Ministerul Educaţiei, Culturii și Cercetării
Universitatea Tehnică a Moldovei

Faculatea Calculatoare, informatică și microelectronică
Departamentul Ingineria Software și Automatică

RAPORT

Lucrare de Laborator nr.3
Disciplina: Analiza și Specificarea Cerințelor Software
Tema : Validarea cerințelor

A efectuat:
st. gr. TI-204 Valciuc Andrei

A verificat:
asist. univ. Crîjanovschi Adriana

Chişinău 2023

Cuprins:

INTRODUCERE

În era tehnologică în continuă evoluție, preocuparea pentru bunăstarea animalelor de companie a devenit o prioritate constantă. Pentru a răspunde acestei nevoi, s-a dezvoltat un sistem inovator de monitorizare a stării animalelor de companie, orientat spre îndeplinirea unor obiective esențiale. În cadrul acestui proiect, accentul principal îl reprezintă identificarea, modelarea și validarea cerințelor funcționale, fundamentale pentru funcționarea optimă a sistemului. Acest proces detaliat de definire a cerințelor asigură o înțelegere profundă a nevoilor utilizatorilor și a contextului în care sistemul va fi implementat.

În realizarea acestui proiect, accentul pe identificarea și modelarea cerințelor funcționale evidențiază necesitatea unei abordări cuprinzătoare. Definirea clară a funcționalităților sistemului, de la monitorizarea sănătății animalelor și până la facilitarea comunicării cu stăpânii, reprezintă elementele cheie care contribuie la o soluție coezivă și orientată către nevoile utilizatorilor.

În paralel, se acordă o atenție deosebită identificării și modelării fluxului de date, element esențial pentru funcționarea corectă a proiectului. Monitorizarea stării animalelor de companie impune colectarea, procesarea și interpretarea eficientă a datelor, iar o înțelegere detaliată a fluxului acestora contribuie la dezvoltarea unei soluții robuste și eficiente.

Pe lângă aspectele funcționale, proiectul acordă o importanță deosebită identificării cerințelor nonfuncționale. Eficiența, securitatea și scalabilitatea sistemului sunt printre prioritățile majore, asigurându-se astfel că soluția implementată nu numai îndeplinește cerințele specifice, dar și se adaptează la nevoile în schimbare ale utilizatorilor și ale mediului înconjurător.

Astfel, această inițiativă reprezintă nu doar o evoluție în monitorizarea stării animalelor de companie, ci și un angajament ferm pentru dezvoltarea unei soluții tehnologice avansate și adaptabile, menite să îmbunătățească semnificativ calitatea vieții animalelor de companie și a stăpânilor lor.

1. IDENTIFICAREA, MODELAREA ȘI VALIDAREA CERINȚELOR FUNCȚIONALE

În procesul de dezvoltare a unui proiect, identificarea, modelarea și validarea cerințelor funcționale reprezintă etape cruciale, fundamentale pentru fundamentarea și implementarea unei soluții coezive și eficiente. În primul rând, identificarea cerințelor funcționale presupune o analiză atentă a nevoilor și așteptărilor utilizatorilor finali. Această fază presupune interacțiuni strânse cu toți actorii implicați pentru a obține o înțelegere profundă a contextului și a obiectivelor proiectului.

Modelarea cerințelor funcționale reprezintă o etapă ulterioară, unde se transformă informațiile obținute într-o structură coerentă și comprehensivă. Această abordare implică definirea clară a funcționalităților sistemului, stabilirea relațiilor dintre acestea și conturarea modului în care acestea vor satisface nevoile utilizatorilor. Modelarea oferă o hartă detaliată a modului în care proiectul va funcționa și cum va adresa cerințele specifice identificate anterior.

Validarea cerințelor funcționale este o etapă critica pentru a se asigura că soluția propusă îndeplinește cu succes obiectivele stabilite. Acest proces presupune testarea fiecărei cerințe în parte pentru a verifica dacă aceasta este bine definită, fezabilă și relevanta în contextul general al proiectului. Validarea contribuie la eliminarea ambiguităților sau a interpretărilor greșite și, în cele din urmă, asigură că soluția propusă va oferi funcționalități conforme cu așteptările inițiale.

În această fază, eforturile se concentrează asupra identificării detaliilor specifice privind funcționalitățile necesare ale sistemului, având în vedere nevoile variate ale utilizatorilor și scopul proiectului. Comunicarea și colaborarea strânsă cu toate părțile interesate sunt esențiale pentru a obține o viziune completă și echilibrată a cerințelor funcționale. Această abordare nu se limitează doar la solicitarea directă de informații, ci implică și observații, analize comparative și studii de caz pentru a identifica nevoile latente și a anticipa cerințele viitoare.

Modelarea ulterioară a cerințelor funcționale presupune transformarea informațiilor adunate într-o structură coerentă și organizată. Această etapă implică definirea detaliată a fiecărei funcționalități, precum și a relațiilor și interdependențelor dintre acestea. Modelarea contribuie la clarificarea cerințelor și la stabilirea unui cadru solid pentru implementarea lor ulterioară în cadrul proiectului.

Validarea cerințelor funcționale reprezintă o etapă crucială pentru asigurarea coerenței și relevanței acestora. Procesul de validare implică verificarea riguroasă a fiecărei cerințe în parte, asigurându-se că aceasta este realizabilă din punct de vedere tehnic, fezabilă în cadrul constrângerilor proiectului și, nu în ultimul rând, că răspunde la necesitățile reale ale utilizatorilor. Testele și simulările sunt adesea utilizate pentru a verifica performanța și eficiența cerințelor, iar feedback-ul primit este integrat într-un proces iterativ de îmbunătățire continuă.

