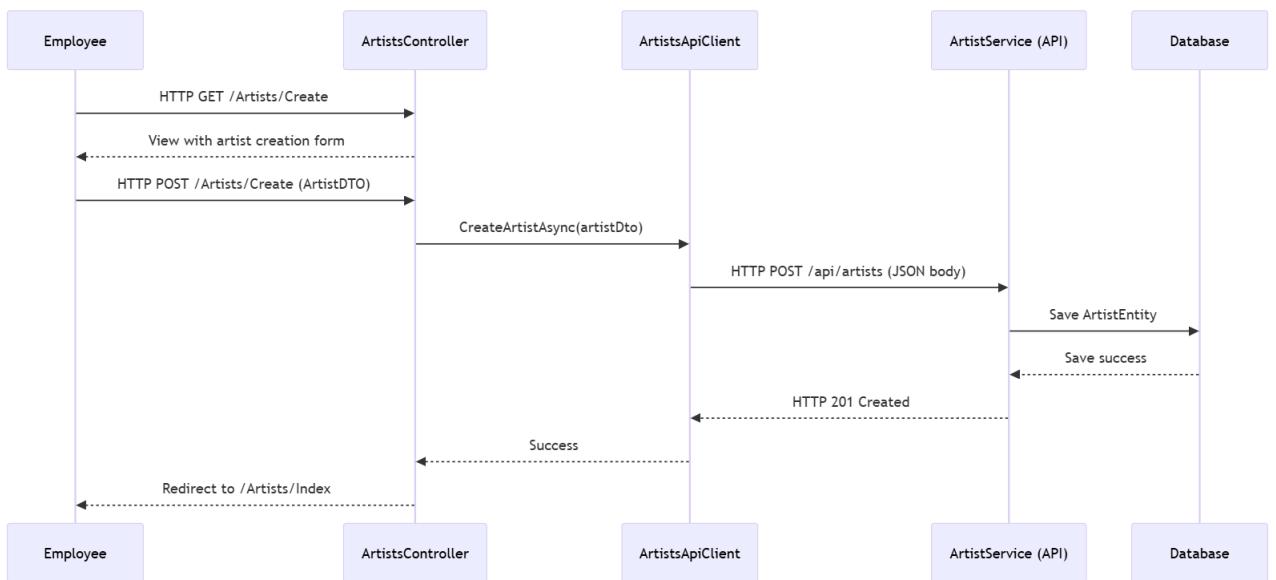


Figura 42: Adaugare artist

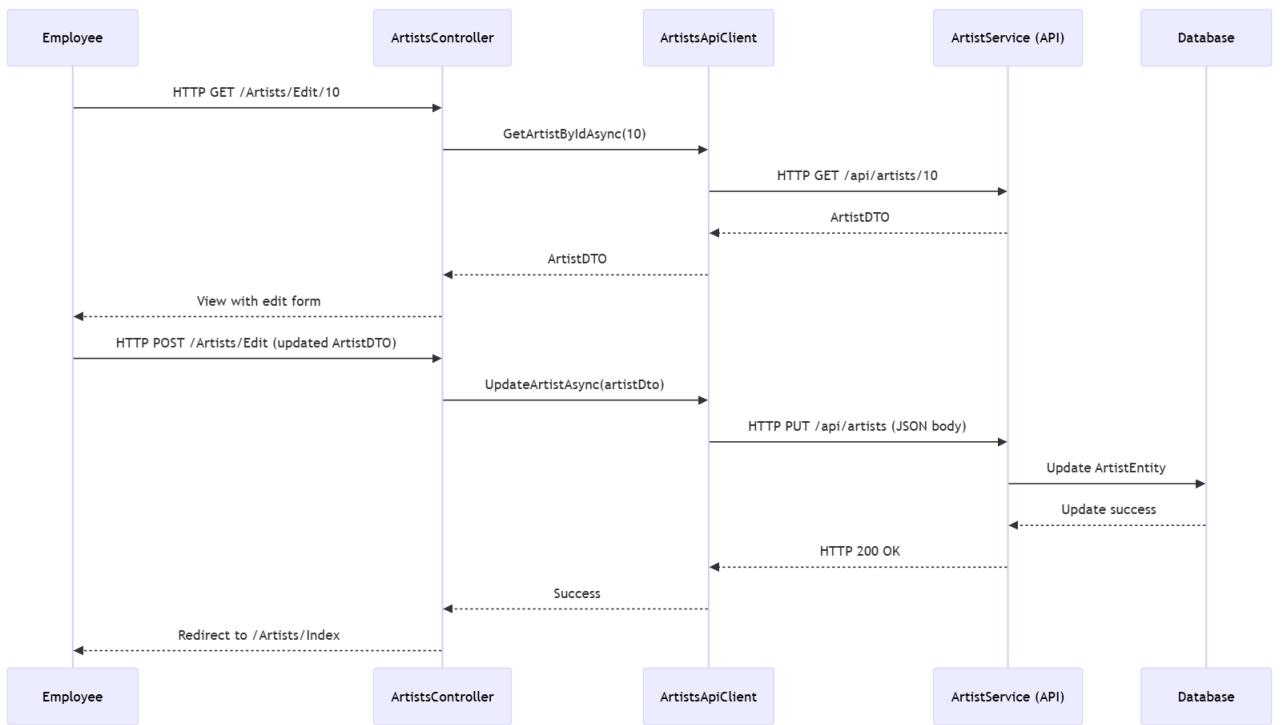


Figura 43: Actualizare artist

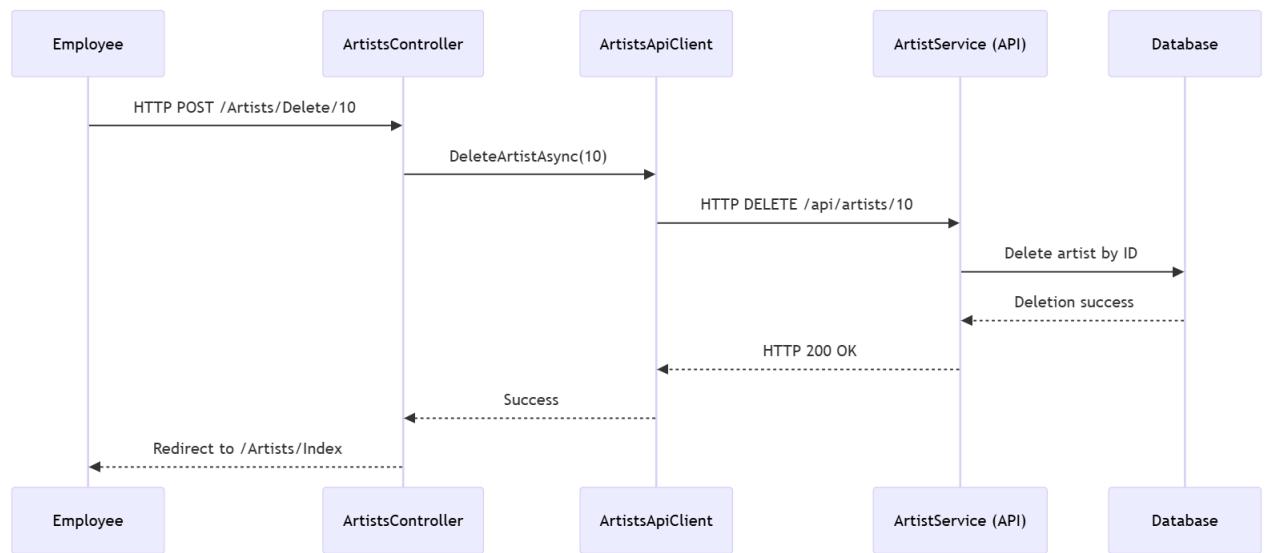


Figura 44: Stergere artist

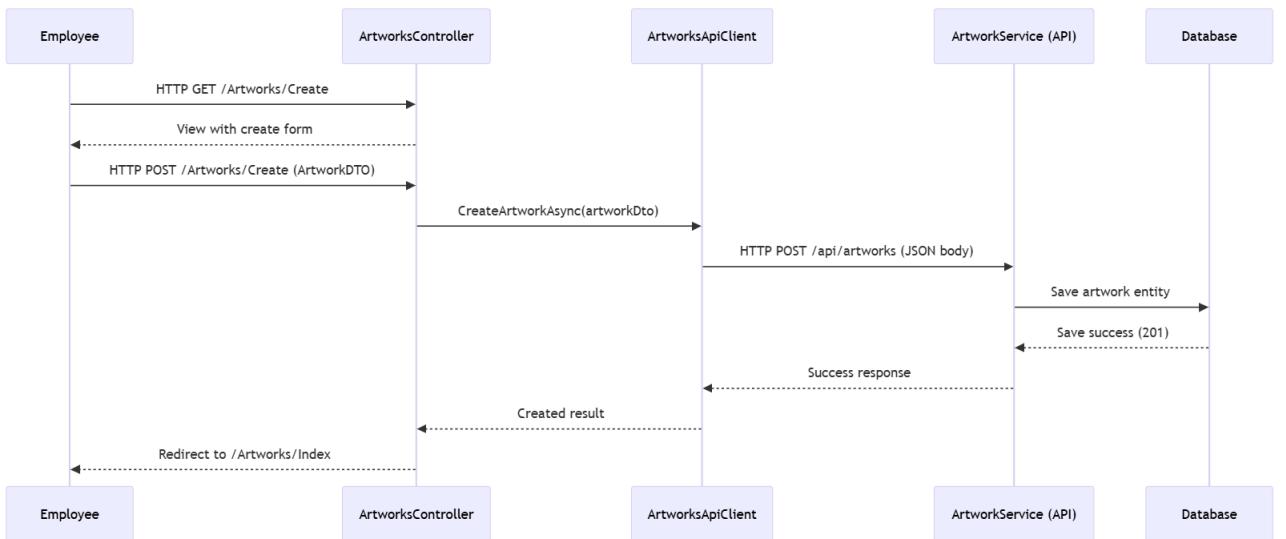


Figura 45: Adaugare opera de artă

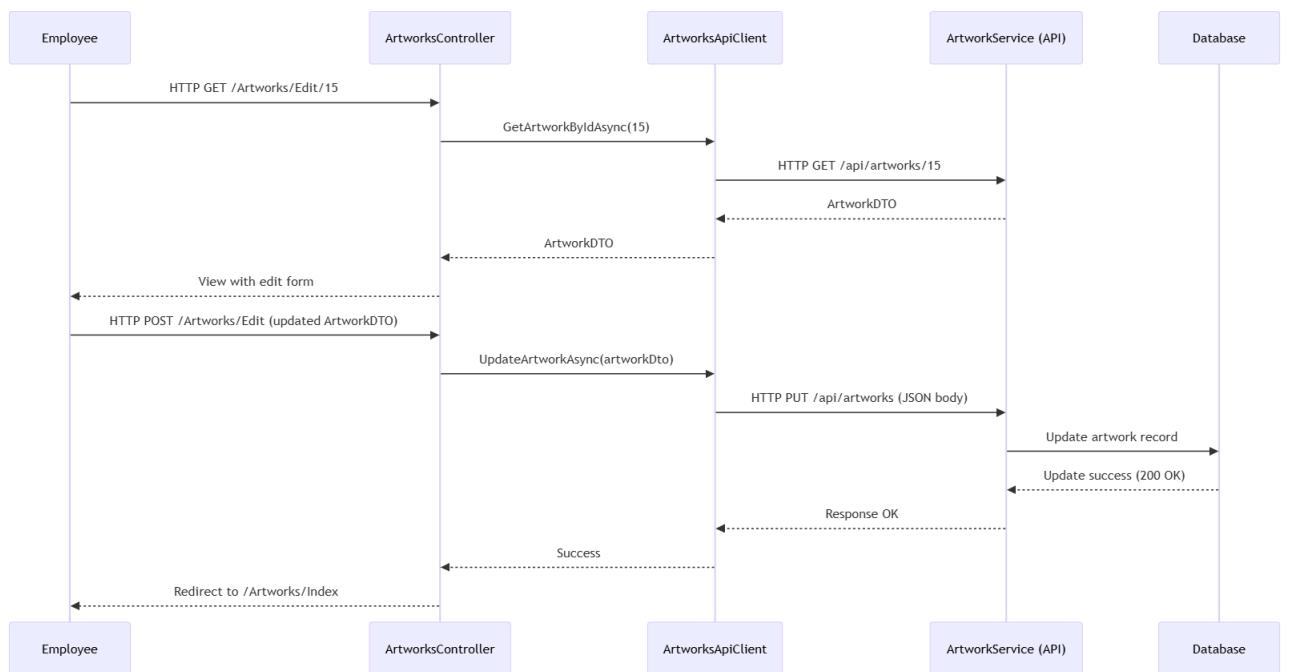


Figura 46: Actualizare opera de artă

