***DOCUMENTAȚIE – TEMA 2 – MUZEE - MVVM***

***Sîrbu Cosmin-Claudiu, grupa 30231***

***1. Introducere:***

Acest proiect, denumit "MuseumApp", reprezintă o aplicație inovatoare concepută pentru a revoluționa experiența vizitatorilor în muzee, aducând în prim-plan tehnologia și accesibilitatea. Într-o lume în care cultura și arta sunt esențiale pentru îmbogățirea spirituală și educațională a societății, MuseumApp servește ca o punte între patrimoniul cultural și publicul modern, facilitând accesul la informații valoroase și îmbunătățind interacțiunea cu operele de artă.

Prin intermediul unei interfețe prietenoase și ușor de utilizat, MuseumApp oferă utilizatorilor posibilitatea de a explora colecții de artă din diverse muzee, oferind detalii fascinante despre operele de artă și artiștii lor. Aplicația încurajează descoperirea și învățarea autonomă, oferind totodată funcționalități interactive care transformă vizita la muzeu într-o experiență personală și profundă.

Dezvoltat cu un puternic accent pe accesibilitate și interactivitate, MuseumApp se adresează atât publicului larg, cât și profesioniștilor din domeniul muzeografic, educatorilor și studenților. Utilizatorii pot căuta opere de artă bazându-se pe criterii diverse, pot să învețe despre istoria și contextul fiecărei piese și pot chiar să participe la tururi virtuale ale expozițiilor.

Pentru muzee, MuseumApp reprezintă un instrument valoros de marketing și educație, permițând instituțiilor să-și extindă audiența și să ofere vizitatorilor o experiență îmbogățită, chiar și dincolo de zidurile fizice ale muzeului. Prin funcționalitățile sale, aplicația deschide noi orizonturi pentru explorarea artelor, făcând cultura mai accesibilă și atrăgătoare pentru publicul din era digitală.

***2. Cerința proiectului:***

Dezvoltati o aplicatie care poate fi utilizata in muzee. Aplicatia va avea 3 tipuri de utilizatori: vizitator al muzeului, angajat al muzeului si administrator.

Utilizatorii de tip vizitator pot efectua urmatoarele operatii fara autentificare:

1. Vizualizarea listei tuturor operelor de arta expuse in muzeu sortate dupa artist.

2. Filtrarea listei operelor de arta plastica dupa urmatoarele criterii: artist, tipul operei de arta, etc.

3. Căutarea unei opere de artă după titlu.

Utilizatorii de tip angajat pot efectua urmatoarele operatii dupa autentificare:

1. Toate operatiile permise utilizatorilor de tip vizitator.

2. Operatiile CRUD in ceea ce priveste persistenta operelor de arta expuse.

3. Salvare liste cu operele de artă expuse în mai multe formate: csv, json, xml, doc.

Utilizatorii de tip administrator pot efectua urmatoarele operatii dupa autentificare:

1. Toate operatiile permise utilizatorilor de tip vizitator.

2. Operatii CRUD pentru informatiile legate de utilizatorii care necesita autentificare.

3. Vizualizarea listei utilizatorilor care necesita autentificare.

4. Filtrarea listei utilizatorilor după tipul utilizatorului.

***3. Instrumente utilizate:***

IntelliJ IDEA este un mediu de dezvoltare integrat (IDE) robust și sofisticat, creat de JetBrains, destinat dezvoltării software-ului cu o eficiență și productivitate îmbunătățite. Apreciat de programatori pentru interfața sa intuitivă și setul bogat de funcționalități, IntelliJ IDEA sprijină mai multe limbaje de programare, inclusiv Java, Kotlin, Scala și Groovy, fiind ideal pentru dezvoltarea aplicațiilor web, mobile și enterprise.

Una dintre caracteristicile principale ale IntelliJ IDEA este capacitatea sa de analiză profundă a codului, care permite identificarea rapidă a erorilor și sugerează optimizări, îmbunătățind calitatea și performanța codului. Autocompletarea inteligentă, refactoringul puternic și sistemul de gestionare a versiunilor sunt alte funcții care fac din IntelliJ o alegere populară printre dezvoltatorii profesioniști.

Prin plugin-uri și personalizări, utilizatorii pot extinde funcționalitățile IDE-ului pentru a se potrivi nevoilor specifice ale proiectului. Fie că este vorba de dezvoltarea unei aplicații Android sau de construirea unui sistem enterprise complex, IntelliJ IDEA oferă un mediu de lucru coerent și puternic, menit să stimuleze creativitatea și eficiența dezvoltatorilor.