Prin aceste etape, procesul impersonal de identificare, modelare și validare a cerințelor funcționale devine un angrenaj esențial în mașinăria proiectului, asigurându-se că soluția propusă nu numai îndeplinește așteptările inițiale, ci și se adaptează la schimbările inevitabile în timpul dezvoltării și implementării. Este un proces continuu, reflexiv și orientat către furnizarea unei soluții durabile și eficiente pe termen lung. Prin urmare, identificarea, modelarea și validarea cerințelor funcționale sunt procese interconectate ce conduc la stabilirea unui temei solide pentru dezvoltarea și implementarea proiectului. Aceste etape, deși desfășurate în mod impersonal, reflectă angajamentul față de calitate, eficiență și relevanță în procesul de proiectare și dezvoltare a soluției propuse.

Diagramele Entity-Relationship (ERD) reprezintă un aspect esențial al proiectării și dezvoltării bazelor de date în domeniul sistemelor informatice. Ele oferă o modalitate vizuală de a conceptualiza structura informațională a unui sistem, precum și modul în care entitățile din acest sistem interacționează între ele. Aceste diagrame sunt utilizate pe scară largă în procesul de proiectare a bazelor de date, având rolul de a facilita înțelegerea complexității structurii datelor și a relațiilor dintre acestea. Entitățile, reprezentate sub forma dreptunghiurilor în diagramele ERD, pot fi diverse, cum ar fi clienți, produse, angajați sau orice alt obiect sau concept care trebuie să fie gestionat într-o bază de date. Atributele, reprezentate sub formă de ovale asociate entităților, descriu caracteristicile sau proprietățile acestora. Prin conexiunile și relațiile definite între entități, se evidențiază modul în care datele sunt stocate și interconectate în cadrul sistemului. Un aspect crucial al diagramei ERD este reprezentarea cheilor primare și a cheilor străine. Cheile primare identifică în mod unic o entitate, iar cheile străine stabilesc legăturile între entități, oferind o modalitate eficientă de a gestiona și accesa datele în mod coerent. Utilitatea principală a diagramei ERD constă în furnizarea unei viziuni cuprinzătoare și accesibile a arhitecturii datelor unui sistem. Aceste diagrame sunt esențiale în comunicarea între membrii echipei de dezvoltare, cu clienții și alte părți interesate, ajutând la clarificarea și alinierea înțelegerii asupra cerințelor proiectului.

În final, diagramele ERD contribuie la optimizarea performanței sistemului prin evidențierea relațiilor și interdependențelor dintre entități. Ele oferă un cadru solid pentru proiectarea și implementarea unei baze de date eficiente, adaptate nevoilor specifice ale aplicației sau sistemului dezvoltat.

Diagrama include cinci entități principale: Proprietar (Owner), Animal de companie (Pet), Vizită la veterinar (VetVisit), Medic veterinar (Veterinarian) și Diagnostic (Diagnosis). Putem observa diagrama creată în figura de mai jos:

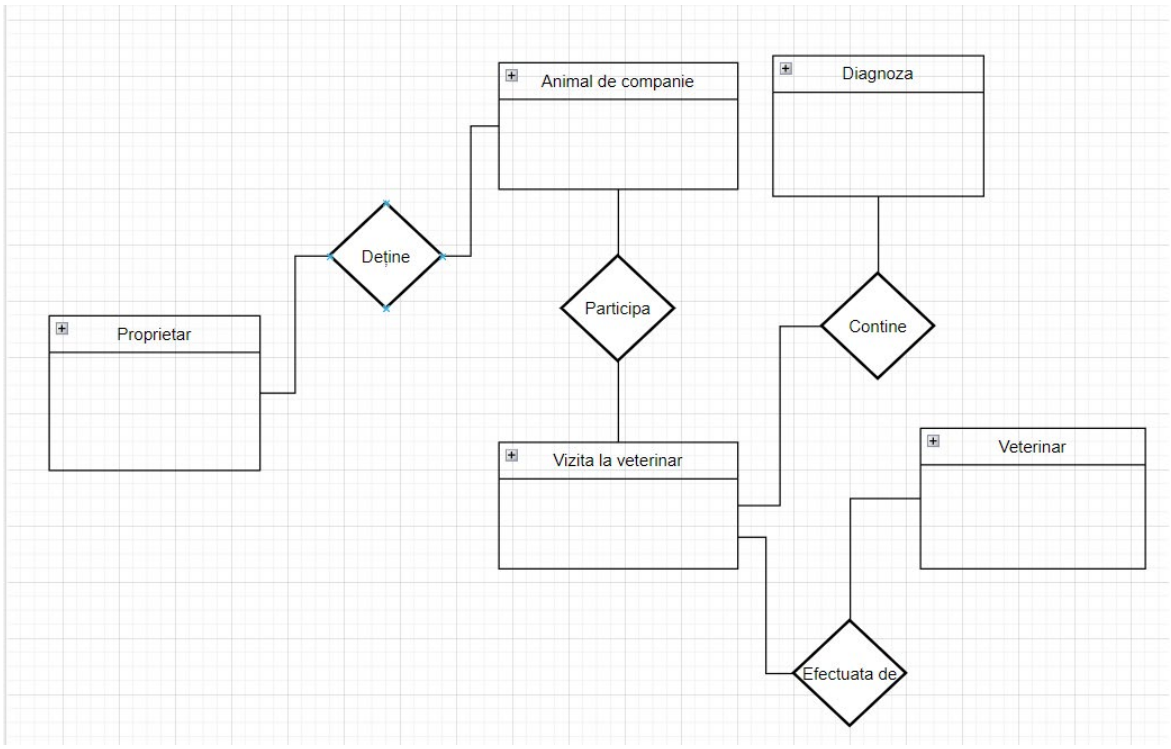

Figura 1.1 – Diagrama ERD brută a sistemului

Relațiile dintre aceste entități sunt stabilite în următorul mod:

Proprietarul este asociat cu Animalul de companie printr-o relație de "Deținere". Astfel, un proprietar poate deține zero sau mai multe animale de companie, în timp ce fiecare animal de companie este deținut de un singur proprietar. Animalul de companie este legat de Vizita la veterinar printr-o relație de "Participare". Un animal de companie poate participa la zero sau mai multe vizite la veterinar, în timp ce fiecare vizită la veterinar este asociată cu un singur animal de companie. Vizita la veterinar este conectată la Medicul veterinar printr-o relație "Efectuare de către". Astfel, o vizită la veterinar este efectuată de un singur medic veterinar, iar un medic veterinar poate efectua zero sau mai multe vizite la veterinar. Vizita la veterinar este, de asemenea, asociată cu Diagnosticul prin intermediul unei relații "Conține". O vizită la veterinar poate conține zero sau mai multe diagnoze, iar fiecare diagnostic este legat de o singură vizită la veterinar.

