Ministerul Educației, Culturii și Cercetării
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică
Departamentul Ingineria Software și Automatică

Lucrare de curs

Disciplina: Analiza și Modelarea Obiect Orientată

Tema: „Analiza și modelarea unei aplicații care va gestiona necesitățile unui club de sport”
„Analysis and modeling of an application that will manage the needs of a sports club”

A efectuat st.gr. TI-207 Bunescu Gabriel.
A verificat lect.univ. Melnic Radu.
asis.univ. Lisnic Inga.

Chișinău 2022

Cuprins:

Introducere

Limbajul Unificat de Modelare UML (Unified Modeling Language) a adus împreună trei metodologii orientate obiect introduse prin munca comună a autorilor Grady Booch, Ivan Jacobson şi Jim Rumbaugh.

Limbajul unificat de modelare a fost definit în intenţia de a introduce o standardizare întipologia, semantică şi reprezentarea rezultatelor (artifactelor) produse de analiza şi proiectarea orientate obiect.

UML este rezultatul unui efort de unificare în care au contribuit elemente dezvoltate de numeroase cercetări şi metode. UML este definit drept o " colecţie a celor mai bune practice "aplicate în modelarea sistemelor informatice de mari dimensiuni şi complexitate.

UML a fost definit pornind de la rolul esenţial pe care-l joacă modelarea în conceperea şi realizarea de sisteme software. UML-lui îi este specific un set de diagrame şi notaţii convenţionale pentru modelarea sistemelor şi care poate fi folosit pentru modelare a diferitelor tipuri de sisteme, pornind de la software până la procese organizaţionale.

O definiţie a limbajului unificat de modelare (UML) ar fi:

„*Limbaj grafic folosit pentru vizualizarea, specificarea, construcţia şi documentarea artifactel or unui sistem software intensiv. UML oferă o modalitate standard de scriere a copiilor sistemului, incluzând atât elemente conceptuale cum ar fi procesele de afaceri şi funcţiile unui sistem, cât şi elemente concrete constând în codurile unor limbaje de programare, schema bazei de date şi componente software reutilizabile*".

Modelarea în acest limbaj se realizeaza prin combinarea notaţiilor UML în cadrul elementelor principale ale acestora denumite diagrame. Având în vedere că UML nu este o metodologie, el nu cere nici un fel de muncă formală. El oferă însă mai multe tipuri de diagrame care folosite cu o anumită metodologie, măresc uşurinţa de înţelegere a aplicaţiei în dezvoltare.[7]

Web site-ul de sport are ca scop prezentarea, motivarea și atragerea a persoanelor de a se ocupa în săli de sport. Vizualizarea imaginilor printre care poate chiar să ajungă și imaginea ta! Comentariilor pe care le lasă antrenori profesioniști aduc ca regulă la motivare de a încerca ceva nou pentru fiecare, ei se staruie să se înpartă cu cît mai multă informație despre cum trebuie să te antrenezi corect, care diete ar fi mai potrivite ce grafic de antrenament să respecți și care sălii de sport sunt dotate cu tot necesar pentru a ajuta oamenii să se aducă la o forma a corpului cît mai potrivită pentru fiecare în parte.

1 Analiza și modelarea orientată pe obiect

Analiza și proiectarea orientate pe obiecte (OOAD) este o abordare tehnică pentru analizarea și proiectarea unei aplicații, a unui sistem sau a unei afaceri prin aplicarea programării orientate pe obiecte, precum și prin utilizarea modelării vizuale pe tot parcursul procesului de dezvoltare a software-ului pentru a ghida comunicarea cu părțile interesate și calitatea produsului.

OOAD în ingineria software modernă se desfășoară de obicei într-un mod iterativ și incremental. Rezultatele activităților OOAD sunt modele de analiză (pentru OOA) și, respectiv, modele de proiectare (pentru OOD). Intenția este ca acestea să fie perfecționate și evoluate continuu, conduse de factori cheie precum riscurile și valoarea afacerii.

Prezentare generală

Paradigma orientată pe obiecte accentuează modularitatea și reutilizarea. Scopul unei abordări orientate pe obiect este de a satisface „principiul deschis-închis”. Un modul este deschis dacă acceptă extensia sau dacă modulul oferă modalități standardizate de a adăuga comportamente noi sau de a descrie stări noi. În paradigma orientată pe obiect, acest lucru este adesea realizat prin crearea unei noi subclase a unei clase existente.

Un modul este închis dacă are o interfață stabilă bine definită pe care trebuie să o utilizeze toate celelalte module și care limitează interacțiunea și erorile potențiale care pot fi introduse într-un modul prin modificările efectuate în altul. În paradigma orientată pe obiecte, acest lucru se realizează prin definirea unor metode care invocă servicii pe obiecte. Metodele pot fi publice sau private, adică anumite comportamente care sunt unice pentru obiect nu sunt expuse altor obiecte. Acest lucru reduce o sursă de multe erori comune în programarea computerelor.

Ciclul de viață al software-ului este de obicei împărțit în etape, mergând de la descrieri abstracte ale problemei la proiecte, apoi la codificare și testare și, în final, la implementare. Primele etape ale acestui proces sunt analiza și proiectarea. Distincția dintre analiză și design este adesea descrisă ca „ce vs. cum”. În analiză, dezvoltatorii lucrează cu utilizatori și experți în domeniu pentru a defini ce ar trebui să facă sistemul. Detaliile de implementare ar trebui să fie în mare parte sau total (în funcție de metoda particulară) ignorate în această fază. Scopul fazei de analiză este de a crea un model funcțional al sistemului, indiferent de constrângeri, cum ar fi tehnologia adecvată.

În analiza orientată pe obiecte, acest lucru se realizează de obicei prin intermediul cazurilor de utilizare și al definițiilor abstracte ale celor mai importante obiecte. Faza de proiectare ulterioară rafinează modelul de analiză și face tehnologia necesară și alte opțiuni de implementare. În proiectarea orientată pe obiecte, accentul se pune pe descrierea diferitelor obiecte, a datelor, a comportamentului și a interacțiunilor acestora. Modelul de proiectare ar trebui să aibă toate detaliile necesare, astfel încât programatorii să poată implementa designul în cod.

Analiza orientata pe obiecte

Scopul oricărei activități de analiză în ciclul de viață al software-ului este de a crea un model al cerințelor funcționale ale sistemului care este independent de constrângerile de implementare.

Principala diferență dintre analiza orientată pe obiect și alte forme de analiză este că prin abordarea orientată pe obiect organizăm cerințele în jurul obiectelor, care integrează atât comportamente (procese), cât și stări (date) modelate după obiecte din lumea reală cu care sistemul interacționează. În alte metodologii de analiză sau tradiționale, cele două aspecte: procesele și datele sunt luate în considerare separat. De exemplu, datele pot fi modelate prin diagrame ER, iar comportamentele prin diagrame de flux sau diagrame de structură.

Modelele comune utilizate în OOA sunt cazurile de utilizare și modelele obiect. Cazurile de utilizare descriu scenarii pentru funcțiile de domeniu standard pe care sistemul trebuie să le realizeze. Modelele de obiecte descriu numele, relațiile de clasă (de exemplu, Cercul este o subclasă a Formei), operațiile și proprietățile obiectelor principale. Modelele sau prototipurile interfeței cu utilizatorul pot fi, de asemenea, create pentru a ajuta la înțelegere.

Modelare orientată pe obiecte

Modelarea orientată pe obiecte (AOOM) este o abordare comună pentru modelarea aplicațiilor, sistemelor și domeniilor de afaceri prin utilizarea paradigmei orientate pe obiecte pe parcursul întregului ciclu de viață al dezvoltării. OOM este o tehnică principală foarte utilizată atât de activitățile OOD, cât și de OOA în ingineria software modernă.

Modelarea orientată pe obiecte se împarte de obicei în două aspecte ale muncii: modelarea comportamentelor dinamice, cum ar fi procesele de afaceri și cazurile de utilizare, și modelarea structurilor statice, cum ar fi clasele și componentele. OOA și OOD sunt cele două niveluri abstracte distincte (adică nivelul de analiză și nivelul de proiectare) în timpul OOM. Unified Modeling Language (UML) și SysML sunt cele două limbaje standard internaționale populare utilizate pentru modelarea orientată pe obiecte.

Beneficiile AOOM sunt:

Utilizatorii au de obicei dificultăți în a înțelege bine documentele cuprinzătoare și codurile limbajului de programare. Diagramele modelului vizual pot fi mai ușor de înțeles și pot permite utilizatorilor și părților interesate să ofere dezvoltatorilor feedback cu privire la cerințele și structura corespunzătoare a sistemului. Un obiectiv cheie al abordării orientate pe obiect este de a reduce „decalajul semantic” dintre sistem și lumea reală și ca sistemul să fie construit folosind o terminologie care este aproape aceeași pe care o folosesc părțile interesate în afacerile de zi cu zi. Modelarea orientată pe obiecte este un instrument esențial pentru a facilita acest lucru.

Abstracție utilă și stabilă

Modelarea ajută la codificare. Un obiectiv al majorității metodologiilor software moderne este de a aborda mai întâi întrebările „ce” și apoi de a aborda întrebările „cum”, adică mai întâi să determinăm

funcționalitatea pe care sistemul trebuie să o furnizeze fără a lua în considerare constrângerile de implementare și apoi să ia în considerare modul de a face soluții specifice acestor abstracte. cerințele și rafinați-le în proiecte și coduri detaliate prin constrângeri precum tehnologia și bugetul. Modelarea orientată pe obiecte permite acest lucru prin producerea de descrieri abstracte și accesibile atât ale cerințelor de sistem, cât și ale proiectelor, adică modele care își definesc structurile și comportamentele esențiale, cum ar fi procesele și obiectele, care sunt active de dezvoltare importante și valoroase, cu niveluri de abstractizare mai ridicate deasupra codului sursă concret și complex.[1]

1.1 Analiza și conceperea sistemelor orientate pe obiect

Analiza orientata pe obiecte

Scopul oricărei activități de analiză în ciclul de viață al software-ului este de a crea un model al cerințelor funcționale ale sistemului care este independent de constrângerile de implementare.

Principala diferență dintre analiza orientată pe obiect și alte forme de analiză este că prin abordarea orientată pe obiect organizăm cerințele în jurul obiectelor, care integrează atât comportamente (procese), cât și stări (date) modelate după obiecte din lumea reală cu care sistemul interacționează. În alte metodologii de analiză sau tradiționale, cele două aspecte: procesele și datele sunt luate în considerare separat. De exemplu, datele pot fi modelate prin diagrame ER, iar comportamentele prin diagrame de flux sau diagrame de structură.

Modelele comune utilizate în OOA sunt cazurile de utilizare și modelele obiect. Cazurile de utilizare descriu scenarii pentru funcțiile de domeniu standard pe care sistemul trebuie să le realizeze. Modelele de obiecte descriu numele, relațiile de clasă (de exemplu, Cercul este o subclasă a Formei), operațiile și proprietățile obiectelor principale. Modelele sau prototipurile interfeței cu utilizatorul pot fi, de asemenea, create pentru a ajuta la înțelegere.

Analiza și proiectarea orientate pe obiecte (O-O) este o abordare care are scopul de a facilita dezvoltarea sistemelor care trebuie să se schimbe rapid ca răspuns la mediile de afaceri dinamice.

Se crede că tehnicile orientate pe obiecte funcționează bine în situațiile în care sistemele informaționale complicate sunt supuse întreținerii, adaptării și reproiectării continue. Abordările orientate pe obiecte folosesc standardul industrial pentru modelarea sistemelor orientate pe obiecte, numit limbaj unificat de modelare (UML), pentru a descompune un sistem într-un model de caz de utilizare.

Programarea orientată pe obiecte diferă de programarea procedurală tradițională prin examinarea obiectelor care fac parte dintr-un sistem. Fiecare obiect este o reprezentare computerizată a unui lucru sau eveniment real. Obiectele pot fi clienți, articole, comenzi și așa mai departe. Obiectele sunt reprezentate și grupate în clase care sunt optime pentru reutilizare și întreținere. O clasă definește setul de atribute și comportamente comune găsite în fiecare obiect din clasă.

Fazele din UML sunt similare cu cele din SDLC. Deoarece aceste două metode împărtășesc modelarea rigidă și exigentă, ele se întâmplă într-un ritm mai lent și mai deliberat decât fazele modelării agile. Analistul trece prin faze de problemă și de identificare, o fază de analiză și o fază de proiectare, așa cum se arată în figura de mai jos.

- Definiți modelul cazului de utilizare.

În această fază analistul identifică actorii și evenimentele majore inițiate de actori. Adesea, analistul va începe prin a desena o diagramă cu figuri care reprezintă actorii și săgeți care arată modul în care actorii se relaționează.

- În timpul fazei de analiză a sistemelor, începeți desenarea diagramelor UML.

În a doua fază, analistul va realiza Diagrame de activitate, care ilustrează toate activitățile majore din cazul de utilizare. În plus, analistul va crea una sau mai multe diagrame de secvență pentru fiecare caz de utilizare, care arată secvența activităților și calendarul acestora. Aceasta este o oportunitate de a reveni și de a revizui cazurile de utilizare, de a le regândi și de a le modifica dacă este necesar.

- Continuând în faza de analiză, elaborați diagrame de clasă.

Substantivele din cazurile de utilizare sunt obiecte care pot fi grupate în clase. De exemplu, fiecare automobil este un obiect care împărtășește caracteristici cu alte automobile. Împreună formează o clasă.

- Încă în faza de analiză, desenați diagrame de diagramă de stat.

Diagramele de clasă sunt folosite pentru a desena diagrame de diagrame de stări, care ajută la înțelegerea proceselor complexe care nu pot fi derivate pe deplin de diagramele de secvență. Diagramele statechart sunt extrem de utile în modificarea diagramelor de clasă, astfel încât procesul iterativ de modelare UML continuă.

- Începeți proiectarea sistemelor prin modificarea diagramelor UML. Apoi completați specificațiile.

Proiectarea sistemelor înseamnă modificarea sistemului existent și asta presupune modificarea diagramelor desenate în faza anterioară. Aceste diagrame pot fi folosite pentru a deriva clase, atributele lor și metode (metodele sunt pur și simplu operații). Analistul va trebui să scrie specificațiile de clasă pentru fiecare clasă, inclusiv atributele, metodele și descrierile acestora. Ei vor dezvolta, de asemenea, specificațiile metodelor care detaliază cerințele de intrare și de ieșire pentru metodă, împreună cu o descriere detaliată a procesării interne a metodei.

- Dezvoltați și documentați sistemul.

UML este, desigur, un limbaj de modelare. Un analist poate crea modele minunate, dar dacă sistemul nu este dezvoltat, nu prea are rost să construim modele. Documentarea este esențială. Cu cât informațiile furnizate echipei de dezvoltare prin documentație și diagrame UML sunt mai complete, cu atât dezvoltarea este mai rapidă și sistemul final de producție este mai solid.[2]

1.2 Limbaj UML

Unified Modeling Language (UML) joacă un rol important în dezvoltarea de software, dar și în pentru sistemele non-software din numeroase domenii, deoarece este o modalitate prin care puteți vizualiza comportamentul și structura unui sistem sau proces. UML ajută la prezentarea posibilelor erori din structurile aplicațiilor, comportamentele sistemelor și alte procese de afaceri.

UML a apărut pentru prima dată în anii 90, datorită lui Grady Booch, Ivar Jacobson și James Rumbaugh, trei ingineri de software care au dorit să dezvolte o modalitate mai puțin haotică de a reprezenta dezvoltarea de software din ce în ce mai complexă, separând, în același timp, metodologia de proces. În prezent, UML este în continuare notația preferată atât pentru dezvoltatori, cât și pentru manageri de proiect, deținători de companii, antreprenori din domeniul tehnologiei și specialiști din diferite domenii.

