

Sistemul de management al proiectelor

1. Descrierea cerințelor sistemului

În cadrul companiei se dorește dezvoltarea unei aplicații Web pentru gestiunea proiectelor care să permită utilizatorilor evidența proiectelor, a activităților specifice fiecărui proiect, a resurselor umane și materiale repartizate pentru implementarea proiectelor. Categoriile de utilizatori ai aplicației sunt: administrator, manager de proiect, membru în echipa proiectului.

Principalele cerințe funcționale ale aplicației au fost grupate în următoarele patru module: Proiecte, Activități, Angajați și Resurse materiale. În fiecare pagină a aplicației va fi inclusă o facilitate de căutare (proiect, activitate, responsabil, resursa).

1) Modulul Proiecte

În cadrul acestui modul, managerii vor putea să adauge proiecte noi, să le urmărească pe cele în desfășurare și să le analizeze pe cele finalizate deja. Un proiect este realizat la comanda fermă a unui client, este coordonat de un manager de proiect (PM) și presupune realizarea mai multor activități. Există mai multe tipuri posibile de proiecte. Până în acest moment au fost realizate proiecte de tipografie (carti) și proiecte de dezvoltare de site-uri Web, dar lista este deschisă.

- **Adaugă proiect** - la definirea unui nou proiect, se vor completa câmpurile: codul și numele proiectului, tipul proiectului, datele de identificare ale clientului (cod, nume, adresa), data de început, data de sfârșit, managerul de proiect, observații și starea proiectului. Un proiect se poate afla în una din stările „propunere”, „în execuție”, „abandonat” și „finalizat”. În momentul creării proiectul va fi în starea „propunere”. După salvarea proiectului, utilizatorul ar putea continua cu adăugarea de activități pentru proiectul curent.
- **Modifică proiect** - pe parcursul ciclului de viață a proiectului, managerul poate să modifice datele, inclusiv să schimbe starea acestuia. La crearea sa, proiectul va trece în starea „propunere”; el va trece automat în starea „în execuție” când cel puțin o activitate va trece în starea „în execuție”; proiectul poate trece în starea „finalizat” numai dacă toate sarcinile alocate sunt în starea „finalizată”; un proiect poate fi anulat numai dacă se află în una din stările „propunere” sau „în execuție”, caz în care vor fi anulate toate activitățile aflate „în execuție”. La finalizarea proiectului, managerul trebuie să completeze numărul de ore-om efectiv de realizare a proiectului, pentru a-l folosi ca referință în estimările viitoare.
- **Anulează proiect** – permite abandonarea unui proiect din varii motive. Proiectul va trece în starea „abandonat”, iar toate activitățile nefinalizate vor fi anulate (trecute în starea „abandonată”).
- **Vizualizează proiecte** – se afișează date generale pentru fiecare proiect, implicit doar pentru cele „în execuție”, iar la cerere toate proiectele (nume proiect, nume client, datele de început și final, starea proiectului etc. De asemenea, la cererea utilizatorului, vor fi afișate și detaliile în legătură cu un proiect: activitățile alocate, responsabilul pentru fiecare sarcină etc. La

vizualizarea detaliilor unui proiect se vor calcula și afișa totaluri privind orele-om planificate, realizate, rămase și extra la nivelul proiectului.

2) Modulul Activități

În acest modul se gestionează activitățile aferente proiectelor create în sistem. Aceste operațiuni pot fi realizate doar de către managerul proiectului.

- **Creare activitate** – crearea unei activități noi poate fi realizată la momentul definirii unui proiect nou (vezi funcționalitatea „Adaugă proiect”), dar și pe parcursul derulării proiectului, atât timp cât el se află „în execuție”. La crearea unei activități trebuie introduse codul și denumirea ei, data de început și termenul de finalizare, codul și numele proiectului, angajatul care va fi responsabil cu realizarea ei (poate fi făcută și ulterior creării ei), descrierea și numărul orelor-om estimate pentru realizarea ei. La momentul creării o activitate va intra în starea „propusă”.
- **Repartizare activitate** - O activitate este repartizată unui angajat care va avea responsabilitatea pentru realizarea ei. Această operațiune este realizată dacă nu se cunoștea responsabilul în momentul creării activității. La repartizarea unei activități, angajatul responsabil va fi înștiințat prin email.
- **Asumare activitate** – După ce a primit, prin email, notificarea privind numirea ca responsabil pentru o activitate, angajatul va trebui să accepte sau să refuze în termen de 3 zile. Dacă timp de 3 zile angajatul nu dă nici un răspuns, atunci managerul de proiect va face o altă repartizare.
- **Modificare activitate** – se pot modifica datele privind activitățile create. Numărul orelor-om estimat nu mai poate fi modificat după începerea realizării activității. Stările unei activități sunt: „propusă”, „repartizată”, „asumată”, „în execuție”, „finalizată”, „abandonată”, „acceptată”, „respinsă”. După finalizare, o activitate este supusă unui proces de acceptare de către client care poate valida rezultatele sau le poate respinge. La finalizarea unei activități, managerul proiectului va completa numărul de ore lucrate, ce va fi utilizat la evaluarea angajaților și a cursului proiectului.
- **Vizualizare activități** – la deschiderea paginii vor fi afișate doar activitățile curente (nefinalizate), ordonate după proiect, alfabetic după denumirea activității sau în funcție de numele responsabilului. În funcție de opțiunile utilizatorului, vor putea fi afișate toate activitățile sau filtrate în funcție de stare, proiect sau responsabil. La vizualizarea sarcinilor se va realiza o formatare condiționată care să evidențieze sarcinile “în execuție” cu termenul depășit, precum și diferențierea celor “în curs de execuție” față de cele finalizate.

3) Modulul Responsabili

În acest modul se vor gestiona datele privind angajații care au primit responsabilități de realizare a unor activități în diferite proiecte.

- **Adaugă responsabil** – se adaugă datele pentru un angajat nou, precum numele, marca, funcția etc.
- **Modifică date responsabil** – permite actualizarea datelor privind angajații.
- **Șterge responsabil.**
- **Asumare activitate.** Se va afișa o listă a activităților repartizate unei persoane și neasumate încă. Din această listă utilizatorul va putea selecta o activitate.

- **Vizualizează date responsabil** – se afișează o listă cu persoanele care pot fi membri în echipa de proiect și datele despre aceștia, precum numele, marca, funcția etc. La cererea utilizatorului se pot afișa detalii privind activitățile repartizate sau realizate de o persoană (codul și denumirea ei, data de început, termenul de finalizare, data finalizării, codul și numele proiectului).

4) Modulul Resurse materiale

În acest modul se realizează gestiunea resurselor materiale necesare pentru derularea fiecărei activități.

- **Adăugă resursă.**
- **Modifică date resursă.**
- **Alocare resursă** – Se specifică resursa materială și cantitatea necesară pentru realizarea unei activități. O resursă de tip material se consumă dintr-o dată (toată cantitatea), la prima utilizare. Este posibil să nu se consume întreaga cantitate planificată, situație în care se va consemna cantitatea consumată efectiv în momentul trecerii activității în starea “finalizată”.
- **Vizualizare resurse** – se afișează datele privind o resursă materială (cod, denumire, um, stoc, etc.). La solicitarea utilizatorului se pot afișa detalii privind activitățile în care a fost utilizată o resursă materială.

2. Arhitectura de referință selectată pentru acest sistem

Arhitectura care va fi utilizată la implementarea aplicației conține următoarele straturi (fig. 1):

1. Presentation Layer: (HTML + Java script) – afișare informații, interacțiuni cu utilizatorul, transformare evenimente UI în apeluri către celelalte straturi ale aplicației. Pentru acest strat s-a utilizat șablonul arhitectural MVC.
2. Domain Layer – conține clasele care mapează conceptele specifice domeniului problemei pentru care se dezvoltă aplicația. Vom avea clase Entity și Service, conform Domain Driven Design (DDD).
3. Data Layer – conține clasele cu rol de asigurare a accesului la stratul de persistență. În aplicația noastră vom utiliza șablonul Repository.
4. Infrastructure Layer – conține toate bibliotecile și API-urile necesare implementării aplicației.