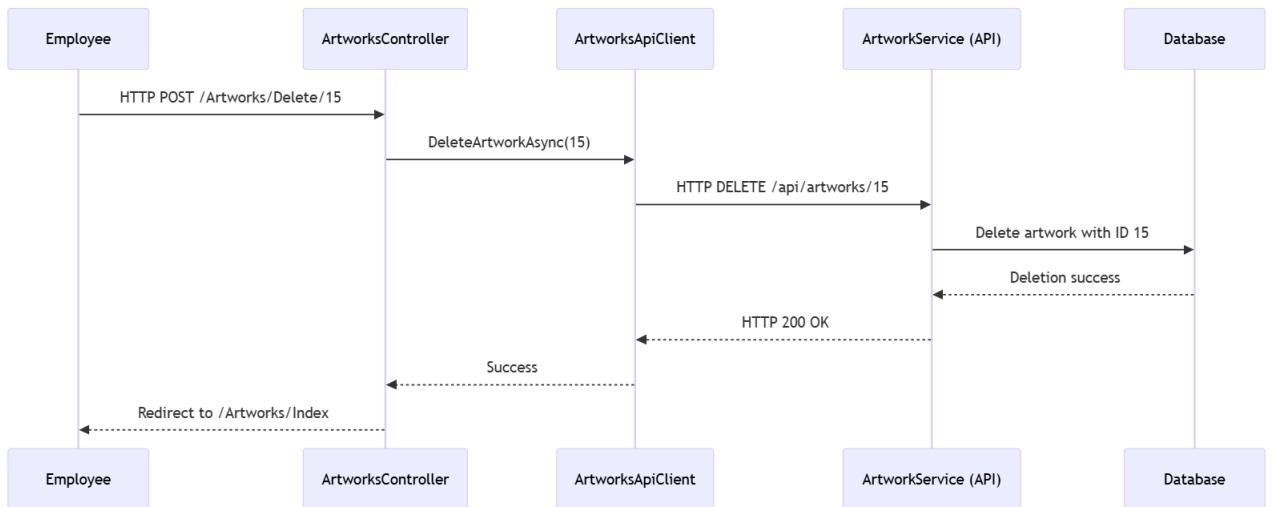


Figura 47: Stergere opera de artă

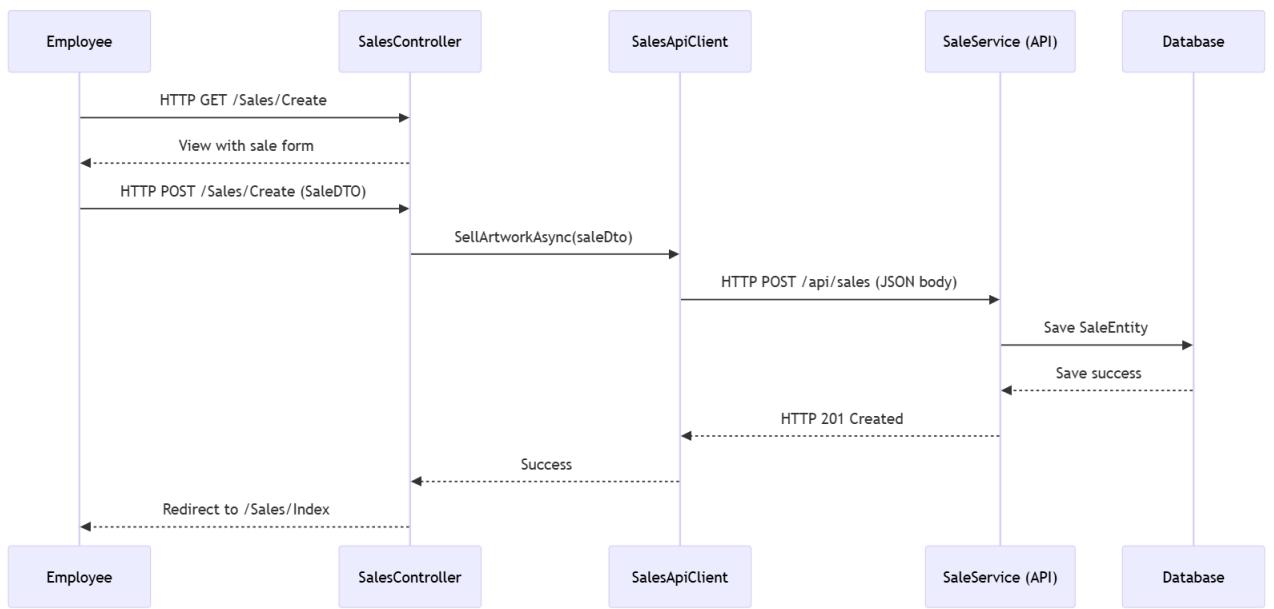


Figura 48: Vanzare opera de artă

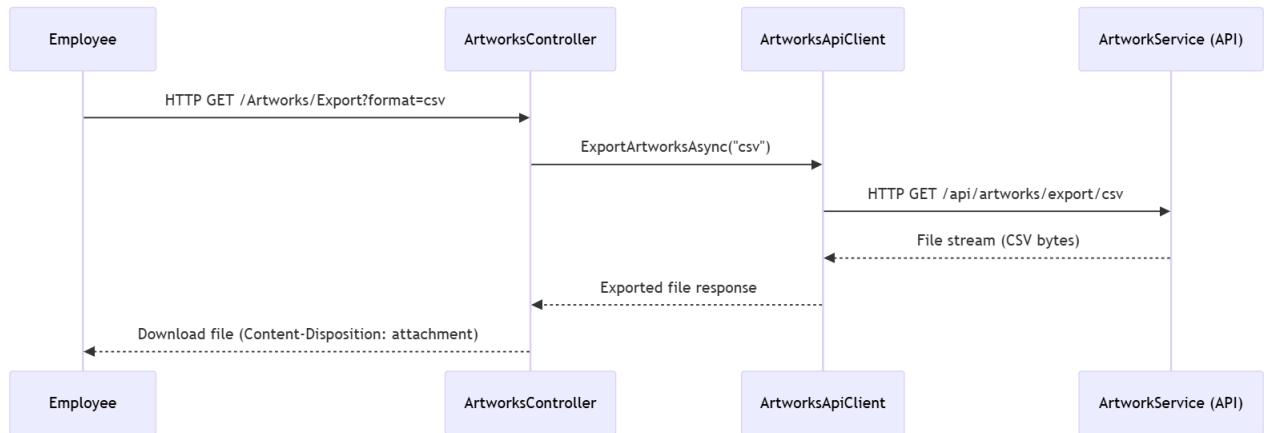


Figura 49: Export opere de artă

10.3 Manager

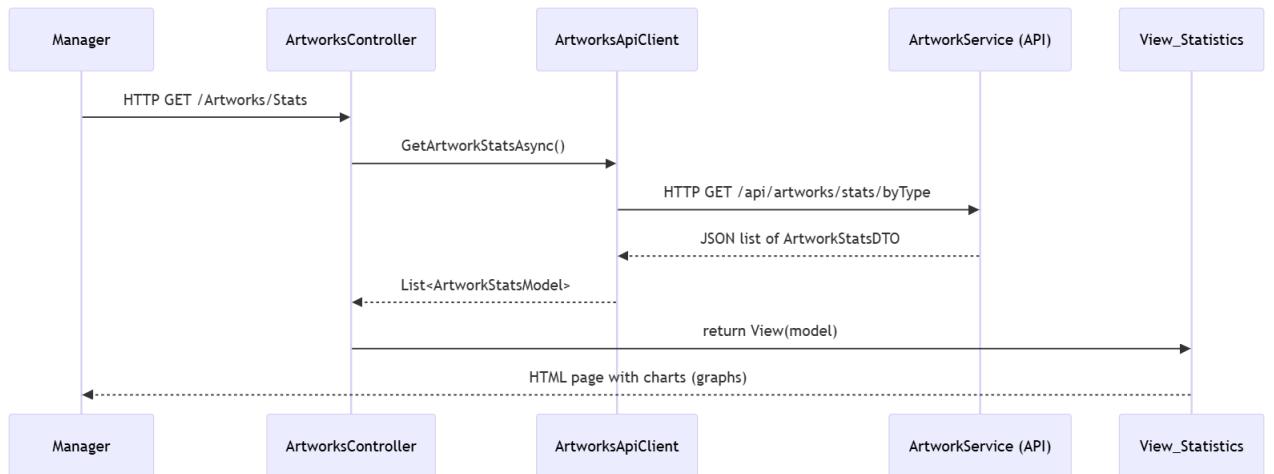


Figura 50: Vizualizare statistici

10.4 Administrator

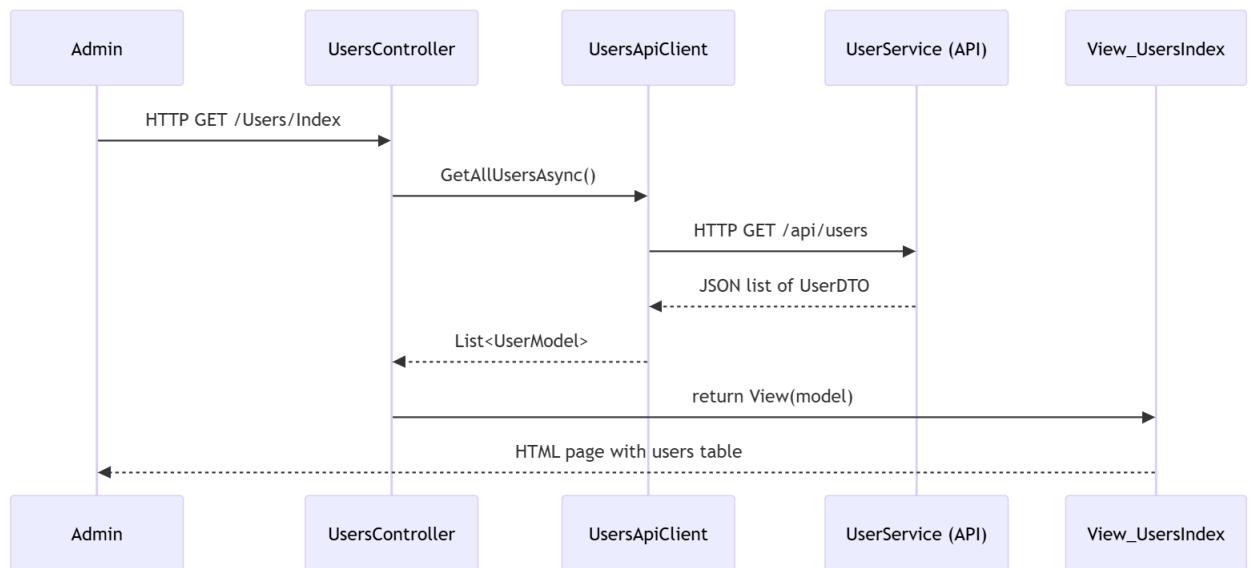