UML sau Unified Modeling Language este un limbaj de descriere grafică pentru modelarea obiectelor în dezvoltarea de software. Dar utilizarea UML nu se limitează la IT, un alt domeniu mare de aplicare practică a UML este modelarea proceselor de afaceri, ingineria sistemelor și maparea structurilor organizaționale. UML permite dezvoltatorilor de software să ajungă la un acord cu privire la notația grafică pentru a reprezenta concepte generaleși să se concentreze pe proiectare și dezvoltare.

Beneficiile UML:

1. Simplifică complexitățile
2. Menține deschise canalele de comunicare
3. Automatizează producția de software și procesele
4. Contribuie la rezolvarea problemelor arhitecturale persistente
5. Sporește calitatea muncii
6. Reduce costurile și timpul de punere în vînzare

Modele baze de date

UML a început să câștige în popularitate și pentru bazele de date de modelare. Aceste modele sunt instrumente vizuale excelente pentru brainstorming, diagrame cu formă liberă și colaborare.

UML nu are specificații pentru modelarea datelor, dar poate fi un instrument util pentru diagrame, în special având în vedere că datele din bazele de date pot fi utilizate în programarea orientată pe obiecte.

Tipuri de modele de baze de date pe care le puteți crea:

1. **Mod de bază de date ierarhică.** Datele acestui model sunt organizate într-o structură arborescentă. Structura este compusă din câteva grupuri denumite segmente. Aceasta utilizează o relație de tip unu la mai mulți. Accesul la date este, de asemenea, previzibil.
2. **Model de rețea**. Acest model ia forma unui grafic, în care tipurile de relații sunt arcuri, iar tipurile de obiecte sunt noduri. Spre deosebire de alte modele de baze de date, schema modelului de rețea nu este limitată la o rețea sau ierarhie.

3. **Model de bază de date orientat pe obiecte**. Acest model utilizează o colecție de obiecte sau elemente de software reutilizabile, cu caracteristici și metode asociate. De exemplu, o bază de date multimedia ar putea avea imagini care nu pot fi stocate într-o bază de date relațională. Sau, o bază de date hipertext permite conectarea cu alte obiecte.
4. **Model relațional**. Aici, datele sunt structurate folosind relații sau structuri matematice de tip grilă, care au coloane și rânduri. Practic, este un tabel.
5. **Modelul obiect relațional**. După cum sugerează numele, acest model este o combinație dintre cele două menționate anterior. Acesta sprijină obiecte, clase, moșteniri și alte elemente orientate pe obiecte, dar acceptă și tipuri de date, structuri tabulare și altele, precum într-un model de date relațional.
6. **Modul entitate-relație**. Acesta este compus din tipuri de entități (persoane, locuri sau lucruri). Prezintă relațiile care există între acestea. Prin definirea entităților, a atributelor acestora și prin prezentarea relațiilor dintre acestea, o diagramă ER ilustrează structura logică și bazele de date.
7. **Model document**. Este conceput pentru stocarea și gestionarea documentelor sau datelor semi-structurate și nu pentru datele atomice. Are o structură arborescentă în care fiecare nod este un obiect ce reprezintă o parte din document.
8. **Model entitate-atribut-valoare**. În modelele EAV sau cu schemă deschisă, datele sunt înregistrate pe trei coloane:
    - Entitatea (ce este descris)
    - Atributul sau parametrul (de ex. nume, descriere, tip de date)
    - Valoarea atributului.
9. **Schemă în stea**. Aceasta este cea mai simplă versiune a unui model dimensional, în care datele sunt aranjate în dimensiuni și fapte. Este utilizată pentru informații de afaceri și stocarea de date deoarece este potrivită pentru interogarea de seturi de big data.

Simplificarea cu software

Indiferent dacă creați modele de baze de date sau diagrame UML, utilizarea unui instrument software simplifică și îmbunătățește procesul. Asigurați-vă că alegeți unul care vă permite:

- Să creați diagrame profesioniste, cu șabloane predefinite și mii de forme într-un ecosistem de conținut care îndeplinește standardele din domeniu, precum UML 2.5, dar și BPMN 2.0 și IEEE.
- Să dați viață diagramelor cu suprapuneri de date, pictograme, culori și grafică, pentru a face datele mai ușor de înțeles, inclusiv prin vizualizarea într-un pas a datelor din Excel.
- Colaborați cu alții folosind lucrul în comun, comentariile și adnotările.
- Comunicați o versiune a adevărului și accesați diagrame de aproape oriunde într-un browser sau în aplicații pentru dispozitive. [3]

Dezavantaje

În ciuda faptului că construirea diagramelor UML are multe dintre avantajele sale, destul de des sunt criticate pentru următoarele neajunsuri:

1. Redundanţă. În marea majoritate a cazurilor, criticii spun că UML-ul este prea mare și complex și de multe ori acest lucru este nejustificat. Include o mulțime de construcții și diagrame redundante sau practic inutile, iar de cele mai multe ori astfel de critici se îndreaptă spre a doua versiune, și nu pe prima, pentru că în revizuirile mai noi sunt mai multe compromisuri „elaborate de comitet”.
2. Diverse inexactități semantice. Deoarece UML este definit printr-o combinație între el însuși, engleză și OCL, îi lipsește rigiditatea inerentă limbilor care sunt precis definite de tehnica descrierii formale. În anumite situații, sintaxa abstractă a OCL, UML și engleză încep să se contrazică, în timp ce în alte cazuri sunt incomplete. Descrierea inexactă a limbajului în sine afectează atât utilizatorii, cât și furnizorii de instrumente, ceea ce duce în cele din urmă la incompatibilitatea instrumentelor din cauza modului unic de interpretare a diferitelor specificații.
3. Probleme în procesul de implementare și studiu. Toate problemele de mai sus creează anumite dificultăți în procesul de introducere și învățare a UML, iar acest lucru este valabil mai ales atunci când conducerea îi obligă pe ingineri să-l folosească forțat, în timp ce aceștia nu au abilități anterioare.
4. Codul oglindește codul. O altă părere este că nu modelele frumoase și atractive sunt importante, ci sistemele de lucru în sine, adică codul este proiectul. În conformitate cu această viziune, este nevoie de a dezvolta o modalitate mai eficientă de a scrie software. UML este apreciat pentru abordările care compilează modele pentru a regenera codul executabil sau sursă. Dar, în realitate, acest lucru poate să nu fie suficient, deoarece limbajului îi lipsește proprietățile de completitudine Turing și fiecare cod generat va fi în cele din urmă limitat de ceea ce poate presupune sau defini un instrument de interpret UML.
5. Nepotrivire de încărcare. Acest termen provine din teoria analizei sistemelor pentru a determina incapacitatea intrării unui anumit sistem de a percepe rezultatul altuia. Ca în oricare sisteme standard notație, UML poate reprezenta unele sisteme într-un mod mai eficient și mai concis decât altele. Astfel, dezvoltatorul este mai înclinat către acele soluții care sunt mai confortabile pentru a țese toate punctele forte ale UML-ului, precum și ale altor limbaje de programare. Această problemă este mai evident în cazul în care limbajul de dezvoltare nu respectă principiile de bază ale doctrinei ortodoxe orientate pe obiecte, adică nu încearcă să funcționeze în conformitate cu principiile OOP.
6. Încearcă să fie versatil. UML este un limbaj de modelare de uz general care caută să fie compatibil cu orice limbaj de procesare disponibil în prezent. În contextul unui anumit proiect, pentru ca echipa de proiectare să poată atinge scopul final, este necesar să se selecteze capacitățile aplicabile

ale limbajului respectiv. În plus, posibilele modalități de limitare a domeniului de aplicare a UML la o anumită zonă trec printr-un formalism care nu este pe deplin formulat, dar care în sine este obiect de critică.

7. Astfel, utilizarea acestui limbaj nu este relevantă în toate situațiile.

Avantajele UML

1. UML folosește simboluri grafice pentru elementele sistemului care se modelează, iar diagramele UML sunt destul de simple de înțeles;
2. UML face posibilă descrierea sistemelor din aproape toate punctele de vedere imaginabile, luând în considerare diferite aspecte;
3. UML este orientat pe obiecte: metodele sale de analiză și construcție sunt apropiate din punct de vedere semantic de metodele de programare utilizate în limbajele OOP moderne;
4. UML este un standard deschis. Standardul se dezvoltă și evoluează de la versiune la versiune, îndeplinind cele mai moderne cerințe de descriere a sistemelor;
5. conține un mecanism de extensie care permite introducerea unor tipuri de text și grafice suplimentare, ceea ce face posibilă utilizarea UML nu numai în domeniul IT.[4]

1.3 Structura generală a limbajului UML

Există două tipuri majore de diagrame UML: diagrame de structură și diagrame comportamentale (iar în aceste categorii există numeroase alte categorii). Aceste variații există pentru a reprezenta numeroasele tipuri de scenarii și diagrame utilizate de diferite tipuri de persoane.

De la clienți și manageri de proiect până la autori tehnici, designeri, analiști, programatori și testeri, fiecare rol va utiliza o diagramă specifică potrivită necesităților lor. Aceasta înseamnă că fiecare configurație necesită subiecte și niveluri de detalii diferite. Obiectivul este ca UML să exprime vizual diagrame care sunt ușor de înțeles pentru toată lumea.

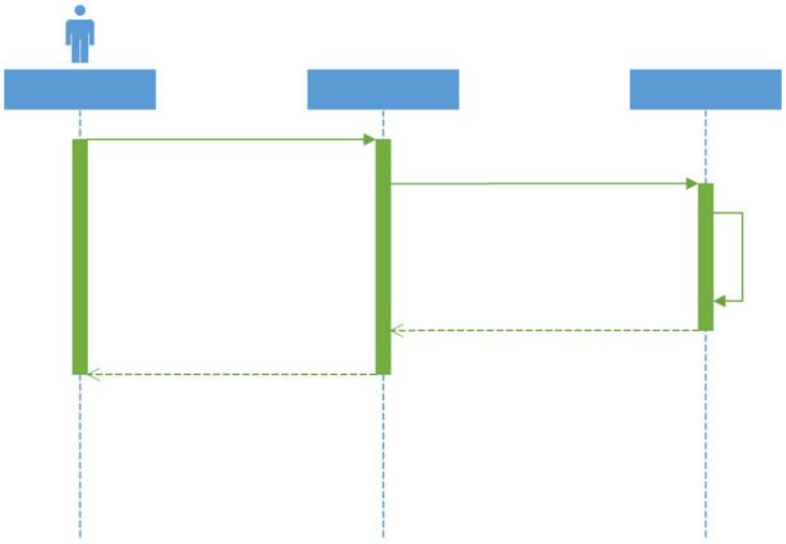

Figura 1 – Diagramă cu succesiune UML de bază.

Diagrame structurale

Diagramele structurale reprezintă structura statică a software-ului sau a sistemului și, de asemenea, prezintă diferite niveluri de abstracție și implementare. Acestea sunt utilizate pentru a vă ajuta să vizualizați diferite structuri care formează un sistem, precum o bază de date sau o aplicație. Acestea indică ierarhia componentelor sau modulelor și modul în care se conectează și interacționează unele cu altele. Aceste instrumente oferă îndrumări și asigură funcționarea tuturor părților unui sistem conform destinației prevăzute în legătură cu celelalte părți.

Diagrame comportamentale

Accentul se pune aici pe aspectele dinamice ale sistemului software sau procesului. Aceste diagrame prezintă funcționalitatea unui sistem și scot în evidență ceea ce trebuie să se întâmple în sistemul care este modelat.

Diagrame UML structurale

1. **Diagramă clase**. Această diagramă, cel mai comun tip în dezvoltarea de software, este utilizată pentru a reprezenta designul logic și fizic al unui sistem și îi indică clasele. Arată similar cu o organigramă, deoarece clasele sunt reprezentate cu casete. Această diagramă oferă o imagine a diferitelor clase și a modului în care sunt interconectate și fiecare clasă are trei compartimente:
   - Secțiunea superioară: numele clasei
   - Secțiunea din mijloc: atributele clasei
   - Secțiunea inferioară: metode sau operațiuni clasă

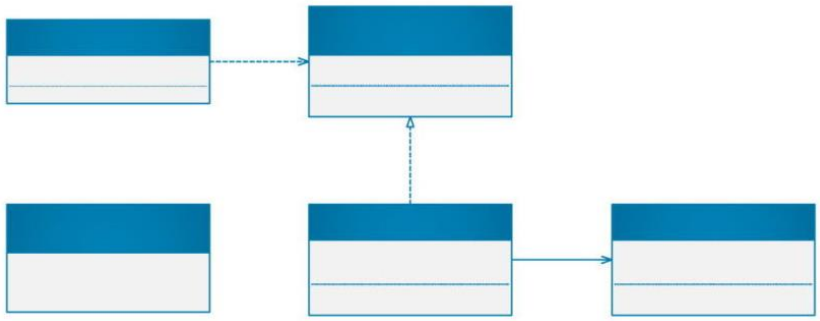

Figura 2 – Diagramă interfață clasă UML

2. **Diagramă obiecte**. Adesea, această diagramă este utilizată ca modalitate de a verifica încă o dată acuratețea unei diagrame de clase. Cu alte cuvinte, va funcționa în practică? Aceasta prezintă obiectele unui sistem și relațiile dintre acestea și oferă o vizualizare mai bună a posibilelor erori de design care trebuie remediate.
3. **Diagramă componente**. Cunoscută și ca organigramă a componentelor, aceasta prezintă grupările logice ale elementelor și relațiile dintre acestea. Cu alte cuvinte, oferă o vizualizare mai simplificată a unui sistem complex prin împărțirea acestuia în componente mai mici. Fiecare dintre părți este

prezentată cu ajutorul unei casete rectangulare, cu același nume scris în interior. Conectorii definesc relația/dependențele dintre diferite componente.

4. **Diagramă structură compozită**. Aceasta este utilizată rareori de persoanele din afara domeniului de dezvoltare de software. De ce? Deși este similară cu o diagramă de clase, este mai detaliată, descriind structura internă a mai multor clase și prezintă interacțiunile dintre acestea. În afara cazului în care sunteți dezvoltator, o imagine de ansamblu conține probabil suficiente informații.
5. **Diagramă implementare**. Această diagramă prezintă componentele hardware (nodurile) și software (artefactele) și relațiile dintre acestea. Aceasta oferă o reprezentare vizuală a locațiilor exacte în care sunt implementate componentele software.
6. **Diagramă pachete**. Aceasta este utilizată pentru a prezenta dependențele dintre pachetele care compun un model. Obiectivul principal este de a prezenta relația dintre diferitele componente mari care formează un sistem complex.
7. **Diagramă profiluri**. Aceasta este mai degrabă un limbaj, decât o diagramă. O diagramă de profiluri contribuie la crearea de noi proprietăți și semnificații pentru diagramele UML prin definirea stereotipurilor particularizate, valorilor etichetate și constrângerilor. Aceste profiluri vă permit să particularizați un metamodel UML pentru diferite platforme (de ex., Java Platform, Enterprise Edition (Java EE) sau Microsoft .NET Framework) și domenii (de ex., modelarea de procese de afaceri, arhitectura dedicată serviciilor, aplicațiilor medicale și altele).

Diagrame UML comportamentale

1. **Diagramă activități**. Aceasta prezintă un proces pas cu pas cu un început și un sfârșit clare. Este un set de activități care trebuie să aibă loc pentru a atinge un obiectiv. Prezintă cum fiecare activitate duce la următoarea și cum sunt toate conectate. Pe lângă dezvoltarea de software, aceasta poate fi utilizată în aproape orice mediu de afaceri. Este denumită și mapare sau modelare de procese de afaceri.

