ROMÂNIA
MINISTERUL APĂRĂRII NAȚIONALE
ACADEMIA TEHNICĂ MILITARĂ
„FERDINAND I”

FACULTATEA DE SISTEME INFORMATICE ȘI SECURITATE CIBERNETICĂ
Specializarea: Calculatoare și sisteme informatice pentru apărare și securitate națională

Implicații ale realocării resurselor asupra fezabilității ordonării activităților

CONDUCĂTOR ȘTIINȚIFIC:

CS III dr. Dana-Mihaela VÎLCU

ABSOLVENT:

Std. Sg. Maj. Mihai-Valentin BADEA

București

-2022-

Abstract

This paper presents a software solution for the Resource-Constrained Project Scheduling Problem. Despite the various variants of the problem which were researched, I implemented the solution for the basic RCPSP with some updates which allowed me to create an application that can be used in real business environment. The main feature of this application is represented by the automated scheduling of the tasks if there is a feasible scheduler. The schedule is determined by an algorithm which considers the duration, the resource usage and the precedence relations between tasks. This make the difference between my application and other task management applications which allows users only to make a “to do” list without the posibility to offer a variant of scheduling.

Rezumat

Această lucrare prezintă o soluție software realizată pentru a rezolva problema ordonării activităților din cadrul proiectelor cu constrângeri pe resurse. În procesul de dezvoltare a aplicației s-au studiat atât problema ordonării activităților în cadrul proiectelor cu constrângeri pe resurse, cât și modul în care task-urile sunt ordonate în sistemele de operare. Astfel, în definirea noțiunii de activitate, s-a realizat o paralelă cu noțiunea de *task* din cadrul teoriei sistemelor de operare. Aplicația implementează un algoritm ce testează fezabilitatea unui set de activități dat, algoritm propus pentru rezolvarea problemei RCPSP standard. Acestui algoritm i-au fost aduse modificări specifice utilizării la nivel organizațional în managementul de proiecte. Diferența majoră dintre aplicația propusă și alte aplicații de Project management constă în faptul că acestea din urmă oferă, de regulă, doar posibilitatea vizualizării planificărilor și posibilitatea ordonării manuale a activităților, pe când aplicația noastră permite o replanificare automată cu respectarea constrângerilor de resurse.

Cuprins

Listă de figuri

Listă de abrevieri

1. RCPSP – Resource-Constrained Project Scheduling Problem
2. PMI – Project Management Institute
3. JVM – Java Virtual Machine
4. UI – User Interface
5. MVC – Model-View-Controller
6. JSON – JavaScript Object Notation
7. TLS – Transport Layer Security

1. Introducere

Lucrarea de față își propune studierea unor modalități de tratare a situațiilor de realocare a resurselor din cadrul managementului activităților proiectelor. De asemenea, în baza studiului efectuat a fost implementată o aplicație prin intermediul căreia managerii de proiecte să observe consecințele produse de realocări ale resurselor și modul în care activitățile vor fi planificate ulterior, precum și sugestii de măsuri care pot fi întreprinse pentru cazul în care realocarea produce ordonări nefezabile a activităților.

1.1. Prezentare generală

Dezvoltarea oricărei organizații, instituții sau companii stă în modalitatea în care acestea își manageriază activitățile. Fie că vorbim despre proiecte ale multinaționalelor în care sunt angrenați sute sau chiar mii de angajați sau despre startup-uri ce au un număr restrâns de angajați, organizarea activității fiecărui membru în parte este extrem de importantă pentru o productivitate crescută. Informații suplimentare privind activitățile, așa cum sunt ele avute în vedere în cadrul managementului de proiecte, sunt prezentate în Capitolul 2. De asemenea, în cursul executării oricărei activități sunt necesare una sau mai multe resurse pe care cel ce are de realizat respectiva activitate le va folosi. Astfel, este extrem de important ca aceste resurse să fie utilizate într-un mod cât mai eficient. Teoria ce modelează această problemă a ordonării activităților cu constrângeri pe resurse este prezentată în cadrul Capitolului 4. Pentru a putea realiza un management al activităților cât mai eficace, va trebui, pentru început, să ne raportăm la un domeniu în care această activitate de ordonare este una principală. Vom lua ca referință modalitatea de ordonare a task-urilor din cadrul teoriei sistemelor de operare prin studierea a ceea ce reprezintă un task și a algoritmilor de ordonare multiprocesor. Aceste noțiuni vor fi prezentate pe larg în Capitolul 3. Următorul pas va fi să observăm asemănările ce pot exista între noțiunea de task din cadrul sistemelor de operare și activitățile din cadrul managementului de proiecte. În final vom analiza modul în care am putea transpune aceste noțiuni și în cadrul activității de Project management, bazându-ne pe asemănările ce există între cele două domenii. Prezentarea implementării aplicației este prezentată în Capitolul 5.

1.2. Utilitatea sistemului

Soluția software propusă în această lucrare oferă, la nivel de organizație, posibilitatea ordonării automate a activităților pe care acestea le au de îndeplinit, în

funcție de resursele de care dispun. De asemenea, prin intermediul interfeței grafice, se poate monitoriza utilizarea resurselor si modalitatea în care sunt organizate activitățile fiecărui angajat.

Fiecărui angajat care se autentifică în aplicație i se oferă posibilitatea de a vizualiza activitățile pe care acesta le are de realizat în următoarea perioadă. În cazul în care se impune, acesta poate realiza realocări ale resurselor în cadrul proiectelor în care este deja asignat sau poate adăuga noi activități. În ambele cazuri, aceste acțiuni vor duce în mod implicit la apelarea algoritmului de ordonare ce va determina dacă există o nouă soluție fezabilă pentru datele actuale.

În cazul aplicației propuse vom considera că o ordonare rezultată este fezabilă în cazul în care toate activitățile au putut fi ordonate în ziua în care acestea au fost planificate fără a fi încălcate constrângerile date de relațiile de precedență sau de numărul limitat de resurse disponibile. Astfel, un set fezabil de activități va fi considerat acela pentru care se poate realiza o ordonare fezabilă. Resursele ce sunt luate în calcul în cadrul aplicației sunt: resursa fizică, resursa umană, dar și timpul. Aplicația descrisă în capitolele următoare a fost prezentată în cadrul *Students' International Conference "CERC"* [14].