Figura 51: Vizualizare utilizatori

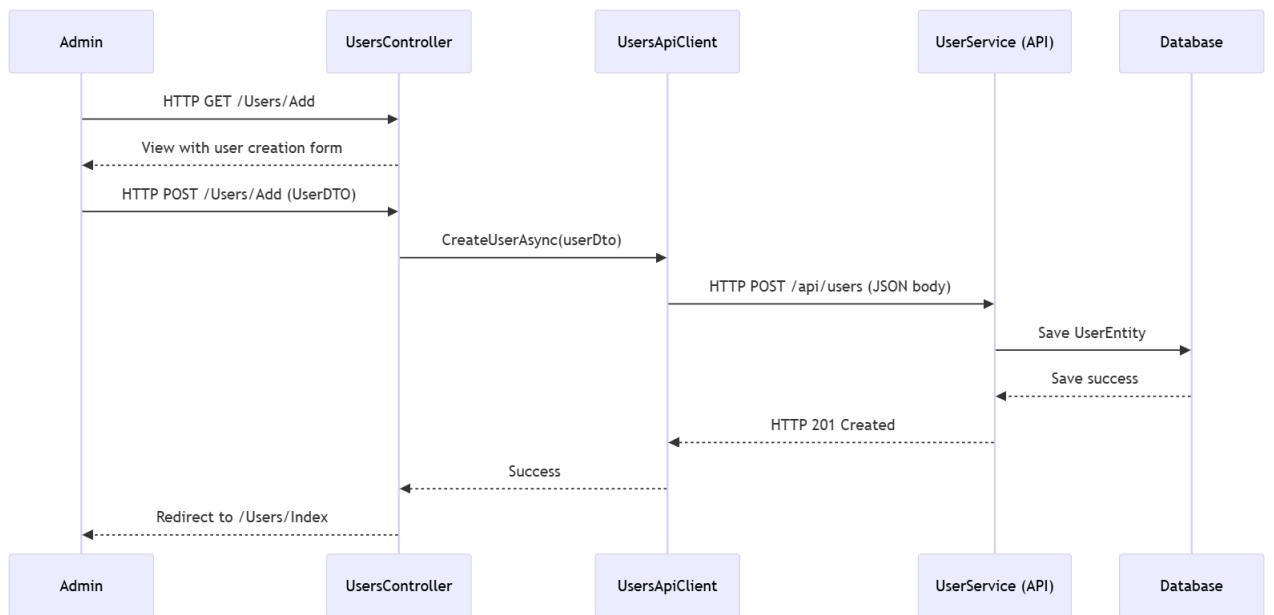


Figura 52: Adaugare utilizator

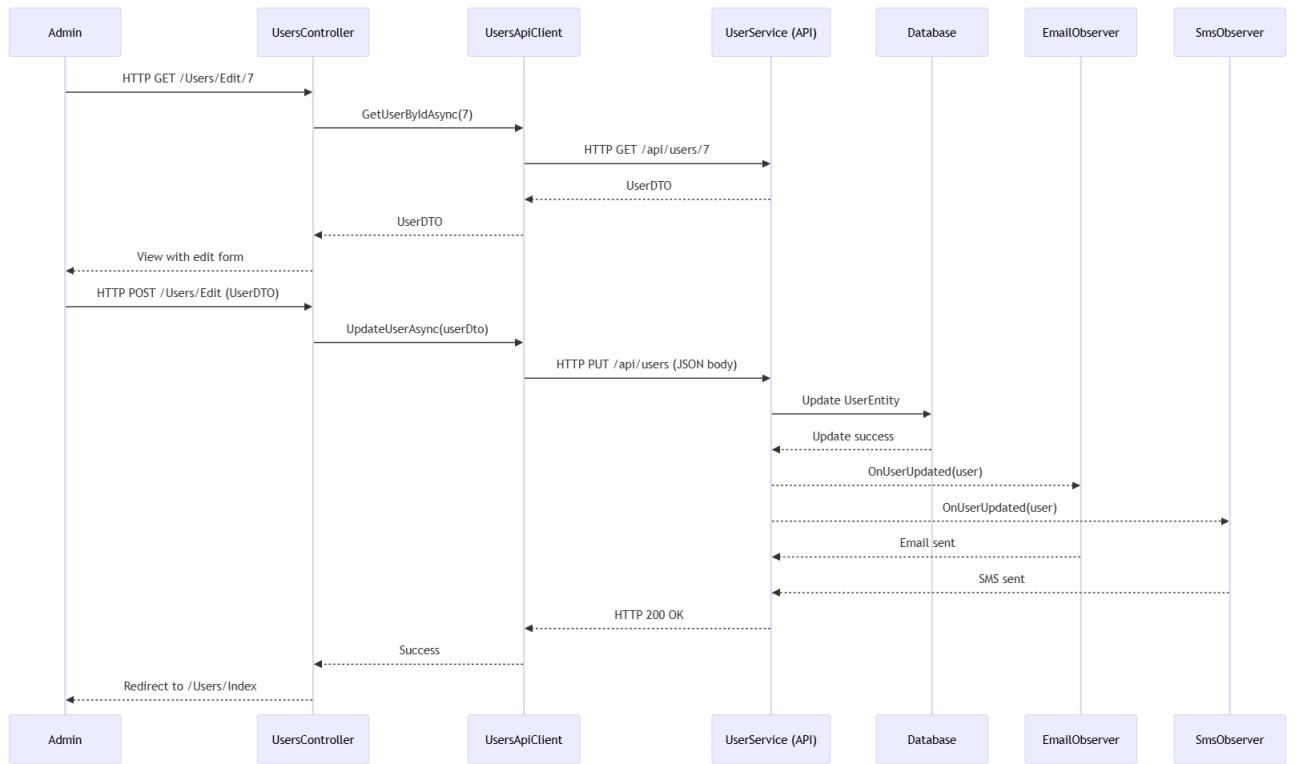


Figura 53: Actualizare utilizator

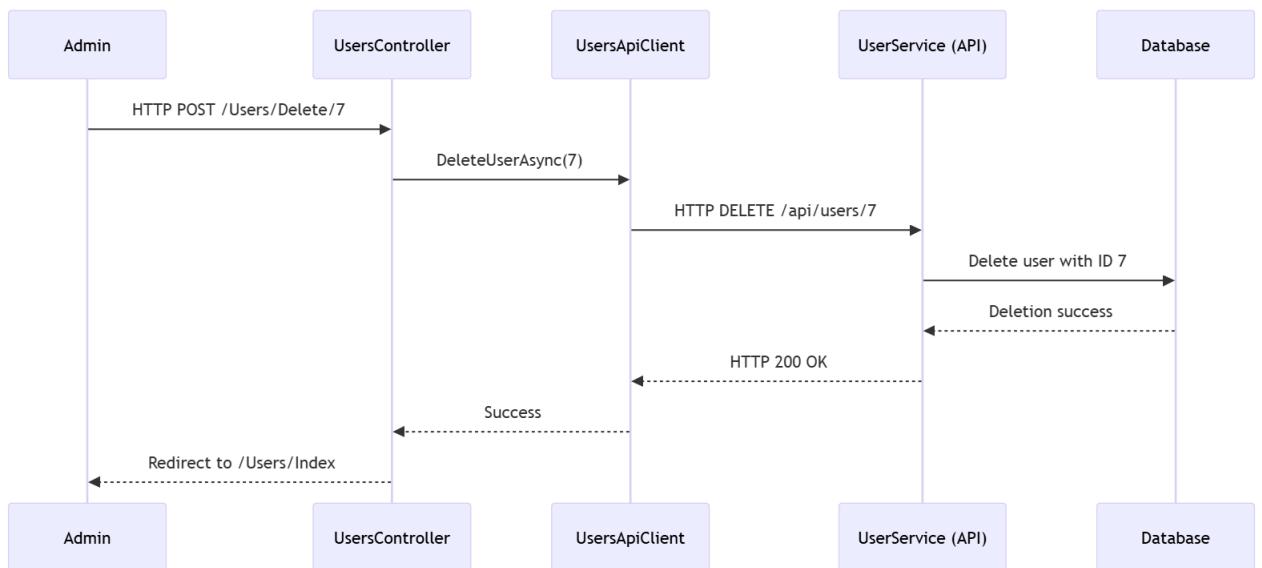


Figura 54: Stergere utilizator

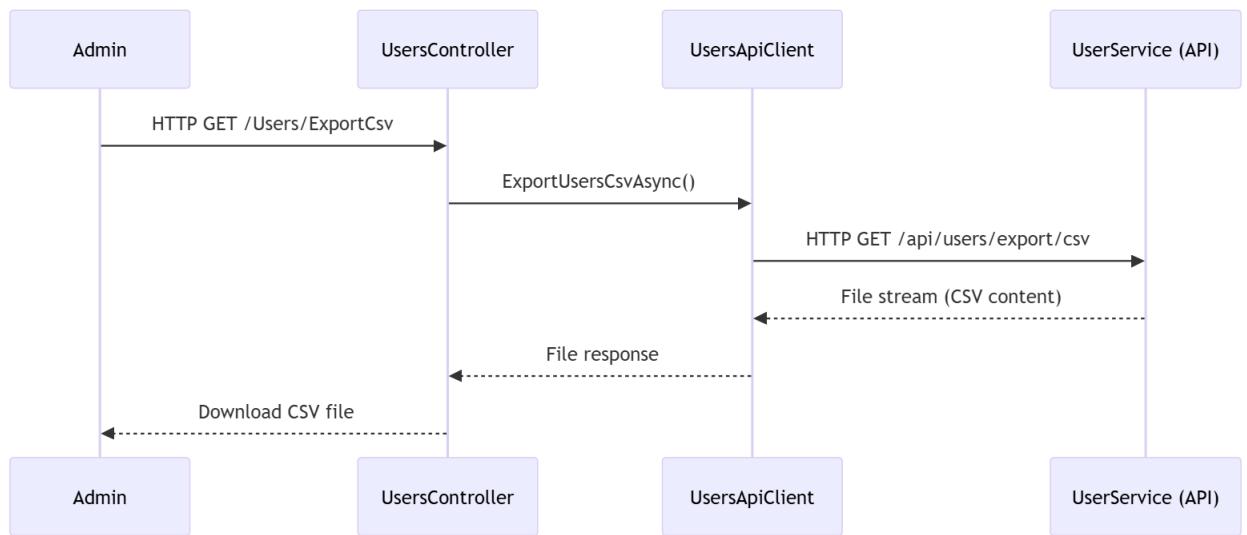


Figura 55: Export utilizatori in format csv

