Logo

Description automatically generated

ACADEMIA DE STUDII ECONOMICE BUCUREȘTI

Facultatea de cibernetică, statistică și informaticǎ economicǎ

**Proiect Modele de proiectare a aplicațiilor de întreprindere**

**Sistem pentru alocarea burselor elevilor sau studenților**

**Profesor coordonator:**

**Prof. Univ. Dr. Paul Pocatilu**

**Studenți:**

Coman Claudia Ana-Maria

Deaconu Andreea-Carmina

Teodorescu Raluca-Elena

București

2025

Cuprins

[Introducere 3](#_Toc187586615)

[Arhitectura sistemului 3](#_Toc187586616)

[Arhitectura aplicației 4](#_Toc187586617)

[Tehnologii Folosite 5](#_Toc187586618)

[Spring Boot: 5](#_Toc187586619)

[Baza de date H2 (In-memory): 5](#_Toc187586620)

[Thymeleaf (Frontend): 5](#_Toc187586621)

[Persistența Datelor 5](#_Toc187586622)

[Descrierea utilizării SGBD-ului H2 5](#_Toc187586623)

[Schema bazei de date (diagrama entitate-relație) 6](#_Toc187586624)

[Justificarea alegerii SGBD-ului în memorie 7](#_Toc187586625)

[Modele de proiectare utilizate 8](#_Toc187586626)

[Abstract Factory 8](#_Toc187586627)

[Composite 9](#_Toc187586628)

[Strategy 9](#_Toc187586629)

[State 9](#_Toc187586630)

[Implementarea aplicației 10](#_Toc187586631)

[Concluzie 11](#_Toc187586632)

# **Introducere**

Proiectul constă în dezvoltarea unui sistem modern și eficient pentru gestionarea burselor studenților, realizat cu ajutorul framework-ului Spring Boot. Acest sistem este conceput pentru a automatiza procesul de evaluare, ierarhizare și alocare a burselor, având ca obiectiv principal crearea unui proces transparent și corect pentru toți studenții eligibili.

Prin integrarea unei baze de date centralizate și utilizarea unor criterii clare de eligibilitate, sistemul facilitează o procesare rapidă și precisă a datelor studenților. De la colectarea informațiilor, la analizarea eligibilității și afișarea rezultatelor, platforma reduce considerabil efortul manual și riscul de erori, oferind în același timp o experiență ușor de utilizat.

**Funcționalități Principale**

Sistemul oferă funcționalități care facilitează gestionarea burselor pentru studenți, automatizând procesul de ierarhizare și alocare a acestora pe baza unor criterii bine definite.

Acesta permite evaluarea candidaturilor, atribuirea burselor corespunzătoare și afișarea detaliilor fiecărei burse, asigurând un proces transparent și eficient. De asemenea, sistemul include opțiuni de sortare și filtrare a datelor pentru o navigare ușoară și rapidă a informațiilor.

1. **Prezentarea informațiilor despre studenți -** Sistemul permite vizualizarea unui tabel cu informațiile studenților, care include: numele, media, anul de studiu, venitul familiei, participarea la competiții, problemele medicale și altele.
2. **Ierarhizarea automată a studenților -** Studenții sunt ierarhizați pe baza mai multor criterii, inclusiv performanța academică (media), venitul familiei, zone defavorizate și alte condiții speciale, iar acest scor determină eligibilitatea pentru diferite tipuri de burse.
3. **Alocarea burselor -** Sistemul alocă automat bursele studenților pe baza tipului acestora (bursa de performanță, bursă de merit, bursă socială, ajutor ocazional). Fiecare bursă atribuită este afișată cu detaliile corespunzătoare.
4. **Afișarea detaliilor burselor -** După procesul de alocare, pentru fiecare student este afișată o listă cu detaliile burselor acordate, incluzând tipul burselor și suma alocată, precum și starea fiecărei burse (ADMISĂ, ÎN EVALUARE, RESPINSĂ).
5. **Sortare și filtrare -** Sistemul permite sortarea și filtrarea datelor studenților în funcție de diferite criterii: numele, media academică, anul de studiu. Acest lucru facilitează o căutare rapidă a informațiilor dorite.