2. Definirea managementului de proiecte

„Activitatea de Project Management reprezintă utilizarea anumitor cunoștințe, abilități, instrumente și tehnici având ca scop final livrarea a ceva de valoare către client” [8]. Acest domeniu este extrem de diversificat deoarece se poate implementa în toate ariile de activitate. Pentru a sublinia acest aspect, pot fi prezentate cu titlul de exemplu următoarele proiecte: dezvoltarea unui software pentru o întreprindere, construcția unei clădiri sau extinderea vânzărilor într-o nouă piață geografică.

Pentru a înțelege managementul proiectelor, vom începe printr-o scurtă definiție a ceea ce reprezintă un proiect. Astfel, conform lucrării [6], un proiect poate fi considerat ca având o serie de activități care:

- au un obiectiv specific ce se concentrează pe adaosul de valoare asupra organizației,
- au definite intervale de timp în care trebuie realizate,
- au anumite limitări,
- necesită o cantitate de resurse pentru a putea fi îndeplinite (umane sau materiale precum: bani, echipament, licențe etc.)

Rezultatul proiectelor poate fi unic (ex. crearea unei noi aplicații ce nu există pe piață) sau repetitiv (ex. oferim servicii de mentenanță ce pot fi oferite și de alte companii), însă acestea trebuie să fie atinse într-o perioadă finită de timp. Deoarece companiile au la dispoziție un număr limitat de resurse, împărțirea acestora în cadrul proiectelor trebuie realizată într-un mod cât mai eficient. Project Management Institute (PMI) identifică în lucrarea *“A Guide to the Project Management Body of Knowledge”* (PMBOK Guide) [9] un număr de cinci grupuri de procese ce au loc în cadrul organizării proiectelor:

1. Inițierea proiectului
    a. Selectarea celor mai bune proiecte având în vedere limitele definite de numărul resurselor
    b. Listarea beneficiilor aduse de către proiect
    c. Pregătirea documentației proiectului
    d. Atribuirea unui Project manager
2. Planificarea proiectului
    a. Definirea cerințelor
    b. Definirea calității și cantității de muncă
    c. Definirea resurselor necesare
    d. Ordonarea activităților
    e. Evaluarea riscurilor

3. Executarea proiectului
    a. Atribuirea membrilor ce vor lua parte la proiect
    b. Coordonarea activităților
    c. Lucrul cu membrii echipei pentru îmbunătățirea muncii
4. Monitorizarea si controlul proiectului
    a. Urmărirea evoluției proiectului
    b. Compararea rezultatelor obținute cu ceea ce se dorea inițial
    c. Analiza variațiilor și a impactului rezultat
    d. Ajustarea detaliilor
5. Finalizarea proiectului
    a. Verificarea faptului că toate activitățile au fost completate
    b. Finalizarea contractului
    c. Remunerația
    d. Finalizarea documentelor aferente muncii depuse

„Un management de succes al proiectelor se poate considera acela ce reușește să obțină o serie de obiective atinse în timp util, fără a depăși costul cerut, utilizând nivelul de tehnologie dorit în timp ce resursele sunt utilizate cât mai eficient, având ca scop final acceptarea de către client a produsului final” [6]. Deoarece fiecare proiect are diferite cerințe venite din partea clientului, activitățile incluse în fiecare grup de procese definit mai sus poate diferi de la proiect la proiect. *PMBOK Guide* [9] identifică diferite metode de împărțire ale acestor activități în funcție de industria în care sunt utilizate. Astfel, acestea sunt considerate bune practici pentru industriile respective și pot fi structurate pentru a crea metodologii de management de proiecte ce vor sta la baza oricărui proiect. Desigur, acestea pot fi personalizate pentru o varietate de proiecte.

O serie de beneficii pe care un management de proiecte eficient le are este următoarea:

- identificarea clară a responsabilităților funcționale pentru a asigura faptul că toate activitățile sunt contabilizate,
- minimizarea necesității unui raport continuu,
- identificarea limitelor de timp pentru ordonarea activităților,
- măsurarea a ceea ce a fost realizat față de ceea ce a fost planificat,
- identificarea prealabilă a problemelor astfel încât acestea pot fi corectate,
- îmbunătățirea capacității de estimare pentru proiectele viitoare

În concluzie, pentru definirea unui proiect în cadrul aplicației se va ține cont de faptul că acesta se va desfășura pe un interval determinat de timp și va conține un număr limitat de resurse.

3. Ordonarea task-urilor în cadrul sistemelor de operare

În capitolul acesta vor fi prezentate noțiuni teoretice ce stau la baza ordonării task-urilor în cadrul sistemelor de operare, prezentându-se astfel principiile și algoritmii folosiți. După ce vom analiza modalitatea în care sistemelor de operare realizează ordonarea task-urilor, vom scoate în evidență caracteristicile comune cu activitățile din cadrul Project Management-ului. Următoarele informații teoretice au fost selectate din cartea *„Multiprocessor On-Line Scheduling of Hard-Real-Time Tasks”* [3].

3.1. Definiții de bază

În general, un task poate fi caracterizat prin următorii parametri întregi:

- S (momentul la care task-ul își începe execuția),
- C (numărul de unități CPU),
- D (deadline-ul)

De asemenea, mai pot fi adăugate și constrângeri adiționale precum relații de precedență sau periodicitate. Starea fiecărui task care și-a început execuția poate fi caracterizat prin doi parametri: timpul de procesare rămas *C(i)* la momentul *t* = i și deadline-ul *D(i)*, momentul până la care un task trebuie sa își termine execuția.

Laxitatea unui task la momentul *t* = i este definită de relația:

$$L(i) = D(i) - C(i) \tag{3.0}$$

Această caracteristică a taskului poate sta, în cadrul unui algoritm de ordonare (ex. LLF) la baza definirii urgenței cu care trebuie executat task-ul respectiv. În mod evident, un task cu laxitate zero trebuie executat imediat și fără întreruperi. Laxitatea negativă a unui task arată faptul că deadline-ul a fost depășit. Problema de organizare a taskurilor [3] poate fi modelată prin configurarea de *„token-uri”* în primul cadran al unui plan cartezian ce are pe axa verticală *C*, iar pe cea orizontală *L*. Astfel, un task *j* având parametrii $C_j(i)$ și $L_j(i)$ la momentul *t* = i, va fi poziționat la $L = L_j(i)$ și $C = C_j(i)$ în planul *L*-*C*. Un astfel de exemplu de reprezentare a unui scenariu în care avem un număr de trei task-uri este prezentat în *Fig. 3.0*. De reținut faptul că o poziție din acest plan poate fi ocupată de mai mult de un task în același moment de timp, în cazul de față observându-se faptul că task-urile 2 si 3 ocupă aceeași poziție. De asemenea, în reprezentarea planului nu este trecut numărul de unități CPU ce pot fi folosite, dar acesta va fi folosit în momentul în care se realizează execuția unui task.