Interacțiunile de principiu, conform arhitecturii de referință. sunt prezentate cu ajutorul diagramei de secvențe din fig. 2.

cmp Arhitectura de referinta

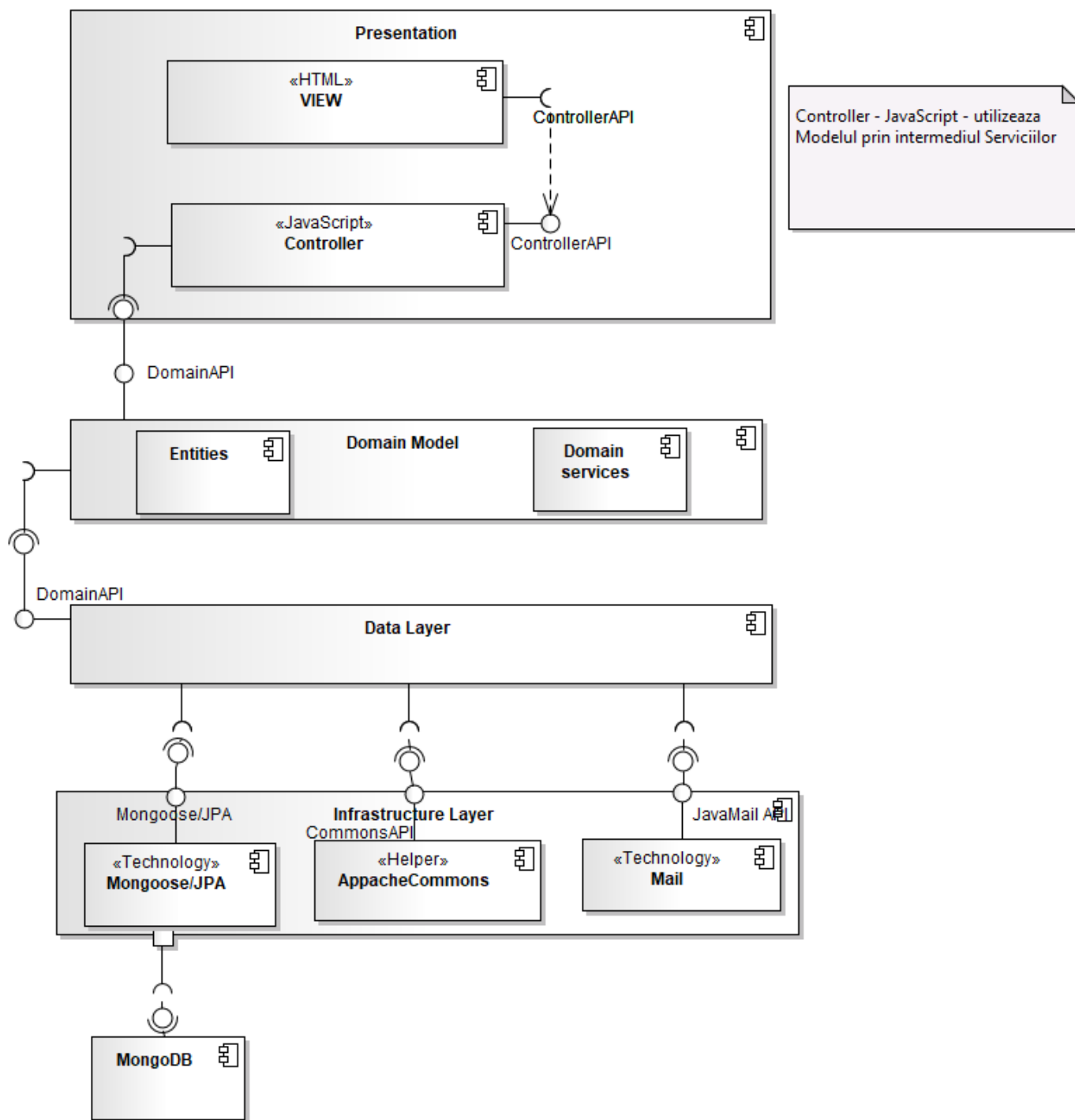


Figura 1 Arhitectura de referință a aplicației

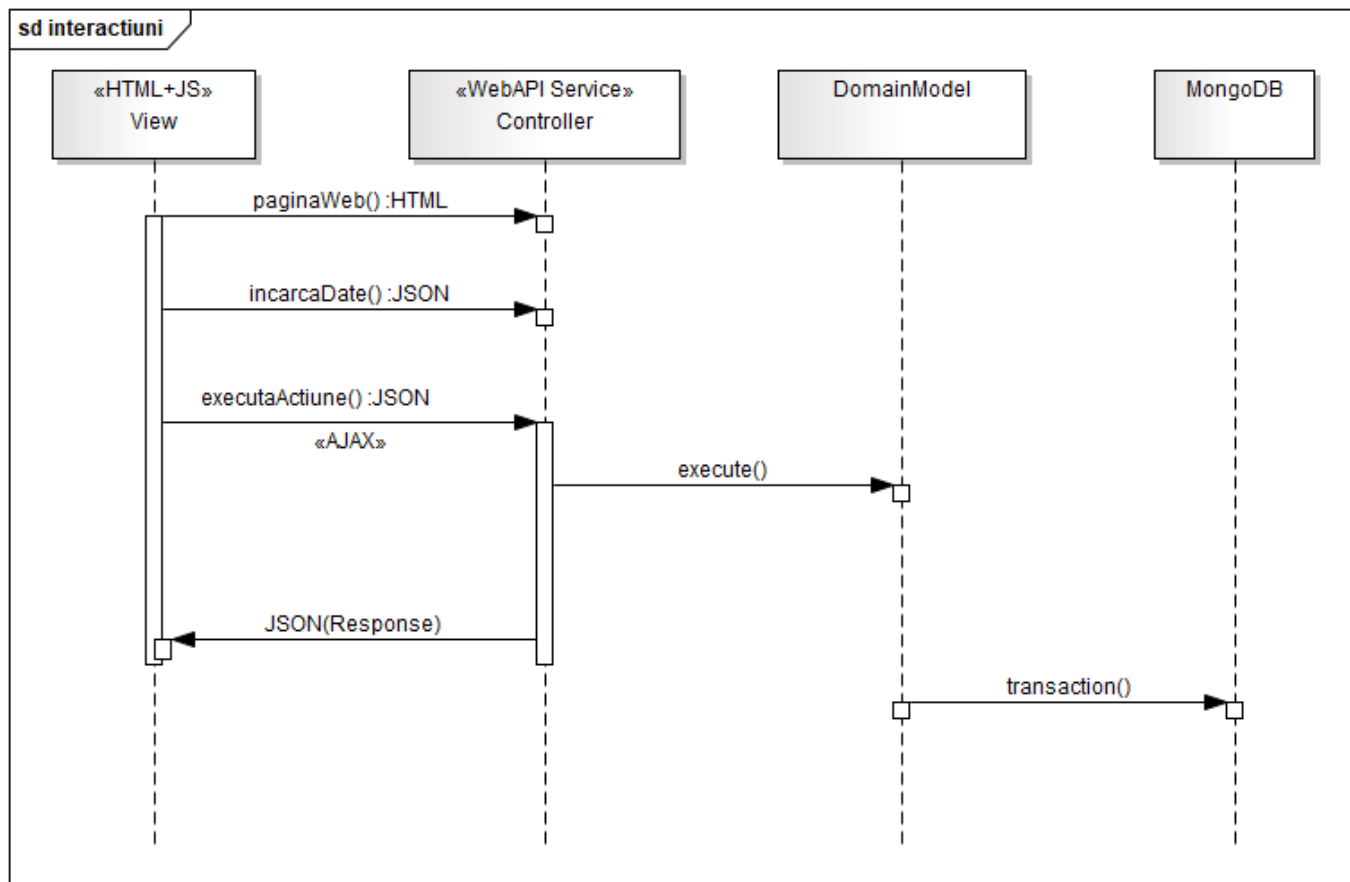


Figura 2 Interacțiunile dintre straturile arhitecturii de referință

3. Structurarea cerințelor funcționale ale aplicației

Pentru structurarea cerințelor funcționale se vor utiliza diagrama de activități și diagrama cazurilor de utilizare. Diagrama de activități poate fi folosită pentru descrierea logicii proceselor economice, respectiv a principalelor funcționalități ale aplicației și a succesiunii în care ele sunt realizate (fig. 3); această diagramă va sta la baza identificării principalelor cazuri de utilizare.

De asemenea, diagrama de activități poate fi utilizată pentru descrierea logicii cazurilor de utilizare sau chiar scenariilor de lucru ceva mai complexe. În proiectul de față nu a fost cazul.