Această diagramă oferă o perspectivă globală asupra interacțiunilor între proprietari, animale de companie, vizite la veterinar, medici veterinari și diagnostice în cadrul sistemului de monitorizare a sănătății animalelor de companie.

Completarea diagramei ERD cu informații despre cardinalitate este esențială pentru a defini clar modul în care entitățile din sistem interacționează. Aceste detalii furnizează un ghid asupra numărului și modului în care entitățile sunt legate între ele, oferind astfel o structură coerentă și bine definită a relațiilor în cadrul aplicației de monitorizare a sănătății animalelor de companie.Cardinalitatea ajută la stabilirea limitelor și a dependențelor între entități, contribuind la evitarea ambiguităților și la clarificarea rolurilor pe care le joacă fiecare entitate în cadrul sistemului. De exemplu, cardinalitatea specifică câți proprietari pot deține câte animale de companie sau câte diagnoze pot fi asociate cu o singură vizită la veterinar. Prin adăugarea acestor detalii, diagrama ERD devine o unealtă mai puternică pentru proiectarea și înțelegerea structurii bazei de date, oferind un cadru coerent pentru dezvoltarea și întreținerea sistemului de monitorizare a stării animalelor de companie. Astfel, cardinalitatea devine un aspect crucial în asigurarea coerenței și eficienței operaționale a aplicației. În figura de mai jos este reprezentată diagrama completată cu cardinalitate:

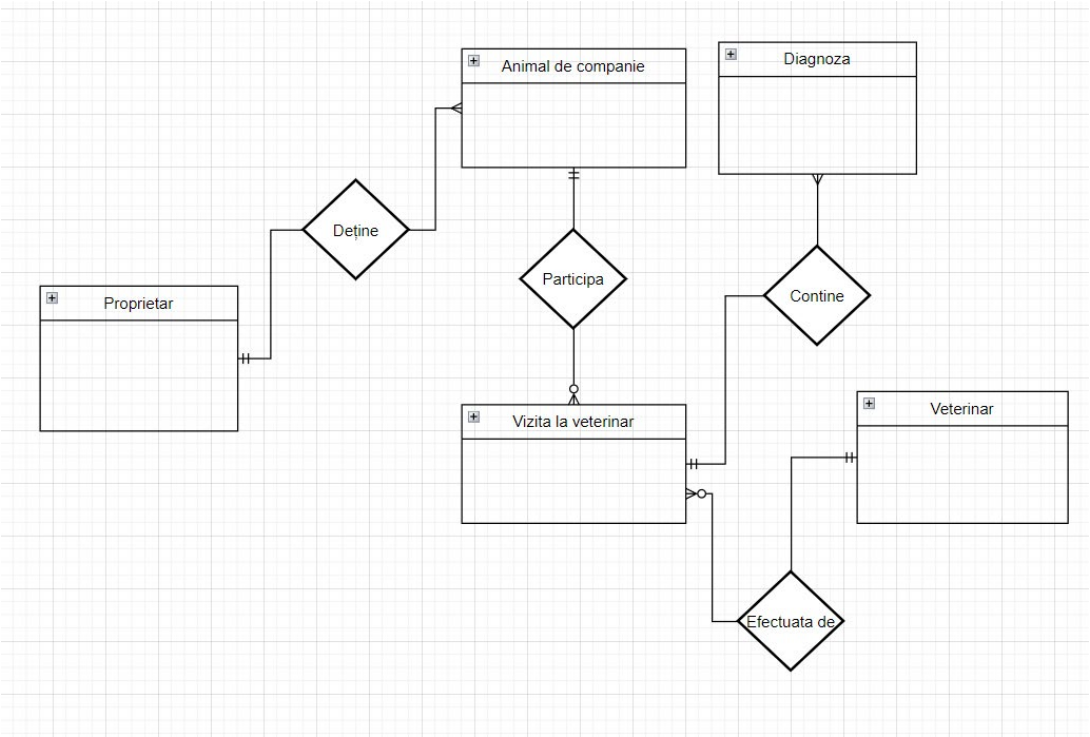

Figura 1.2 – Diagrama ERD cu cardinalitatea completă

În contextul diagramei ERD pentru aplicația de monitorizare a stării animalelor de companie, cardinalitatea oferă claritate cu privire la relațiile dintre entități. Această informație esențială subliniază gradul de dependență și conectivitate dintre diversele componente ale sistemului. De exemplu, cardinalitatea dintre entitățile "Proprietar" și "Animal de companie" indică faptul că un proprietar poate fi asociat cu zero sau mai multe animale de companie, în timp ce fiecare animal de companie are exact un proprietar. Această relație clarifică modul în care proprietarii sunt legați de animalele de companie pe

care le dețin. Similar, în relația dintre "Animal de companie" și "Vizită la veterinar", cardinalitatea arată că un animal de companie poate participa la zero sau mai multe vizite la veterinar, dar fiecare vizită la veterinar este asociată cu un singur animal de companie. Aceasta conturează modul în care informațiile despre vizitele la veterinar sunt legate de fiecare animal de companie în parte. În ansamblu, completarea diagramei ERD cu detalii privind cardinalitatea contribuie la o înțelegere mai profundă a modului în care entitățile interacționează, furnizând un cadru clar pentru proiectarea și gestionarea datelor în cadrul aplicației.