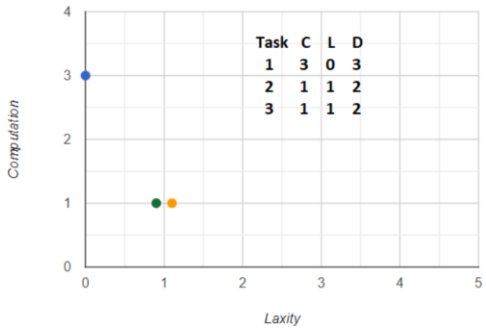

Fig. 3.0 – Reprezentarea a 3 task-uri în planul cartezian L-C

În cazul în care avem la dispoziție un număr de n procesoare și m task-uri ($n < m$) ce se pot executa la momentul t = i, atunci se vor putea executa un număr de n din totalul de m task-uri posibile. Reprezentând acest lucru tot în planul *L-C* menționat anterior, execuția task-urilor se traduce prin deplasarea în jos cu o unitate pe axa paralelă cu *C* și, simultan, spre stânga tot cu o unitate pe axa paralelă cu *L*. Astfel, ecuațiile ce prezintă execuția unui task sunt următoarele:

$$L(i+1) = L(i) \tag{3.1}$$

$$C(i+1) = C(i) - 1 \tag{3.2}$$

Pentru un task ce nu este executat, ecuațiile sunt următoarele : $L(i+1) = L(i) - 1$ si $C(i+1) = C(i)$. Algoritmul de ordonare este cel ce impune ce set de task-uri se va executa la un moment de timp. În cazul în care un task ajunge în al doilea cadran al planului, atunci acesta și-a ratat deadline-ul și, drept urmare, ordonarea nu este una fezabilă. Contrar acestui caz, dacă un task ajunge la valoarea 0 pe axa C, iar valoarea laxității este pozitivă, atunci task-ul a fost executat cu succes înainte de deadline.

3.2. Ordonarea în sisteme uni-procesor și multi-procesor

În cazul sistemelor cu un singur procesor există algoritmi de ordonare ce pot fi considerați optimi, contrar sistemelor multi-procesor unde studiile au arătat că problema găsirii unui algoritm optim nu este rezolvabilă.

În continuare, voi prezenta doi algoritmi ce sunt folosiți pentru ordonarea task-urilor: *Earliest Deadline* si *Least Laxity*. **Earliest Deadline** presupune execuția task-ului cu cel mai apropiat deadline, iar **Least Laxity** presupune execuția task-ului ce are cea mai mică valoarea a laxității. În ambele cazuri, dacă există task-uri situate pe aceeași poziție, se va alege unul în mod arbitrar.

În lucrarea sa [7], Dertouzos a demonstrat caracterul optim al algoritmului Earliest Deadline prin faptul că întotdeauna este posibil ca o ordonare fezabilă să fie transformată într-o ordonare ce implementează algoritmul Earliest Deadline. O astfel de situație este prezentată in *Fig. 3.1.* unde avem două task-uri: primul are deadline-ul la momentul *t*=3 și are nevoie de două unități de procesare, iar al doilea are deadline-ul la momentul *t*=2 si are nevoie de o singură unitate de procesare. În cazul *a)* se prezintă o ordonare fezabilă în care prima dată este executat task-ul cu deadline-ul mai mare, în locul task-ului cu deadline-ul mai mic. Având in vedere cele menționate mai sus, dacă înlocuim varianta de la punctul *a)* cu implementarea algoritmului Earliest Deadline, vom obține de asemenea o soluție fezabilă, așa cum se poate observa în aceeași figură la subpunctul *b)*.

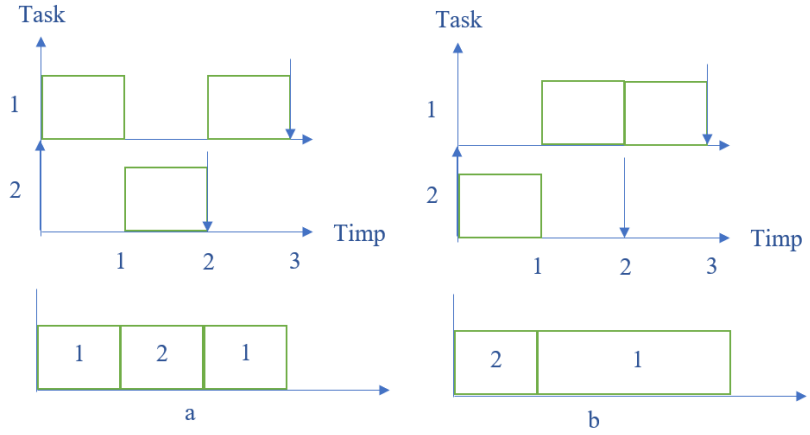

Fig. 3.1 – Ilustrarea caracterului optim al algoritmului Earliest Deadline

Din nefericire, caracterul optim al algoritmului nu se menține și în cazul sistemelor multi-procesor. Pentru a demonstra acest fapt, vom considera problema de ordonare reprezentată în *Fig. 3.2.a)* în care avem trei task-uri ce trebuie executate si un număr n=2 procesoare. Task-urile 2 si 3 au ambele deadline-ul peste două unități de timp din momentul în care se începe ordonarea, iar task-ul numărul 1 are deadline-ul peste trei unități de timp. Pentru a își finaliza execuția cu succes, task-ul 1 are nevoie de trei unități de procesare, iar celelalte două au nevoie de o unitate de procesare. Algoritmul Earliest Deadline ar presupune prima dată executarea task-urilor 2 si 3, lucru ce ar rezulta în mod imediat trecerea task-ului 1 în al doilea cadran al planului *L-C* prezentat la începutul capitolului. Astfel, nu există o ordonare fezabilă ce se poate realiza pentru scenariul dat în cazul în care se dorește folosirea algoritmului Earliest Deadline.