3.1 Diagrama de activități pentru descrierea procesului economic

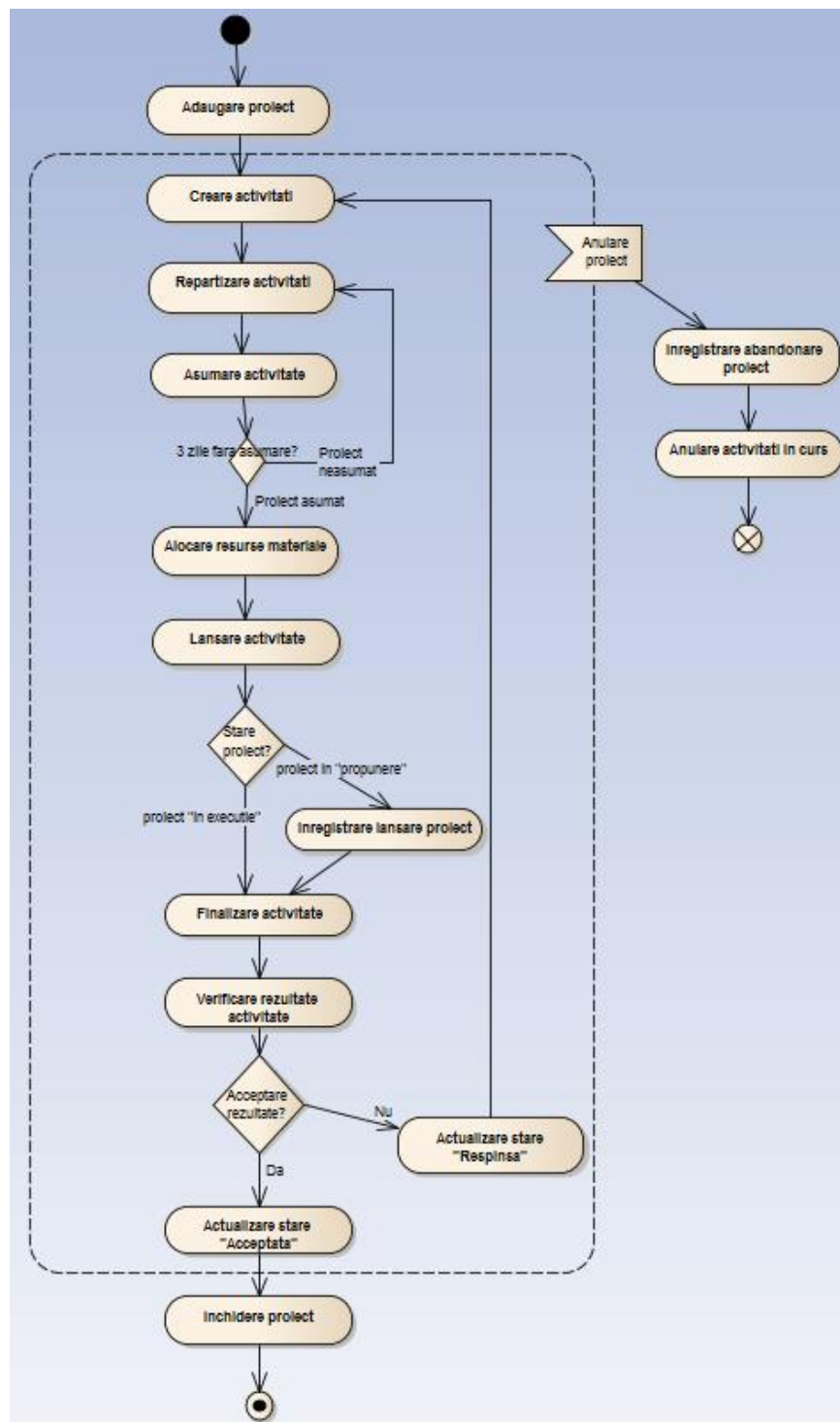


Figura 3 Diagrama de proces pentru managementul proiectelor

3.2 Diagramele cazurilor de utilizare

Avand în vedere complexitatea funcțională a aplicației, s-a pornit cu crearea unei diagrame de pachete (fig. 4), în care se realizează partiționarea funcționala a aplicației. Diagrama prezintă cele 4 funcțiuni principale ale aplicației, sub forma de pachete. Crearea ei a fost necesară pentru a controla complexitatea aplicației, dat fiind că diagramele cazurilor de utilizare nu pot fi descompuse ca și diagramele fluxurilor de date (vezi disciplina ASI din anul 3!)

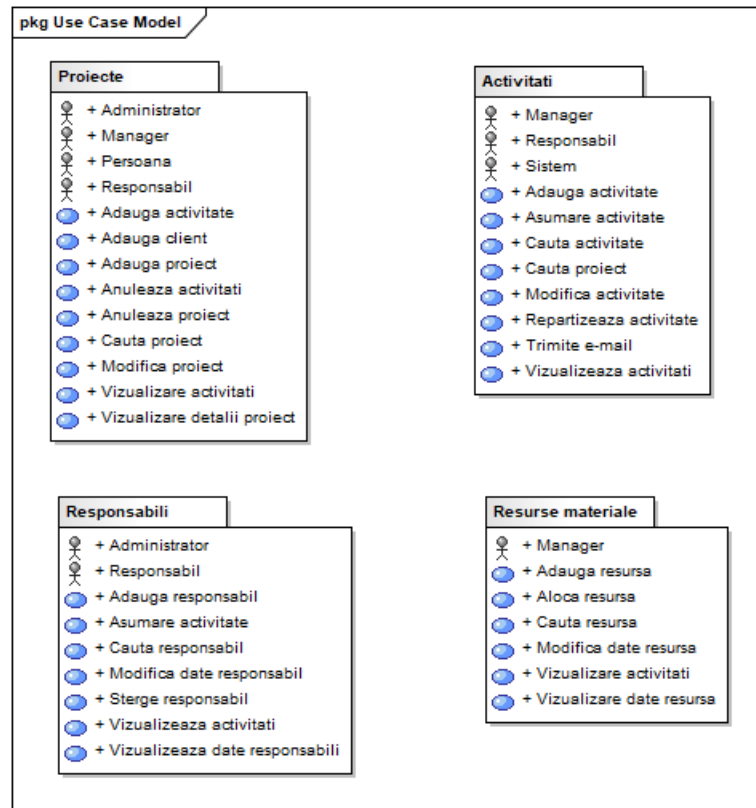


Figura 4 Funcțiunile aplicației

Pentru fiecare dintre cele patru funcțiuni principale descrise în primul capitol și evidențiate în diagrama de pachete se va construi câte o diagramă a cazurilor de utilizare. În continuare vor fi prezentate cele 4 diagrame (fig. 5-8).

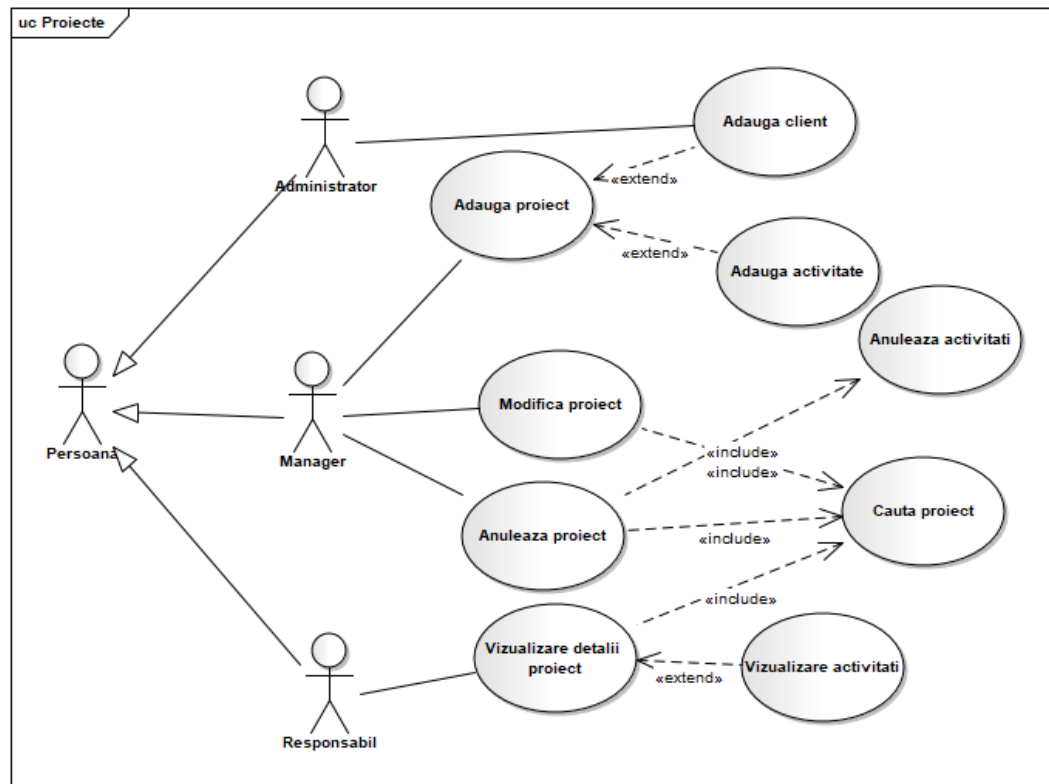


Figura 5 DCU pentru modulul „Proiecte”

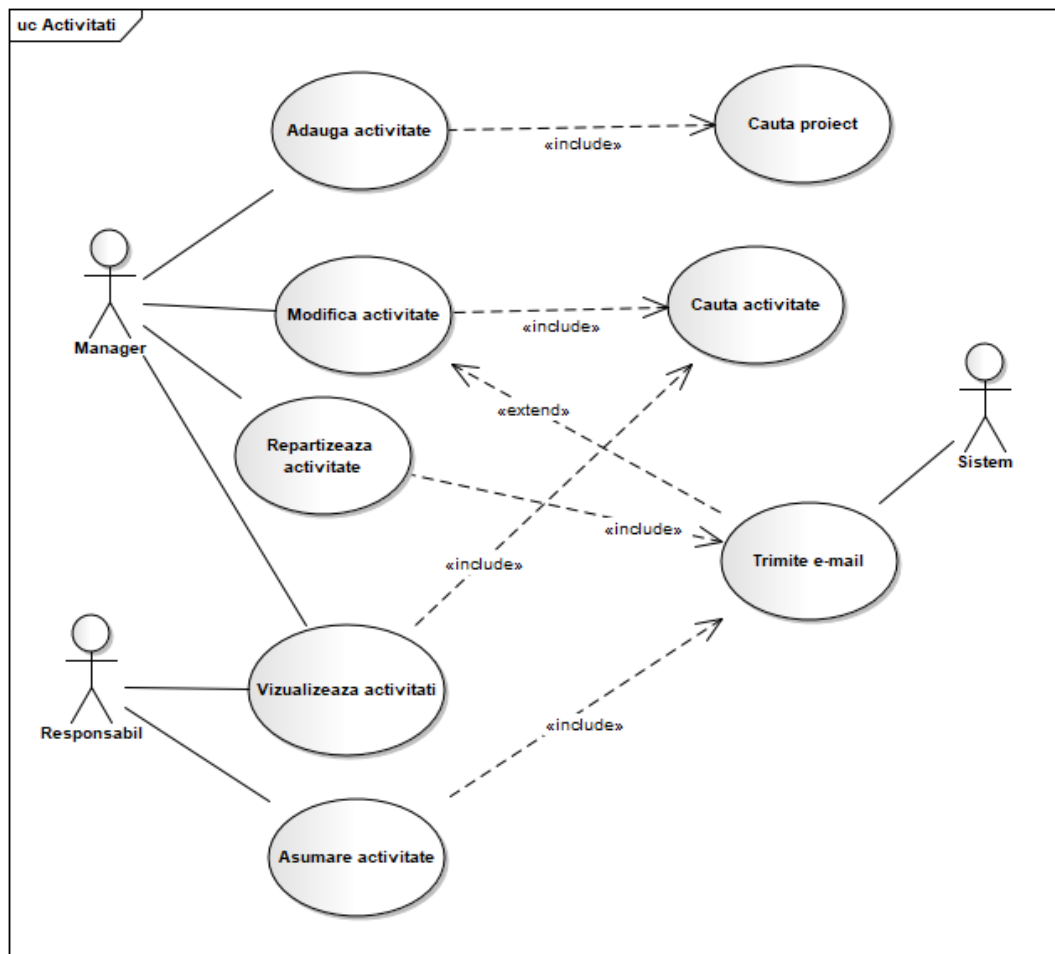


Figura 6 DCU pentru modulul „Activități”