***4. Justificarea limbajului ales:***

Java este un limbaj de programare robust și versatil, ales datorită abilității sale de a răspunde eficient unei game largi de necesități de dezvoltare software. Este apreciat pentru portabilitatea sa, codul Java putând rula nealterat pe diverse platforme, fără a necesita modificări, grație sloganului „Write Once, Run Anywhere”. Această caracteristică îl face ideal pentru dezvoltarea de aplicații cross-platform, de la aplicații desktop și mobile, până la soluții complexe pentru servere.

Java se bucură de o comunitate vastă de dezvoltatori, ceea ce îi conferă un avantaj semnificativ prin bibliotecile extinse și framework-urile disponibile, facilitând astfel dezvoltarea rapidă și eficientă a aplicațiilor. De asemenea, Java este recunoscut pentru gestionarea automată a memoriei prin colectorul de gunoi, eliminând unele dintre cele mai comune probleme și erori asociate cu managementul memoriei.

Siguranța tipurilor, orientarea obiect și scalabilitatea sunt alte aspecte care contribuie la popularitatea Java. Aceste caracteristici permit dezvoltarea de sisteme robuste și sigure, care pot evolua ușor pe măsură ce necesitățile se schimbă. În plus, Java continuă să inoveze și să se adapteze la noile tehnologii, asigurându-și relevanța și eficacitatea într-o lume a tehnologiei în constantă schimbare. Astfel, pasiunea pentru Java nu este doar o preferință personală, ci și o alegere strategică pentru dezvoltarea de soluții software durabile și performante.

5. ***Diagrame UML:***

O imagine care conține text, diagramă, Plan, Tipărire

Descriere generată automat

*Figura 1. Diagrama de clase restransă.*

Această diagramă, din Figura 1, reprezintă o structură de clasă detaliată pentru o aplicație care folosește arhitectura Model-View-ViewModel (MVVM). MVVM este un pattern arhitectural folosit în dezvoltarea de aplicații, care separă logica de business de interfața cu utilizatorul, facilitând testarea și mentenanța.

În centrul diagramei, se găsesc clasele ViewModel care conțin logica de prezentare. Aceste clase fac legătura între Model și View, preluând datele și pregătindu-le pentru afișare. Ele sunt responsabile pentru gestionarea acțiunilor utilizatorului, actualizând modelul și răspunzând la schimbările acestuia.

Pe partea dreaptă, sunt reprezentate clasele Model care gestionează datele, regulile de business și operațiile de persistență. Clasele aici, cum ar fi `User` și `ArtWork`, par a fi entități de business care stochează date și sunt utilizate de ViewModel pentru a îndeplini cerințele aplicației.

Pe stânga, se observă View-urile care sunt interfețele grafice prin care utilizatorii interacționează cu aplicația. Ele sunt pasive, primind date de la ViewModel pentru a le afișa.

Secțiunea de jos a diagramei cuprinde un set de comenzi sub `Command`, care par să execute operații specifice la acțiunea utilizatorului, urmând pattern-ul de comandă pentru a decupla solicitarea unei operații de executantul acesteia.

Diagrama reflectă o separare clară a responsabilităților, o trăsătură esențială pentru menținerea și extinderea aplicației pe măsură ce aceasta evoluează. Prin distribuirea responsabilităților în diverse clase specializate, diagrama sugerează că aplicația poate fi adaptată ușor pentru a răspunde la noi cerințe fără a perturba alte părți ale codului.

O imagine care conține captură de ecran, alb și negru

Descriere generată automat

*Figura 2. Diagrama Use-Case*

Această diagramă, din Figura 2 de cazuri de utilizare prezintă interacțiunile diferitelor tipuri de utilizatori cu un sistem, probabil un software de management al unui muzeu sau o galerie de artă. Există trei roluri de utilizatori identificate: Vizitator, Angajat și Admin, fiecare având seturi distincte de interacțiuni posibile cu sistemul.

Vizitatorii pot vizualiza opera de artă și pot efectua acțiuni de filtrare a operelor după diferite criterii sau pot căuta opere după titlu. Aceste acțiuni sunt tipice pentru un utilizator obișnuit care navighează printr-o colecție de artă online.