# **Arhitectura sistemului**

Aplicația se bazează pe o arhitectură client-server, în care frontend-ul și backend-ul interacționează pentru a oferi funcționalitățile de alocare a burselor pentru studenți. Detaliile aplicației sunt gestionate pe partea de backend, iar datele sunt stocate într-o bază de date in-memory (H2). Aplicația poate fi accesată printr-o interfață web simplă și folosind atât frontend-ul, cât și consola.

Arhitectura MVC controleaza app prin controller.start(), iar view e doar de afisat (printr-un final Scanner), iar Controller ul are un final CommandView ,filtru(),viz(),cautare(),modif() si start(), la toate facem load si folosim afisarea din view, cu trim() la citire. Metoda start() cu while true la fel ca la citireOptiune, afisam meniu, citim optiune si switch cu case-urile. In main avem: view, controller(view), controller.start()

Arhitectura MVP are in main: presenter=new presenter, view=new view(presenter), presenter.setview(view) deoarece depin una de alta, iar presenterul are iView. Acum start-ul e prin view. In view am si Scanner si Presenter si start() -> while true: afMeniu(), int optiune, presenter.handle(optiune) ca sa am switch ul in presenter.

## **Arhitectura aplicației**

1. **Frontend**: Interfața utilizatorului este construită folosind **Thymeleaf**, un motor de template integrat în Spring, care permite generarea dinamică a paginilor HTML. Acesta oferă o experiență ușor de utilizat, facilitând vizualizarea datelor și interacțiunea cu sistemul printr-o prezentare clară și interactivă a informațiilor.
2. **Backend (Spring Boot + Java)**: Partea de server, construită folosind framework-ul Spring Boot, care procesează cererile, execută logica de alocare a burselor și se conectează la baza de date H2 pentru stocarea informațiilor.
3. **Baza de date H2 (In-memory)**: H2 este o bază de date ușor de utilizat, care se execută direct în memorie, fiind ideală pentru aplicații mici, fără necesitatea de a configura o bază de date externă.
4. **Console Output**: Aplicația afișează în consolă atât informațiile despre studenți, cât și ierarhizarea și alocarea burselor.

Aplicația este structurată pe mai multe pachete pentru a respecta principiile de modularitate și separare a responsabilităților. Structura de pachete a aplicației arata astfel:

A screenshot of a computer

Description automatically generated

Figure 1. Structură pachete

Pachetul ***designPatterns*** este organizat în mai multe subpachete, fiecare destinat unui anumit model de design utilizat în aplicație (Abstract Factory, Composite, State și Strategy). Pachetul ***utils*** conține subpachetele:

1. ***model***: clasele de tip "model" care reprezintă entitățile (Student, Bursa).
2. ***controller***: clasele care gestionează cererile HTTP și coordonează fluxul de date între modelul de date și interfața utilizator, folosind un model MVC tradițional pentru a răspunde cu pagini web.
3. ***service***: logica de afaceri, cum ar fi ierarhizarea studenților, alocarea burselor și validarea acestora.
4. ***repository***: Conține interfețele și clasele care interacționează direct cu baza de date (utilizează Spring Data JPA).

# **Tehnologii Folosite**

## **Spring Boot:**

* **Spring Boot** este folosit pentru a construi rapid și eficient backend-ul aplicației, având avantajul unei configurații implicite și integrări cu diverse librării, inclusiv Spring Data JPA pentru gestionarea bazei de date.
* **Spring Data JPA**: Permite interacțiunea ușoară cu baza de date H2 in-memory.
* **Spring Web**: Folosit pentru crearea API-urilor RESTful care permit comunicarea între frontend și backend.

## **Baza de date H2 (In-memory):**

* **H2** este folosită în acest proiect pentru a stoca datele despre studenți și burse fără a fi nevoie de o configurație complexă a unei baze de date externe.

## **Thymeleaf (Frontend):**

* **Thymeleaf** este utilizat pentru structurarea dinamică a paginilor web, permițând integrarea datelor server-side direct în HTML. Interfața este simplă și intuitivă, oferind utilizatorilor o modalitate ușoară de a vizualiza informațiile și de a interacționa cu sistemul.