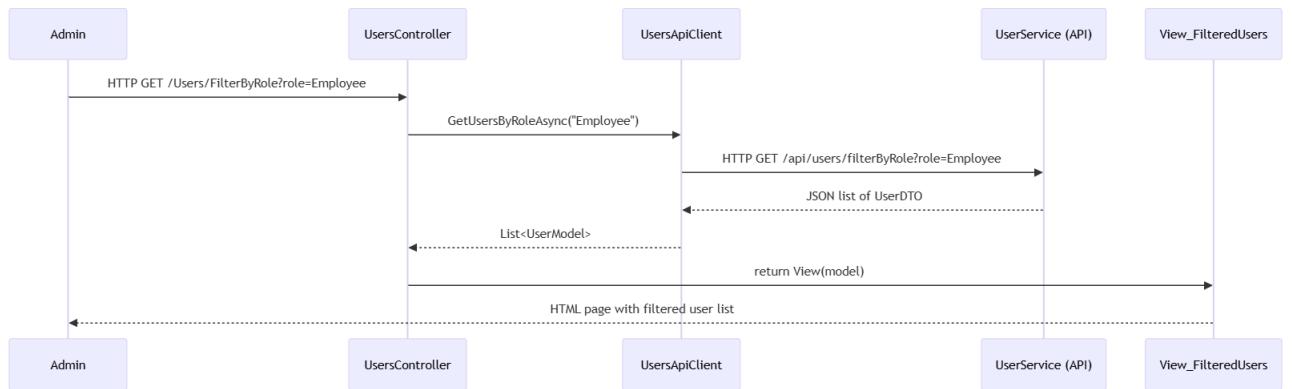


Figura 56: Filtrare utilizatori

11 Diagrama de componente

Diagrama de componente are rolul de a arăta modul în care este organizată aplicația la nivel logic și cum interacționează diferitele părți ale sistemului. În cadrul proiectului, am folosit o arhitectură bazată pe microservicii, fiecare având propriul său rol și propria bază de date.