Atributele adaugă un nivel de detaliu esențial pentru fiecare entitate, definind proprietățile specifice care vor fi stocate și gestionate în cadrul sistemului. De exemplu, în cazul entității "Proprietar", atributele precum "Nume", "Prenume", "Adresă" și "Număr de telefon" sunt vitale pentru a identifica și diferența proprietarii în sistem.Aceste atribute oferă, de asemenea, informații despre caracteristicile animalelor de companie ("Animal de companie"), vizitelor la veterinar ("Vizită la veterinar"), medicii veterinari ("Medic veterinar") și diagnozele asociate vizitelor la veterinar ("Diagnostic"). Ele reprezintă esența datelor manipulate de aplicație și furnizează o bază solidă pentru gestionarea eficientă a informațiilor.Prin completarea diagramei ERD cu atribute, se obține o viziune cuprinzătoare asupra structurii și complexității datelor, facilitând astfel procesele de dezvoltare, întreținere și analiză a sistemului. În figura de mai jos putem observa diagrama ERD completată cu atribute:

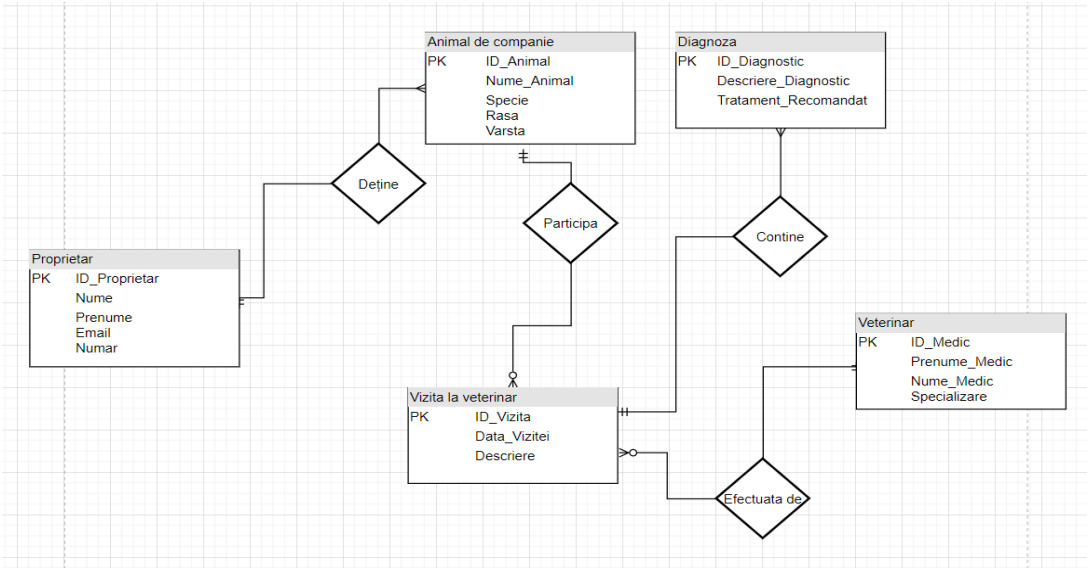

Figura 1.3 – Diagrama ERD completată cu atribute

Diagrama ERD completată cu atribute oferă o imagine detaliată și comprehensivă a modului în care datele sunt structurate și relaționate în cadrul aplicației de monitorizare a stării animalelor de companie.

Entitate: Proprietar

- ID_Proprietar (cheie primară)
- Nume

- Prenume
- Adresă
- Număr de telefon
- Email

Entitate: Animal de companie

- ID_Animal (cheie primară)
- Nume_Animal
- Specie
- Rasă
- Vârstă
- Sex
- Data_nașterii
- ID_Proprietar (cheie străină referențiată către ID_Proprietar în entitatea Proprietar)

Entitate: Vizită la veterinar

- ID_Vizita (cheie primară)
- Data_Vizitei
- Descriere_Problema
- ID_Animal (cheie străină referențiată către ID_Animal în entitatea Animal de companie)

Entitate: Medic veterinar

- ID_Medic (cheie primară)
- Nume_Medic
- Prenume_Medic
- Specializare

Entitate: Diagnostic

- ID_Diagnostic (cheie primară)
- Descriere_Diagnostic
- Tratament_Recomandat
- ID_Vizita (cheie străină referențiată către ID_Vizita în entitatea Vizită la veterinar)

Aceste atribute adaugă detalii esențiale la fiecare entitate, definind specificațiile pentru fiecare categorie de date. Prin intermediul cheilor primare și străine, se stabilesc relațiile între entități, facilitând astfel manipularea și interconectarea datelor în cadrul sistemului de gestionare a stării animalelor de companie.

2. IDENTIFICAREA ȘI MODELAREA FLUXULUI DE DATE

Diagramele DFD (Diagrama Fluxurilor de Date) sunt o modalitate grafică de a reprezenta procesele, fluxurile de date și depozitele de date într-un sistem informatic. Aceste diagrame sunt utilizate pentru a modela și descrie modul în care informațiile circulă într-un sistem, evidențiind interacțiunile dintre diferitele sale componente.

În general, diagramele DFD sunt compuse din procese, fluxuri de date și depozite de date. Procesele reprezintă activitățile sau operațiunile care transformă intrările (fluxurile de date) în ieșiri (fluxuri de date). Fluxurile de date sunt căile pe care informațiile călătoresc între procese, iar depozitele de date sunt locurile în care datele sunt stocate și recuperate. Diagramele DFD sunt folosite pentru a oferi o perspectivă macroscopică asupra sistemelor, evidențiind modul în care datele sunt prelucrate și cum circulă prin diversele componente ale sistemului. Acestea sunt utile în fazele incipiente ale dezvoltării software pentru a înțelege cerințele sistemului și pentru a comunica aceste informații între membrii echipei de dezvoltare și clienți.