# **Persistența Datelor**

## **Descrierea utilizării SGBD-ului H2**

În cadrul proiectului, se utilizează H2 Database Engine, un sistem de gestionare a bazelor de date relaționale. Acesta este ales pentru simplitatea sa de configurare și integrarea perfectă cu Spring Boot, fiind ideal pentru dezvoltare și teste. Configurarea H2 este realizată pentru a opera în modul in-memory, ceea ce permite un acces rapid la date și evită persistența pe disc.

H2 este integrat prin Spring Boot, utilizând setări standardizate în fișierul *application.properties* pentru a defini conexiunile și comportamentul bazei de date. Configurația H2 este definită tot în fișierul *application.properties*, unde sunt specificate URL-ul de conexiune, driver-ul, numele de utilizator și parola.

A screenshot of a computer

Description automatically generated

Figure 2. Fișier H2

Aceste setări asigură că baza de date este accesibilă și configurată corespunzător pentru Hibernate, care gestionează operatiunile ORM.

A screenshot of a computer

Description automatically generated

Figure 3. Conectare H2

Pentru manipularea datelor, se folosește JPA împreună cu Hibernate, care facilitează maparea obiectelor Java pe structuri relaționale și invers, simplificând astfel operatiunile CRUD pe entitățile definite în proiect. Aceasta este o soluție eficientă și practică pentru gestionarea datelor în medii de dezvoltare, oferind performanțe și flexibilitate în testarea aplicațiilor Java.

## **Schema bazei de date (diagrama entitate-relație)**

Diagrama entitate-relație pentru baza de date utilizată în cadrul proiectului descrie structura și relațiile dintre tabelele **Studenti** și **Burse**.

A screenshot of a computer

Description automatically generated

Figure 4. Diagrama ER a bazei de date

Tabela Studenti stochează informații despre studenți relevante pentru condițiile de acordare a burselor. Fiecare student poate avea asociate una sau mai multe burse, reflectate în tabela Burse care include detalii precum tipul bursei, suma alocată, starea bursei și legătura către studentul beneficiar printr-un foreign key către Studenti.

A screenshot of a computer

Description automatically generated

Figure 5. Tabela Studenti

A table with numbers and letters

Description automatically generated

Figure 6. Tabela Bursa

Această structură permite gestionarea eficientă a datelor necesare pentru atribuirea burselor, oferind o vizualizare clară a relațiilor dintre studenți și bursele pe care le pot primi, îmbunătățind astfel managementul și raportarea în cadrul aplicației.

## **Justificarea alegerii SGBD-ului în memorie**

Alegerea sistemului de gestionare a bazelor de date H2  este motivată de eficiența pe care o oferă în procesul de dezvoltare. Folosirea H2 simplifică configurarea inițială prin eliminarea necesității instalării unei baze de date externe și oferă performanțe superioare datorită accesului direct la date din memoria RAM. Acest lucru permite și o resetare rapidă a stării datelor, esențială pentru testarea aplicației.

# **Modele de proiectare utilizate**

## **Abstract Factory**

Modelul **Abstract Factory** este un tipar de proiectare creațional care centralizează logica de creare a unor familii de obiecte (în cazul de față, diferite tipuri de burse), asigurând astfel coerență și ușurință în extinderea proiectului. Implementarea Abstract Factory din acest proiect se concentrează pe decuplarea codului client de detaliile specifice fiecărui tip de bursă (Merit, Performanță, Socială și Ajutor Ocazional).

**Single Responsibility Principle (SRP)**

Fiecare fabrică concretă (ex. BursaMeritFactory, BursaPerformantaFactory) are o singură responsabilitate: crearea instanței de Bursa aferentă tipului său. Astfel, logica de instanțiere pentru bursele respective nu este duplicată în alte zone ale aplicației. În plus, clasele Student și Bursa sunt proiectate respectând principiile SOLID, fiecare clasă fiind responsabilă exclusiv pentru gestionarea propriilor atribute.