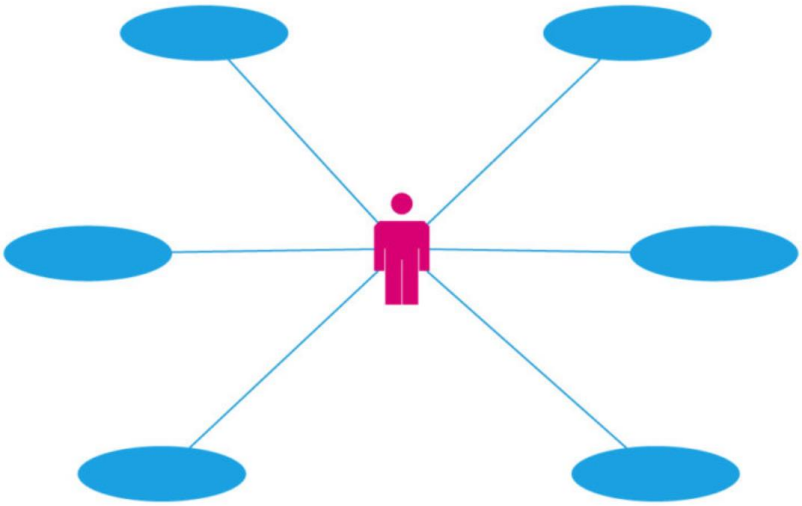

Figura 3 – Diagramă UML de bază cu cazuri de utilizare

2. **Diagramă cazuri de utilizare**. Aceasta descrie ce face un sistem dar nu și cum o face. Un caz de utilizare este un set de evenimente care se produc atunci când un "actor" utilizează un sistem pentru a finaliza un proces. Un actor este definit ca orice interacționează cu sistemul (persoană, organizație sau aplicație) din afara sistemului. Așadar, o diagramă de cazuri de utilizare descrie vizual un set de succesiuni și reprezintă cerințele funcționale ale sistemului.
3. **Diagramă prezentare generală interacțiuni**. Adesea complexă, această diagramă este similară cu diagrama de activități deoarece ambele prezintă o succesiune pas cu pas a activităților. Însă, o diagramă de prezentare generală a interacțiunilor este o diagramă de activități compusă din diferite diagrame de interacțiuni. Acestea utilizează aceleași adnotări ca o diagramă de activități (inițial, final, decizie, fuzionare, bifurcație și noduri de îmbinare) cu adăugarea de elemente precum interacțiunea, utilizarea interacțiunii, constrângerea de timp și constrângerea de durată.
4. **Diagramă sincronizare**. Atunci când sincronizarea intră în primplan, este utilizată această diagramă UML. Cunoscută și sub numele de diagramă de determinare a succesiunii sau diagramă de evenimente, nu prezintă modul în care interacționează obiectele sau în care se modifică unele pe altele. Funcțional, prezintă modul în care obiectele și actorii acționează într-o cronologie. Accentul se pune aici pe cât de mult durează evenimentele și pe modificările care se produc în funcție de constrângerile de durată. Printre părțile principale ale diagramei de sincronizare se numără:
    - Linia vieții: participant individual
    - Cronologie stare: diferitele stări prin care trece linia vieții în cadrul unui canal
    - Constrângere de timp: timpul necesar pentru îndeplinirea unei constrângeri
    - Constrângere de timp: un timp în care ceva trebuie îndeplinit de către participant
    - Producere distrugere: unde se încheie linia vieții unui obiect. După producerea distrugerii într-o linie a vieții acesta nu va mai apărea niciodată.
5. **Diagramă mașină stare**. Denumită și diagramă de stare, aceasta se aplică atunci când comportamentul unui obiect este complex și detaliile sunt esențiale. Ajută la descrierea comportamentului unui obiect (sau, uneori, a unui operator) și a modului în care se schimbă în baza evenimentelor interne și externe.
6. **Diagramă succesiune**. Populară nu numai în comunitatea designerilor, această diagramă cu aspect vizual atrăgător este bună pentru prezentarea tuturor tipurilor de procese de afaceri. Aceasta dezvăluie pur și simplu structura unui sistem, prezentând în mod cronologic esența mesajelor și interacțiunilor dintre actori și obiecte. Diagramele de succesiuni prezintă iterații și ramificații simple. Sunt favorabile pentru multi-tasking.
7. **Diagramă comunicare**. O diagramă de comunicare sau colaborare este similară unei diagrame de succesiune. Totuși, aceasta pune accent pe comunicarea dintre obiecte. Aceasta prezintă organizarea obiectelor care participă într-o interacțiune și are iterații și ramificații mai complexe.[3]

2 Modelarea și proiectarea sistemul informatic

Proiectarea corecta din start a unui proiect poate economisi multe resurse și este o cheie spre succes. Întrucît nu toți membrii echipei sunt programatori este nevoie de ales o modelare vizuală care să fie pe înțelesul tuturor. Modelarea constă în prezentarea schematică a proceselor din cadrul sistemului. Unul din limbaje cunoscute pe larg este UML (The Unified Modeling language). Caracteristicile principale ale limbajului UML sunt:

- limbajul UML conține mai multe tipuri de diagrame care descriu funcționalitatea sistemului(diagrama de clasă, de colaborare, de secvență, etc.);
- limbajul UML permite modelarea detaliată a fiecarei stări dintr-un proces;
- bazat pe modelarea orientată pe obiect;
- permite exportarea codului în baza diagramelor.

Enterprise Architect este un soft care oferă mediul de dezvoltare necesar pentru limbajul de programare UML. Este foarte comod și util din punct de vedere al interfeței, oferind o gamă largă de posibilități. Prin urmare, pentru modelare sistemului se va folosi acest instrument fiindcă este comod și accesibil.[6]

2.1 Reprezentarea diagramelor în limbajul UML

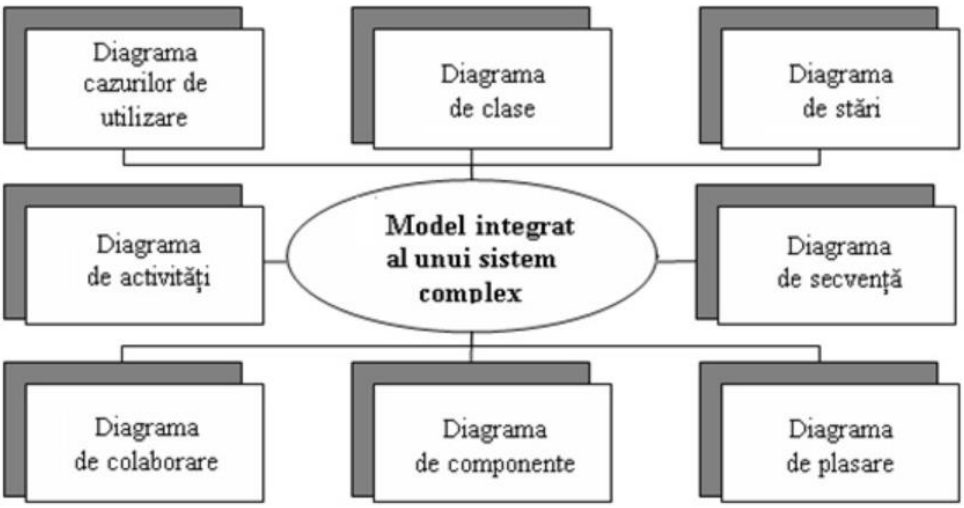

Figura 4 – Model al unui sistem complex în notația UML.

Modelarea vizuala in UML poate fi reprezentat ca un oarecare proces de lansare pe niveluri de la cel mai general și abstract model conceptual al sistemului inițial către modelul logic și mai apoi fizic, ce corespunde unui sistem de program. Pentru atingerea acestui scop de la început se crează un model în forma de diagrama cazurilor de utilizare care descrie destinația funcțională a sistemului sau cu alte cuvinte descrie ceea ce sistemul va executa în procesul sau de funcționare. Diagrama cazurilor de utilizare reprezintă un model inițial conceptual al unui sistem în procesul de proiectare și exploatare.

Proiectarea unei diagrame a cazurilor de utilizare urmarește scopurile:

- determinarea limitelor comune și a contextului domeniului de modelare la etapele inițiale de proiectare a unui sistem;
- formularea cerințelor comune către comportare funcțională și sistemului proiecat;
- elaborarea modelului inițial conceptual al unui sistem pentru detalierea de mai tîrziu în forma modelelor logice și fizice;
- pregătirea documentației inițiale pentru interacțiunea elaboratorilor unui sistem cu clienții și utilizatorii.[5]

Diagrame UML

În cadrul limbajului UML toate reprezentările modelului unui sistem complex sunt fixate în calitate de construcţii speciale grafice care deseori sunt reprezentate sub formă de graf conex cu noduri – entităţi şi muchii – relaţii. În UML sunt definite nouă tipuri de diagrame[6]:

- Diagrame cazurilor de utilizare (use case diagram)
- Diagrame de clase (class diagram)
- Diagrame de comportament (behavior diagrams)
    - Diagrame de stări (statechart diagram)
    - Diagrame de activităţi (activity diagram)
    - Diagrame de interacţiune (interaction diagrams)
        - Diagrame de secvenţă (sequence diagram)
        - Diagrame de colaborare (collaboration diagram)
- Diagrame de realizare (implementation diagrams)
    - Diagrame de componente (component diagram)
    - Diagrame de plasare (deployment diagram)

2.1.1 Imaginea generală asupra sistemului

În limbajul UML, imaginea generală asupra sistemul se realizează prin diagrama Use Case care descrie procesul de funcționare a unui model funcțional. Pentru acesta se folosește un actor prin care se arată interecțiunea cu mai multe procese din sistem.

Diagrama cazurilor de utilizare reprezintă un model iniţial conceptual al unui sistem în procesul de proiectare şi exploatare.

Proiectarea a unei diagrame a cazurilor de utilizare urmăreşte scopurile următoare:

- determinarea limitelor comune şi a contextului domeniului de modelare la etapele iniţiale de proiectare a unui sistem;
- formularea cerinţelor comune către comportare funcţională a sistemului proiectat;
- elaborarea modelului iniţial conceptual al unui sistem pentru detalierea de mai tîrziu în forma modelelor logice şi fizice;

- pregătirea documentării iniţiale pentru interacţiunea elaboratorilor unui sitem cu clienţii şi utilizatorii.

Structura sau elementul standart al limbajului UML – caz de utilizare se foloseşte pentru specificarea particularităţilor comune ale comportării unui sistem sau a oricărei alte entităti în domeniul de lucru fără cercetarea structurii interne a acestei entităţi. Fiecare caz de utilizare determină o succesiune de acţiuni care trebuie sa fie executate de către sistemul poiectat la colaborarea lui cu actorul corespunzător.

Diagrama cazurilor de utilizare poate fi completată cu text explicativ, care desfăşoară sensul sau semantica componentelor ce o compun. Acest text se numeşte adnotare sau scenariu.

Cazul de utilizare aparte se notează cu o elipsă în interiorul căreia se conţine denumirea prescurtată sau numele în formă de verb cu cuvinte explicative.[8]

Figura 5 reprezintă actorul Utilizatorul cu toate cazurile de utilizare din cadrul aplicații. Rolul lui este de a utiliza aplicați pentru a beneficia de sală de sport și a se folosi de toate oportunitățile pe care lui-a i se oferă la maxim.

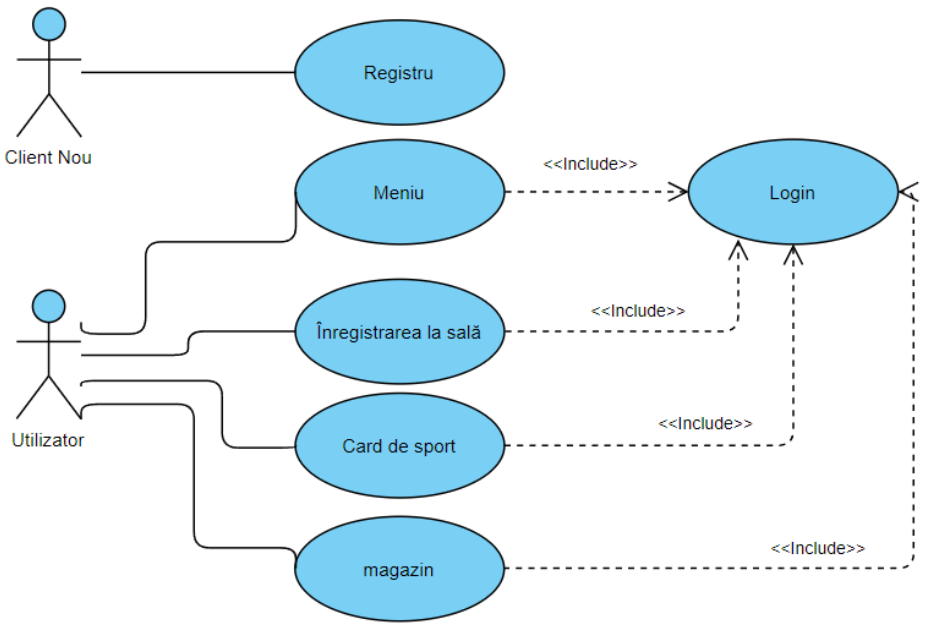

Figura 5 – Interacțiunea Utilizatorului cu aplicația

În figura 6 poate fi observată oportunitățile pe care ne poate înregistrări la o aplicație a unei Săli de sport. Astfel, Utilizatorul poate să î-și aleagă Antrenorul care să îl ajute, domeniul de antrenament care el vrea să se antreneze, poate să î-și facă un grafic cît mai bun pentru a vizita sala de sport și a se antrena cu placere. Și un plus mare este că el poate să își aleagă surplusurile de care el vrea să se folosească după antrenament.

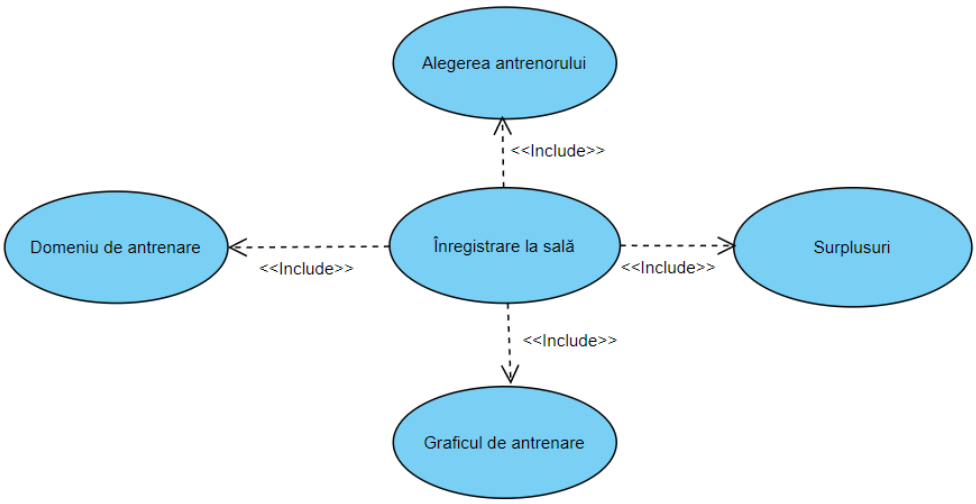

Figura 6 – Procesul de înregistrare la sală

În figura 7 se observă că în aplicația dată este prezent și un magazin care îi permite Utilizatorul să beneficieze de diferite obiecte necesare în timpul unui antrenament perfect ,ca de exemplu: îmbracăminte potrivit[, proteine calitative, bare și gantele de antrenament și unul din cele mai importante fitness braslet care îi va permite mereu să afle multă informație despre rezultatele lui după antrenament sau în timpul lui, pentru a observa schimbarile mai bine care se petrec cu el.