Prin intermediul diagramei DFD, proiectanții de sisteme și dezvoltatorii pot obține o înțelegere vizuală a logicii de funcționare a unui sistem, contribuind la identificarea posibilelor puncte de congestie, a necesităților de securitate și a altor aspecte relevante. Aceste diagrame sunt utile pentru a evalua arhitectura generală a unui sistem și pentru a planifica etapele ulterioare ale dezvoltării software, inclusiv detaliile proceselor individuale și interfețele utilizatorilor. Prin utilizarea diagramei DFD, echipele de dezvoltare pot identifica clar fluxurile de date între diferitele componente ale unui sistem, precum și interdependențele și relațiile dintre acestea. Aceasta facilitează comunicarea și colaborarea între membrii echipei, ajutând la alinierea asupra cerințelor și a funcționalităților esențiale ale sistemului. De asemenea, diagramele DFD sunt utile pentru a evidenția modul în care datele sunt colectate, prelucrate și stocate în cadrul unui sistem. Aceasta poate conduce la identificarea posibilelor vulnerabilități de securitate sau a punctelor critice care ar putea necesita atenție sporită în procesul de dezvoltare.

Pe măsură ce proiectul avansează, diagramele DFD pot evolua pentru a include niveluri de detaliu mai ridicate, detaliind subprocesoarele, interacțiunile complexe și dependențele mai fine ale sistemului. Aceste actualizări permit echipei să abordeze aspecte mai specifice și să asigure coerența între conceptul inițial și implementarea practică a sistemului.

Diagrama DFD (Diagrama Fluxurilor de Date) de nivel 0 este o reprezentare grafică a unui sistem, care evidențiază procesele majore, fluxurile principale de date și depozitele de date la un nivel de abstractizare înalt. Această diagramă furnizează o privire de ansamblu asupra sistemului, identificând elementele cheie și interacțiunile dintre acestea. Diagrama DFD de nivel 0 oferă o privire de ansamblu clară asupra sistemului, identificând modul în care datele intră și ies din sistem, precum și procesele cheie care transformă aceste date. Este adesea utilizată în fazele incipiente ale dezvoltării software sau în procesul de analiză și proiectare pentru a stabili o bază conceptuală pentru sistem. Pe măsură ce proiectul avansează, diagramele de nivel inferior (nivel 1, nivel 2 etc.) pot fi create pentru a detalia fiecare proces și interacțiune în parte. În figura de mai jos putem observa diagrama DFD de nivel 0 pentru sistemul de monitorizare a stării animalelor de companie:

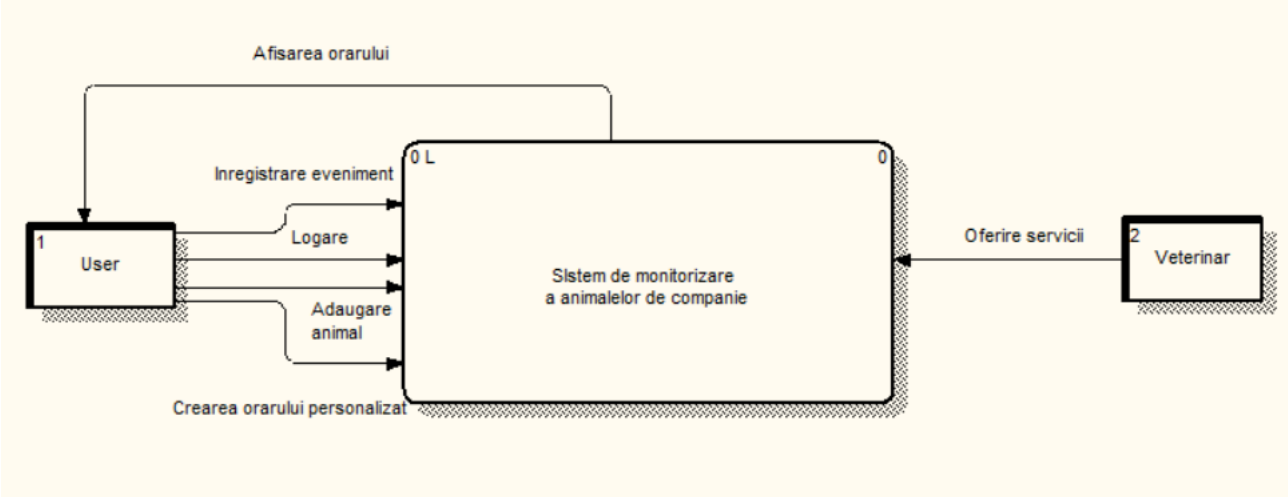

Figura 2.1 Diagrama DFD de nivel 0

În diagrama dată putem observa fluxul de date la un nivel de abstractizare înalt. În cazul dat utilizatorul este capabil să se logheze/înregistreze, să adauge animal de companie, să înregistreze un eveniment legat de anumit animal și să creeze un orar personalizat creat din o serie de evenimente înregistrate. Veterinarul prin sistemul dat oferă servicii iar sistemul al rândul său afișează răspunsul.