În partea dreaptă a diagramei se află **Frontend-ul**, realizat cu *ASP.NET Web MVC*. Acesta comunică doar cu **API Gateway-ul**, care este punctul unic de acces spre restul serviciilor.

API Gateway-ul este implementat cu ajutorul bibliotecii *Ocelot* și are rolul de a redirecționa cererile primite către microserviciile corespunzătoare, în funcție de configurația definită în fișierul *ocelot.json*.

În partea stângă a diagramei sunt prezentate cele 4 **microservicii**:

- **ArtworkService** – gestionează operele de artă.
- **ArtistService** – se ocupă de artiști.
- **UserService** – gestionează utilizatorii și autentificarea.
- **SaleService** – se ocupă de înregistrarea vânzărilor.

Fiecare microserviciu este conectat la o bază de date proprie, respectând principiul *Database-per-Service*. Acest lucru oferă independentă între servicii și facilitează scalabilitatea și întreținerea pe termen lung.

Legăturile dintre componente sunt de tip **REST API**, iar porturile sunt gestionate individual pentru fiecare serviciu.

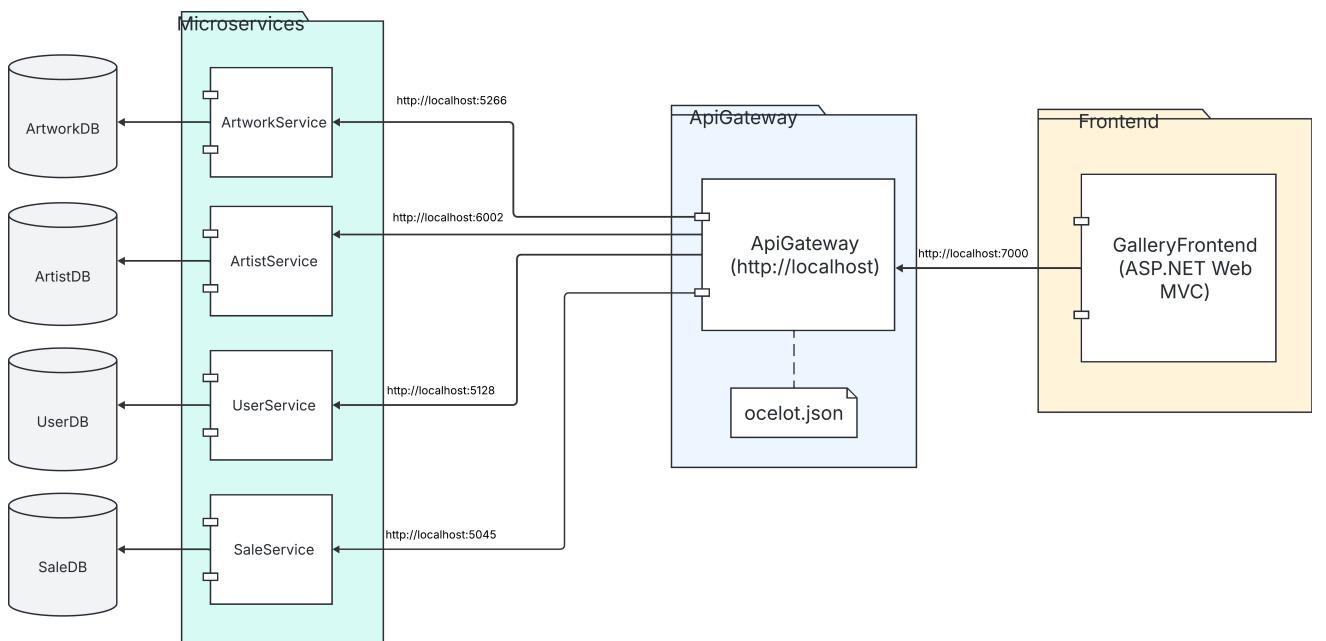


Figura 57: Diagrama de componente

12 Diagrama de dezvoltare

Diagrama de dezvoltare (sau deployment) prezintă modul în care componentele aplicației sunt distribuite și interconectate în mediul de rulare. Aceasta oferă o imagine de ansamblu asupra arhitecturii fizice a sistemului, evidențiind serverele, aplicațiile și bazele de date implicate.

În figura de mai sus sunt reprezentate următoarele componente:

- **GalleryFrontend** – aplicația ASP.NET MVC care rulează pe un server de tip client. Aceasta este interfața cu utilizatorul și se accesează prin browser.
- **API Gateway** – este un server intermediar care primește toate cererile din partea frontend-ului și le redirecționează către microserviciile corespunzătoare. Configurația acestuia este definită în fișierul `ocelot.json`.
- **Microserviciile** – sistemul este împărțit în patru microservicii independente. Aceste servicii rulează izolat și sunt accesate doar prin intermediul gateway-ului.
- **SQL Server** – conține patru baze de date distințe, câte una pentru fiecare microserviciu.

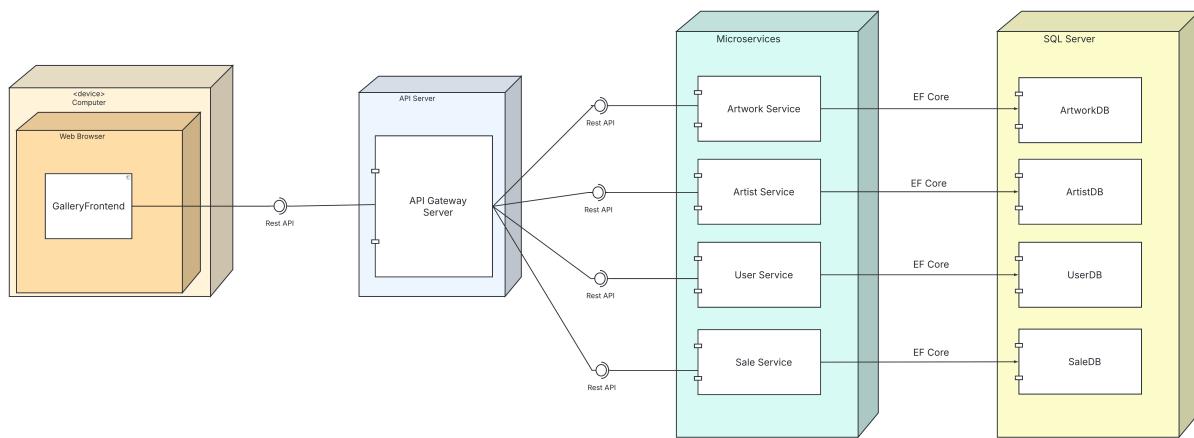


Figura 58: Diagrama de dezvoltare

13 Descrierea Aplicatiei

Aplicația a fost construită folosind o arhitectură de tip microservicii, în care funcționalitățile sunt împărțite în patru servicii independente: ArtistService, ArtworkService, UserService și SaleService. Partea de interfață (frontend) a fost realizată cu ASP.NET Web MVC, iar comunicarea între părți se face printr-un API Gateway, configurat cu Ocelot.

Pentru a structura codul într-un mod clar, reutilizabil și ușor de întreținut, am folosit mai multe şabloane de proiectare (design patterns), atât în backend, cât și în frontend.

Şabloane de proiectare utilizate

- **Builder** – folosit pentru a crea entități într-un mod centralizat și controlat. Am implementat câte o clasă pentru transformarea obiectelor din domeniu în entități de bază de date.
- **Facade** – utilizat pentru a ascunde complexitatea internă a serviciilor. În fiecare microserviciu am definit câte o clasă Fațadă care oferă metode simplificate pentru operațiile importante.
- **Strategy** – aplicat în cazul exportului de date (ex: export opere în format CSV, XML sau JSON). Am creat o interfață comună și implementări specifice pentru fiecare format, ușor de extins.
- **Observer** – folosit în UserService pentru notificarea automată a utilizatorilor (prin email și SMS) în momentul în care un cont este modificat. Am înregistrat observatorii într-un subiect (notificator) care se ocupă de distribuirea notificărilor.
- **Template Method** – utilizat în frontend, în controller-ele pentru roluri (Admin, Employee, Manager, Visitor), unde am extras comportamentul comun într-un controller de bază. Clasele derivate implementează pașii specifici fiecărui rol.