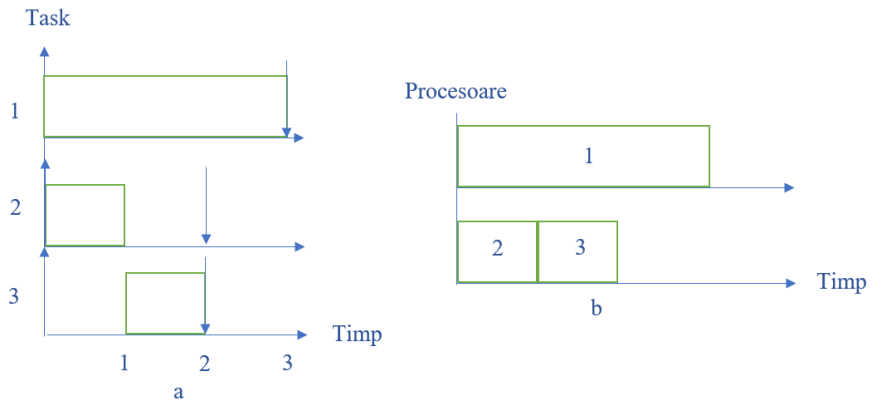

Fig. 3.2 – Exemplificarea nemenținerii caracterului optim al algoritmului Earliest Deadline în cadrul sistemelor multi-procesor

Totuși, scenariul prezentat în figura de mai sus poate avea o ordonare fezabilă în cazul în care este folosit algoritmul Least Laxity. Implementarea acestuia este reprezentată prin diagrama din *Fig. 3.2.b).*

Cele de mai sus demonstrează caracterul optim al algoritmului Earliest Deadline în cazul sistemelor uni-procesor și modul în care acesta realizează ordonarea task-urilor. O caracteristică ce face ca prezentul algoritm să fie considerat „best effort" este reprezentată de atributele task-urilor pe care acesta le necesită: algoritmul va realiza ordonarea acestora doar în funcție de deadline-ul precizat, necontând timpul de procesare și nici măcar start-time-ul. Astfel, ar fi foarte util dacă am putea găsi un algoritm ce are această proprietate pentru sistemele multi-procesor. La momentul actual nu se cunoaște un algoritm care să aibă aceasta proprietate.

În mod particular, în cazul în care nu cunoaștem *a priori* niciunul din următorii parametri: 1) deadline-uri, 2) timpi de procesare sau 3) start-time-uri, atunci pentru orice algoritm propus există un scenariu de task-uri pentru care nu s-ar putea găsi o soluție fezabilă.

3.2.1. Task-uri periodice

Pentru a vorbi despre task-urile periodice, se va introduce următoarea noțiune: factorul de utilizare (U). Factorul de utilizare al unui set de task-uri este egal cu $\sum_i C_i/D_i$. Un task poate solicita un nou timp de procesare de îndată ce deadline-ul acestuia expiră. Un aspect evident este reprezentat de faptul că $U \leq n$ reprezintă o condiție necesară pentru ordonarea unui set de task-uri, având la dispoziție n procesoare. Posibilitatea că $U \leq n$ ar reprezenta, de asemenea, și o condiție suficientă pentru a se putea realiza o ordonare fezabilă reprezintă o problemă discutabilă. Teorema următoare oferă un răspuns parțial la această problemă:

Teoremă [3]*:* „Presupunem Γ un set de *m* task-uri periodice având factorul de utilizare $U \leq n$. Notăm cu $T = CMMDC(D_1, \dots, D_m)$ și $t = CMMDC(T, T * \frac{C_1}{D_1}, \dots, T * \frac{C_m}{D_m})$. Atunci, o condiție suficientă pentru a putea ordona Γ având la dispoziție *n* procesoare este ca *t* să fie număr întreg.”

În demonstrarea acestei teoreme se presupune că *t* este un număr întreg. Acest lucru ar rezulta în faptul că $T, T * \frac{C_1}{D_1}, \dots, T * \frac{C_m}{D_m}$ vor fi toate numere întregi. Strategia propusă este să se execute taskul 1 pentru $T * \frac{C_1}{D_1}$ unități de timp, taskul 2 pentru $T * \frac{C_2}{D_2}$ unități de timp și așa mai departe. Pentru un task *j*, fiecare cerere va solicita un număr de C_j unități de timp și care va trebui să se execute în D_j unități de timp. Putem diviza D_j în $\frac{D_j}{T}$ părți, având fiecare dimensiunea T. Pentru fiecare parte vom aloca un număr de $T * \frac{C_j}{D_j}$ unități de procesare pentru taskul *j*. Așadar, pentru fiecare task în parte, timpul total de procesare va fi dat de urmarea formulă: $C_j = \left(\frac{D_j}{T}\right) * \left(T * \frac{C_j}{D_j}\right)$. De aici putem deduce faptul că timpul total de procesare al întregului set Γ va fi $T * \left(\sum_i \frac{C_i}{D_i}\right) = T * U$. Un astfel de exemplu este ilustrat în următoarea figură:

Task	Cj	Dj
J1	2	6
J2	4	6
J3	2	12
J4	20	24

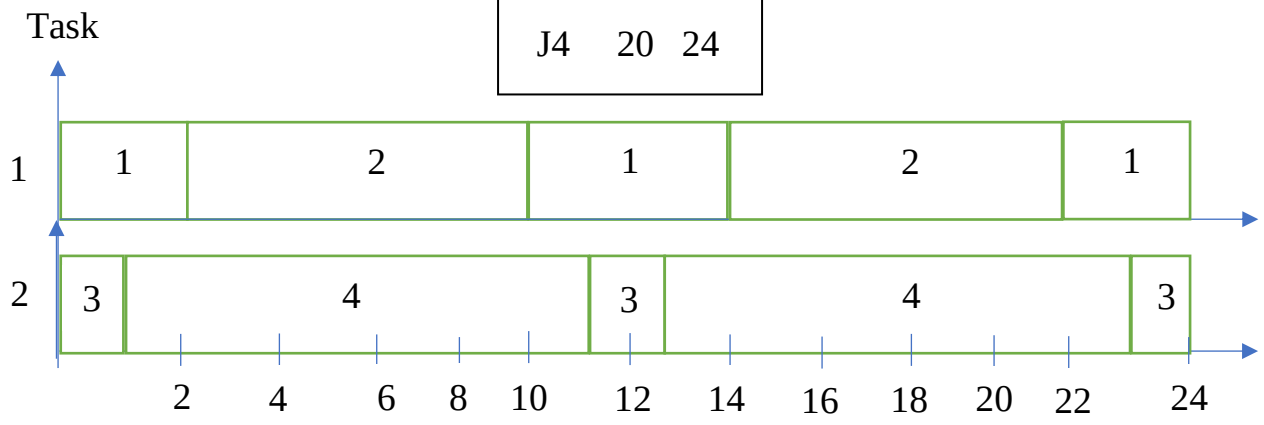

Fig. 3.3. – Scenariu cu 4 taskuri periodice și 2 procesoare

În concluzie, având în vedere aspectele prezentate în cadrul acestui capitol, elementele ce pot fi utile în rezolvarea problemei propuse sunt reprezentate de caracteristicile task-urilor:

- Starttime-ul
- Deadline-ul
- Timpul de procesare
- Periodicitatea

Algoritmii de ordonare multiprocesor nu sunt utili pentru problema ordonării cu constrângeri pe resurse deoarece aceștia nu iau în calcul alte elemente decât numărul procesoarelor și caracteristicile task-urilor.