La nivelul unei diagrame DFD de nivel 1, se oferă o detaliere mai profundă a proceselor identificate la nivelul 0. Această diagrame evidențiază subprocesele și fluxurile de date care apar în interiorul fiecărui proces de nivel superior, precizând mai multe aspecte ale funcționării sistemului fără a intra în detalii exhaustive. Ea servește ca un pas intermediar între diagrama de nivel 0 și cele de nivel inferior, oferind o viziune mai detaliată asupra modului în care datele sunt prelucrate în cadrul fiecărui proces identificat. Această abordare a nivelului 1 este esențială pentru a îndeplini obiectivele și detaliile necesare ale proiectului. În figura 2.2 este reprezentată diagrama DFD de nievl 1 pentru sistemul nostru

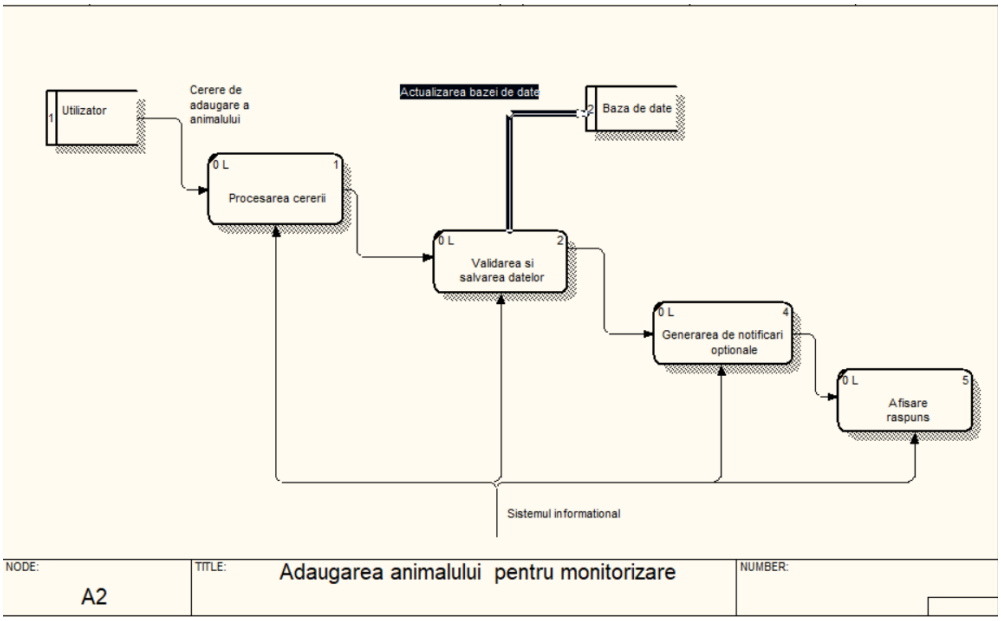

Figura 2.2 – Diagrama DFD de nivelul 1

În această diagramă este reprezentat procesul de adaugare a animalului de companie de către utilizator. Deci are loc trimiterea cererii de adăugare de către user a animalului, are loc procesarea cererii după care validarea și actualizarea bazei de date. În următorul pas sunt generate notificări personalizate opționale și afișarea răspunsului.

La nivelul unei diagrame DFD de nivel 2, detaliile sistemului sunt extinse încă o dată pentru a evidenția subprocesele și fluxurile de date în interiorul proceselor identificate la nivelul 1. Această diagrame oferă o viziune mai detaliată asupra modului în care fiecare proces major de nivel superior este descompus în subprocesoare și cum datele circulă între acestea.

Subprocesele individuale din cadrul unui proces de nivel 1 sunt reprezentate în această diagramă, iar fluxurile de date care le conectează evidențiază modul în care datele sunt prelucrate și transformate la

un nivel mai granular. Depozitele de date asociate cu subprocesele pot fi, de asemenea, incluse pentru a arăta modul în care datele sunt stocate temporar.

Această diagrame DFD de nivel 2 este utilă pentru a oferi o înțelegere detaliată a funcționării interne a fiecărui proces de nivel 1, fără a se abate în mod excesiv în detaliile operaționale. Este utilă în fazele ulterioare ale dezvoltării software, atunci când este necesar să se exploreze în profunzime activitățile și fluxurile de date din cadrul fiecărui proces identificat la nivelurile superioare.

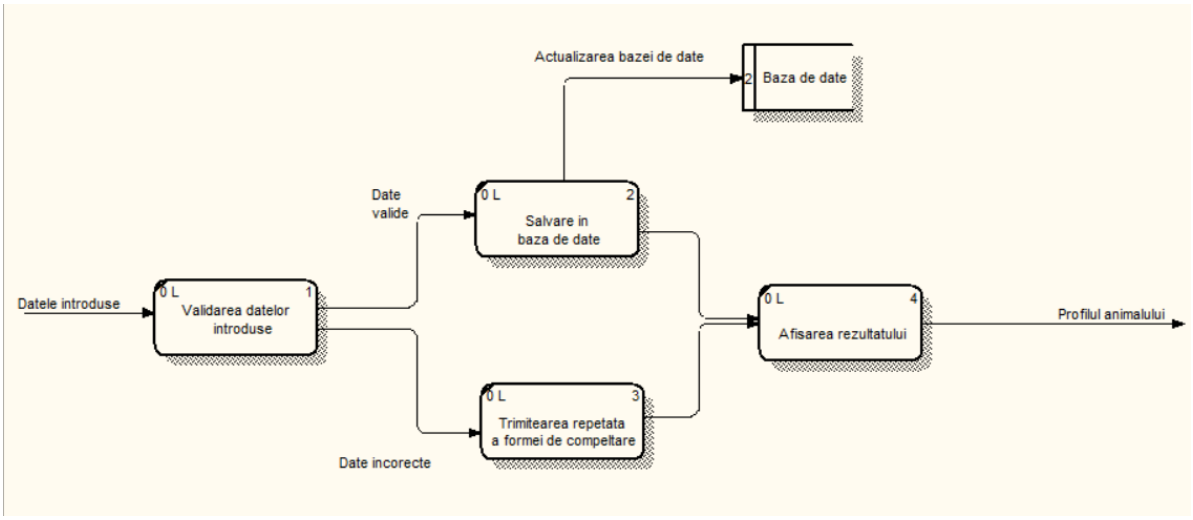

Figura 2.3 – Diagrama DFD de nivelul 2

În figura de mai sus putem observa diagrama DFD de nivelul 2 pentru procesul de validare a datelor. Fluxul de date de la utilizator trece prin procesul de validare conform regulilor de utilizare a platformei. Dacă datele sunt valide atunci are loc procesul de salvare și actualizare a bazei de date. În caz contrar are loc transmiterea repetată a formei de completare. În final este afișat rezultatul.