Angajații au un set mai extins de responsabilități: adăugarea, ștergerea și actualizarea informațiilor despre operele de artă. De asemenea, pot genera rapoarte cu diverse formate de fișiere, ceea ce indică responsabilitățile lor de gestionare și de raportare în sistem.

Adminii au controlul complet, putând gestiona atât opera de artă, cât și utilizatorii sistemului. Ei pot adăuga, șterge, actualiza și căuta utilizatori, precum și să filtreze lista de utilizatori, sugerând că au acces la funcționalități administrative în vederea întreținerii sistemului.

Diagrama subliniază separarea clară a privilegiilor și funcțiilor între diferitele nivele de acces ale utilizatorilor, asigurând securitatea și organizarea eficientă a informațiilor din sistem.

O imagine care conține captură de ecran, diagramă

Descriere generată automat

*Figura 3. Diagrama UML – IntelliJ*

Această diagramă UML din Figura 3 este mai complexă și mai detaliată, ilustrând multiple relații și ierarhii între clasele unui sistem software. Diagrama arată structura internă a unui program, probabil același care gestionează opere de artă și utilizatori, cu o varietate de clase care reflectă diferitele entități și procese prezentate în cazurile de utilizare anterioare. Din cauza complexității și densității, o analiză mai detaliată ar necesita o vizualizare mai clară și posibil o explicație suplimentară a componentelor individuale.

***6. Descrierea aplicației:***

MuseumApp pare a fi o aplicație software structurată folosind pattern-ul arhitectural MVVM pentru a gestiona opera de artă și utilizatorii unui muzeu. În componenta de `Model`, există entități cum ar fi `ArtWork`, `User`, și diferite tipuri de utilizatori precum `MuseumAdmin`, `MuseumEmployee`, și `MuseumVisitor`, care reprezintă modelele de date și actorii sistemului.

Sub directorul `Command`, sunt definite o serie de acțiuni, cum ar fi `CAddArtWork`, `CSearchArtWork`, `CUpdateUser`, și multe altele, ce par a fi responsabile pentru executarea operațiilor specifice, fiecare comandă corespunzând probabil unui caz de utilizare specific în sistem.

`Repo` conține `ArtWorkRepository` și `UserRepository`, care sugerează obiecte care facilitează interacțiunea cu baza de date sau sursa de date pentru opera de artă și utilizatori, respectiv.

În secțiunea `View`, sunt listate UI-urile ca `AdminUI`, `ArtWorkListView`, și `Login`, indicând diferitele interfețe grafice prin care utilizatorii interacționează cu aplicația.

În cele din urmă, `ViewModel` conține obiecte cum ar fi `ArtWorkListViewModel` și `LoginViewModel`, care servesc ca legătură între model și interfața grafică, gestionând logica de afișare și răspunzând la acțiunile utilizatorului.

Această structură a proiectului sugerează o aplicație bine organizată care permite utilizatorilor să efectueze operații complexe într-un mediu muzeal, cum ar fi căutarea și administrarea operelor de artă și a conturilor de utilizator, cu o separație clară a responsabilităților care facilitează mentenanța și extensibilitatea.

În continuare vom lua pe pachete si vom descrie codul:

*Pachetul Model:*

Pachetul Model din proiectul MuseumApp cuprinde entitățile de bază ale aplicației, reprezentând structura de date și logica de afaceri. Clasa `ArtWork` reprezintă o operă de artă din cadrul muzeului, având proprietăți precum titlu, artist, anul creării și tipul operei de artă, fie că este vorba de pictură, sculptură sau altă formă de artă. Această clasă funcționează ca un DTO (Data Transfer Object), facilitând transferul de informații între straturile aplicației și baza de date.

Clasele `MuseumAdmin`, `MuseumEmployee` și `MuseumVisitor` extind clasa `User`, oferind roluri și permisiuni specifice în cadrul aplicației. De exemplu, un `MuseumAdmin` ar putea avea drepturi depline pentru a gestiona toate datele din sistem, în timp ce un `MuseumEmployee` ar putea avea acces limitat, concentrându-se pe operațiunile zilnice ale muzeului. `MuseumVisitor` ar putea reprezenta un utilizator al aplicației care nu necesită credențiale de autentificare, având acces doar la funcționalități de bază, cum ar fi vizualizarea operelor de artă.

Clasa `User` este o clasă abstractă de bază pentru toate tipurile de utilizatori ai aplicației, gestionând informații comune precum numele de utilizator, parola și tipul utilizatorului. Într-un mediu de producție real, ar trebui să implementăm măsuri de securitate robuste, deoarece în acest cod de exemplu, parolele sunt gestionate ca șiruri de caractere în text clar, ceea ce nu este sigur.