4. Stadiul actual

4.1. Literatura de specialitate

În această secțiunea voi prezenta baza științifică ce motivează alegerea temei curente și importanța pe care o are în cadrul desfășurării activităților de Project management.

Articolul *„An updated survey of variants and extensions of the resource-constrained project scheduling problem”* [4] definește problema ordonării activităților cu constrângeri de resurse, având scopul de a găsi o ordonare a activităților fără să fie încălcate constrângerile date de relațiile de precedență și de numărul de resurse disponibil, astfel încât timpul de îndeplinire al tuturor activităților să fie minim. Se specifică faptul că această problemă a devenit una standard în domeniul organizării proiectelor, atrăgând un număr mare de cercetători ce au venit cu diverse proceduri de rezolvare.

În cadrul articolului sunt prezentate pe scurt diferite variații ale unor soluții implementate de către cercetători în domeniu.

În Capitolul 2 se specifică faptul că problema de bază a ordonării activităților cu constrângeri pe resurse necesită procesarea task-urilor în mod non-preemptiv, ceea ce înseamnă ca odată începută o activitate, aceasta își continuă execuția până în punctul în care este finalizată. Articolul precizează faptul că în lucrarea *„Handbook on Project Management and Scheduling Vol.1 ”*[1] autorii oferă o soluție în care activitățile pot fi întrerupte fără nicio restricție. Tot în cadrul aceluiași capitol, este abordată problema numărului de resurse solicitat de către fiecare activitate în parte. Se specifică faptul că în cadrul problemei standard, orice activitate A_i care necesită o cantitate de r_{ik} din resursa k, nu își va schimba această valoare pe parcursul procesării acesteia. Hartmann, prin lucrarea sa [5], abordează și situația în care activitatea A_i necesită o cantitate de r_{ikt} unități din resursa k la perioada t din timpul său total de procesare.

O carte ce prezintă foarte detaliat problema organizării proiectelor cu constrângeri pe resurse atât din punct de vedere teoretic, cât și din punct de vedere al algoritmilor folosiți este *„Resource-Constrained Project Scheduling: Models Algorithms, Extensions and Applications”* [2]. În capitolul 1 al cărții este definită problema ordonării activităților cu constrângeri pe resurse ca fiind o problemă de optimizare combinatorială. O problemă de optimizare combinatorială este definită de o soluție discretă în spațiul X sau care poate fi redusă la un set discret, și printr-un subset de soluții fezabile $Y \subseteq X$ asociat unei funcții obiectiv $f : Y \to \mathbb{R}$. Așadar, o problemă de optimizare combinatorială presupune găsirea unei soluții fezabile $y \in Y$ astfel

încât $f(y)$ este minimizată sau maximizată. Astfel, aspectul cel mai important pe care îl urmărim și în cadrul acestei lucrări este să găsim ordonarea ce respectă constrângerile date și care are durata cea mai mică.

Având la bază informația teoretică despre probleme de optimizare combinatorială, autorul definește problema curentă de ordonare a activităților cu constrângeri asupra resurselor ca fiind dată de următorul tuplu : (V, p, E, R, B, b).

Un set $V = \{A_0, A_1, \dots, A_{n+1}\}$ este folosit pentru a identifica activitățile ce constituie baza proiectului. Prin convenție, activitatea A_0 indică începerea procesului de ordonare, iar activitatea A_{n+1} indică, în mod simetric, finalizarea acestuia. Mulțimea rămasă prin eliminarea celor două activități, $A = \{A_1, A_2, \dots, A_n\}$, reprezintă activitățile propriu-zise ce trebuie să fie executate.

Duratele de execuție ale activităților sunt reprezentate printr-un vector p in $\mathbb{N}^{n+2}$, unde p_i reprezintă durata activității A_i. O caracteristică ce trebuie menționată este legată de valorile speciale folosite pentru activitățile A_0 si A_{n+1} , acestea având duratele asociate $p_0 = p_{n+1} = 0$.

Relațiile de precedență sunt definite de o colecție E având perechi de forma (A_i , A_j) care denotă faptul că activitatea A_i va preceda activitatea A_j . Prin definirea acestor relații de precedență se poate crea un graf G(V,E) în care fiecărui nod îi este asociat o activitate. Pentru ca relațiile de precedență să aibă sens, în cadrul grafului nu vor exista cicluri. Deoarece precedența reprezintă o relație binară tranzitivă, prezența unei rute în cadrul grafului G de la nodul A_i către nodul A_j denotă faptul că A_i va începe prima. Având in vedere cele doua activități ce reprezintă începutul, respectiv finalizarea ordonării, rezultă faptul că activitatea A_0 va preceda toate celelalte activități, iar activitatea A_{n+1} va fi succesorul acestora.

Resursele sunt organizate sub forma unei mulțimi $R = \{R_1, \dots, R_m\}$.

Disponibilitatea resurselor este reprezentată de un vector B in $\mathbb{N}^q$, unde B_k reprezintă numărul disponibil din resursa R_k .

Cererile de resurse sunt reținute într-o matrice de dimensiuni $(n + 2) * q$ notată cu b. Astfel, b_{ik} reprezintă cantitatea din resursa R_k folosită în fiecare perioadă de timp pentru executarea activității A_i .