3. Identificarea cerințelor non-funcționale

Cerintele nefunctionale reprezinta un aspect esential in dezvoltarea si proiectarea oricarui sistem software sau produs IT. In contrast cu cerintele functionale, care se concentreaza pe ce anume trebuie sa faca sistemul, cerintele nefunctionale se refera la modul in care sistemul trebuie sa functioneze. Aceste cerinte reprezinta criterii de performanta, securitate, scalabilitate, usurinta in utilizare, si multe altele, care afecteaza experienta utilizatorului final si eficienta sistemului in sine.

Cerintele nefunctionale pot varia considerabil in functie de tipul de proiect si de industrie. Ele pot include aspecte precum timpul de raspuns al aplicatiei, disponibilitatea sistemului, securitatea datelor, interoperabilitatea cu alte sisteme, costurile de mentenanta si multe altele. Aceste cerinte sunt adesea mai dificil de definit si masurat decat cerintele functionale, dar sunt la fel de importante pentru succesul unui proiect. În acest context, gestionarea cerintelor nefunctionale devine cruciala pentru echipele de dezvoltare. Ele trebuie sa le identifice, sa le prioritizeze si sa le integreze in procesul de dezvoltare pentru a asigura ca produsul final atinge nivelul de performanta, securitate si calitate dorit. Neglijarea acestor cerinte poate duce la esecul unui proiect sau la nevoia de a face modificari costisitoare in fazele ulterioare ale dezvoltarii.Pe măsură ce proiectul avansează, este important să se monitorizeze și să se măsoare constant conformitatea cu aceste cerințe. De exemplu, în ceea ce privește performanța unei aplicații web, trebuie să se monitorizeze timpul de încărcare al paginilor și să se asigure că se încadrează în limitele stabilite. În cazul securității, se pot efectua teste de penetrare sau audituri pentru a verifica dacă datele sunt protejate corespunzător.Un alt aspect important este gestionarea schimbărilor. De-a lungul vieții unui proiect, cerințele nefuncționale pot evolua sau se pot schimba în funcție de feedbackul utilizatorilor, noile tehnologii sau alte factori. Este vital să se aibă un proces bine definit pentru a gestiona aceste schimbări, pentru a nu afecta calitatea și performanța produsului final.

Prin urmare, cerințele nefuncționale joacă un rol esențial în dezvoltarea de succes a oricărui produs software sau sistem IT. Ele influențează experiența utilizatorului, securitatea și performanța produsului și nu trebuie ignorate sau subestimate. Prin gestionarea atentă a acestor cerințe pe parcursul întregului ciclu de dezvoltare, echipele pot crea produse de înaltă calitate care să satisfacă nevoile utilizatorilor și să atingă obiectivele proiectului.

Cerințele non-funcționale, cunoscute și sub numele de cerințe de performanță sau cerințe de calitate, reprezintă o componentă esențială a specificațiilor în dezvoltarea software-ului. Acestea se concentrează asupra caracteristicilor sistemului care nu țin de funcționalitatea directă a acestuia, ci se referă la modul în care sistemul trebuie să se comporte și să ofere o experiență de utilizare de înaltă calitate. Deși nu definesc comportamentul principal al aplicației, ele influențează puternic modul în care utilizatorii percep și interacționează cu sistemul, performanța sa generală și durabilitatea în timp. Iată o definiție mai detaliată și o dezvoltare a semnificației cerințelor non-funcționale:

Cerințele non-funcționale sunt specificații care descriu calitățile, caracteristicile și constrângerile sistemului în dezvoltarea software-ului, cu excepția funcționalității directe. Ele se referă la "cum" ar trebui să se comporte un sistem, mai degrabă decât la "ce" ar trebui să facă. Aceste cerințe acoperă o gamă largă de aspecte, inclusiv performanța, fiabilitatea, securitatea, scalabilitatea, usabilitatea și multe altele. Ele sunt adesea critice pentru succesul unui produs software și pot avea un impact semnificativ asupra satisfacției utilizatorilor, costurilor de întreținere și imagini organizaționale.

Aceste cerințe sunt la fel de importante ca și cerințele funcționale, poate chiar mai importante. Ele sunt spre deosebire de cerințele funcționale, care descriu ceea ce face de fapt un sistem sau un produs și răspunsul acestuia la intrări și acțiuni. Aceste cerințe sunt vitale pentru succesul sistemelor software. Dacă nu sunt abordate corespunzător, apar rezultate nedorite, cum ar fi utilizatori, dezvoltatori și clienți nesatisfăcuți și depășiri de planificare și buget pentru a corecta software-ul care a fost dezvoltat fără cerințele nefuncționale în minte. Terminologia inconsecventă, definițiile confuze și absența unei scheme de clasificare universal acceptate fac ca înțelegerea cerințelor nefuncționale să fie o provocare. Cerință nefuncțională – o specificație a cât de bine trebuie să funcționeze un sistem software.

Cerințele non-funcționale pot aborda următoarele aspecte cum ar fi:

- preformanța;
- fiabilitatea;
- securitatea;
- scalabilitatea;
- compatibilitatea;
- mentenabilitatea;6-

Cerințele non-funcționale sunt de o importanță vitală pentru succesul oricărui proiect software, deoarece ele determină modul în care un sistem oferă performanță, calitate și experiență utilizator. Neglijarea acestor cerințe poate duce la nemulțumirea utilizatorilor, vulnerabilități de securitate și costuri suplimentare pentru remedierea problemelor. Prin urmare, aceste cerințe necesită o gestionare atentă și un echilibru adecvat în dezvoltarea software-ului.