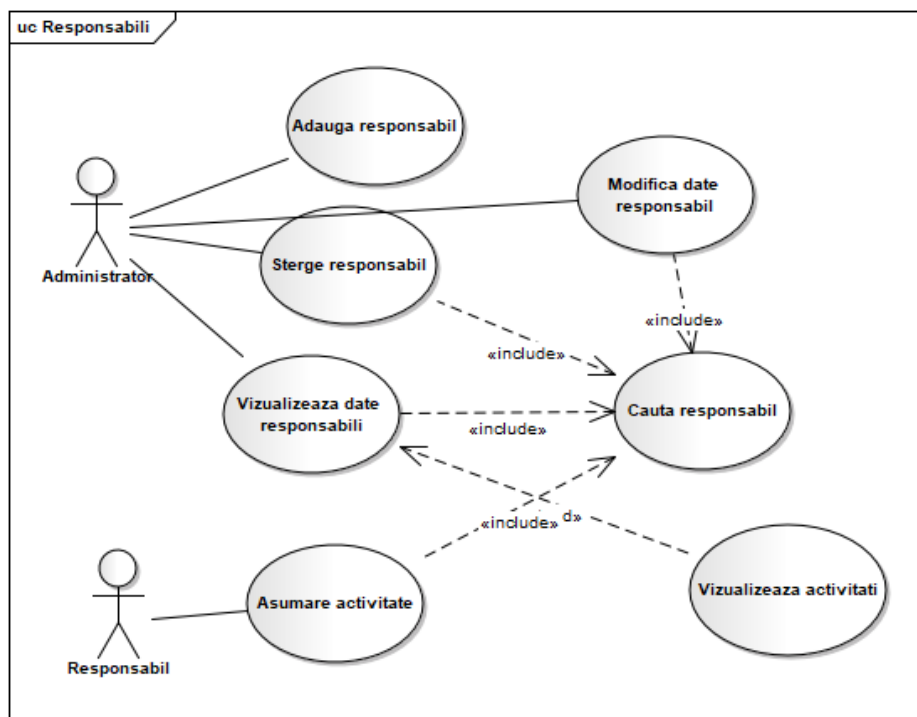


Figura 7 DCU pentru modulul „Responsabili”

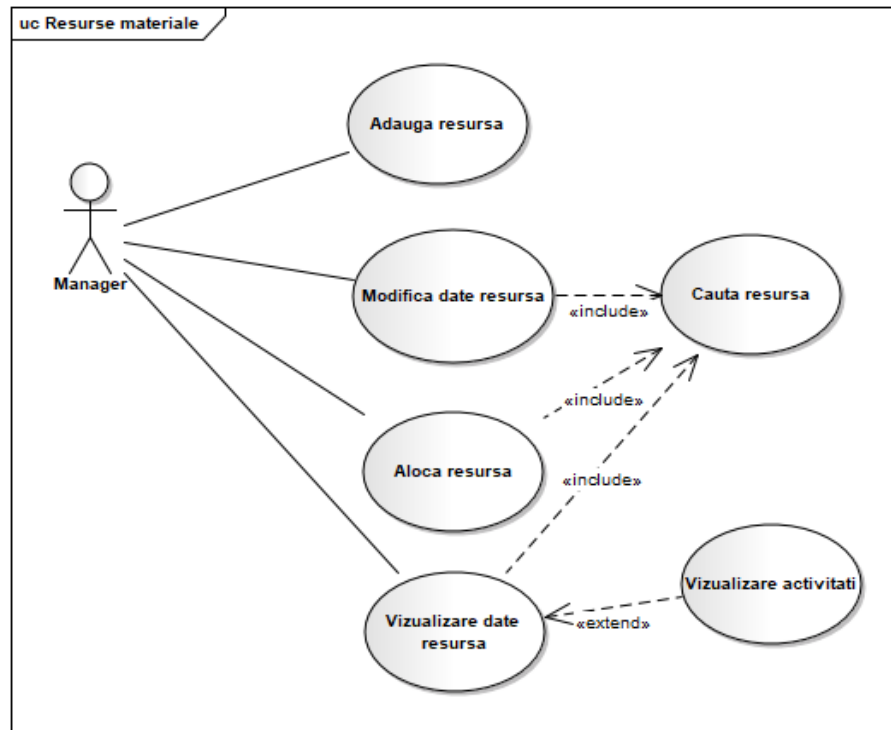


Figura 8 DCU pentru modulul „Resurse materiale”

3.3 Descrierea narativă a cazurilor de utilizare

Descrierea cazurilor de utilizare poate fi realizată sub formă narativă, folosind un șablon sau cu ajutorul limbajului Gherkin. Cazurile de utilizare pot fi detaliate și sub formă grafică, cu ajutorul diagramelor de activități.

În acest paragraf va fi prezentat un șablon pentru descrierea detaliată a cazurilor de utilizare, iar în următorul va fi utilizat limbajul Gherkin.

Numerotare: CU1

Nume CU: Adaugă proiect

Obiectiv: Înregistrarea cu succes în sistem a unui proiect nou.

Autorii: DuNe

Actorii: Managerul de proiect

Prioritate: 1

Precondiții: Utilizatorul este conectat cu un cont care are drepturi de manager de proiect

Post-condiții: 1. Proiectul este înregistrat în baza de date

Scenariul principal: Înregistrarea cu succes a noului proiect, fără crearea de activități

Scenarii alternative: 1. Înregistrarea cu succes a noului proiect pentru un client nou
2. Înregistrarea cu succes a noului proiect cu adăugarea de activități

Scenarii excepție: 1. Abandonarea operațiunii de înregistrare a proiectului

Pașii scenariului principal:

1. Utilizatorul inițiază operațiunea de adăugare a proiectului
2. Utilizatorul introduce codul și numele proiectului, alege tipul proiectului (tipografie sau dezvoltare software)
3. Utilizatorul introduce datele de identificare ale clientului (codul sau numele)
4. Sistemul validează clientul și afișează datele acestuia (codul, numele, adresa, banca etc.)

5. Utilizatorul introduce datele de început și finalizare a proiectului.
6. Sistemul validează datele (data de finalizare să fie ulterioară celei de inițiere). Dacă datele introduse nu sunt corecte, atunci se va relua pasul 5.
7. Utilizatorul alege persoana care va fi managerul de proiect. Valoarea implicită va fi numele utilizatorului conectat.
8. Utilizatorul introduce observațiile privind proiectul.
9. Utilizatorul salvează proiectul.
10. Sistemul verifică dacă datele introduse sunt complete și corecte.
11. Sistemul înregistrează proiectul cu starea “propus”.

Pașii scenariului alternativ 1:

Dacă la pasul 4 din scenariul principal sistemul nu validează clientul (nu există în baza de date), atunci:

1. Sistemul afișează un mesaj de atenționare
2. *[Punct de extensie: CU7 Adăugare client (Client inexistent)]*
3. Cazul de utilizare se continuă cu pasul 5.

Pașii scenariului alternativ 2:

Dacă oricând înainte de pasul 9 utilizatorul alege să adauge activități, atunci:

1. *[Punct de extensie: CU4 Adăugare activitate]*
2. Cazul de utilizare se continuă cu pasul curent.

Pașii scenariului de excepție:

Dacă în oricare moment înainte de pasul 9 al scenariului principal utilizatorul abandonează operațiunea de adăugare, atunci:

1. Sistemul avertizează utilizatorul că datele introduse nu vor fi salvate.
2. Cazul de utilizare se încheie, fără salvarea datelor introduse.

Number: CU5

Nume CU: Anulează proiect

Obiectiv: Înregistrarea în sistem a unui proiect abandonat

Autor: DuNe

Actor: Manager

Nivel de prioritate: 1

Pre-condiții: 1. Utilizatorul este conectat cu un cont care are drepturi de manager de proiect

2. Proiectul este deja înregistrat în sistem și se află în una din stările „propunere” sau „în execuție”

Post-condiții: actualizarea în baza de date a stării proiectului în “abandonat” și a stării tuturor activităților asociate proiectului și aflate în una din stările „propusă”, „repartizată”, „asumată” sau “în execuție” în noua stare “abandonat”.

Scenariul principal: Înregistrarea cu succes în sistem a stării proiectului în “abandonat”.