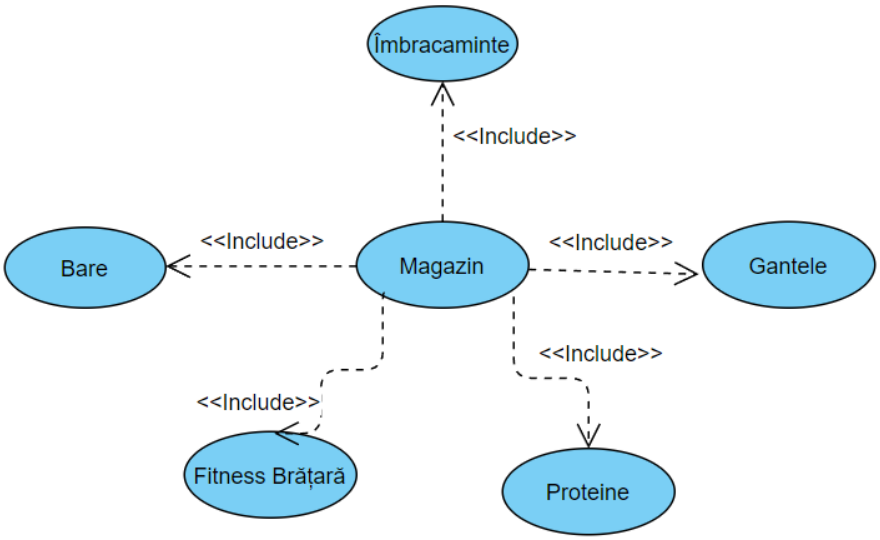

Figura 7 – Magazinul propus de aplicație

Figura 8 reprezintă actorul Director cu toate cazurile de utilizare din cadrul aplicației. Rolul principal al directorului este gestionarea aplicației pentru a controla că totul să fie în regulă. Dacă aparatele de antrenat sunt satisfăcătoare, are destuli antrenori calificați, se respectă toate normele și condițiile pentru sport pitanie și dacă surplusurile sunt destule pentru a le satisface toate dorințele clienților săi.

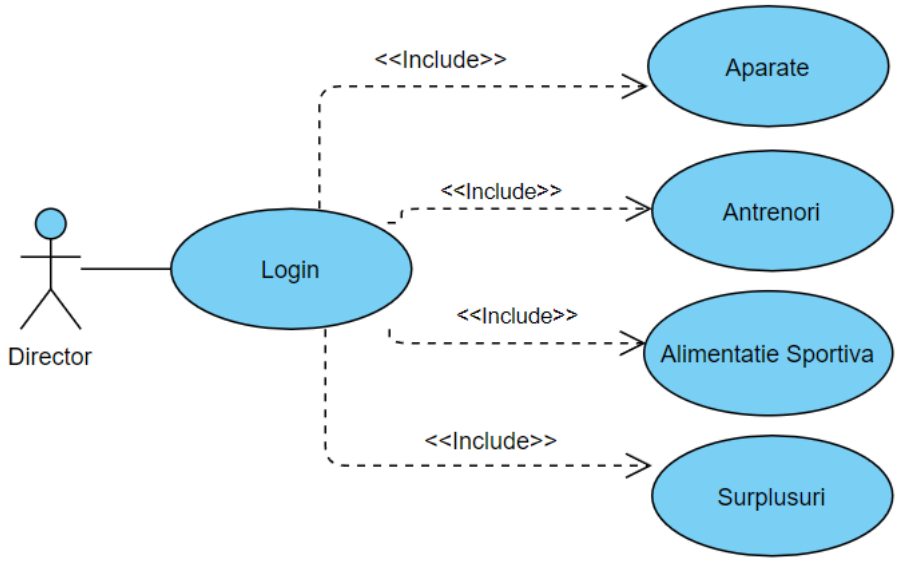

Figura 8 – Interacțiunea Directorului

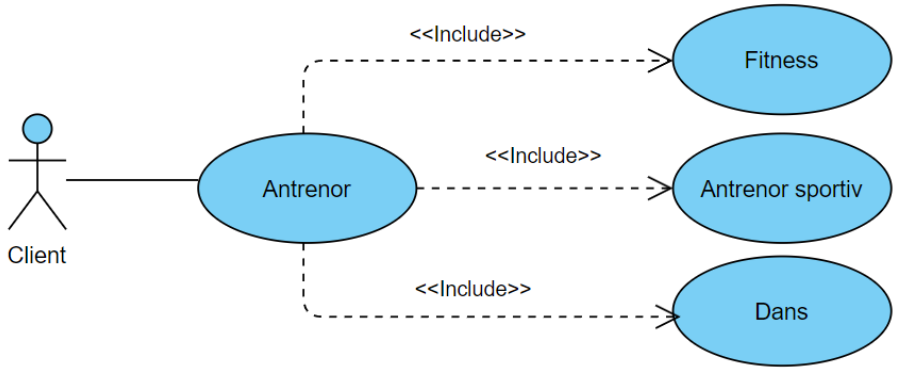

Figura 9 – Tipuri de Antrenori

În figura 9 putem observa că clientul poate interacționa și alege dintr-un numar de categorii de antrenori: Fitness trener, antrenor de sport și antrenor de dans.

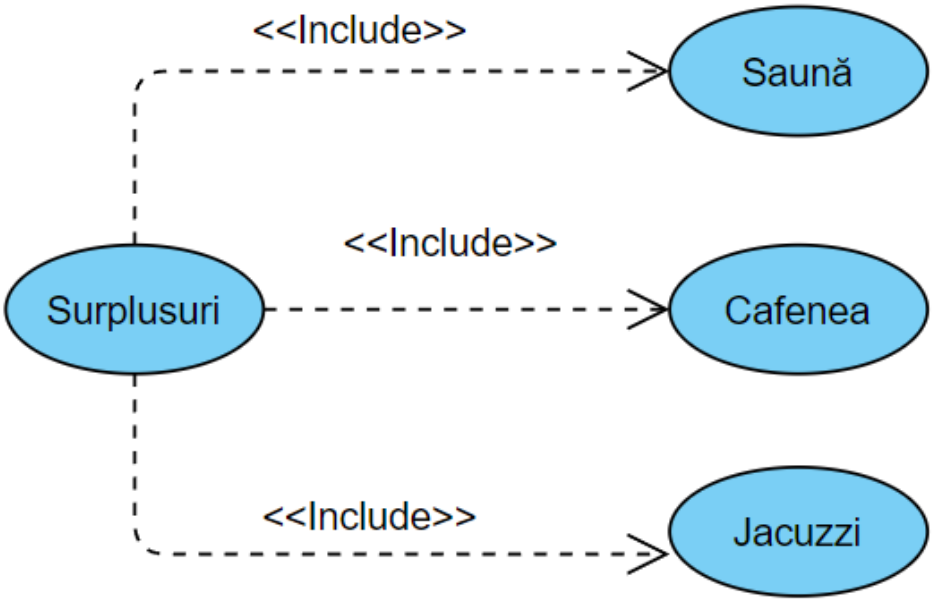

Figura 10 – Metode de relaxare

În figura 10 sunt descrise surplusurile pe care sale de sport le poate oferi clienților după antrenament ca: Saună, Cafenea și Jacuzzi.

În figura 11 sunt prezentate un regim de Alimentație Sportivă pentru clienți pentru ai ajuta ca să ajungă mai ușor la rezultate bune și să nu aibă probleme cu sănătatea. Adică, Cină AP după antrenament, Dietă în timpul exercițiilor în sală, Vitamine pentru AP și sport, Nutriție adecvată pentru meniu de alimnentație pentru fiecare zi.

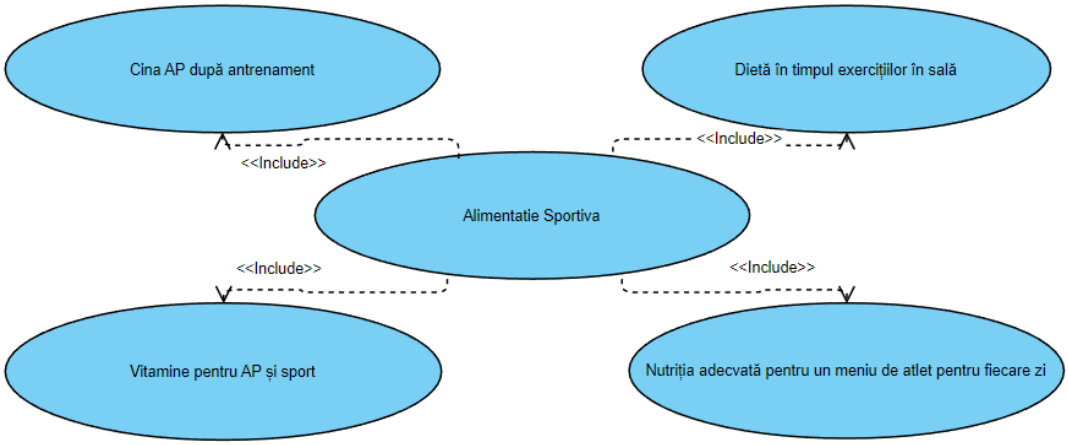

Figura 11 – Regim de alimentație

În figura 12 sunt prezentate aparatele pe care le oferă sala de sport și de care pot beneficia clienții ei. Ca de exemplu de apărate: Benzi de alergare, Steppers, Tabelele de inversare, Aparate de vâsle.

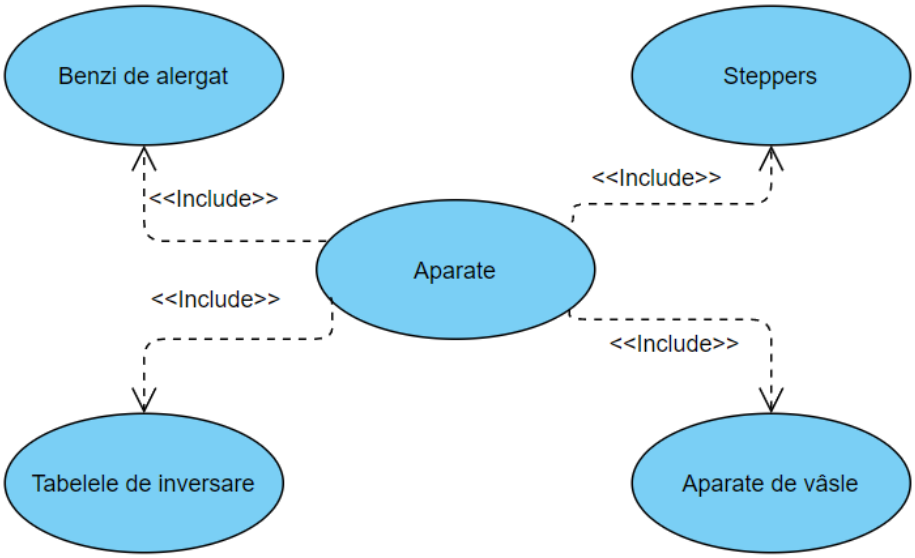

Figura 12 – Apărate de sport

2.1.2 Descrierea scenariilor de utilizare a aplicației

Descrierea scenariilor de utilizare a aplicației în limbajul UML se realizează prin diagrama de secvență, care descrie ordonat interacțiunea obiectelor. Ea descrie setul de acțiuni care se execută asincron sau sincron pentru executarea unui funcțional.

În diagrama de se descriu numai obiectele care sunt direct implicate în interacţiune şi nu reflectă asocierile statice cu alte obiecte. Pentru diagrama de secvenţă momentul principal este dinamica colaborării între obiecte în timp. Grafic fiecare obiect se reprezintă printr-un dreptunghi şi se plasează în partea de sus a ciclului său de viaţă. În interiorul dreptunghiului se indică numele obiectului şi numele clasei despărţite prin două puncte. Totodată toată înregistrare se subliniază, ce indică că obiectul este exemplarul unei clase. În caz dacă numele obiectului lipseşte, atunci se indică numai numele clasei şi obiectul se consideră anonim.

A două măsură a diagramei de secvenţă este axa verticală (de sus în jos). Momentului iniţial de timp îi corespunde partea de sus al diagramei. Totodată colaborarea obiectelor este realizată prin mesajele transferate. Mesajele se reprezintă sub forma de săgeţi drepte cu numele mesajelor, ele de asemenea sunt într-o ordine anumită în timp. Cu alte cuvinte, mesajele plasate în diagrama de secvenţă mai sus sunt iniţiate mai devreme decît cele din jos. Totodată proporţiile pe axa temporală nu se indică fiindcă diagrama de secvenţă modelează doar ordonarea în timp a legăturilor de tip «mai devreme–mai tîrziu».

Linia de viaţă a obiectului (object lifeline) se reprezintă printr-o linie verticală punctată asociată cu un singur obiect în diagrama de secvenţă. Linia de viaţă defineşte intervalul de timp în care obiectul există şi interacţionează cu sistemul dat.[8]

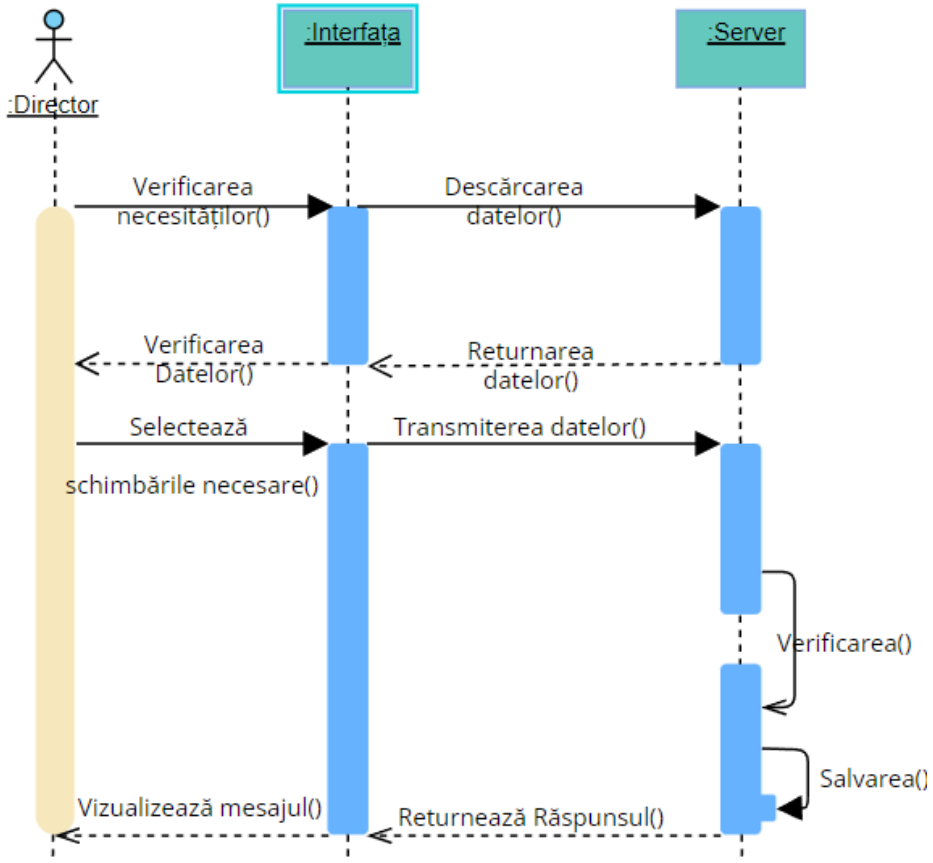

Figura 13 – Interacțiunea Directorului cu schimbărilor necesare pentru sala de sport

În figura 13 este prezentată interacțiunea Directorului cu schimbărilor necesare pentru sala de sport. Acest drept îl are doar Directorul. Întîi de toate el va verifica dacă totul necesar este disponibil, pentru

aceasta se va crea cerere la server pentru a descărca datele. Mai apoi în mod sincron se returnează datele și se vizualizează de către director. După care se fac schimbările dacă sunt necesare și datele se transmit la server, unde se validează. Dacă validare a fost cu succes, datele se inserează în baza de date. În caz contrar se returnează mesaj de eroare către utilizator. Și ultimul pas este primirea răspunsului și vizualizarea lui.

În figura 14 se descrie interacțiunea clientului cu magazinul din aplicatie. Pentru aceasta este necesar în mod sincron de a deschide magazinul, după care se crează cerere către server pentru a primi acces, după care se returnează răspunsul și se vizualizează de către client. În mod asincron se selectează categoria și deja în mod sincron se selectează produsele dorite, se crează cerere de procurare a acestor produse, se verifică dacă sunt în stoc și se returnează răspunsul și se afisează. Se crează cerere de salvare și primirea mesajului că salvarea sa făcut cu succes.