O ordonare reprezintă un punct S in $\mathbb{R}^{n+2}$ unde S_i reprezintă punctul de început al activității A_i . C_i reprezintă timpul de completare al activității C_i , unde $C_i = S_i + p_i$. S_0 reprezintă un punct de referință pentru începutul proiectului, astfel, având in vedere mențiunile anterioare, $S_0 = 0$. Pentru ca o ordonare să fie considerată fezabilă, aceasta trebuie să respecte simultan atât relațiile de precedență (1.0), cât și constrângerile rezultate din numărul limitat de resurse disponibile (1.1). Mulțimea $Å_t$

reprezintă totalitatea activităților ce se află în execuție la momentul de timp t : $Å_t = \{A_i \in Å \mid S_i \leq t \leq S_i + p_i\}$.

$$S_j - S_i \geq p_i,\ \forall (A_i, A_j) \in E \quad (4.0)$$

$$\sum_{A_i \in Å_t} b_{ik} \leq B_k \forall\ R_k \in R,\ \forall\ t \geq 0 \quad (4.1)$$

Timpul de finalizare al ordonării este egal cu timpul de început al activității S_{n+1}. Autorul folosește termenul de *„makespan”* pentru a reprezenta timpul necesar finalizării tuturor activităților. Astfel, având în vedere toate cele prezentate mai sus, autorii lucrării [2] definesc Resource-Constrained Project Scheduling Problem (RCPSP) ca fiind problema găsirii unei ordonări non-preemptive S având makespan-ul minim S_{n+1} care respectă constrângerile date de relațiile de precedență (1.0) și de resurse (1.1).

4.2. Soluții similare

În clipa de față există numeroase aplicații de *Project Management* ce pot fi folosite pentru gestionarea activităților. Însă, principala problemă pe care am întâlnit-o în analizarea acestora a fost reprezentată de faptul că aceste aplicații nu realizează în mod automat o ordonare a activităților pe care utilizatorul le definește. Astfel, rolul acestora este mai degrabă acela de a fi un „to do list”.

Pe lângă soluțiile gratuite accesibile pe Internet, sau demo-urile celor ce necesită un abonament plătit, soluția prezentată în această lucrare implementează și o sugestie de ordonare a activităților, lucru ce poate fi util organizațiilor în realizarea obiectului lor de activitate. În continuare, voi prezenta câteva soluții software similare ce pot fi găsite pe Internet, prezentând totodată modul acestora de utilizare. Aceste aplicații au fost alese datorită faptului că sunt cele mai sugerate de către motorul de căutare Google.

4.2.1. Trello

Trello [10] reprezintă o aplicație web în care utilizatorii își pot organiza activitățile pe care le au de realizat în cadrul organizațiilor din care fac parte. Aplicația permite crearea de proiecte în care activitățile sunt împărțite pe liste de lucru. Ca și parametrii, fiecărei activități îi sunt atribuite o descriere, o perioadă în care aceasta trebuie realizată și un membru al echipei ce trebuie să o execute. De asemenea, se mai pot atașa link-uri sau fișiere necesare completării activității. În varianta demo gratuită se pot afișa activitățile doar sub forma unui Board. Pentru varianta plătită a aplicației există beneficii suplimentare precum: conectarea la

Google Drive, Dropbox si altele; posibilitatea de vizualizare a activităților sub forma unui calendar, lucru ce face mult mai ușor de urmărit evoluția acestora.

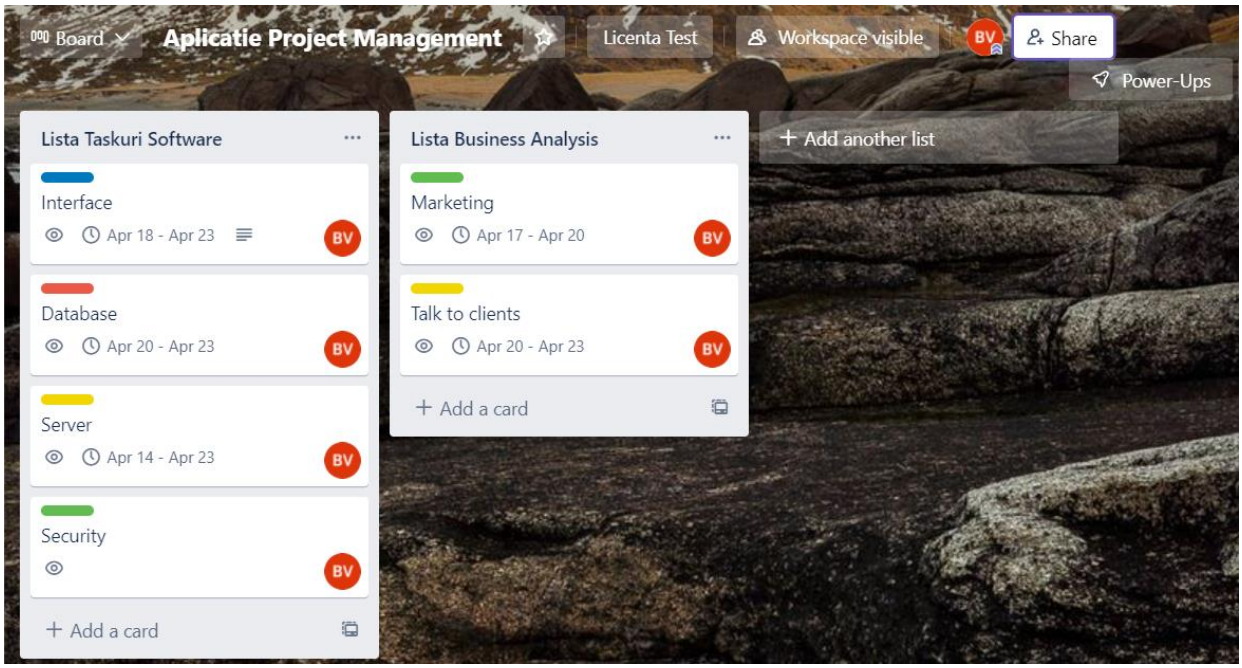

Fig. 4.0 - Interfața grafică Trello

4.2.2. Asana

O aplicație ce oferă mai multe module pentru utilizatori este Asana [11]. Aceasta oferă în mod implicit trei moduri de vizualizare a activităților: Calendar, Board si List. În ceea ce privește parametrii pe care o activitate îi are, aceștia sunt : utilizatorul ce trebuie să îndeplinească task-ul, perioada de desfășurare a activității si prioritatea. În cadrul aplicației sunt definite trei niveluri de prioritate: Low, Medium si High. Pe lângă acestea, o caracteristică foarte interesantă este reprezentată de posibilitatea de a defini relații de precedență, astfel putând arăta dacă execuția unei activități blochează sau este blocată de o alta. Săgețile ce se observă în imaginea următoare reprezintă relațiile de precedență definite de utilizator. În cazul în care se încearcă schimbarea perioadei de execuție a unei activități care nu ar respecta relațiile definite, se va realiza în mod automat o schimbare în perioada de execuție și pentru activitatea ce nu ar mai îndeplini condițiile impuse.