În esență, pachetul Model oferă o fundație solidă pentru reprezentarea datelor în cadrul aplicației, asigurând că toate informațiile esențiale sunt încapsulate și pot fi gestionate într-un mod care respectă principiile SOLID de programare orientată pe obiecte. Aceste clase permit o manipulare ușoară a datelor și oferă o punte între interfața utilizatorului, logica de afaceri și persistența datelor.

*Pachetul ViewModel:*

Pachetul `ViewModel` din aplicația MuseumApp este esențial pentru funcționarea modelului MVVM, servind ca un intermediar între View (interfața grafică) și Model (logică și date). Acesta conține mai multe clase ViewModel, fiecare destinată să gestioneze datele și logica de prezentare pentru diferite părți ale aplicației.

- `AdminViewModel`: Gestionează interfața și logica specifică pentru utilizatorii admin, inclusiv funcționalități de administrare.

- `ArtWorkListViewModel`: Se ocupă cu prezentarea listei de opere de artă, facilitând operații cum ar fi filtrarea și selecția operei de artă.

- `EmployeeViewModel`: Similar cu AdminViewModel, această clasă este orientată către nevoile și funcțiile specifice angajaților.

- `LoginViewModel`: Gestionează logica necesară pentru autentificarea utilizatorilor.

- `MainScreenViewModel`: Este responsabilă pentru logica de pe ecranul principal al aplicației, coordonând celelalte ViewModel-uri.

- `UsersFilterViewModel`: Specializată în filtrarea utilizatorilor, oferind funcționalități de căutare și sortare.

Fiecare ViewModel conține metode și proprietăți care reflectă acțiunile și datele necesare pentru View. Acestea reacționează la intrările utilizatorilor, solicită date de la Model și notifică View cu privire la orice schimbări necesare pentru a reflecta starea actuală a datelor sau rezultatul unei acțiuni.

Această structură asigură separarea clară a codului de prezentare de logica de afaceri și de accesul la date, permițând dezvoltarea independentă și testarea fiecărei componente, îmbunătățind, de asemenea, mentenanța și scalabilitatea aplicației.

*Pachetul Repo:*

Pachetul Repo este inima funcționalității de backend a aplicației MuseumApp, gestionând persistența datelor și comunicarea cu baza de date. Acest pachet conține clasele și interfețele necesare pentru a interoga și manipula datele referitoare la operele de artă și utilizatorii sistemului.

`ArtWorkRepository` este responsabil pentru operațiunile CRUD (Create, Read, Update, Delete) pentru obiectele ArtWork. Aici sunt definite metode precum `addArtwork`, care inserează o nouă operă de artă în baza de date, `getAllArtworks`, care recuperează toate operele de artă, sau `deleteArtwork`, care șterge o operă de artă specifică. Această clasă folosește `ConnectionBD` pentru a stabili conexiunea cu baza de date și pentru a executa interogările SQL necesare.

Clasa `ConnectionBD` este utilitarul care gestionează conexiunea cu baza de date, asigurându-se că toate tranzacțiile sunt executate într-un mod sigur și eficient. Aceasta poate folosi un pool de conexiuni pentru a optimiza accesul la resursele de rețea și la baza de date.

`DAOException` este o clasă de excepții care este aruncată când apar erori în operațiunile de baza de date, oferind informații detaliate despre problema întâmpinată pentru a facilita depanarea.

Interfețele `IArtWork` și `IUser` definesc contractele pentru operarea cu obiectele de tip ArtWork și User. Acestea sunt esențiale pentru a asigura decuplarea logică și flexibilitatea codului, permițând substituirea implementărilor pentru diferite surse de date sau pentru simularea comportamentului în testele unitare.

`UserRepository` îndeplinește un rol similar cu `ArtWorkRepository`, dar pentru entitățile de tip User. Acesta oferă metode pentru adăugarea, actualizarea, ștergerea și interogarea detaliilor utilizatorilor, precum și pentru autentificarea acestora în sistem.

În concluzie, pachetul Repo este crucial pentru asigurarea interacțiunii eficiente între aplicație și baza de date, encapsulând logica specifică de acces la date, oferind astfel o separare clară între straturile de logică de afaceri și persistență, ceea ce duce la o arhitectură de software solidă și ușor de întreținut.

