Documentație

# Proiectare

## Scopul aplicației

Se urmărește realizarea unei aplicații pentru gestionarea documentelor și cererilor de documente pentru primăria Cluj-Napoca.

## Cerințe

### Cerințe funcționale

* Persistența și consistența datelor și operații CRUD. Acest lucru se va realiza printr-o bază de date în MySQL, accesată printr-un framework de Object Relational Mapping , mai exact Hibernate
* Role-based access control. Rolurile sunt user-ul normal și adminul. Ele vor fi prezentate ulterior în detaliu
* Validarea datelor. Inputurile din câmpurile din interfața grafică trebuie verificate înainte de a fi introduse în baza de date. Acest lucru se va face în layer-ul de Service
* Interfață grafică. Of course, clienții din afara sferei de programatori au nevoie de o interfață grafică ușor de folosit. Ea va fi realizată în Swing

### Cerințe non funcționale

* Generare automată de documente
* Filtre și sortare pentru operațiile de căutare
* Afișarea rezultatelor din căutări în real time

## Diagrama Use Case

Diagram

Description automatically generated

## Descriere Scenarii de utilizare mai speciale

Nume scenariu: Creare request

Actor principal: User normal

Prerequisites: Utilizatorul este logat în aplicație

Scenariu principal de succes:

1. Utilizatorul intră în meniul de request-uri
2. Utilizatorul apasă pe un buton de creare request
3. Utilizatorul este redirecționat la o fereastră din care va alege tipul de document pentru care vrea să facă request
4. Utilizatorul este redirecționat la o fereastră în care completează datele pentru document
5. Utilizatorul apasă pe un buton de confirmare care îl va anunța dacă cererea lui a fost trimisă

Extensie:

1. Utilizatorul deja a trimis 3 cereri pe anul curent și nu are destule proprietăți pentru cereri suplimentare. În acest caz, el va primi o fereastră cu un mesaj de eroare când apasă pe buton la pasul 2
2. Utilizatorul nu completează un câmp din pasul 4, caz în care se va rămâne în acel pas, el fiind atenționat de acest lucru

Nume scenariu: Creare tip document

Actor principal: Admin

Prerequisites: Adminul este logat în aplicație

Scenariu principal de succes:

1. Adminul intră într-un meniu de gestionare a documentelor
2. Adminul apasă un buton de ”Creare nou tip document”
3. Adminului i se deschide o fereastră în care i se cere să inroducă informații cu privire la document (numele tipului de document, câmpuri, tipul câmpurilor etc.)
4. Adminul confirmă informațiile introduse printr-un buton, primind un mesaj de confirmare

Extensie: Adminul introduce un tip care are același nume cu un tip deja existent. În acest caz se va rămâne în pasul 3 până când se dă un nume diferit

## Arhitectura sistemului

Arhitectura sistemului va fi una multi-layer. Pentru depanare ușoară și organizare mai bună, se cere separarea diferitelor etape de procesare în aplicație în layere (Model, Repository, Service, Presentation)

* Model va conține clasele corespunzătoare modelelor din bazele de date
* Repository va avea diferite metode în care se fac metodele de interogări pe tabele
* Service va realiza validarea datelor pentru a avea consistență
* Presentation va conține clasele de interfață grafică și controller-ele pentru interacțiunea utilizator-aplicație

## Modelul bazei de date

Deși tabelele vor fi generate prin clasele de Java, tot trebuie să existe o idee despre relațiile care vor exista între ele, precum și o idee despre ce date trebuie să persiste. Notă: Fără tabelul de PersonalProperty

Prima variantă

Aici un câmp din tipurile de document ar fi tratat ca o entitate. Nu știu cât de eficient ar fi asta în ceea ce privește timpul de acces pentru ar reconstitui un document

Graphical user interface, application, Teams

Description automatically generated

Notă: ENUM-ul -urile au următoarele opțiuni în funcție de tabel

User: role: USER,ADMIN

Request: Status: SENT,OPENED, APPROVED, REJECTED

PersonalProperty: type: HOUSE, APARTAMENT

DocumentField: field\_type: NUMERIC, DECIMAL, STRING

Varianta a doua

Graphical user interface, application

Description automatically generated

Enum-urile având aceleași semnificații

Aici pentru fiecare tip de document m-am gândit să existe tabele generate dinamic cu câmpuri diferite în funcție de nevoie. Mai exotic de implementat

## Diagrama de clase

TBA

Notă: Nu pot estima niciodată absolut toate componentele mai speciale de care am nevoie în interfața grafică. Uneori fac clase separate, alteori doar fac metode speciale de styleComponent() in clasa generică de fereastră

## Paginile interfeței grafice

Diagrama următoare prezintă conceptual ferestrele din interfața grafică prin care trece un utilizator pentru a realiza funcționalitățile la care are acces. Ea este similară puțin cu un State Diagram cu trecerea între opțiuni combinată cu un Class Diagram (mai exact enumerarea componentelor grafice gen butoane și inputuri care ajută la îndeplinirea funcționalităților) . Nu știu dacă există astfel de diagrame în realitate, dar pe mine mă ajută să organizez mai bine interfața grafică. Cum se citește asta? Săgețile simple albastre pe care scrie condiții și acțiuni sunt la ce fereastră va fi redirecționat User-ul atunci când ia acțiunea respectivă și condițile sunt îndeplinite. Săgețile negre sunt relații normale de la UML

Diagram

Description automatically generated

# Detalii de implementare

## Tehnologii

Aplicația în sine se cere a fi implementată în Java

Modul de acces la baza date, dar și crearea modelului în aceasta se va realiza folosind framework-ul Hibernate. Baza de date în sine și dialectul ei va fi în MySQL

Interfața grafică va fi realizată în Java Swing

## Design Patterns

## Algoritmi și structuri de date folosite

## Biblioteci folosite

# Testare

# Concluzii

# Dezvoltări ulterioare

# Bibliografie

<https://www.baeldung.com/hibernate-criteria-queries>