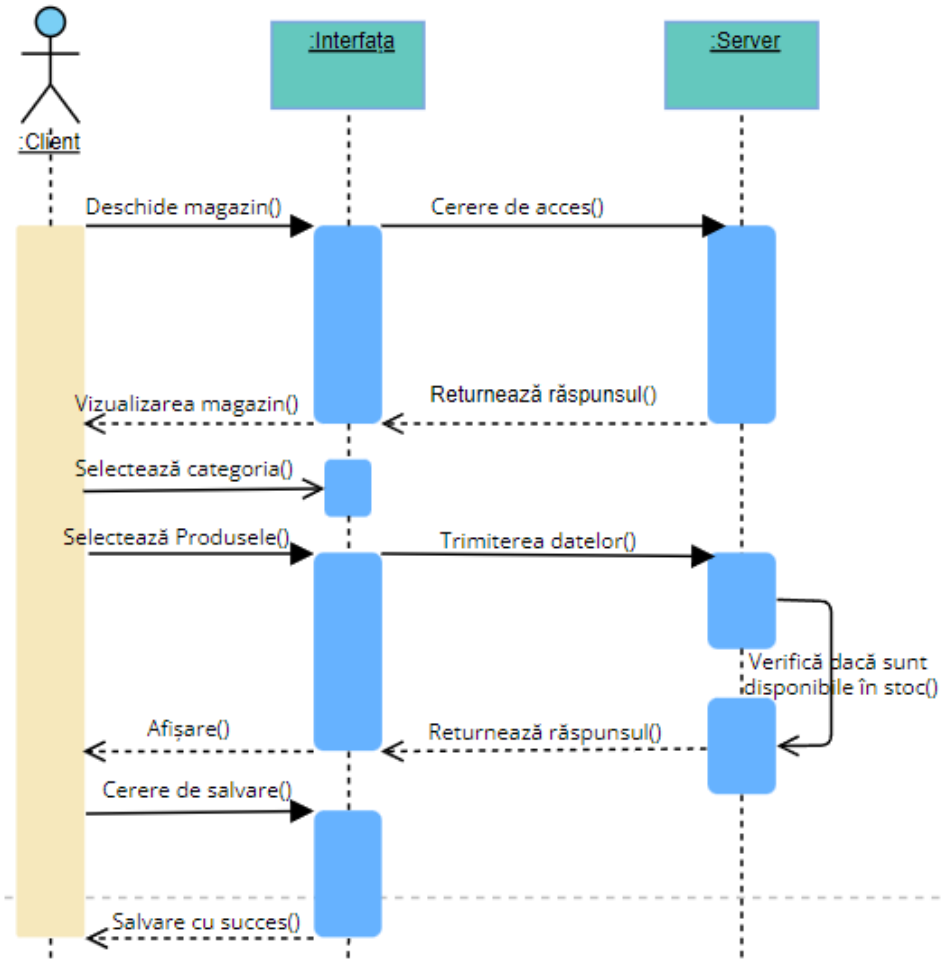

Figura 14 – Interacțiunea clientului cu magazinul din aplicație

În figura 15 se descrie interacțiunea Utilizatorul cu aplicația. În mod sincron se formează cerere de înregistrare la sală, se trimite cerere către server și se returnează răspunsul, apoi se vizualizează de către utilizator. Urmatorul pas ar fi în mod asincron, utilizatorul își va selecta antrenorul dorit, domeniul de antrenament, graficul potrivit și surplusurile dorite. Apoi în mod sincron salvează, după care cerere este

trimisă catre server care verifică(asincron) și se salvează cerere creată(sincron), după care se returnează răspunsul și se vizualizează de catre utilizator.

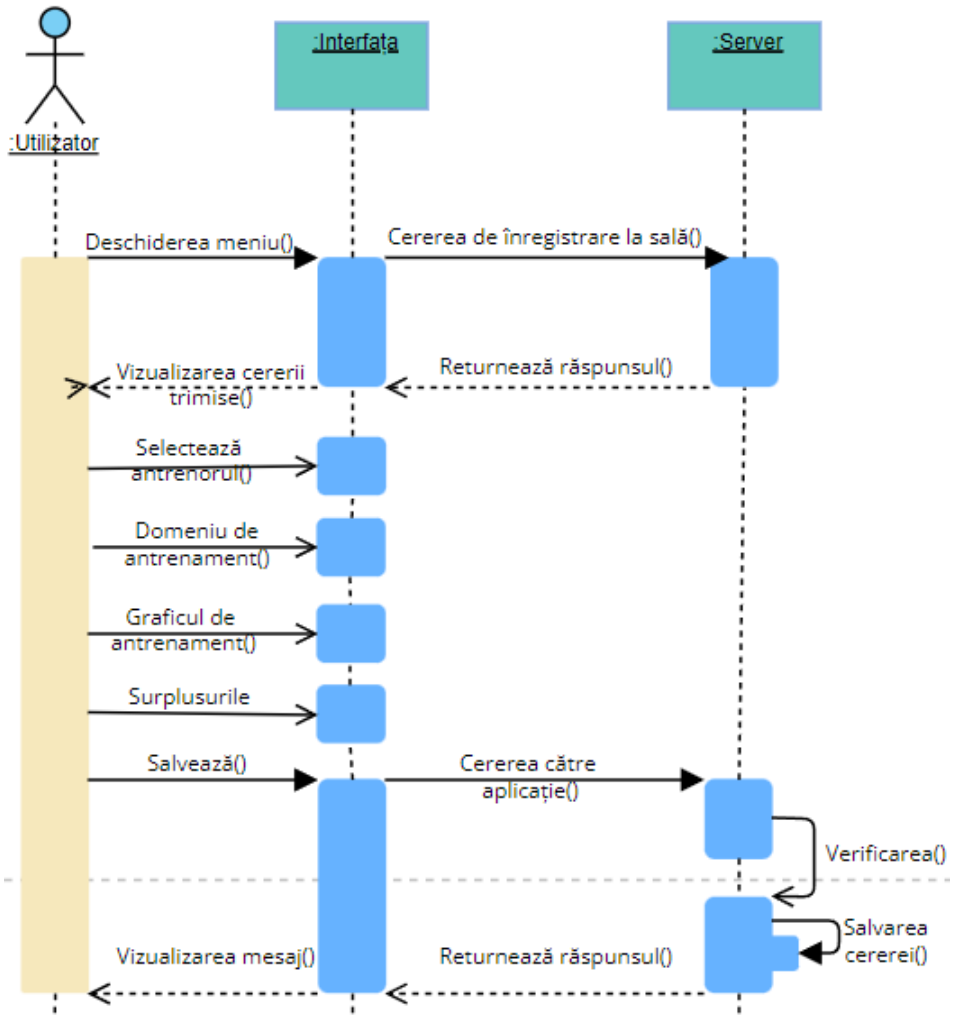

Figura 15 – Interacțiunea Utilizatorul cu aplicația

2.1.3Analiza rezultatelor modelării din diagramele cazurilor de utilizare și dezvoltare în diagramele de colaborare.

Diagrama de colaborare este o diagramă de interacțiune care subliniază organizarea structurală care trimit și primesc masaje. Diagrama de colaborare realizează o descriere non-secvențială a modului în care interacționează obiectul. Diagramele de colaborare sunt în particular indicate pentru faza exploratorie, care corespunde căutării obiectelor.

Diagramele de colaborare sunt utilizate pentru:

- A descrie diverse scenarii de funcționare a aplicației prin reprezentarea fluxurilor de mesaje dintre obiecte;
- A prezenta organizarea spațială a obiectelor și legăturile dintre acestea;

Diagramele de colaborare conţin :

- obiecte - reprezentarea lor grafică sub formă de dreptunghiuri
- legături între obiecte- reprezentate grafic prin linii de conectare

- mesaje - reprezentate ca etichete ale legăturilor, şi care conţin o săgeată îndreptată spre obiectul server (receptor al mesajului).[8]

În figura 16 – este reprezentat Directorul și interacțiunile pentru aplicația unei sale de sport cu ajutorul diagramelor de colaborare. Unde directorul verifică datele și dacă sunt necesare schimbările, el le face, după care se verifică si se salvează.

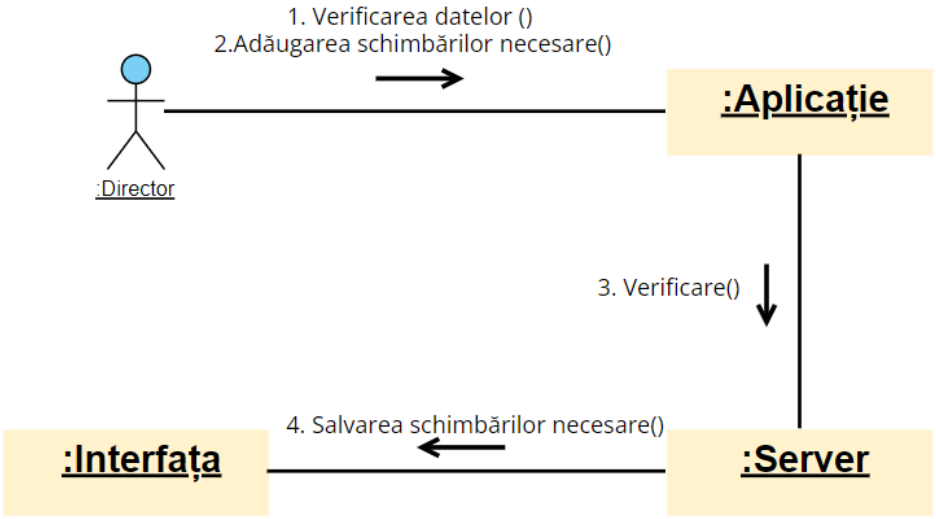

Figura 16 – Interacțiunea Directorului cu schimbărilor necesare pentru sala de sport

În figura 17 – este reprezentat Clientul și interacțiunile lui cu magazinul cu ajutorul diagramelor de colaborare. Unde are loc selectarea produselor dorite, verificara dacă sunt în stoc și salvarea cumpărăturilor dorite.

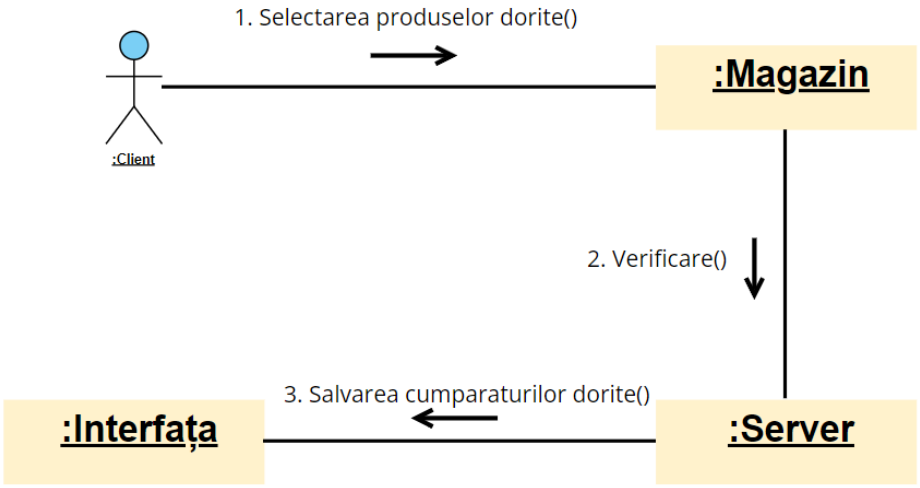

Figura 17 – Interacțiunea clientului cu magazinul din aplicație

În figura 18 – este reprezentat Utilizatorul și interacțiunile lui cu aplicația cu ajutorul diagramelor de colaborare. Unde are loc înregistrarea la sală, după care el î-și alege antrenorul, domeniul de antrenamen, graficul de antrenament și surplusurile dorite, după care urmatorul pas este verificcarea și deja salvarea.

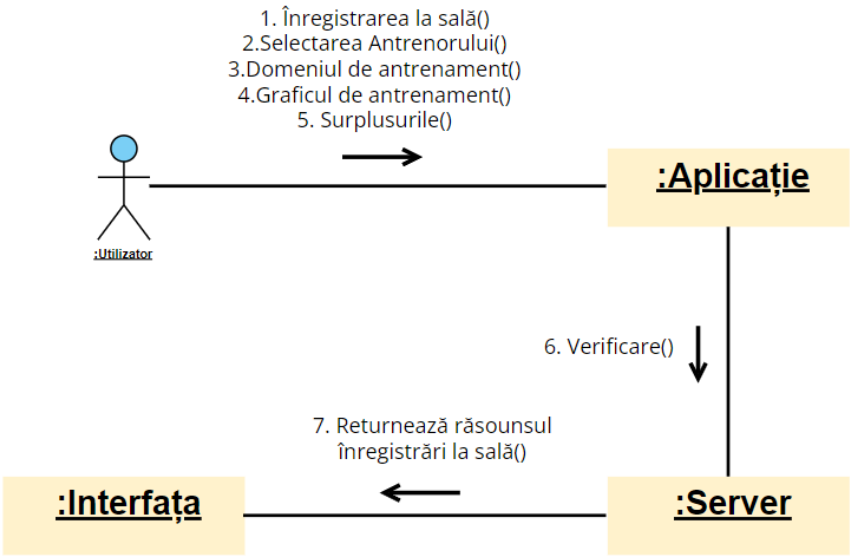

Figura 18 – Interacțiunea Utilizatorul cu aplicația

2.1.4 Diagrama de clase

Diagrama de clase (class diagram) se utilizează pentru reprezentarea structurii statice a unui model de sistem în terminologia claselor programării OO. Diagrama de clase poate reflecta diferite legături între entitățile domeniului de obiecte (obiecte și subsisteme) și descrie structura lor internă și tipurile de relații.

Diagrama de clase reprezintă un graf cu noduri – elemente de tip «clasificatori» care sunt legate prin diferite tipuri de relaţii de structură. Trebuie de menţionat că diagrama de clase poate conţine interfeţe, pachete, relaţii şi chiar exemplare, aşa ca obiecte şi legături. Prin diagrama de clase se subînţelege modelul structural static al sistemului proiectat, de accea diagrama de clase deseori se socoate o reprezentare grafică a legăturilor structurale ale modelului logic al sistemului care sunt independente şi invariante în timp.[8]

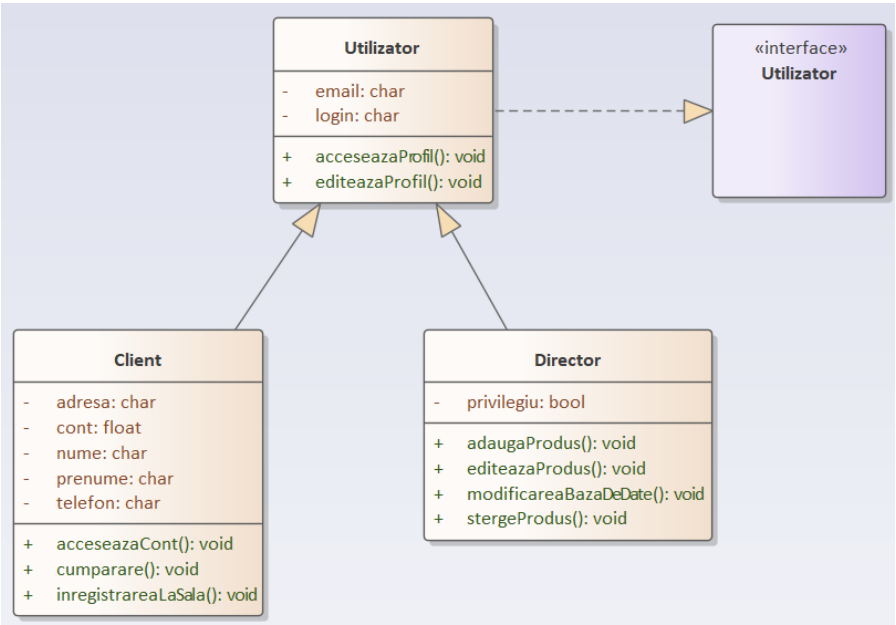

Figura 19 – Descrierea structurii clasei ”Utilizator”

În figura 19 avem clasa de bază „Utilizator”, iar de la clasa de bază sunt generalizate încă două clase: clasa „Client” si clasa „Director”. Clasa „Client” preia informația de la utilizator și mai adaugă informații personale care vor trebuie în timpul comandarii unui produs sau înregistrării la sală. De asemenea în clasa „Client” apare starea contului ce poate fi modificata doar de client în orice timp sau în timpul achitării unui produs sau înregistrării la sală. În clasa „Director” apare variabila „privilegiu”, cu aceasta variabilă verificăm dacă utilizatorul face parte din clasa „Director”, de asemenea apar metode noi ca adaugarea unui produs, editarea unei produs etc.