13.1 Vizitator

GalleryFrontend Home Privacy

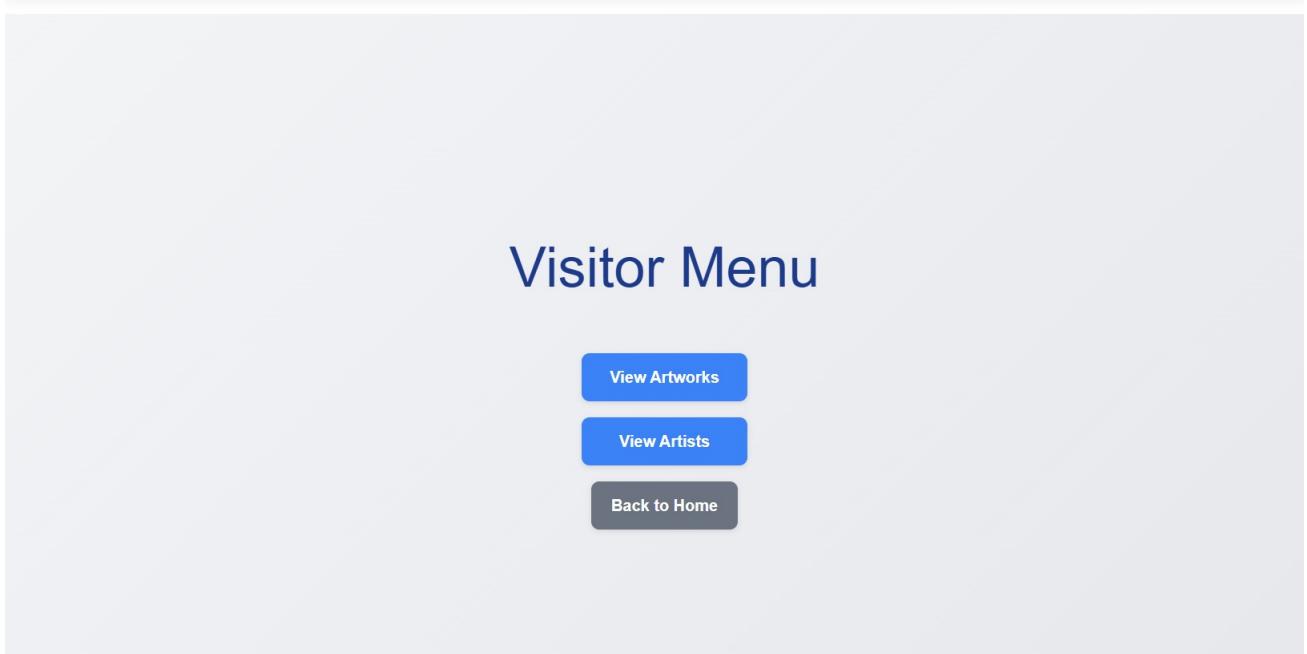


Figura 59: Dashboard-ul vizitatorului

GalleryFrontend Home Privacy

Artists				
Search by name				Search
Name	Birth Date	Birthplace	Nationality	Photo
Leonardo da Vinci	1452-04-15	Vinci, Italia	italian	<button>View</button>
Claude Monet	1840-11-14	Paris, Franta	francez	<button>View</button>
Pablo Picasso	1881-10-25	Malaga, Spania	spaniol	<button>View</button>
Frida Kahlo	1907-07-06	Coyocan, Mexic	mexicana	<button>View</button>
Constantin Brancusi	1876-06-07	Hobrita, Romania	roman	<button>View</button>

Figura 60: Pagina cu artiști

Available Artworks					
Title	Year	Type	Artist	Price	Images
Mona Lisa	1504	Painting	Leonardo da Vinci	1000000	<button>View</button>
The Last Supper	1498	Painting	Leonardo da Vinci	950000	<button>View</button>
Water Lilies	1906	Painting	Claude Monet	500000	<button>View</button>
Impression, Sunrise	1872	Painting	Claude Monet	700000	<button>View</button>

Figura 61: Pagina cu operele de arta

13.2 Angajat

GalleryFrontend Home Privacy

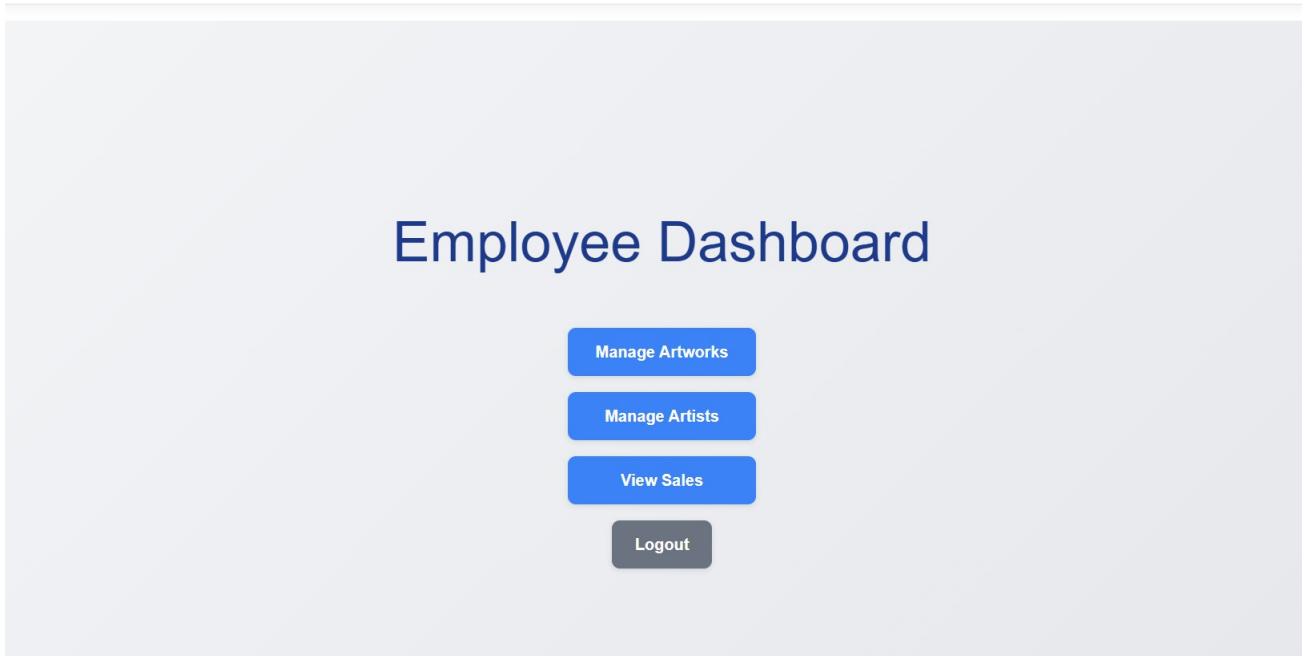


Figura 62: Dashboard-ul anagajatului

GalleryFrontend Home Privacy

Manage Artists					
Name	Birth Date	Birthplace	Nationality	Photo	Actions
Leonardo da Vinci	1452-04-15	Vinci, Italia	italian	<button>View</button>	<button>Edit</button> <button>Delete</button>
Claude Monet	1840-11-14	Paris, Franta	francez	<button>View</button>	<button>Edit</button> <button>Delete</button>
Pablo Picasso	1881-10-25	Malaga, Spania	spaniol	<button>View</button>	<button>Edit</button> <button>Delete</button>
Frida Kahlo	1907-07-06	Coyocan, Mexic	mexicana	<button>View</button>	<button>Edit</button> <button>Delete</button>

Figura 63: Pagina cu gestionarea artistilor

Manage Artworks					
<input type="text" value="Search by title"/> -- Select Type -- -- Select Artist --			<input type="button" value="Search"/> <input type="button" value="Filter by Type"/> <input type="button" value="Filter by Artist"/>		
<input type="button" value="Add New Artwork"/> <input type="button" value="Export CSV"/> <input type="button" value="Export JSON"/> <input type="button" value="Export XML"/>					
Title	Year	Type	Artist	Price	Actions
Mona Lisa	1504	Painting	Leonardo da Vinci	1000000	<input type="button" value="Edit"/> <input type="button" value="Delete"/> <input type="button" value="Add I..."/> <input type="button" value="Sell"/> <input type="button" value="View ..."/>
The Last Supper	1498	Painting	Leonardo da Vinci	950000	<input type="button" value="Edit"/> <input type="button" value="Delete"/> <input type="button" value="Add I..."/> <input type="button" value="Sell"/> <input type="button" value="View ..."/>
Water Lilies	1906	Painting	Claude Monet	500000	<input type="button" value="Edit"/> <input type="button" value="Delete"/> <input type="button" value="Add I..."/> <input type="button" value="Sell"/> <input type="button" value="View ..."/>

Figura 64: Pagina cu gestionarea operelor de artă

Sales							
Total Sales Value: 4650000 €							
Sale ID	Artwork Title		Employee Name	Sale Date	Price	Actions	
1	Mona Lisa		Employee One	2025-05-06	1000000 €	<input type="button" value="Dow..."/>	
2	Lobster Telephone		Employee One	2025-05-06	450000 €	<input type="button" value="Dow..."/>	
3	Bird in Space		Employee One	2025-05-07	900000 €	<input type="button" value="Dow..."/>	
4	Lobster Telephone		Manager2	2025-05-08	450000 €	<input type="button" value="Dow..."/>	
5	Seascapes Caribbean Sea		Manager2	2025-05-09	350000 €	<input type="button" value="Dow..."/>	
6	exemplu opera		Employee One	2025-05-20	1500000 €	<input type="button" value="Dow..."/>	