Scenarii alternative: Nu este cazul

Scenarii de excepție: 1. Anularea unui proiect aflat în starea “abandonat” sau “finalizat”.

2. Renunțarea la operațiunea de anulare a proiectului.

Pașii scenariului principal:

1. Utilizatorul inițiază operațiunea de anulare a proiectului
2. Utilizatorul alege proiectul [incluziune UC Cauta proiect]
3. Sistemul validează proiectul (proiectul trebuie să fie în una dintre stările „propunere” sau „în execuție”).
4. Sistemul afișează detaliile proiectului (numele, datele de început și finalizare ale proiectului, starea proiectului și activitățile corespunzătoare proiectului).

5. Sistemul solicită confirmarea de către utilizator a anulării tuturor activităților asociate proiectului (schimbării stării activităților în “abandonată”).
6. Utilizatorul confirmă anularea activităților.
7. Sistemul verifică datele introduse.
8. Sistemul actualizează starea proiectului în “abandonat”.

Pașii scenariului de excepție 1:

În cazul în care la pasul 7 din scenariul principal proiectul se află în una din stările “finalizat” sau “anulat”, atunci:

1. Sistemul avertizează utilizatorul că proiectul nu poate fi anulat.
2. Cazul de utilizare se încheie, fără anularea proiectului.

Pașii scenariului de excepție 2:

Dacă în oricare moment înainte de pasul 7 al scenariului principal utilizatorul renunță la operațiunea de anulare a proiectului, atunci:

1. Sistemul avertizează utilizatorul că proiectul nu va fi anulat.
2. Cazul de utilizare se încheie, fără anularea proiectului.

3.4 Descrierea cazurilor de utilizare (comportamentului aplicației) folosind limbajul Gherkin

Gherkin este un limbaj care permite descrierea comportamentului unei aplicații software, fără a detalia maniera în care acel comportament este implementat.

Pentru fiecare funcționalitate (caz de utilizare) se va crea un fișier plain text cu extensia *.feature*. Sintaxa generică a acestui fișier va fi cea de mai jos:

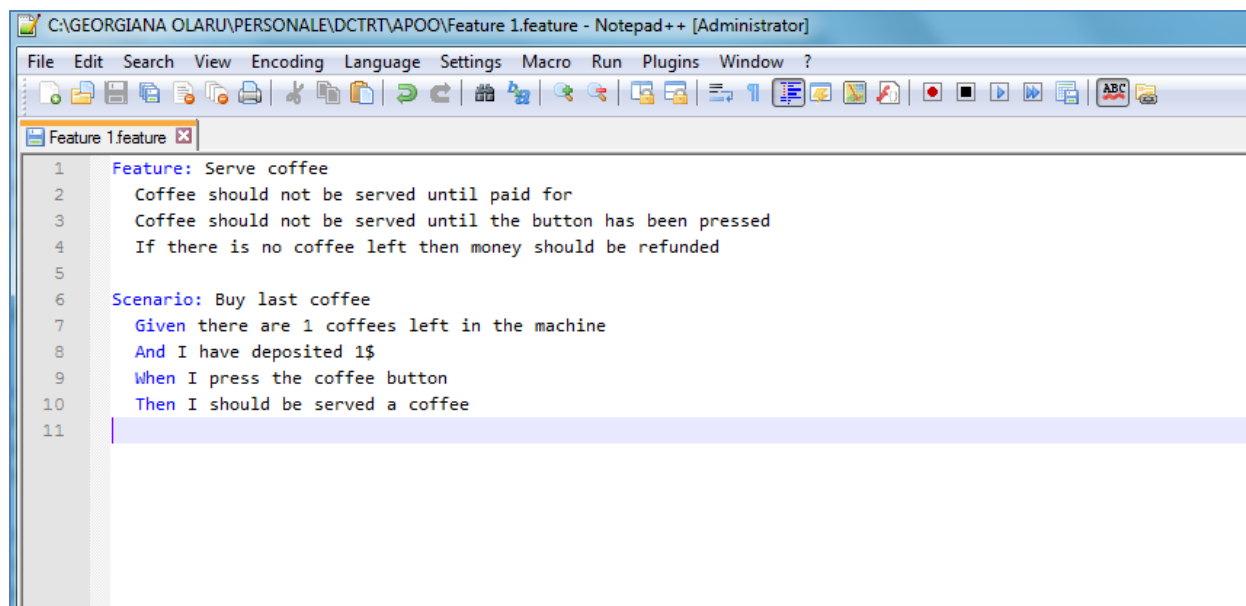
```

1: Feature: Some terse yet descriptive text of what is desired
2:   Textual description of the business value of this feature
3:   Business rules that govern the scope of the feature
4:   Any additional information that will make the feature easier to understand
5:
6:   Scenario: Some determinable business situation
7:     Given some precondition
8:     And some other precondition
9:     When some action by the actor
10:    And some other action
11:    And yet another action
12:    Then some testable outcome is achieved
13:    And something else we can check happens too
14:
15:   Scenario: A different situation
16: Given some precondition
17:    And some other precondition
18:    When some action by the actor
19:    And some other action
20:    And yet another action
21:    Then some testable outcome is achieved
22:   .....
```

Cuvintele marcate în bold sunt cuvinte cheie, elemente ce structurează descrierea cazurilor de utilizare prin limbajul Gherkin.

Se poate folosi Notepad++ pentru a descrie cazurile de utilizare folosind Gherkin. Se recomanda instalarea acestui plugin pentru a evidenția cuvintele cheie Gherkin. <http://productive.me/blog/cucumbergherkin-syntax-highlighting-for-notepad>

Rezultatul ar trebui să arate în genul celui de mai jos :



```
1 Feature: Serve coffee
2   Coffee should not be served until paid for
3   Coffee should not be served until the button has been pressed
4   If there is no coffee left then money should be refunded
5
6 Scenario: Buy last coffee
7   Given there are 1 coffees left in the machine
8   And I have deposited 1$
9   When I press the coffee button
10  Then I should be served a coffee
11
```

Pe lângă scenariile, mai există încă două elemente de structură opționale, pe care va trebui să le descoperiți singuri cu ajutorul documentației existente în linkurile de mai jos. Acestea sunt **Scenario outline** și **Background**.

Resurse generice pentru Gherkin:

1. <https://cucumber.io/docs/reference>
2. <https://github.com/cucumber/cucumber/wiki/Gherkin>
3. <http://docs.behat.org/en/v2.5/guides/1.gherkin.html>
4. <http://morelia.readthedocs.io/en/latest/gherkin.html>

Găsiți exemple de .features ale unor proiecte open source în linkurile următoare:

<https://github.com/cucumber/cucumber-ruby/tree/master/features>

<https://github.com/esambo/TimeFliesBy/tree/master/features>

<https://github.com/diaspora/diaspora/tree/master/features>

În figura 9 este prezentată descrierea cazului de utilizare Adaugă proiect folosind limbajul Gherkin,

```

Scenario: Adaugă proiect
  Given Utilizatorul este conectat cu un cont care are drepturi de manager de proiect
  When Utilizatorul inițiază operațiunea de adăugare a proiectului
  And Utilizatorul introduce codul și numele proiectului, alege tipul proiectului (tipografie sau construcții)
  And Utilizatorul introduce datele de identificare ale clientului (codul sau numele)
  And Utilizatorul introduce datele de început și finalizare a proiectului.
  And Utilizatorul alege persoana care va fi managerul de proiect
  And Utilizatorul introduce observațiile privind proiectul.
  Then Proiectul este salvat în baza de date.

Scenario: Adauga proiect fara selectarea tipului de proiect
  Given Utilizatorul este conectat cu un cont care are drepturi de manager de proiect
  When Utilizatorul inițiază operațiunea de adăugare a proiectului
  And Utilizatorul introduce codul și numele proiectului
  And Utilizatorul introduce datele de identificare ale clientului (codul sau numele)
  And Utilizatorul introduce datele de început și finalizare a proiectului.
  And Utilizatorul alege persoana care va fi managerul de proiect
  And Utilizatorul introduce observațiile privind proiectul.
  Then Utilizatorul este atenționat printr-un mesaj "Alege tipul de proiect!".

Scenario: Adauga proiect fara alegerea responsabilului
  Given Utilizatorul este conectat cu un cont care are drepturi de manager de proiect
  When Utilizatorul inițiază operațiunea de adăugare a proiectului
  And Utilizatorul introduce codul și numele proiectului
  And Utilizatorul introduce datele de identificare ale clientului (codul sau numele)
  And Utilizatorul introduce datele de început și finalizare a proiectului.
  And Utilizatorul alege persoana care va fi managerul de proiect
  And Utilizatorul introduce observațiile privind proiectul.
  Then Proiectul este salvat în baza de date, iar utilizatorul logat este salvat ca responsabil proiect

Scenario: Adauga proiect de pe un cont de utilizator care nu are rolul de manager
  Given Utilizatorul logat nu are profilul de manager
  When Utilizatorul deschide aplicatia
  Then Opțiunea de adăugare a unui proiect nu este disponibilă

```

Figura 9 Descrierea „Adauga proiect” folosind Gherkin

4 Structura aplicației. Diagrame de clase

Structura aplicației este prezentată cu ajutorul diagramelor de clase. Pentru o mai mare claritate, au fost construite 3 diagrame de clase, câte una pentru fiecare strat din arhitectura aplicației. Pentru asigurarea unei perspective de ansamblu, în diagramele de clase au fost incluse și clase din alte straturi și care se regăsesc și în celelalte diagrame, ele putând fi ușor distinse prin faptul că nu au incluse atributele și metodele.