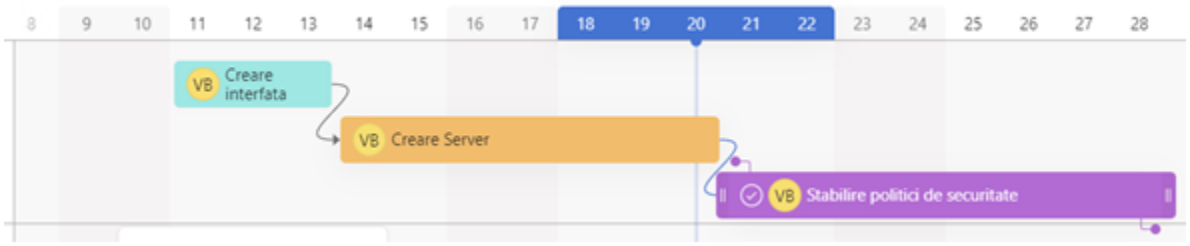

Fig. 4.1 - Asana – afișarea timeline-ului

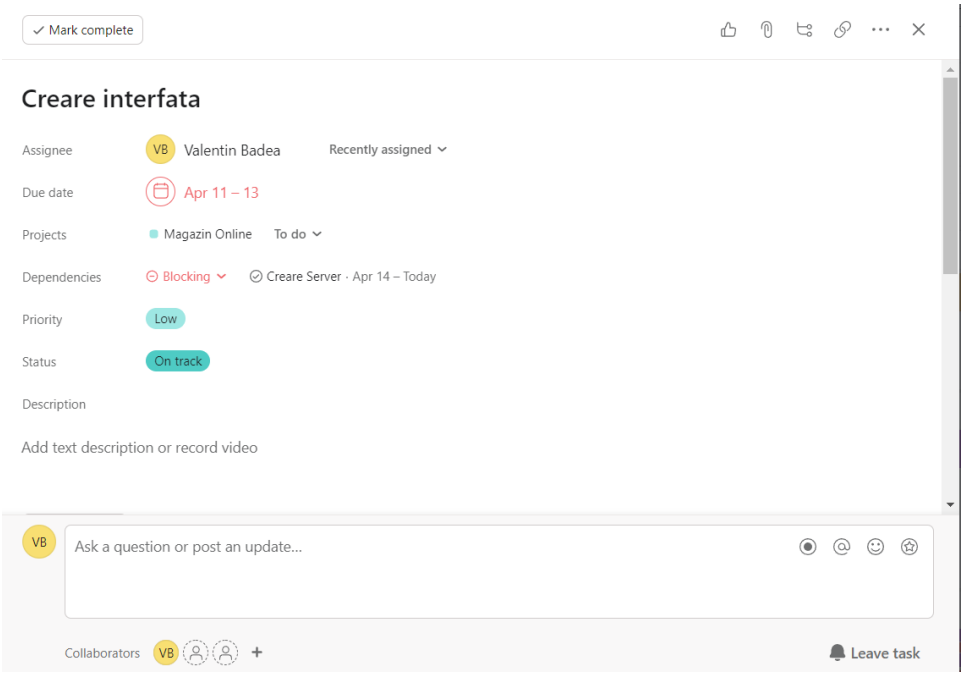

Fig. 4.2 - Asana – Caracteristicile unei activități

4.2.3. Jira

Un exemplu de aplicație folosită foarte frecvent în industria IT este Jira [12]. Aplicația are o interfață ce permite o foarte ușoară creare și vizualizare a activităților din cadrul proiectelor.

In cadrul tab-ului „Board” se pot vizualiza activitățile împărțite pe trei categorii:

- „to do” – activități ce urmează a fi realizate
- „in progress” – activități curente ce se află in desfășurare
- „done” – activități ce au fost finalizate

În cadrul tab-ului „Roadmap” se pot vizualiza activitățile din punct de vedere temporal. Consider că acest mod de vizualizare este mult mai intuitiv pentru

utilizatori, dar în continuare problema ordinii în care se vor executa aceste activități este nedeterminată și rămâne astfel la latitudinea celui ce le execută.