**Open/Closed Principle (OCP)**

Codul client și cel al fabricilor existente nu necesită modificare când se adaugă un nou tip de bursă. Pentru a adăuga un alt tip de bursă, este suficient să se creeze o nouă fabrică concretă care implementează interfața BursaFactory și să fie înregistrată în BursaFactoryProvider. Aplicația rămâne astfel open for extension, but closed for modification.

**Dependency Inversion Principle (DIP)**

Clasele care utilizează bursele nu depind de implementările concrete, ci de abstracția BursaFactory. Acest lucru reduce puternic cuplarea dintre componente și face sistemul mai robust în fața schimbărilor.

**Interfața BursaFactory**

Definește metoda createBursa(), care reprezintă comportamentul de bază pe care toate fabricile de burse trebuie să îl implementeze.

**Fabrici Concrete**

Fiecare tip de bursă (Merit, Performanță, Socială, Ajutor Ocazional) are propria fabrică (de ex. BursaMeritFactory, BursaPerformantaFactory), ceea ce respectă SRP: fiecare fabrică concretă se ocupă exclusiv de crearea obiectelor acelui tip de bursă, fără să amestece logica altor burse.

**BursaFactoryProvider**

Conține metoda statică getFactory(TipBursa tip), care, pe baza enum-ului TipBursa, returnează o instanță de fabrică potrivită. Această abordare facilitează extinderea și mentenanța codului, respectând OCP, deoarece o nouă fabrică se poate atașa fără a modifica logicile deja existente pentru celelalte tipuri.

## **Composite**

În cadrul acestui proiect, am utilizat modelul **Composite** pentru a gestiona și manipula bursele studenților, în special pentru filtrarea și sortarea acestora. Modelul Composite este o tehnică de design structural care permite tratamentul uniform al obiectelor individuale și al colecțiilor de obiecte, considerându-le într-un mod similar.

Astfel, pentru gestionarea studenților și a burselor acestora, am folosit două componente principale:

1. **StudentLeaf** – Reprezintă o unitate de bază (un student), care poate fi considerat un "leaf" (frunză) în ierarhia structurii de date. Aceasta conține informații despre un singur student și include operațiile de filtrare și sortare (sau, în cazul "leaf"-urilor, evitarea sortării).
2. **StudentComposite** – Este un container care conține mai multe instanțe ale obiectelor StudentComponent (atât frunze, cât și compozite). Acesta gestionează colectarea studenților și permite aplicarea de operații comune asupra întregii colecții, cum ar fi sortarea și filtrarea acestora.

Modelul Composite facilitează adăugarea de funcționalități precum sortarea și filtrarea, aplicabile uniform fiecărui student, tratat individual sau în grup. Acesta distinge clar între gestionarea și vizualizarea studenților, permițând manipularea independentă a fiecărui StudentComponent și formarea de structuri complexe din elemente simple și colecții.

## **Strategy**

Modelul de proiectare **Strategy** permite definirea unui set de algoritmi, care pot fi selectați dinamic în funcție de context, fără a modifica codul client. În cadrul acestui proiect, modelul este utilizat pentru a calcula suma alocată burselor studenților, în funcție de tipul bursei: bursă de merit, bursă socială sau ajutor ocazional.

Fiecare tip de bursă are reguli specifice de calcul al sumei finale, iar implementarea este realizată prin intermediul interfeței BursaStrategy și a claselor concrete care implementează această interfață (BursaMeritStrategy, BursaSocialaStrategy, AjutorOcazionalStrategy).Folosind interfața BursaStrategy, codul care aplică strategiile nu depinde de implementările concrete.

## **State**

Modelul de proiectare **State** este utilizat pentru a reprezenta diferitele stări ale unei burse și pentru a defini comportamentele asociate fiecărei stări. O bursă poate trece prin următoarele stări:

1. **Admisa**: Bursa a fost aprobată, iar suma alocată este pregătită pentru distribuire. Aceasta poate fi creată și salvată în baza de date.
2. **In Evaluare**: Bursa se află în procesul de evaluare în funcție de valoarea variabilei *venitPerCapita* sau daca studentul este admis pentru ajutor ocazional, fiind nevoie de o anchetă.
3. **Respinsa**: Bursa a fost respinsă din cauza neîndeplinirii criteriilor.