*Pachetul View:*

Pachetul `View` din aplicația MuseumApp conține interfețele utilizator prin care interacționează cu sistemul. Acesta cuprinde componente UI specifice diferitelor funcționalități ale aplicației, reflectând separarea clară între diversele aspecte ale interfeței:

- `AdminUI`: Conține interfețe grafice dedicate administrației, unde administratorii pot gestiona diverse aspecte ale sistemului.

- `ArtWorkListView`: Evidențiază interfețele pentru listarea și gestionarea operelor de artă, oferind vizualizare și interacțiune specifică acestui segment.

- `EmployeeUI`: Include formularele și elementele vizuale necesare angajaților pentru a-și îndeplini sarcinile.

- `Login`: Un ecran esențial care gestionează autentificarea utilizatorilor, probabil oferind câmpuri pentru numele de utilizator și parolă.

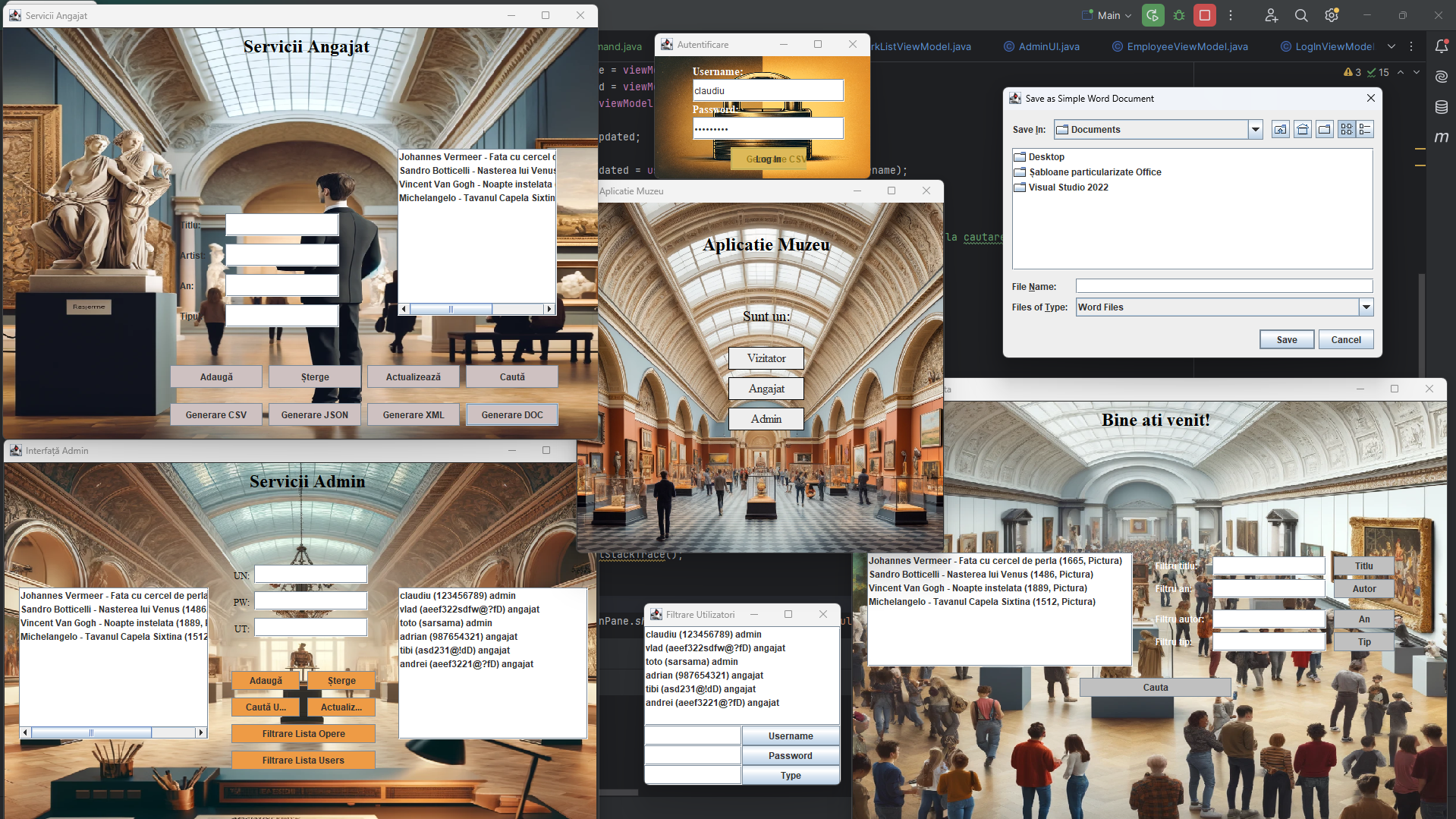
- `MainScreen`: Este ecranul central al aplicației, care coordonează navigarea principală și accesul la diverse funcționalități.