Figura 65: Pagina cu gestionare vanzari opere

13.3 Manager

GalleryFrontend Home Privacy

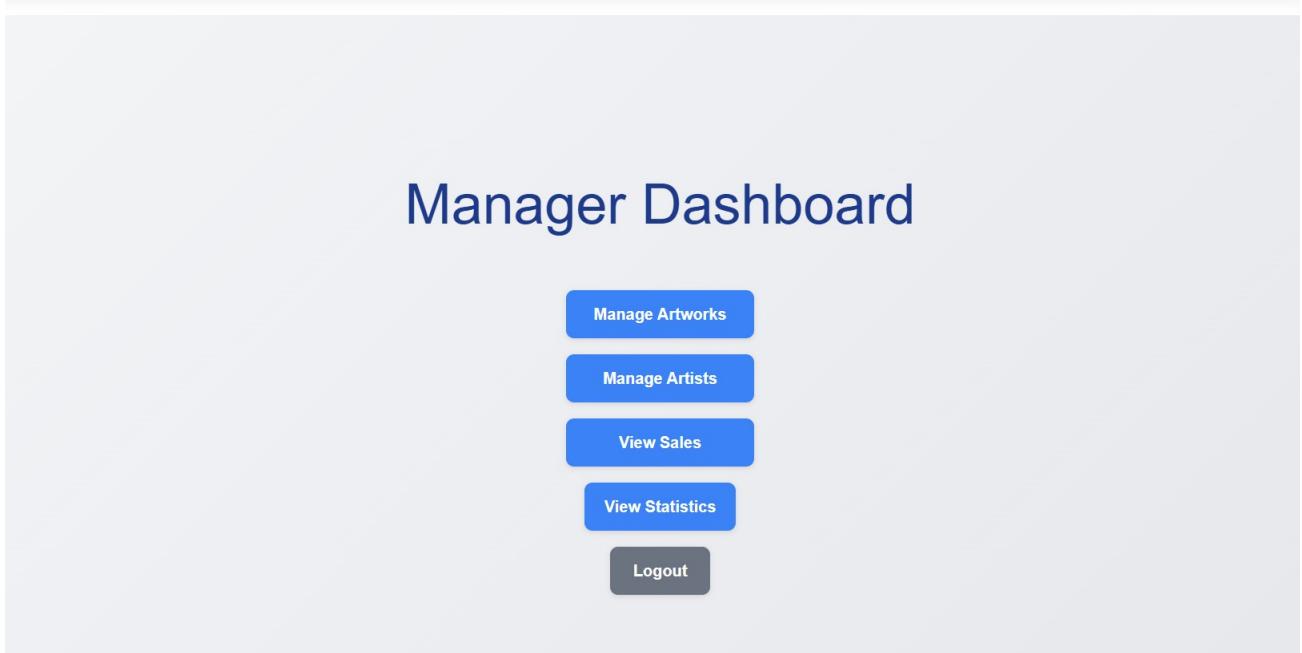


Figura 66: Dashboard-ul managerului

GalleryFrontend Home Privacy

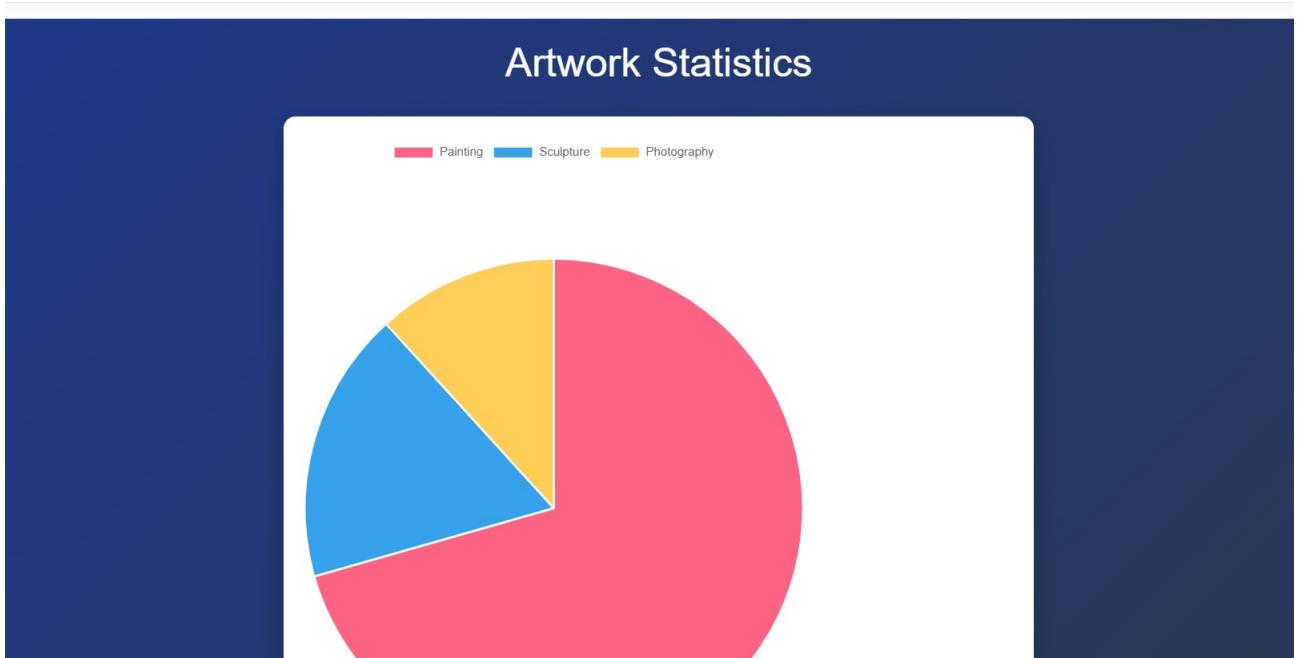


Figura 67: Pagina cu statistici legate de operele de artă

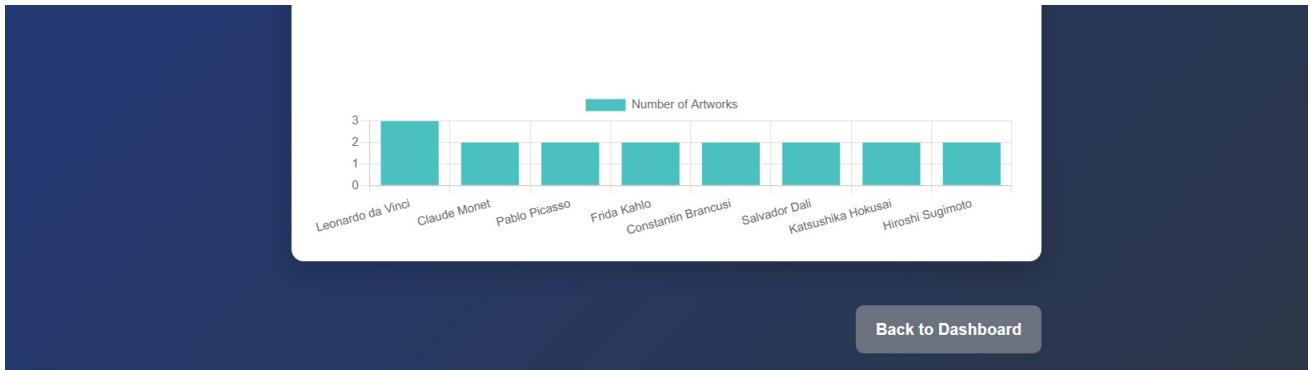


Figura 68: Pagina cu statistici legate de operele de artă

13.4 Administrator

GalleryFrontend Home Privacy

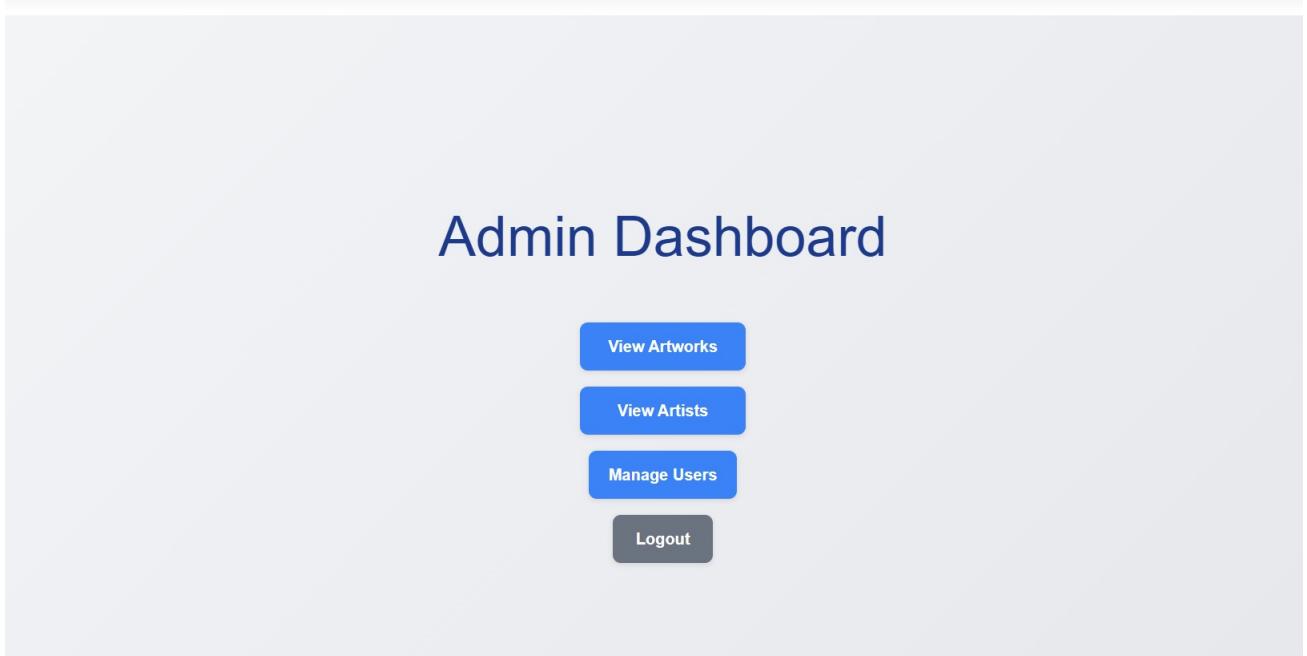


Figura 69: Dashboard-ul administratorului

GalleryFrontend Home Privacy

User Management					
Name	Email	Phone	Role	Actions	
Employee One	employee1@example.com	0748883817	Employee	<button>Edit</button>	<button>Delete</button>
Employee2	employee2@example.com	0748883833	Employee	<button>Edit</button>	<button>Delete</button>
Manager1	manager1@example.com	0558883833	Manager	<button>Edit</button>	<button>Delete</button>
Manager2	manager2@example.com	0988838123	Manager	<button>Edit</button>	<button>Delete</button>
Octavian	tavimirisan26@gmail.com	0748883817	Manager	<button>Edit</button>	<button>Delete</button>