Stările bursei sunt implementate ca obiecte separate care respectă o interfață comună. Fiecare stare implementează comportamentele specifice, cum ar fi notificarea studentului sau actualizarea statusului în baza de date.

Separarea comportamentelor în funcție de stări permite evitarea codului complicat bazat pe condiții (*if-else*) și face mai ușoară înțelegerea și extinderea aplicației. În plus, dacă este nevoie de adăugarea unei noi stări în viitor (de exemplu, *Revoked* – bursă retrasă), aceasta poate fi implementată ca o nouă clasă fără a afecta codul existent.

# **Implementarea aplicației**

Aplicația oferă flexibilitate în ceea ce privește testarea și rularea, permițând utilizatorilor să interacționeze cu funcționalitățile atât prin interfața grafică, cât și direct din consola de comandă. Aceasta include:

1. **Interfața Grafică Web:** Utilizatorii pot naviga prin paginile web pentru a accesa și gestiona datele. ([*http://localhost:8080/students*](http://localhost:8080/students))
2. **Consola de Comandă:** Aplicația include de asemenea un modul consolă, accesibil imediat după pornirea aplicației. Acest modul oferă un meniu simplu, bazat pe text, prin care utilizatorii pot executa operații direct din linia de comandă.

Operațiile legate de studenți si bursele lor sunt gestionate în back-end prin intermediul clasei *StudentController*, folosind următoarele end-point-uri:

1. **GET /students** - obținem o listă cu studenți ce poate fi sortata/filtrată
2. **POST /alocare-burse** – bursele sunt alocate conform cerințelor și utilizatorul este mai apoi redirecționat către /students/burse
3. **GET /burse** - afișează detaliile legate de studenți si burse în urma alocării, permitând sortarea si filtrarea acestora

În continuare, clasele *BursaService* și *StudentService* facilitează interacțiunea dintre aplicație și baza de date, oferind metode pentru extragerea datelor și alocarea burselor.

Din punct de vedere al criteriilor de acceptare pentru cele patru tipuri de burse s-au folosit următoarele condiții:

1. **Bursa de Performanță** este destinată studenților dintr-un an care se clasează în primii 2% din totalul studenților pe baza mediei academice.
2. **Bursa de Merit** se acordă studenților dintr-un an care urmează imediat după primii 2% și se încadrează în următorii 10% din totalul studenților, considerând mediile obținute.
3. **Bursa Socială** este destinată studenților cu venituri familiale modeste. Studenții ale căror familii au un venit per membru de familie sub 1500 lei sunt considerați eligibili pentru această bursă.
4. **Bursa de Ajutor Ocazional** este acordată în circumstanțe speciale, cum ar fi cazurile în care un student se confruntă cu probleme medicale grave sau decese în familie.

Fiecare tip de bursă este creat având sume pre-stabilite prin intermediul modelului de proiectare **Abstract Factory**. După stabilirea sumei inițiale, se aplică modelul **Strategy** pentru a adapta suma finală a bursei în funcție de circumstanțele specifice ale fiecărui student.

În plus, modelul **State** este folosit pentru a gestiona cele trei tipuri de stări în care se pot afla cererile pentru burse. Operațiile de sortare și filtrare sunt gestionate de modelul **Composite**, oferind flexibilitate aplicației.

În ultimul rând, utilizatorul poate vizualiza lista studenților, bursele acordate și motivele pentru care unele cereri au fost respinse.

# **Concluzie**

Implementarea sistemului de gestionare a burselor prin modelele de proiectare Abstract Factory, Strategy, State și Composite a optimizat configurarea, calculul și gestionarea stărilor burselor, demonstrând eficiența aplicării principiilor de inginerie software în dezvoltarea de soluții robuste. Flexibilitatea interacțiunii prin interfața utilizator și consolă asigură accesibilitate și ușurință în testare și utilizare, reflectând adaptabilitatea sistemului în diferite scenarii de utilizare.