- `UsersFilterUI`: Specializată în filtrarea și căutarea utilizatorilor, cu interfețe care permit aplicarea de criterii specifice de filtrare.

Fiecare element UI este însoțit de un fișier `.form`, ceea ce sugerează utilizarea unei unelte de design vizual pentru crearea interfețelor, cum ar fi IntelliJ IDEA's GUI Designer sau similar. Aceasta înseamnă că UI-ul este probabil construit cu componente Swing în Java, permițând o definire vizuală și o implementare codificată a aspectelor interfeței grafice.

Aceste componente sunt legate de claselor lor `ViewModel` corespunzătoare pentru a afișa datele și a gestiona evenimentele generate de utilizator, respectând principiile MVVM de separare a logicii de prezentare de logica de afaceri și de gestionare a datelor.

Aplicatia vizualizată arată ca și în figura 4:



*Figura 4. Rularea aplicației și vizualizarea fiecărei ferestre.*

7. Concluzii:

MuseumApp se profilează ca o soluție software avansată și atent structurată, menită să îmbogățească experiența vizitatorilor de muzee și să eficientizeze gestionarea internă a instituțiilor muzeale. Bazată pe arhitectura MVVM, aplicația promovează separarea responsabilităților, ceea ce facilitează întreținerea și scalarea pe măsură ce noile cerințe emerg sau se schimbă. Fiecare pachet și componentă au roluri clar definite, asigurând o coeziune și o decuplare adecvate între diversele straturi ale aplicației.

Prin intermediul interfețelor prietenoase ale pachetului `View`, utilizatorii pot interacționa ușor cu sistemul, fie că sunt vizitatori, angajați sau administratori. Fiecare dintre aceste roluri este dotat cu funcționalități specifice, de la simpla vizualizare a operelor de artă până la gestionarea complexă a datelor de utilizator și opere de artă. Acest lucru nu doar că îmbunătățește accesibilitatea și utilizarea, dar sporește și valoarea educativă și culturală a aplicației.

Pachetul `ViewModel` adaugă un nivel esențial de abstracție între interfața utilizatorului și model, manipulând datele și pregătindu-le pentru prezentare. Astfel, logica de afaceri rămâne flexibilă și testabilă, iar interfața utilizatorului devine mai adaptabilă și mai ușor de actualizat. Componentele `Command` din proiectul MuseumApp exemplifică utilizarea eficientă a pattern-urilor de design în execuția acțiunilor, de la operații CRUD până la generarea de rapoarte, îmbunătățind în mod semnificativ interactivitatea și performanța.

Pachetul `Model` oferă structura de date și regulile de business, permițând ca aplicația să funcționeze într-un mod logic și coerent. Acesta stă la baza fiecărui proces din sistem, asigurând consistența și integritatea informațiilor gestionate. Pe de altă parte, `Repo` servește drept legătură crucială cu baza de date, facilitând interogările și persistența datelor, aspecte cheie pentru fiabilitatea aplicației.

Aplicația MuseumApp, prin utilizarea IDE-ului IntelliJ și limbajului Java, demonstrează capacitatea tehnologiei moderne de a răspunde nevoilor dinamicului sector cultural. Se evidențiază angajamentul pentru inovație și adaptabilitate, recunoscând importanța accesului la cultura și artă în format digital. MuseumApp nu este doar un produs software, ci o fereastră spre patrimoniul cultural, o resursă educativă și un catalizator pentru explorarea continuă și aprecierea artei.

În concluzie, MuseumApp reprezintă un proiect ambițios și bine gândit, cu potențial semnificativ pentru industria muzeală și pentru publicul său. Îmbinând tehnologia avansată cu nevoia de acces la cultura și educație artistică, proiectul poate servi drept model pentru alte inițiative care își propun să îmbogățească domeniul cultural prin digitalizare. Este un exemplu despre cum tehnologia poate fi folosită pentru a aduce istoria și arta mai aproape de publicul contemporan și pentru a deschide noi căi de interacțiune și învățare.