Figura 70: Pagina cu gestionarea utilizatorilor

13.5 Shared

GalleryFrontend Home Privacy

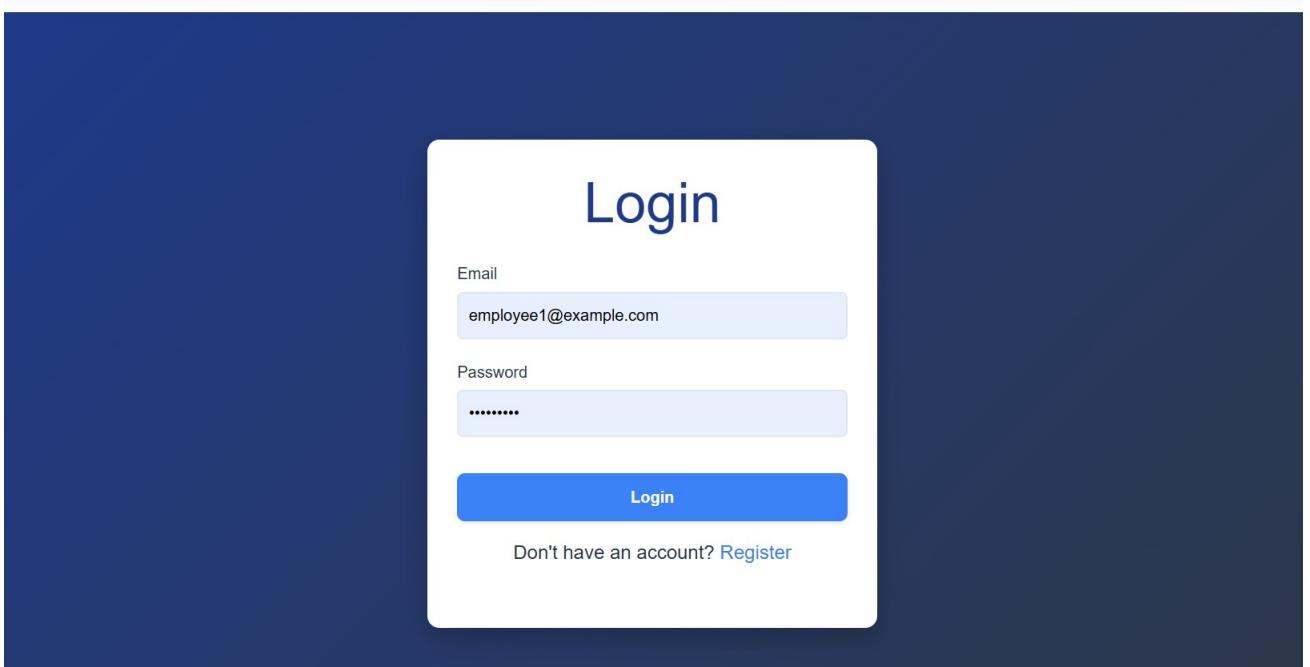


Figura 71: Pagina cu autentificarea in aplicatie

GalleryFrontend Home Privacy

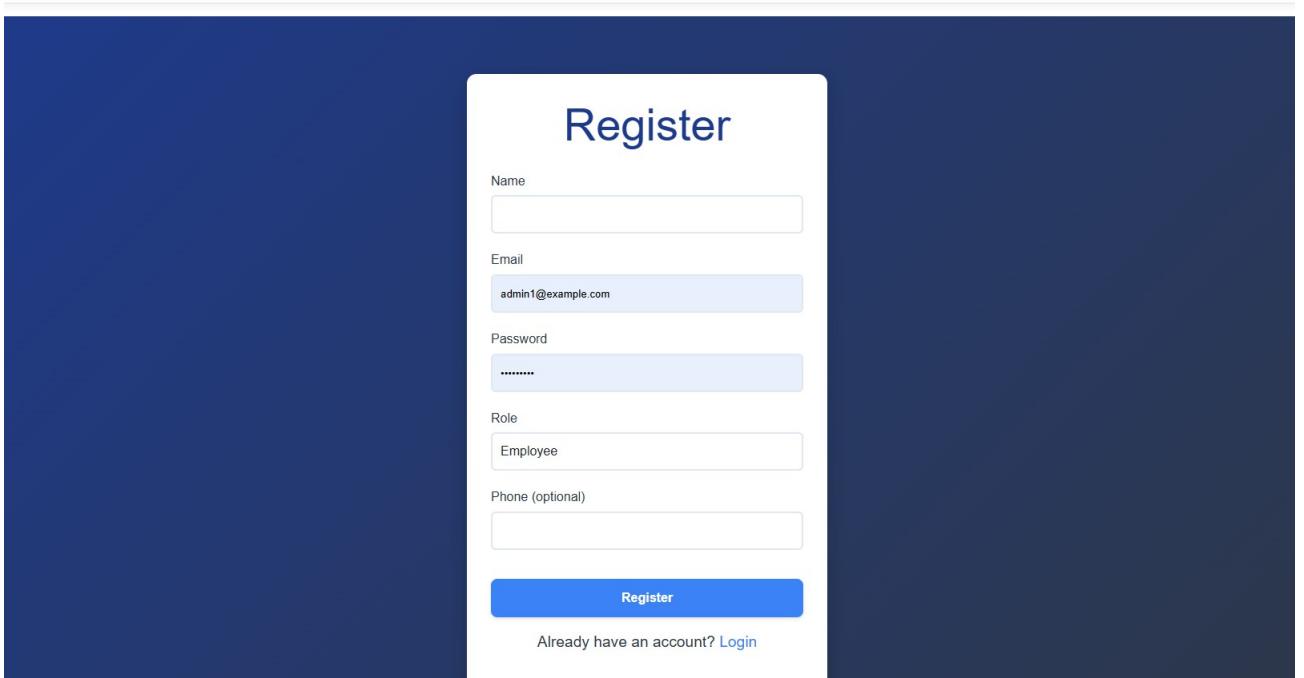


Figura 72: Pagina cu inregistrarea pe platforma

Concluzii

În urma realizării acestui proiect, am învățat:

1. Cum să dezvolt o aplicație modulară folosind arhitectura cu microservicii.
2. Cum să folosesc un API Gateway pentru a centraliza comunicația între servicii.
3. Cum să structurez corect codul în pachete: Controller, Service, Domain, Infrastructure.
4. Cum se aplică sabloane de proiectare precum Factory, Facade, Strategy, Observer și Template Method.
5. Cum se comunică între frontend și backend folosind API-uri REST.
6. Cum se construiește o interfață MVC care interacționează cu servicii externe.
7. Importanța diagramelor UML pentru înțelegerea și documentarea sistemului.
8. Cum să lucrez organizat, documentat și orientat pe funcționalitate reală.