Domain Model (Business Layer)

În fig. 10 este prezentată **diagrama de clase specifică domeniului problemei (domain model)**. Aici au fost incluse următoarele grupe de clase:

- Clasele de tip Entity care mapează obiectele din domeniul problemei (business entities) (clase precum Proiect, Persoana, Client). Aceste clase vor cuprinde doar atributele (starea) și metodele care gestionează valorile atributelor (get-uri, set-uri și metodele pentru gestionarea atributelor de tip colecție). De dragul simplității, metodele get și set nu au fost incluse în model.

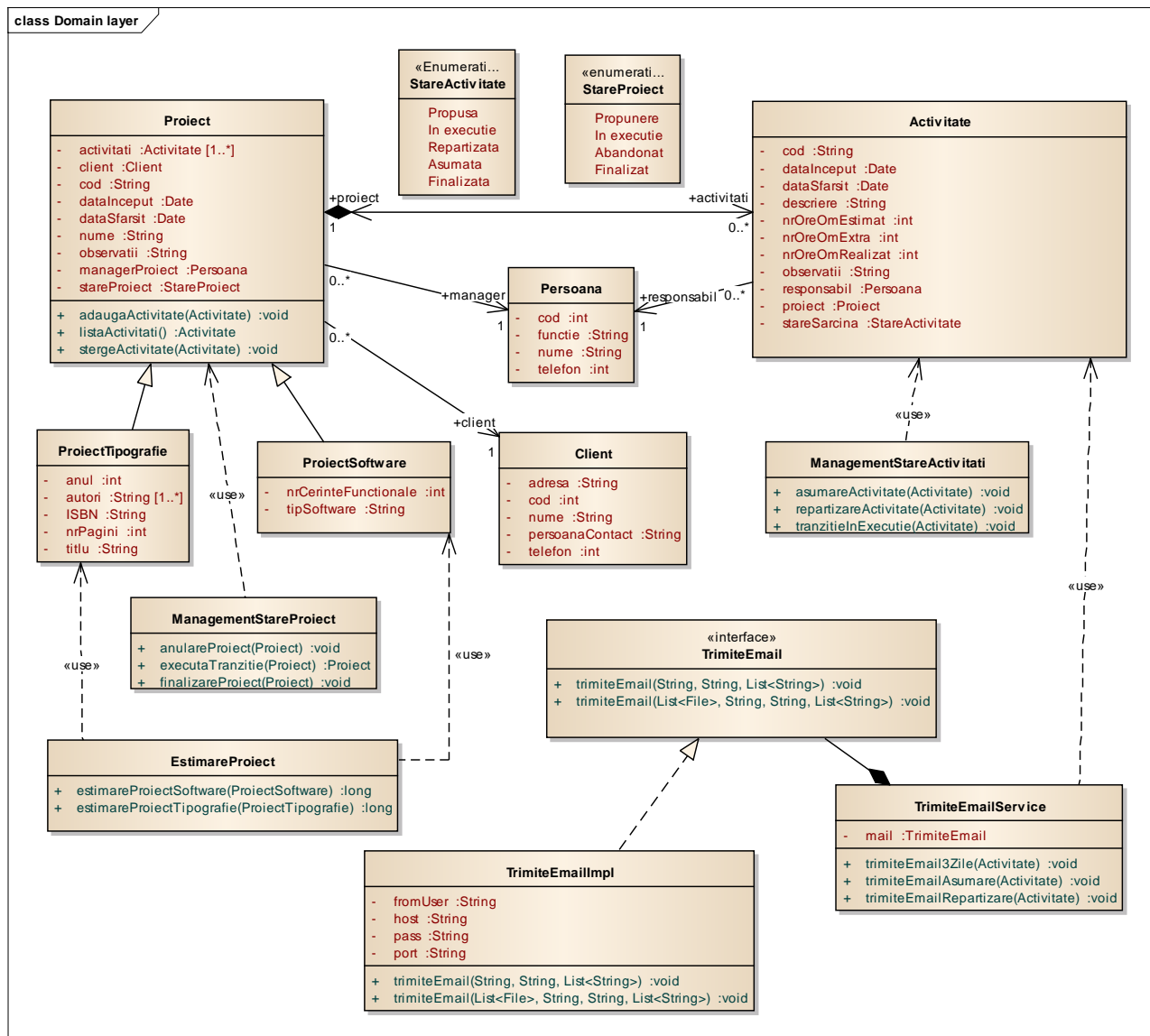


Figura 10 Diagrama de clase pentru modelul domeniului (business layer)

- Clasele care implementează funcționalitatea specifică domeniului problemei (business logic sau service objects în Domain Driven Design) (clase precum `ManagementStareActivitati`, `EstimareProiect`, `TrimateEmailImpl`). Astfel de clase sunt necesare atunci când există unele operațiuni (comportament) specifice domeniului problemei (unele verbe) care nu aparțin strict unor entități (clase de tip Entity), deci nu pot fi incluse în acestea. Așadar, scopul claselor din acest grup este doar de a oferi funcționalități specifice domeniului și vor face referire la clasele de tip Entity care conțin datele necesare în implementarea funcționalităților (vezi relațiile de dependență `<use>` din fig. 10). La modul general, clasele din acest grup vor implementa operațiuni de prelucrarea, transformarea și managementul datelor aplicației, implementarea regulilor afacerii (business rules) sau asigurarea validității datelor din aplicație. Un exemplu mai elocvent ar fi clasa `CalculSalarii`, care implementează funcționalitatea (comportamentul) specific domeniului privind calculul drepturilor salariale, indemnizațiile de concediu etc. Pentru implementarea acestor operațiuni/funcționalități, clasa `CalculSalarii` va face referire la obiecte din clase de tip Entity

(precum Angajat, Pontaj, Deduceri etc.) pentru a obține datele necesare în prelucrări. Pentru mai multe detalii vezi Service objects in Domain Driven Design.

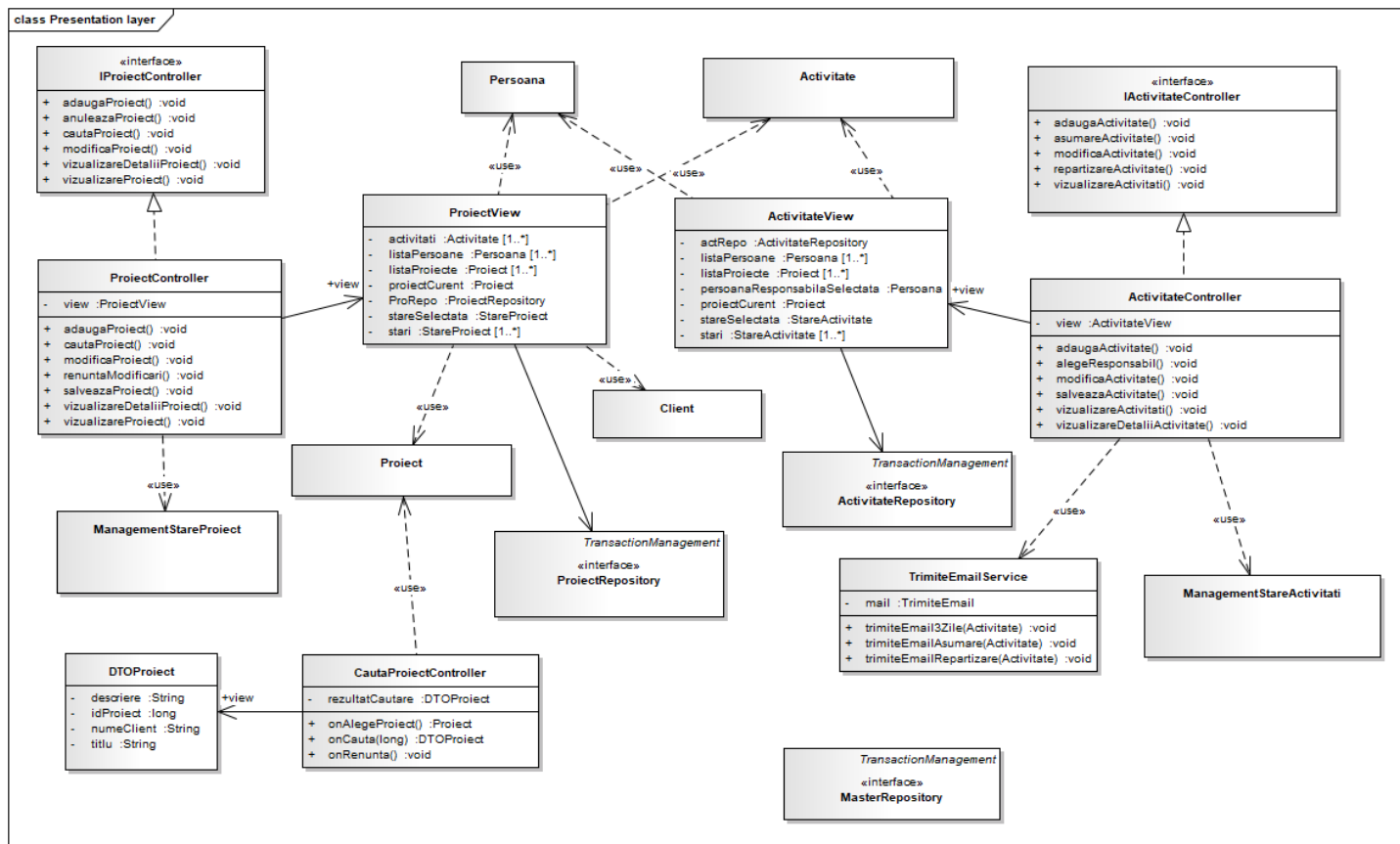
- Clasa ManagementStareActivitati – expune operațiile necesare pentru execuția tranziției între diverse stări prin care trece un obiect Activitate. Conform diagramei stărilor de tranziție din figura 13, tranziția este restricționată prin reguli clar definite. Astfel, sunt necesare operații prin care vor fi controlate valorile atributului Activitate.stare, adică respectarea ordinii descrise în fig. 13, validarea alegerilor făcute de utilizator sau execuția înapoi a fluxului descris în fig. 13. Toate aceste aspecte reprezintă funcționalitate/comportament specific domeniului problemei.
- Clasele de tip *enumeration* – StareActivitate și StareProiect, care conține valorile posibile pentru atributul *stare* din clasele Activitate și Proiect.
- Interfața TrimiteEmail este realizată cu scop de a exemplifica procesul de ascundere a detaliilor legate de tehnologia utilizată, de pilda API-ul JavaMail. S-ar fi putut invoca serviciile API-ului JavaMail chiar din clasele de implementare a Modelului (TrimiteEmailService), însă s-ar fi creat o dependență față de această tehnologie (API) astfel că schimbarea tehnologiei ar fi determinat modificări în stratul Model al aplicației. Soluția oferită izolează aspectele tehnologice (JavaEmail) la nivelul clasei TrimiteEmailImpl iar eventualele modificări ulterioare se vor limita la această clasă, fără a afecta modelul. Este de asemenea, un exemplu de reutilizare prin compunere, în contrast cu reutilizarea prin moștenire care expune și interfețele oferite de super-clase. În modelul reutilizării prin compunere, funcționalitatea reutilizată NU este expusă mai departe clienților noilor funcții.