În figura 20 avem clasa „Înregistrarea la sală” care reprezintă pagina de înregistrare la sală, adică care domeniu dorim, grafic potrivit, antrenor și preț accesibil. În aceasta pagină putem găsi recomandație despre antrenori, graficuri aranjate și surplusuri pe care le oferă sala. De asemenea în partea cea mai de jos a paginii putem găsi comentarii ,dacă sunt, lăsate de către alți utilizatori.

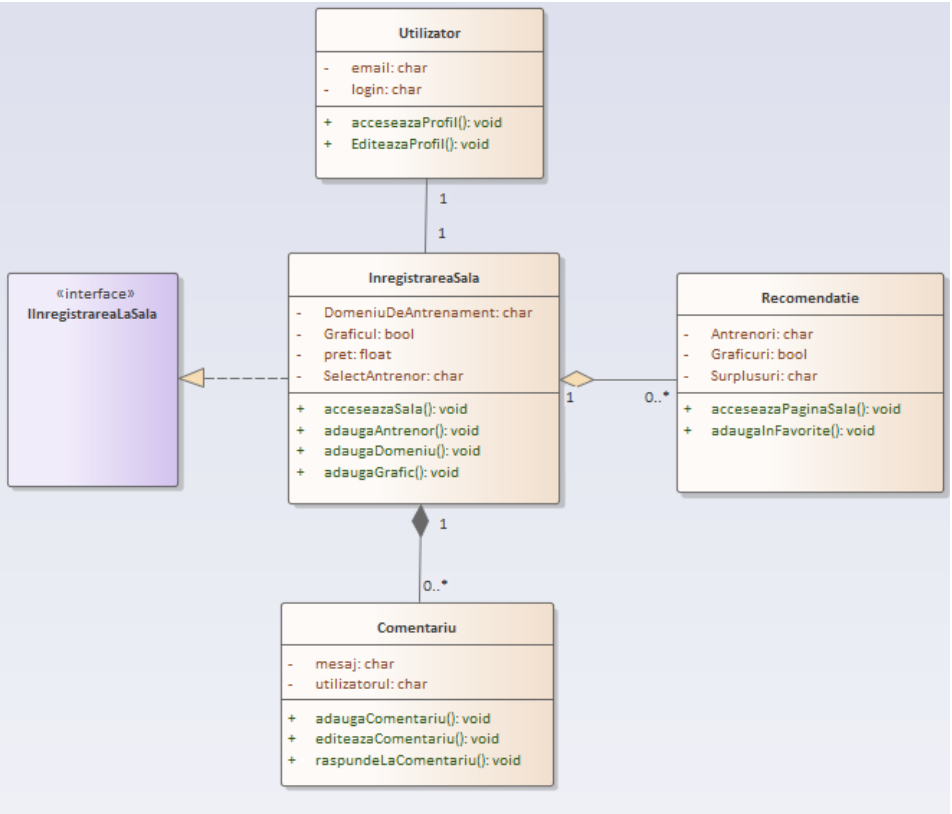

Figura 20 – Descrierea structurii clasei ”Înregistrarea la sală”

În figura 21 avem clasa „Magazin” ce conține o listă de produse care sunt distribuite în diferite categorii. Fiecare produs are o imagine, o denumire, un producător și un preț, informația despre produs și imagine sunt stocate în baza de date. Clasa “Magazin” poate avea de la 0 până la o infinitate de produse, iar un utilizator poate avea doar un Magazin și o funcție de cautare. De asemenea fiecare produs în mod obligator are o imagine atașată la ea.

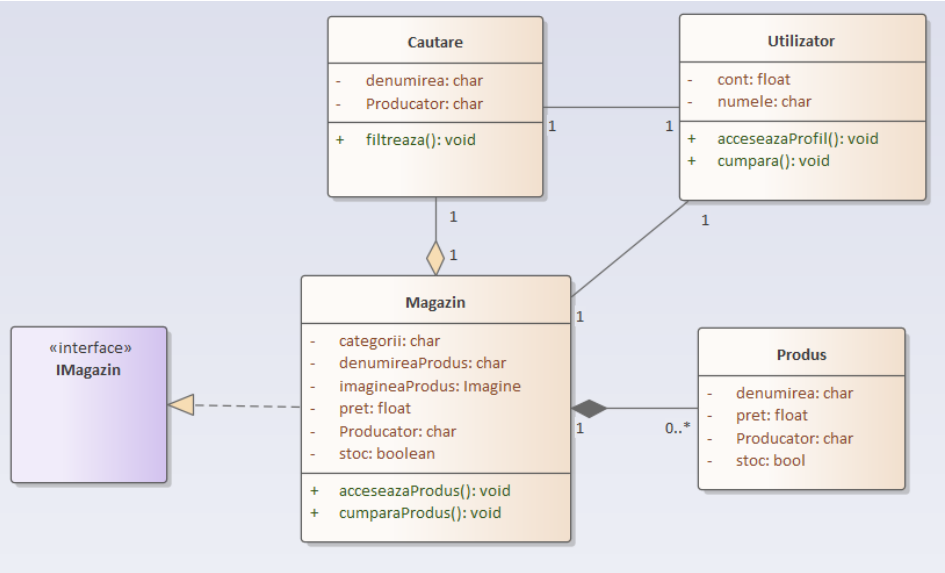

Figura 21 – Descrierea structurii clasei ”Magazin”

2.1.5 Diagrama de stare și diagramelor de activități

Diagrama de stare descrie procesul de modificare a stărilor numai pentru o clasă, pentru un exemplar a unei clase, adică de a modela toate modificările posibile în starea unui propriu obiect. În urma căruia modificarea stării obiectului poate fi provocată de influenţa externă a altor obiecte sau din exterior.

Diagrama de stare se utilizează pentru descrierea consecutivităților de stări posibile și treceri care în ansamblu caracterizează comportamentul elementelor pe parcursul ciclului său de viață.

Automatul în UML reprezintă o formalizare pentru modelarea comportamentului elementelor modelului şi a sistemului întreg. În metamodelul UML automatul este un pachet, în care sunt definite o mulţime de definiţii, necesare pentru reprezentarea comportamentului entităţii modelate în formă de spaţiu discret cu un număr finit de stări şi treceri. În metamodelul UML automatul este un pachet care definește o mulțime de definiții necesare pentru reprezentarea comportamentului entități modelate cu un număr finit de stări și tranziții.

Starea în UML este subînțeleasă ca o metaclasă abstractă care se utilizează pentru modelarea situațiilor separate pe parcursul căreia se execută anumite condiții.

Starea poate fi în formă de valori concrete a atributului clasei sau obiectului, în acest caz modificarea anumitelor valorilor va respinge modificarea clasei modelate sau obiectului.

Starea conține 3 acțiuni:

- entry
- do
- exit

tranziţie - O simplă *tranziţie* reprezintă o relaţie între două stări consecutive indicînd faptul schimbării a unei stări cu altă. Prezenţa obiectului modelat în prima stare va efectua anumite acţiuni, dar

trecerea în starea a doua va fi atunci cînd anumite acţiuni vor fi terminate şi după îndeplinirea anumitor condiţii adăugătoare.

Diagrama de activitate - Pentru modelarea procesului de executare a operaţiilor în limbajul UML se utilizează aşa numitele diagrame de activităţi. Notaţia grafica acceptată pentru aceste diagrame are mult comun cu notaţia diagramei de stări ce se evidenţiază prin notarea stărilor şi tranziţiilor.

Deosebirea constă în semantica stărilor care sunt utilizate pentru prezentarea acţiunilor dar nu activităţilor, şi în aceea că tranziţiile evenimentelor nu sunt etichetate. Fiecare stare în diagrama de activităţi corespunde executării unei operaţiuni elimentare, dar trecere în altă stare se execută numai la terminarea operaţiei în starea precedentă.

Starea activităţii este un caz particular a stării. Starea activităţii nu poate avea tranziţii interne fiindcă ea este elementară. Starea activităţii se utilizează pentru modelarea unui pas de executarea a algoritmului (procedurii) sau a unui flux de control.[8]

În diagrama de stare reprezentată în Figura 22. sunt reprezentate toate stările posibile și trecerile care caracterizează acțiunea de vânzare a unui produs sau de achitare abonamentului la sală din aplicație. Pentru inceput clientul alege cum dorește să achite produs sau abonamentului la sală, cu cardul sau cash. Dacă a ales achitarea cu cardul atunci mai întâii el introduce datele cardului și datele sale persoanle, apoi datele cardului se duc la verificare, iar dacă verificarea a fost cu succes atunci clientul confirmă comanda prin intermediul aplicații băncii de care se folosește și după confirmare primește datele despre comandă efectuată, dacă verificarea a depistat erori, atunci clientul trebuie din nou să introducă datele cardului corecte. Apoi pagina cu tranzacție se închide. Iar dacă a ales platirea cu cash, atunci clientul introduce datele personale și primeste data livrării produs sau că achitare abonamentului la sală a fost efectuată cu succes și va putea să fregventeze sală de sport.

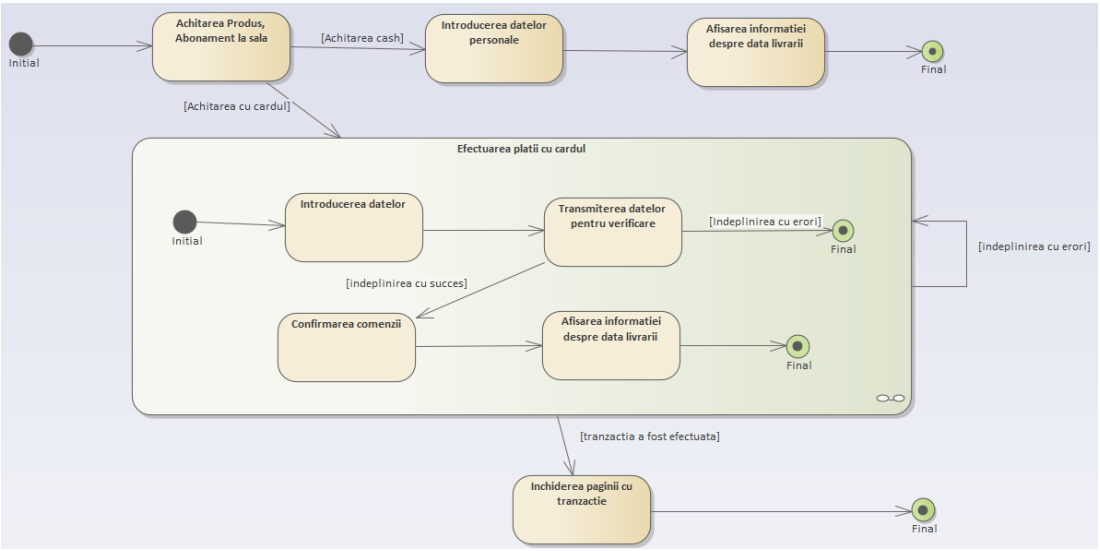

Figura 22 – Cumpararea unui produs, achitare abonamentului la sală de sport

În diagrama de stare reprezentată în Figura 23 sunt reprezentate toate stările posibile și trecerile care caracterizează acțiunea de înregistrare și logare pentru Clinții. Pașii pentru a ne înregistra sunt introducerea nume/prenume, email și parola dorită. Pentru a ne loga este necesar doar să introducem email și parola. Și pentru ambele cazuri dacă ceva nu merge, dă o eroare, atunci putem să începem totul de la început cît Sign, cît și Login.

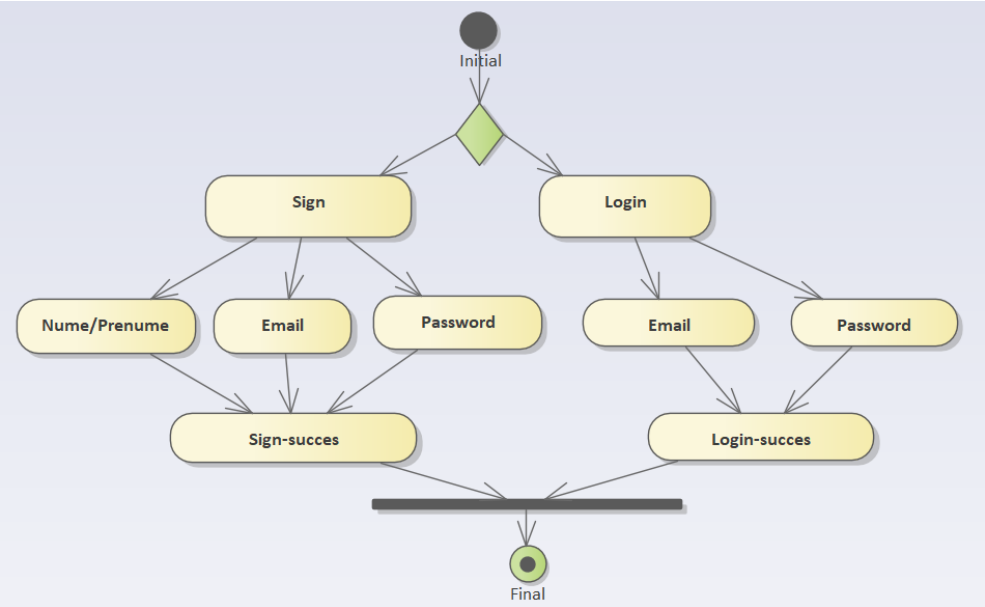

Figura 23 – Înregistrarea și Logarea

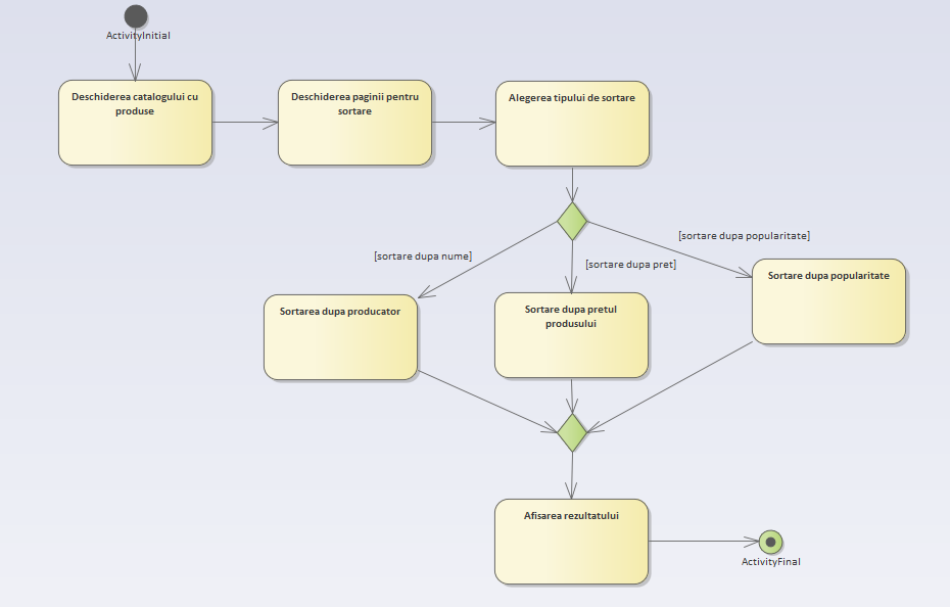

Figura 24 – Sortarea Produselor

În diagrama de stare reprezentată în Figura 24 este prezentată prima stare de activitate. Pentru sortarea Produselor, în primul rând trebuie să deschidem catalogul cu produse, apoi deschidem pagina unde se va efectua sortarea. După ce pagina a fost accesată, utilizatorul alege tipul de sortare (după nume, după preț sau după popularitatea produsului). În urma alegerii se va declanșa procesul de sortare dorit, apoi rezultatele vor fi afișate pe pagina de sortare deschisă.