3.1 Descrierea cerințelor non-funcționale

Cerințele non-funcționale, cum ar fi performanța, fiabilitatea, securitatea, scalabilitatea, compatibilitatea și mentenabilitatea, sunt cruciale pentru proiectarea și dezvoltarea software-ului de succes. Iată o descriere a fiecărei cerințe non-funcționale:

Performanța: Performanța se referă la capacitatea software-ului de a răspunde rapid și eficient la cerințele utilizatorilor. Aceasta poate include timpul de răspuns, viteza de încărcare a paginilor, eficiența utilizării resurselor hardware (cum ar fi CPU și memorie), și capacitatea de a face față volumului mare de date sau utilizatori. Performanța bună este esențială pentru a oferi o experiență de utilizare fluidă și pentru a satisface așteptările utilizatorilor.

Fiabilitatea: Fiabilitatea se referă la capacitatea software-ului de a funcționa în mod constant și de a evita erorile sau căderile neașteptate. Un sistem fiabil trebuie să fie robust și să poată gestiona erori și situații de excepție fără a afecta operațiunile critice. Fiabilitatea este crucială în aplicații care nu își permit căderi sau indisponibilități.

Securitatea: Securitatea se referă la protejarea sistemului și a datelor împotriva accesului neautorizat, a atacurilor cibernetice și a vulnerabilităților. Cerințele de securitate pot include autentificarea utilizatorilor, criptarea datelor, gestionarea acceselor, protecția împotriva malware-ului și a altor amenințări. Asigurarea securității este esențială pentru a proteja datele și confidențialitatea utilizatorilor.

Scalabilitatea: Scalabilitatea se referă la capacitatea software-ului de a crește sau de a se adapta la schimbări în cerințele de volum sau de utilizare. Un sistem scalabil poate crește fără a suferi degradări semnificative ale performanței. Scalabilitatea este importantă pentru a face față creșterii numărului de utilizatori, a datelor sau a cerințelor noi fără a necesita modificări majore ale infrastructurii sau a codului.

Compatibilitatea: Compatibilitatea se referă la capacitatea software-ului de a funcționa corect pe diferite platforme, dispozitive sau sisteme de operare. Acest lucru asigură că utilizatorii pot accesa și utiliza aplicația pe o gamă largă de medii și dispozitive. Compatibilitatea este importantă pentru a ajunge la un public mai mare și pentru a evita frustrarea utilizatorilor.

Mentenabilitatea: Mentenabilitatea se referă la ușurința cu care software-ul poate fi întreținut, actualizat și extins. Un cod bine documentat, modular și ușor de înțeles facilitează gestionarea schimbărilor și remedierea problemelor. Mentenabilitatea este importantă pentru a reduce costurile de întreținere și pentru a menține software-ul relevant pe termen lung.

Toate aceste cerințe non-funcționale sunt interconectate și contribuie la dezvoltarea unui software de calitate, care să ofere o experiență pozitivă utilizatorilor și să îndeplinească standardele de performanță, fiabilitate, securitate, scalabilitate, compatibilitate și mentenabilitate

CONCLUZII

În concluzie, în procesul de proiectare și dezvoltare a unui sistem, identificarea, modelarea și validarea cerințelor funcționale reprezintă un pas esențial pentru a asigura că sistemul îndeplinește scopul și funcțiile preconizate. Aceste cerințe oferă o structură clară a comportamentului sistemului, definind funcționalitățile sale cheie și modul în care acestea trebuie să interacționeze.

Identificarea și modelarea fluxului de date constituie un alt aspect crucial, contribuind la realizarea unei diagrame DFD care ilustrează modul în care informațiile circulă prin sistem. Aceasta furnizează o viziune de ansamblu asupra proceselor, fluxurilor de date și depozitelor de date, facilitând înțelegerea modului în care datele sunt prelucrate și gestionate. Pe lângă cerințele funcționale, identificarea cerințelor non-funcționale adaugă o dimensiune importantă proiectului. Aceste cerințe acoperă aspecte precum performanța, securitatea, scalabilitatea și altele, care sunt esențiale pentru a asigura calitatea și fiabilitatea sistemului pe termen lung. Prin abordarea acestor obiective, se creează un cadru solid pentru proiectarea și implementarea unui sistem care să îndeplinească atât cerințele funcționale, cât și cele non-funcționale. Identificarea și modelarea fluxului de date, împreună cu validarea cerințelor funcționale, conduc la o mai bună înțelegere a modului în care sistemul va opera și cum vor interacționa diferitele sale componente. Această abordare riguroasă asigură că proiectul respectă cerințele clienților și ale utilizatorilor finali, contribuind la succesul și eficacitatea sistemului în funcțiune. În plus, identificarea cerințelor non-funcționale aduce în prim-plan aspecte critice ale sistemului care pot influența experiența utilizatorului, performanța și securitatea acestuia. Abordarea acestor cerințe într-un mod sistematic contribuie la crearea unui mediu robust și adaptabil, capabil să răspundă nevoilor variate ale utilizatorilor și să facă față unor provocări diverse. Procesul de identificare, modelare și validare a cerințelor, în special în contextul fluxului de date, nu doar asigură funcționalitatea corectă a sistemului, ci și eficiența și integritatea acestuia. O diagrame DFD bine realizată servește ca instrument vizual puternic pentru echipele de dezvoltare, facilitând comunicarea și înțelegerea proceselor, indiferent de complexitatea sistemului.

Prin urmare, abordarea sistematică a identificării cerințelor funcționale și non-funcționale, precum și modelarea fluxului de date, este fundamentală pentru succesul unui proiect. Aceste eforturi colective aduc claritate în procesul de dezvoltare a sistemului, permitând echipei să răspundă eficient la cerințele clienților și să creeze soluții software robuste și adaptabile într-o varietate de medii și scenarii.