Presentation Layer

În figura 11 este prezentată diagrama de clase care conține clasele specifice stratului interfeței utilizator (Presentation Layer). În acest strat sunt incluse clasele care implementează datele expuse în paginile Web și funcționalitatea aferentă evenimentelor din interfața grafică utilizator, adică orchestrarea interacțiunilor utilizatorului cu aplicația așa cum au fost descrise prin cazurile de utilizare. Pentru acest strat a fost utilizat șablonul arhitectural Model-View-Controller (MVC).

În această diagramă se regăsesc următoarele grupe de clase:

- Clase cu rol de View și Controller (vezi arhitectura de referință) – s-a creat câte o astfel de clasă pentru fiecare interfață utilizator (pagină HTML). Pentru detalii vezi modelul de proiect de la disciplina PSI și alte informații privind MVC.
- Clase de tip Interface pentru implementarea funcționalităților aplicației. S-a creat câte o clasă pentru fiecare diagramă a cazurilor de utilizare (aici am inclus doar primele 2), iar fiecare clasă conține câte o metodă pentru fiecare caz de utilizare concret (primary). Aceste interfețe vor fi implementate de clasele cu rol de controller. Acest stil de lucru ar facilita traducerea cazurilor de utilizare în codul aplicației, garantând astfel implementarea tuturor funcționalităților/cazurilor de utilizare.



Data Layer

Acest strat furnizează funcționalitatea necesară celorlalte părți/componente ale aplicației pentru accesarea datelor din stratul de persistență (baza de date sau altă metode de persistență). În acest sens, am optat pentru șablonul Repository (se putea utiliza DAO - Data Access Objects), iar diagrama de clase este prezentată în figura 12.

Utilizarea unui sablon precum Repository are drept scop *ascunderea implementarii privind accesul la stratul de persistență* (utilizarea unui framework ORM precum JPA sau, în aplicația noastră, biblioteca ODM Mongoose pentru MongoDB și Node.js) și *furnizarea funcționalității necesare pentru accesarea datelor*. Pentru operații CRUD, toate celelalte componente ale aplicației vor folosi în mod exclusiv API-ul specific aplicației, pus la dispoziție sub forma de clase/componente Repository.

În aplicația noastră am creat trei astfel de clase/componente: `ProiectRepository`, `ActivitateRepository` și `MasterRepository` (vezi figura 12). Un aspect important pe care trebuie să-l observăm în diagrama din fig. 12 este legat de ascunderea implementării accesului la date (așa cum spuneam mai sus, unul din obiectivele principale). Clasa `AbstractRepository` încapsulează într-o manieră înalt abstractizată operațiile fundamentale CRUD folosind funcțiile Mongoose. Ascunderea implementării presupune ca un client să nu poată utiliza în mod direct această clasă. Acest deziderat se implementează prin declararea clasei ca fiind de tip *abstract*. Ulterior, implementarea componentelor `ProiectRepositoryImpl`, `ActivitateRepositoryImpl` și `MasterRepositoryImpl` extind și utilizează această clasă oferind mai departe clienților metode CRUD specifice aplicației, bine definite și non-ambigue, prin intermediul celor trei interfețe.

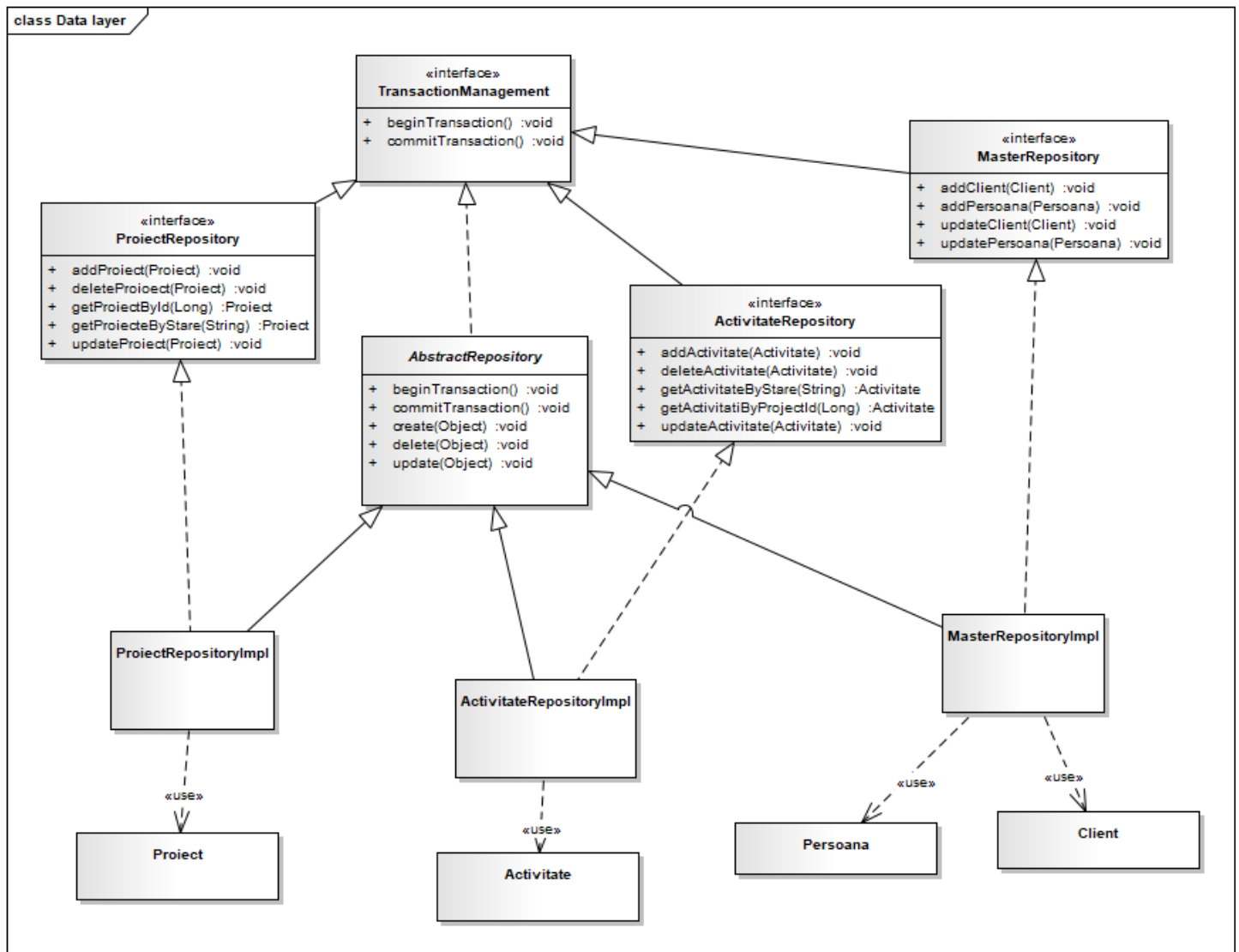


Figura 12 Diagrama de clase pentru Data Layer

In dezvoltarea componentelor noastre este important de inteles un lucru: desi clasele de implementare, **ProjectRepositoryImpl**, **ActivitateRepositoryImpl** si **MasterRepositoryImpl**, extind clasa **AbstractRepository**, iar metodele acestora nu sunt nici macar declarate „*protected*”, totusi o aplicatie client nu va putea utiliza aceste metode pentru simplul fapt ca aplicatiile-client lucreaza, la nivel de declaratii de variabile, numai cu interfetele, nu si cu implementarea. Adica, un client va declara o variabila de genul:

```
DocumentRepository repo= new DocumentRepositoryDefault()
```