Figura 25 – Înregistrarea la sală de sport

În diagrama de stare reprezentată în Figura 25 este prezentată accesarea catalogului de înregistrare la sală de sport. Pentru a ne înregistra la sală trebuie să îndeplinim cîțiva pași foarte importanți și primul din ei este alegerea antrenorului și domeniu de antrenament, după care ne vom alege graficul potrivit pentru noi. Și deja surplusurile dorite pe care le oferă sala de sport pentru a ne odihni după antrenament. După aceasta pe pagină vom avea două butoane, unul execută înregistrarea la sală și al doilea anulează toate setările făcute anterior. Și deja la urmă putem să selectăm din nou pentru a ne înregistra la sală sau să închidem aplicația.

2.1.6 Diagramelor de componente şi de plasare

Diagrama de componente permite determinarea arhitecturii sistemului elaborat prin stabilirea dependenţei între componentele de program în calitate de care poate fi codul iniţial, binar şi executabil. În mai multe domenii de elaborare modul şi componenta corespund fişierului. Săgeţile punctate care leagă modulele arată relaţiile de dependenţa analogice celor ce au loc la compilarea codurilor sursei iniţiale. Elementul grafic de bază al diagramei de componente sunt componentele, interfeţele şi dependenţele între ele.

Diagrama de componente se elaborează pentru urmatoarele scopuri:

- Vizualizarea structurii comune a codului sursă a unui sistem de program.
- Specificarea variantei executabile a unui sistem de program.

- Asigurarea utilizării repetate a unor fragmente ale codului sursă.
- Reprezentarea conceptuală şi fizică a schemelor bazei de date.

Diagrama de componente asigură trecere coordonată de la reprezentare logică spre o realizare a unui proiect în formă de cod sursă. Unele componente pot exista numai la etapa compilării codului sursei, altele – la etapa realizării lui. Diagrama de componente reflectă dependenţele între componente la cercetarea componentelor în calitate de clasificatori.

Node – element real al unui sistem care prezintă un mijloc de calcul cu un anumit volum de memorie și cu capacitatea de prelucrare a informației. Există în timpul funcționării unui produs soft.

Device – element de intrare/ieșire.

Există 3 tipuri de componente:

- regrupare – pagini web, biblioteci dinamice sau fișiere help;
- produs de lucru – fișierele document, code sursă;
- executare – fișierele executabile (.exe)

Stereotipurile:

- library – se referă la primul tip de componente, reprezintă bibliotecile dinamice sau statice;
- table – se referă la primul tip de componente, este componentul ce conține tabele de bază de date;
- file – se referă la al 2-lea tip, conține fișierul cu text inițial al codului;
- document – se referă la al 2-lea tip, reprezintă documente de tip txt, doc, xls;
- executable – se referă la al 3-lea tip, pentru fișiere de tip .exe.

Diagrama de plasare este specifică pentru vizualizarea elementelor şi componentelor a programului, ce există numai la etapa executării lui (runtime). În urma căruia sunt prezentate numai componente – exemplare a programului, care sunt fişiere de executare sau librăriile dinamice. Acele componente, care nu sunt utilizate la etapa executării, în diagrama de plasare nu sunt indicate. Componente cu texte iniţiale a programului pot fi numai în diagrama de componente. În diagrama de rezervare nu sunt indicate.

Diagrama de rezervare conţine reprezentarea grafică a procesorilor, echipamentelor, proceselor şi legăturilor între ele. În deosebire de diagrama reprezentării logice, diagrama de plasare este un sistem unic, deoarece trebuie să desfăşoare toate particularităţile realizării ei. Această diagramă finalizează procesul pentru un sistem concret de programare şi elaborarea ei este ultima etapa de specificarea a modelului.

Acopurile, ce sunt urmărite în timpul elaborării diagramei de plasare:

- De definit distribuirea componentelor a sistemului după nodurile fizice.
- De prezentat legăturile fizice între toate nodurile de realizare a sistemului la etapa de executare.

- De a găsi locurile înguste a sistemului şi de a reconfigura topologia ei pentru atingerea producerii necesare.
- Pentru garantarea cerinţelor diagramei de plasare se elaborează împreună cu analistele a sistemului, ingineri de reţele şi altele. Apoi vom examena elemente din care sunt conţinute diagramele de plasare.

Nodul (node) reprezintă un anumit element fizic a sitemului, care are o anumită resursă de calculare. Ca resursă de calculare a nodului poate fi o valoarea electronică sau magnitioptică a memoriei sau procesorului.

Pe lîngă reprezentarea nodurilor în diagrama de plasare sunt indicate relaţiile între ele. În calitate de relaţii sunt conectări fizice între noduri şi dependenţa între noduri şi componente, reprezentarea cărora poate fi în diagramele de plasare.

În afară de conectări în diagrama de plasare pot fi relaţiile de dependenţă între nod şi componentele lui. Acest caz este alternativ pentru reprezentarea componentelor depuse înăuntrul simbolului al nodului, ce nu este întodeauna comod, deoarece simbolul devine valoros. De aceea cînd există mulţimea de componente desfăşurate în nod, o informaţie respectivă poate fi reprezentată în fel de relaţie de dependenţă.

Diagrama de plasare poate avea o structură mai compusă, care include componentele, interfeţe şi alte echipamente. În diagrama de plasare mai jos este prezentat fragmentul reprezentării fizice a sistemului serviciului îndreptat pentru clienţii băncii. Nodurile acestui sistem sunt terminalul îndreptat (nodul - tip) şi serverul băncii (nodul - exemplar).[8]

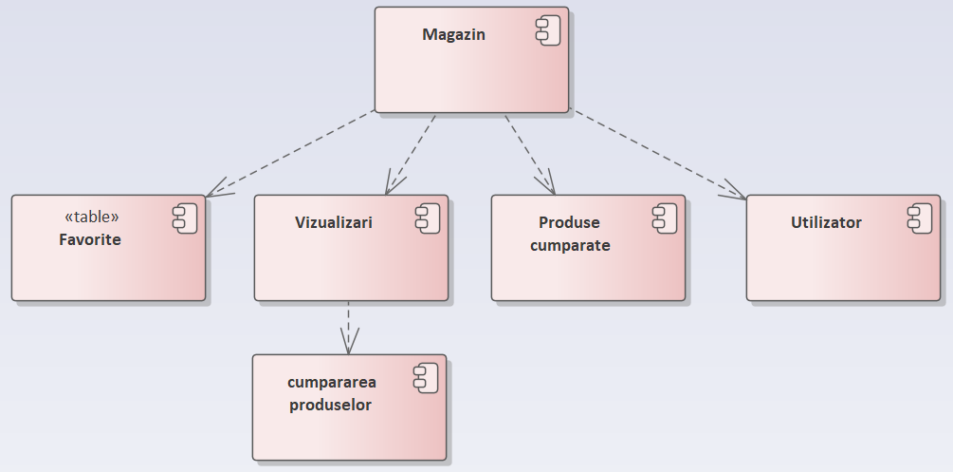

Figura 26 – Componentele magazinului

Pentru a descrie componentele unui magazin a aplicației am folosit diagrama de componente reprezentată în Figura 26. Mapa principală a aplicației conține componenta principala magazin, care conține aplicațiile interioare, care se divizează în 4 componente, și anume produse cumpărate deja, utilizator, adică login sau sign, favorite care li-am selectat și Vizualizări care mai are o componentă integrată în ea care se numește cumpărarea produselor.

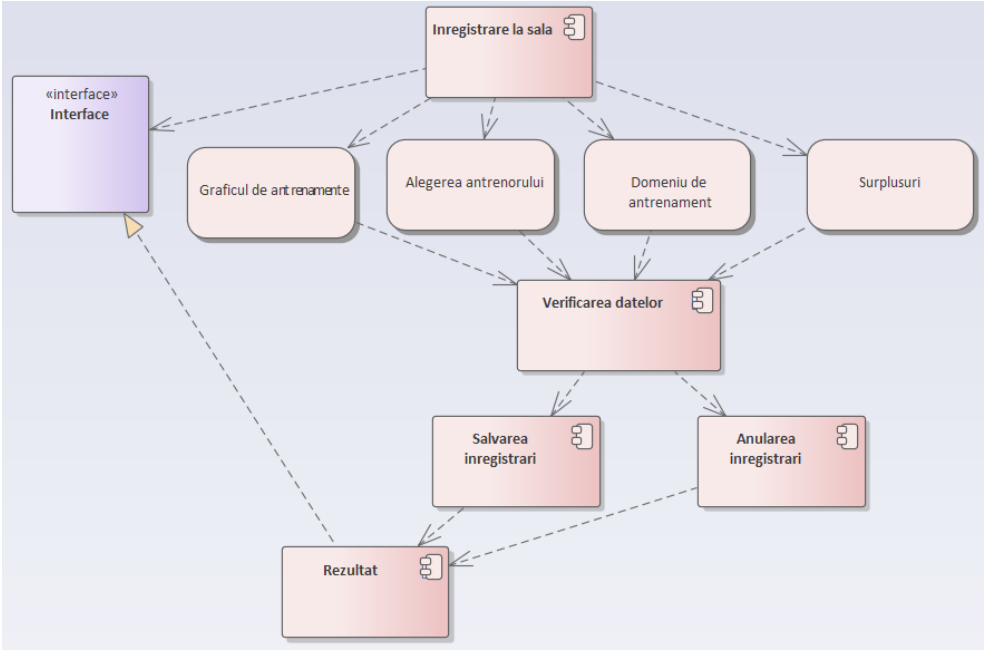

Figura 27 – Componentele pentru înregistrare la sală

Componentele principale ale paginii Înregistrare la sală pentru aplicație este demonstrată în Figura 27. Când intrăm pe pagina Înregistrare la sală principală, avem mai multe opțiuni, Interfața aplicației și încă 4 componente, care realizează alegerea pentru înregistrare la sală care sunt: Alegerea antrenorului, graficul de antrenament, domeniu de antrenament și surplusuri. După care ele se combină într-o componentă care verifică datele. Următorul pas este că avem două componente una salvează înregistrarea la sală și una care anulează selectările făcute anterior, unde rezultatul va venit în componenta de rezultate de la care se duce rezultatul prin linia de Realizarea către interfață.

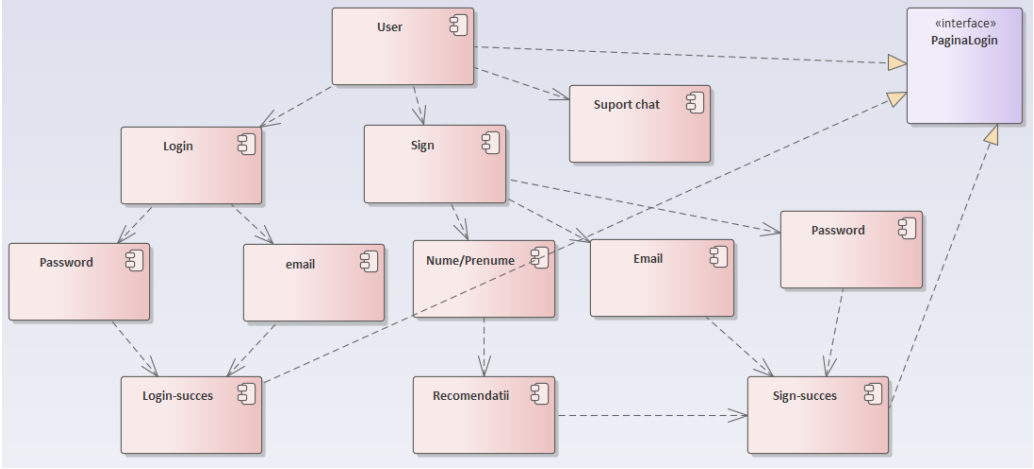

Figura 28 – Componentele principale pe pagina User a aplicației

Componentele principale ale paginii User pentru aplicație este demonstrată în Figura 28. Când intrăm pe pagina User principală, avem mai multe opțiuni, putem accesa pagina de Login sau de Sign, suport chat și interfața aplicației. Pagina Login se în parte în două diviziuni pe care trebuie să le completăm pe ambele Password și Email, după ele se combină într-o componentă care se numește Login-succes care se duce prin linie de Realizare la Interfață. Pagina Sign se înparte în 3 componente care trebuiesc îndeplinite

pentru a face înregistrarea, ele sunt: Nume/Prenume, Email, Password. Componenta Nume/Prenume mai are o componentă care ne va arăta recomandările și toate împreună se introduc în componența Sign-succes și prin linia de Realizare se duce către interfață. Și ultima componenta care se duce de la User se numește Suport Chat, pentru rezolvarea problemelor care apar la înregistrare sau logare.

Diagramele de plasare.

În diagrama de deplasare din figura 29 este prezentată interacțiunea dintre *SQL Services*, serverul bazei de date și dispozitivele Utilizatorilor care accesează *SQL Services.*

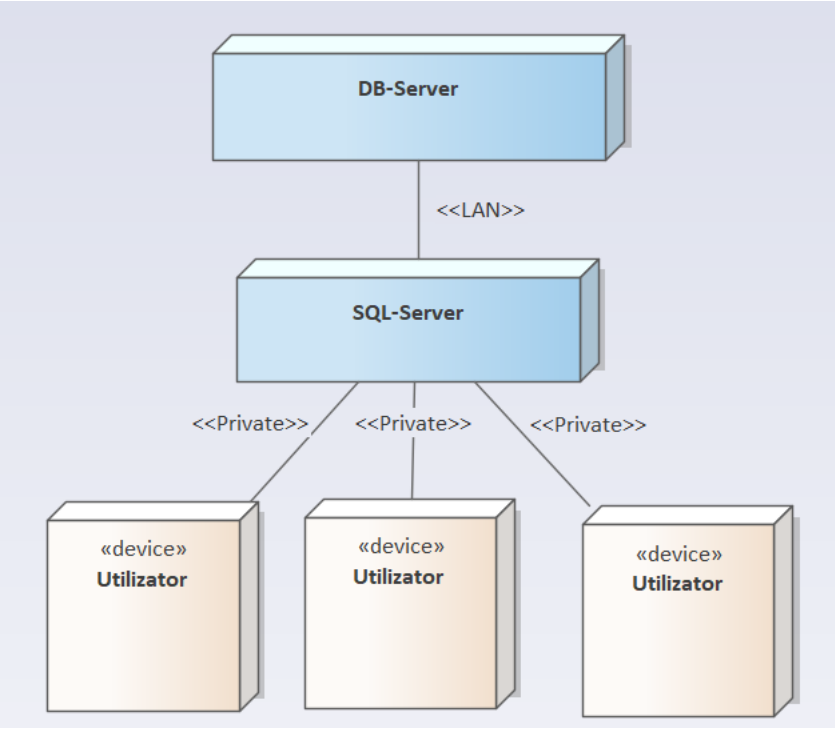

Figura 29 – Diagrama de plasare.

3 Prezentarea generală a web site-ului de sport

În versiunea finală sa primit site-ului, am realizat pagina de Home, About, Courses, Gallery, Blog, înregistrare/logare, Contact me. Navigarea între aceste pagini se realizează bine, prin butoanele care sunt aplasate deasupra sau la sfîrșit de pagină.

Vizualizarea rezultatului:

Pagina principală ne face cunoștință cu site-ul crea, ne spune care este denumirea site-ului, ne arată tematica lui și bara de meniu, care ne spune ce pagini există pe site-ul dat.