Mai departe, prin variabila *repo*, vor fi accesibile doar metodele declarate de interfata **ProjectRepository**, indiferent cate alte metode pot fi disponibile in clasa **ProjectRepositoryImpl**. Exemplul nostru este mult simplificat, pentru claritate. In practica se utilizeaza sablonul Factory pentru a obtine o instanta corespunzatoare unei interfete. Totusi, NU trebuie sa intelegem aici ca trebuie neaparat sa existe vreun mecanism de securitate prin care clientul sa nu poata declara variabila ca fiind de tipul clasei de implementare in locul interfetei. Trebuie doar sa retinem ca paradigma „***program to interfaces, not to implementations***” (adica utilizarea exclusiv a interfetelor – numite si API - in declararea variabilelor) este o metoda formala („formal method” -

cea mai simplă din câte există) **respectată cu strictete** în practică din simplul motiv că asigură supraviețuirea codului-client atunci când se utilizează o nouă versiune a aceluși API (de cele mai multe ori, necesară datorită îmbunătățirilor aduse, pe măsura ce respectiva evoluează). Imaginați-vă că o aplicație pe care o scrieți acum, utilizând versiunea Hibernate inclusă în proiect, va trebui să supraviețuiască atunci când, peste câțiva ani, veți include o nouă versiune Hibernate care oferă o funcționalitate suplimentară, de care aveți nevoie. În aceste situații este esențial ca: 1) furnizorul componentei să respecte „contractele” vechi (interfețele inițiale) odată cu adăugarea de funcționalitate nouă 2) clientul să nu utilizeze decât interfețe publice, care nu trebuie să se schimbe, ale componentelor și să nu încerce utilizarea unor funcționalități interne ale componentei expunându-se, astfel, riscului de incompatibilitate cu versiunile următoare.

5. Diagrama de stare

Diagrama de stări din fig. 13 reflectă stările posibile ale unui obiect Activitate.

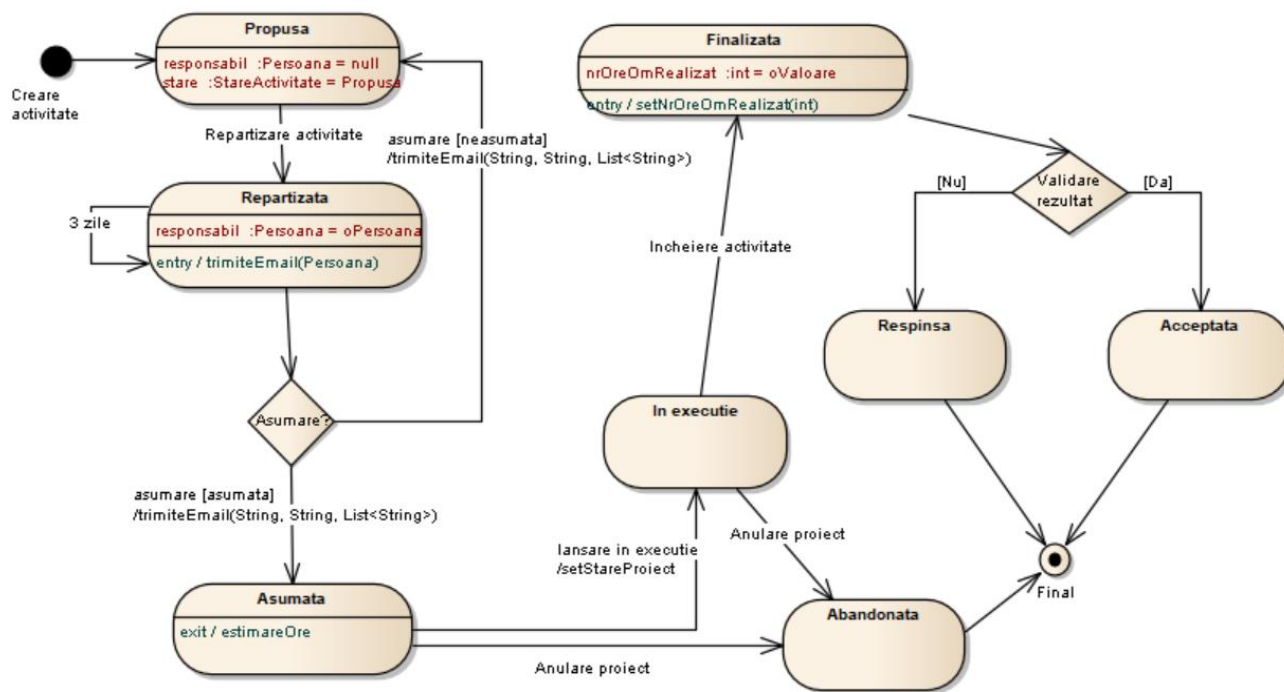


Figura 13 Diagrama de stări pentru obiectele de tip Sarcină

6. Diagrame de secvențe – dinamica modelului (corespunde diagramelor de activități)

Implementarea unei funcționalități, corespunzătoare unui caz de utilizare, este descrisă cu diagrama de secvențe. Se va crea câte o diagramă pentru fiecare caz de utilizare sau chiar scenariu de lucru.

În acest paragraf sunt descrise aspectele dinamice ale sistemului pentru implementarea funcționalității specifice cazului de utilizare „Adăuga activitate”. A fost construită diagrama de secvențe din figura 14.

Adaugă activitate

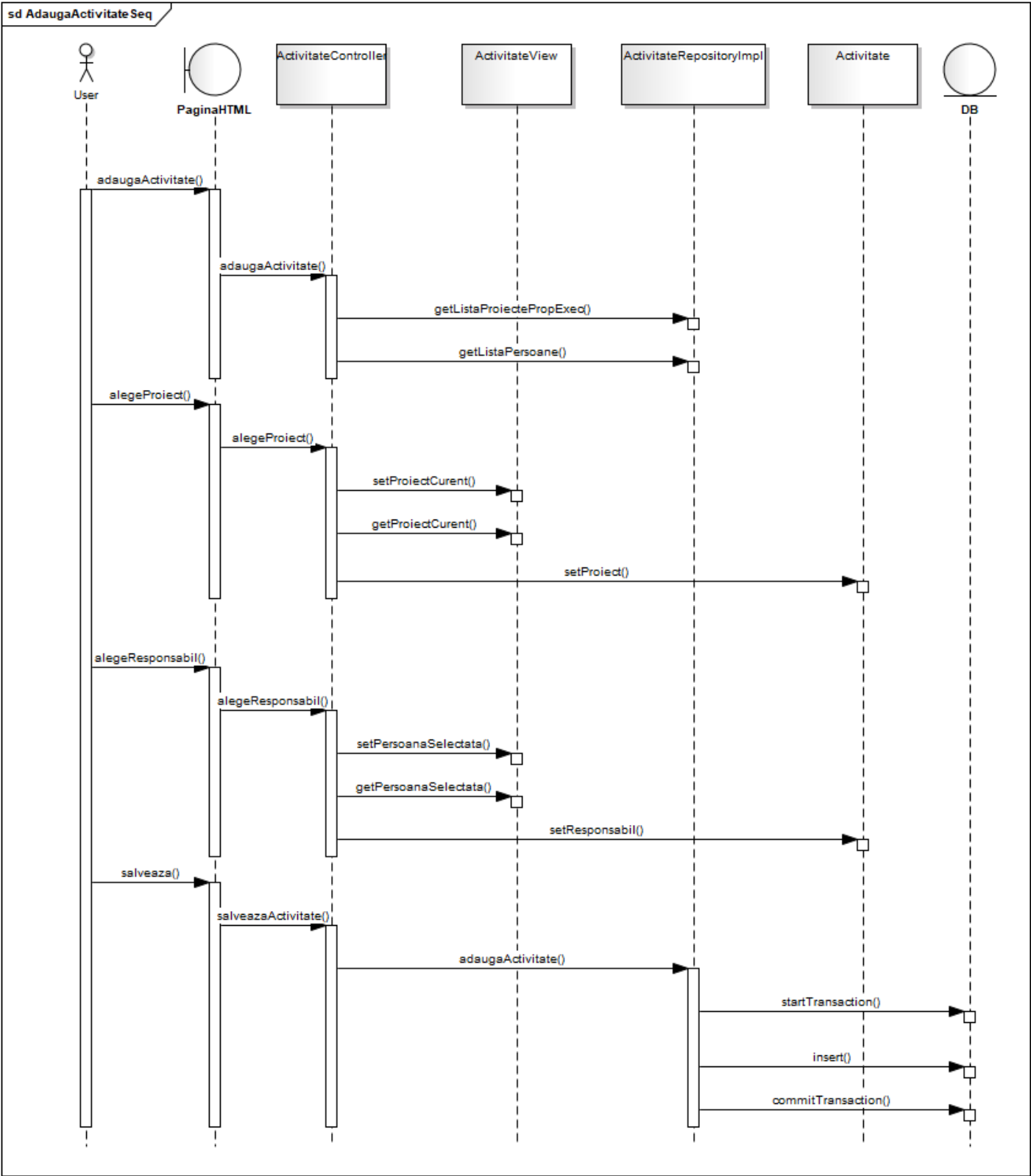


Figura 15 Diagrama de secvențe pentru cazul de utilizare Adauga activitate