Figura 29 – Pagina principală.

Pagina About Me este creată pentru informația despre o persoană care se antrenează și dorește să se înpartă cu detalii despre antrenamentele sale, sau cu sfaturi.

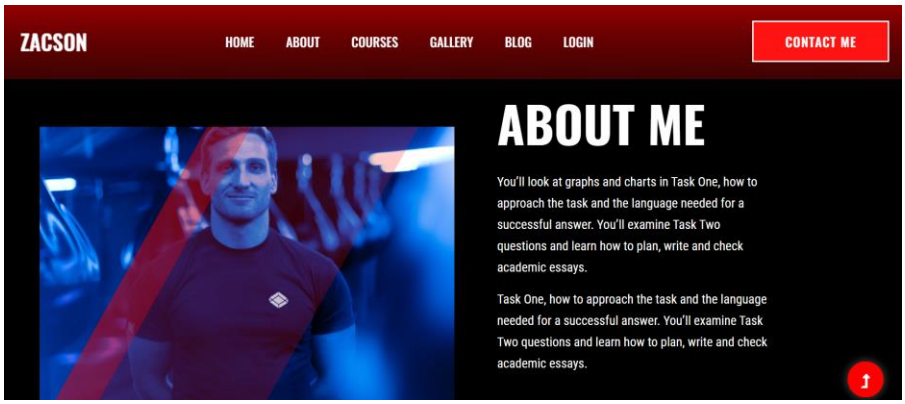

Figura 30 – Pagina Despre mine.

Pagina Courses este alcătuită din o mulțime de cursuri pe care se poate de înregistrat pentru fregventare.

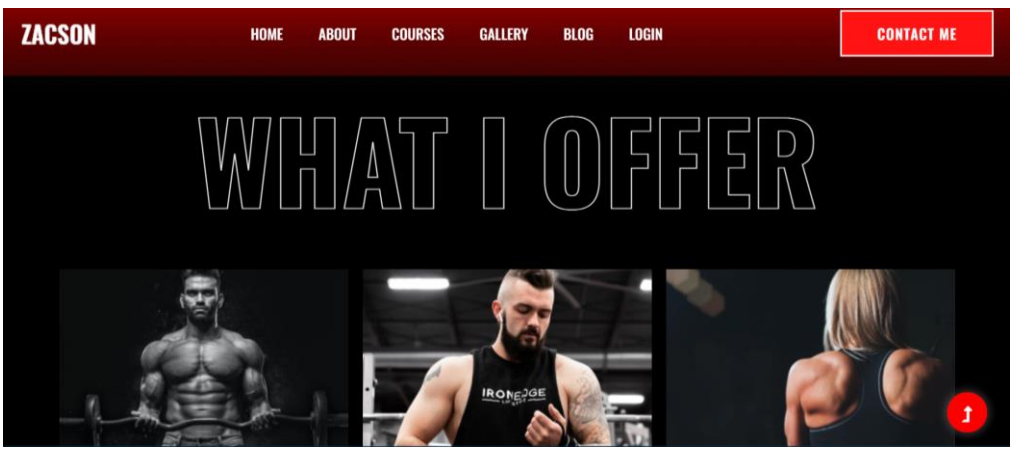

Figura 31 – Pagina Cursurilor.

Pe pagina Gallery este reprezentat multe imagini a persoanelor în catrul antrenamentelor sale în salele de sport.

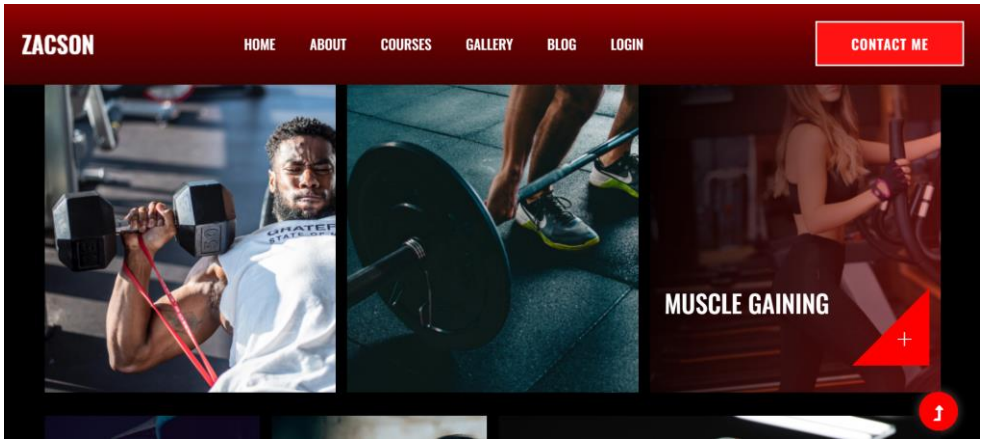

Figura 32 – Gallery.

Pagina Blog se ocupă cu informarea și discuția între persoane pe diferite teme sportive. Ca de exemplu ce rație trebuie sa menții, ce tipuri de grafic este mai potrivit și multe altele.

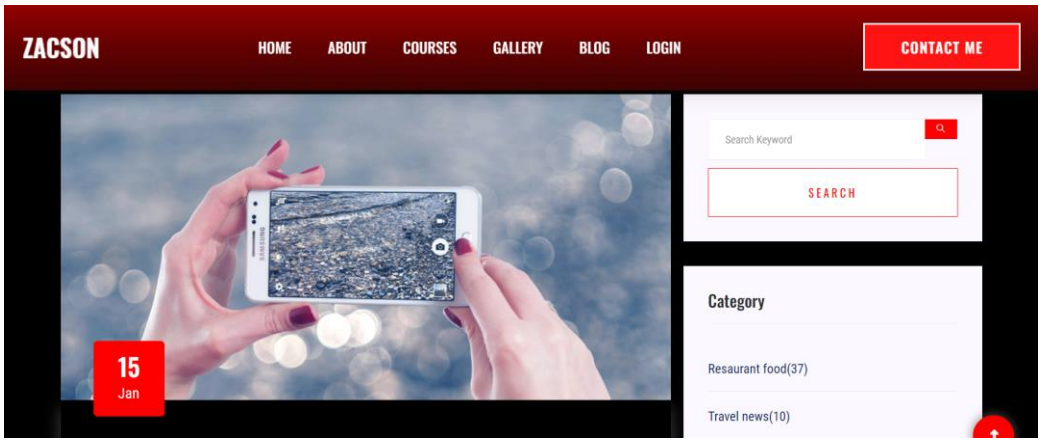

Figura 33 – Pagina Blog.

Pentru a contacta administratorul întrați pe pagina Contact Me pentru a putea scrie un mesaj.

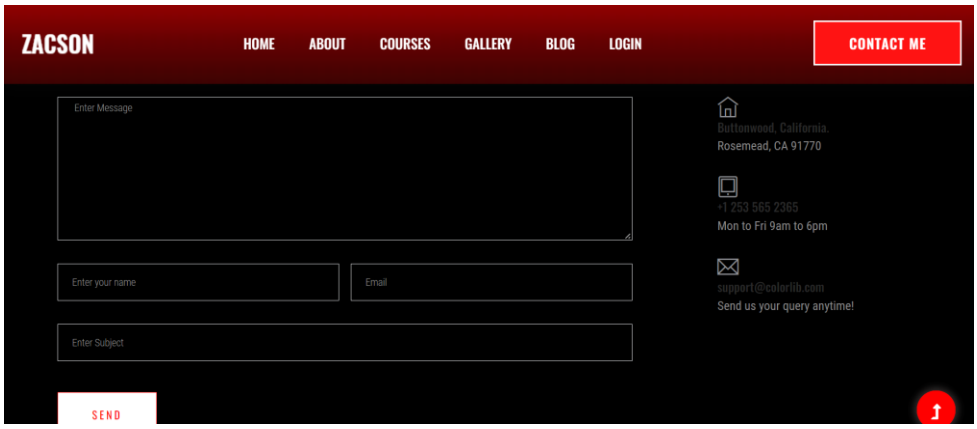

Figura 34 – Pagina de Contact.

Pentru a ne putea înregistra pe site întrăm pe pagina login și alegem Sign up și Login pentru a ne putea loga pe site.

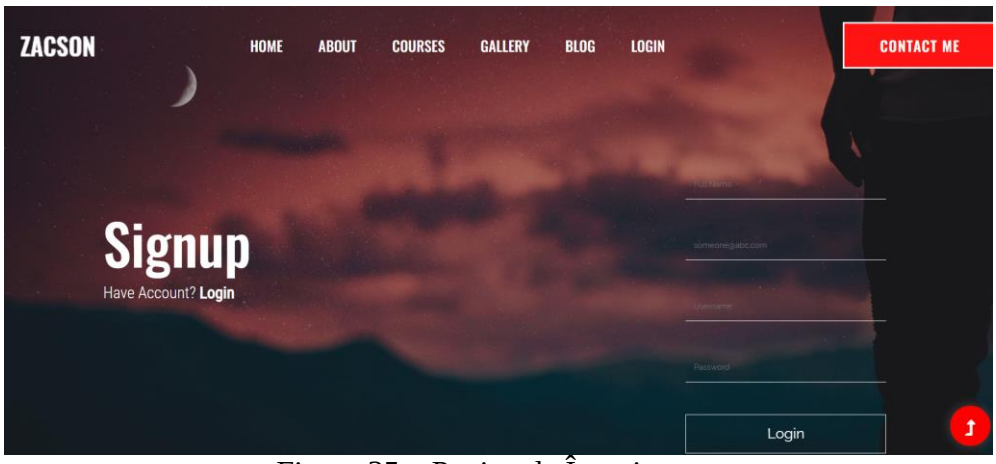

Figura 35 – Pagina de Înregistrare.

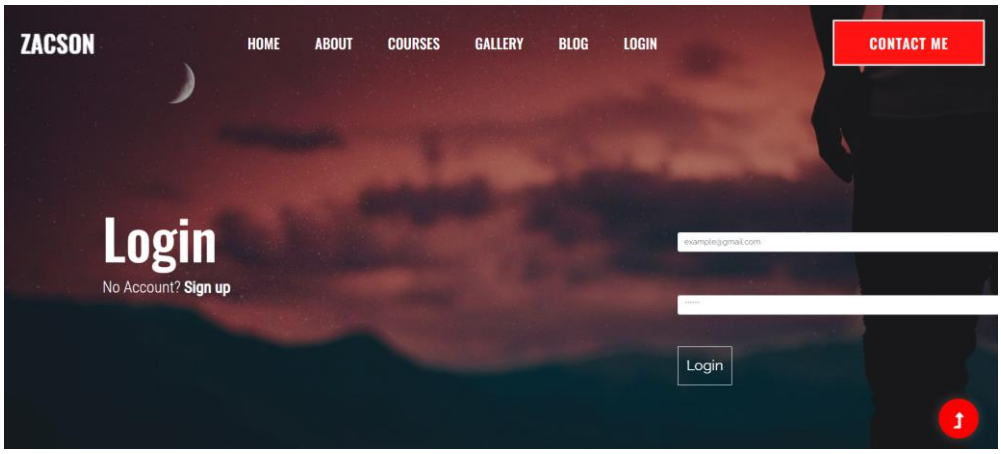

Figura 36 – Pagina de Login.

Concluzie

În această lucrare am studiat tipurile de diagrame din limbajul UML, am elaborat aceste diagrame la tema „Analiza si modelarea unei aplicatii care va gestiona necesitatile unui club de sport”. În realizarea sarcinii propuse m-a ajutat foarte mult conspectul de la cursul de AMOO, însa am utilizat și surse adiționale cum ar fi tutoriale, literatura adăugătoare utilizatînd instrumentul „Enterprise Architect”.

În concluzie, scopul UML poate fi definit ca un mecanism simplu de modelare pentru a modela toate sistemele practice posibile în mediul complex de astăzi.

Una dintre condiţiile ce trebuie îndeplinite ca un proiect să aibă succes este aceea ca cerinţele proiectului să fie definite într-o manieră care să permită o uşoară înţelegere, indiferent de nivelul de pregătire informatică al celui care este implicat în proiect. De asemenea, modificările ce apar pe parcurs în cerinţe trebuie să fie cu uşurinţă asimilate de către membrii echipei de dezvoltare.

Fazele de dezvoltare a unui sistem informatic prin limbajul UML:

Analiza cerintelor - are ca principal rol definirea claselor, obiectelor, mecanismelor specifice procesului modelat. Clasele vor fi stabilite în paralel cu relaţiile lor specifice şi descrise în cadrul diagramei claselor, în timp ce relaţiile dintre ele vor fi surprinse în modelele dinamice UML.

Analiza si abstractizarea - Această fază are ca principal rol definirea claselor, obiectelor, mecanismelor specifice procesului modelat. Clasele vor fi stabilite în paralel cu relaţiile lor specifice şi descrise în cadrul diagramei claselor, în timp ce relaţiile dintre ele vor fi surprinse în modelele dinamice UML.

Design - Faza de design asigură transferarea rezultatelor din faza anterioară către o soluţie tehnică. Vor fi adăugate noi clase în infrastructura tehnică: interfaţa utilizator, dirijarea BD în vederea memorării datelor în BDOO, comunicaţia cu alte sisteme şi interfaţa cu echipamentele hardware.

Programarea sau constructia - Prin intermediul fazei de programare, clasele obţinute la faza de design sunt transformate în cod sursă prin intermediul unui limbaj de programare orientat obiect.

Testarea - Sunt efectuate mai multe tipuri de testări:

- teste pe unitate - au loc individual asupra claselor sau grupurilor de clase şi sunt realizate exclusiv de către programatori. Sunt utilizate diagramele claselor şi specificaţiile de clasă;
- teste integratoare – vizează integrările componentelor şi claselor în ordinea în care acestea cooperează. Sunt utilizate diagramele componentelor şi de colaborare;
- testele sistemului realizat – sunt aplicate asupra unei „cutii negre” pentru a valida sistemul şi a verifica cooperarea acestuia cu utilizatorii;
- testele de acceptare - permit verificarea de către clientul și agradului în care acesta satisface cerinţele specificate în prima fază a dezvoltării aplicaţiei. Sunt utilizate diagramele cazurilor deutilizare (use-case).

Bibliografie:

1. Analiză și proiectare orientată pe obiecte (OOAD). [Resursă electronică], accesat la data 16.04.2022. – Regim de acces: https://cybarlab.com/ooad
2. Analiza și proiectarea sistemelor orientate pe obiecte. [Resursă electronică], accesat la data 16.04.2022. – Regim de acces: https://www.w3computing.com/systemsanalysis/object-oriented-systems-analysis-design/
3. Ghidul simplu pentru diagrame UML [Resursă electronică], accesat la data 17.04.2022. – Regim de acces: https://www.microsoft.com/ro-ro/microsoft-365/business-insights-ideas/resources/guide-to-uml-diagramming-and-database-modeling
4. Caracteristicile generale ale limbajului UML. [Resursă electronică], accesat la data 17.04.2022. – Regim de acces: https://mkr-novo2.ru/ro/installation-and-configuration/obshchaya-harakteristika-yazyka-uml-osnovy-unificirovannogo-yazyka-modelirovaniya.html
5. Tipuri de diagrame. [Resursă electronică], accesat la data 18.04.2022. – Regim de acces: https://personal.ucoz.net/index/diagrame_uml/0-26
6. MODELAREA SISTEMELOR INFORMATICE [Resursă electronică], accesat la data 18.04.2022. – Regim de acces: https://www.scritub.com/stiinta/informatica/MODELAREA-SISTEMELOR-INFORMATI12121131520.php
7. Proiectarea obiectuală a sistemelor informatice [Resursă electronică], accesat la data 18.04.2022. – Regim de acces: https://www.academia.edu/5211344/Proiectarea_obiectual%C4%83_a_sistemelor_informatice
8. AMOO Radu Melnic Partea_1_RO [resursă electronică], accesat la data de 22.04.2022