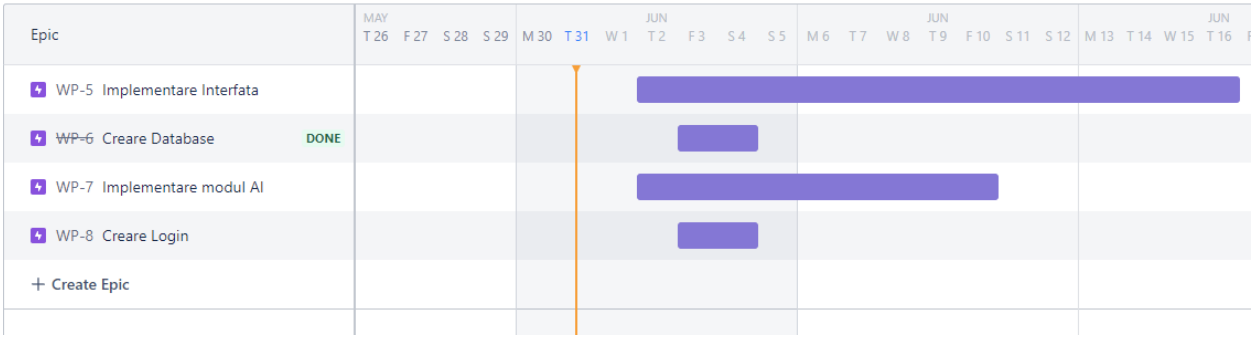

Fig. 4.3 - Jira Roadmap – aplicație de project management

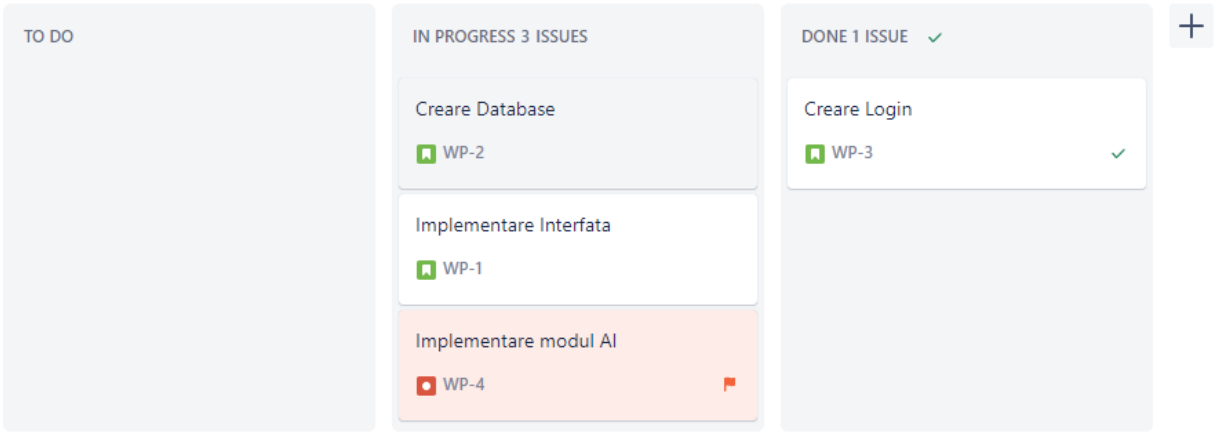

Fig. 4.4 - Jira Board – aplicație de project management

În concluzie, noțiunile prezentate în [2] sunt suficiente pentru a putea realiza ceea ce aplicația are ca obiectiv: realizarea unei ordonări a activităților cu constrângeri pe resurse. Deoarece în această lucrare nu se menționează cum ar trebui definită o activitate, s-au folosit noțiunile din Capitolul 3 pentru a realiza o paralelă între noțiunea de task din cadrul *Sistemelor de Operare* și activitatea din cadrul *Project Managementului*. Totodată, pentru interfața grafică a aplicației, s-a ținut cont de modurile similare în care aplicațiile din aceeași arie sunt construite.

5. Structura aplicației

5.1. Cerințele aplicației

Aplicația este capabilă să ruleze pe majoritatea platformelor datorită faptului că programele Java sunt compilate în bytecode, iar mai apoi Java Virtual Machine (JVM) îl execută la run-time. Astfel, pe orice platformă pentru care există o JVM se poate rula același cod. Compatibilitatea cross-platform permite o experiență run-time consistentă atât pentru dezvoltatori, cât și pentru utilizatori.

Aplicația oferă o gamă largă de funcționalități pe lângă funcția de baza a acesteia, aceea de a realiza o ordonare automată a activităților în cadrul proiectelor. Pentru a realiza această ordonare s-a implementat algoritmul propus în cadrul lucrării [2] ce testează fezabilitatea ordonării unui set de activități dat. Desigur, pentru ca aplicația să se muleze cât mai mult pe un mediu real de Project management, s-a decis introducerea mai multor elemente ce se regăsesc în practică. Astfel, în cadrul aplicației noastre, un utilizator nu poate avea mai mult de o activitate atribuită la un moment de timp, lucru ce presupune mai multe restricții asupra ordonării. Totodată, în cazul în care un set de activități nu poate fi ordonat în termen de 24 h, atunci rezultă faptul că nu există o ordonare fezabilă, iar ca rezultat, doar o parte din activități vor fi ordonate.

Pentru interacțiunea cu utilizatorul se va implementa o interfață grafică folosindu-se UI[1] framework-ul JavaFX prin intermediul căreia utilizatorul va putea accesa funcționalitățile implementate.

În implementarea proiectului, se va folosi o abordare orientată pe obiecte ce va oferi scalabilitate si posibilitatea adăugării simple de noi funcționalități.

5.2. Definirea arhitecturii aplicației

Aplicația este realizată folosindu-se arhitectura Client-Server. Astfel, server-ul va reprezenta componenta ce manageriază și livrează majoritatea datelor folosite de către client. Conexiunea dintre cele două entități este realizată prin intermediul socket-urilor, pe un port stabilit dinainte. Mesajele trimise între cele două entități vor implementa formatul de reprezentare JavaScript Object Notation (JSON).

5.2.1. Proiecte Maven

Maven [13] este un sistem de build și administrare a proiectelor. Funcționalitățile sale principale sunt: descrierea procesului de build al software-ului și descrierea dependențelor acestuia. Aceste proiecte conțin un fișier denumit

[1] User Interface

pom.xml (Project Object Model), ce are o structură specifică. POM-ul conține informații despre module, precum și despre dependențele proiectului. Utilizarea acestui tip de proiect ne-a permis o foarte ușoară descărcare dinamică a bibliotecilor Java utile pe parcursul desfășurării procesului de realizare a aplicației. Astfel, prin adăugarea acestor dependențe, am fost scutiți de timpul ce ar fi fost necesar descărcării fiecărei biblioteci în parte.

5.2.2. Modelul arhitectural Model-View-Controller

În realizarea aplicației (Client) s-a folosit modelul arhitectural Model-View-Controller (MVC). Acest pattern realizează separarea aplicației în trei părți principale: model-ul, view-ul și controller-ul. Fiecare dintre aceste trei elemente sunt construite astfel încât să manipuleze anumite aspecte specifice de dezvoltare a aplicației. Acesta reprezintă, în momentul de față, unul dintre cele mai frecvente Framework-uri folosite în industrie deoarece permite o ușoară scalabilitate a proiectelor. În continuare vom detalia fiecare dintre cele trei elemente.

5.2.2.1. Model

Componenta Model corespunde tuturor obiectelor cu care user-ul lucrează. În principiu, acesta va reprezenta datele ce sunt transferate între View si Controller. Spre exemplu, un task va fi modelat prin intermediul unei clase TaskModel ce va conține toate atributele și funcționalitățile acestuia.

5.2.2.2. View

Componenta View va corespunde tuturor interfețelor grafice ale aplicației. De exemplu, TaskView va include o componentă UI ce va conține elemente precum: text box, combo box, table view etc. cu care utilizatorul final va interacționa.

5.2.2.3. Controller

Controller-ul acționează ca o interfață între componentele View si Model pentru a procesa toate cererile primite, manipulând datele prin intermediul componentei Model și interacționând cu componenta View pentru a reda rezultatul final. De exemplu, Task Controller va manipula toate inputurile venite de la TaskView pentru a realiza update-uri în baza de date prin intermediul TaskModel.

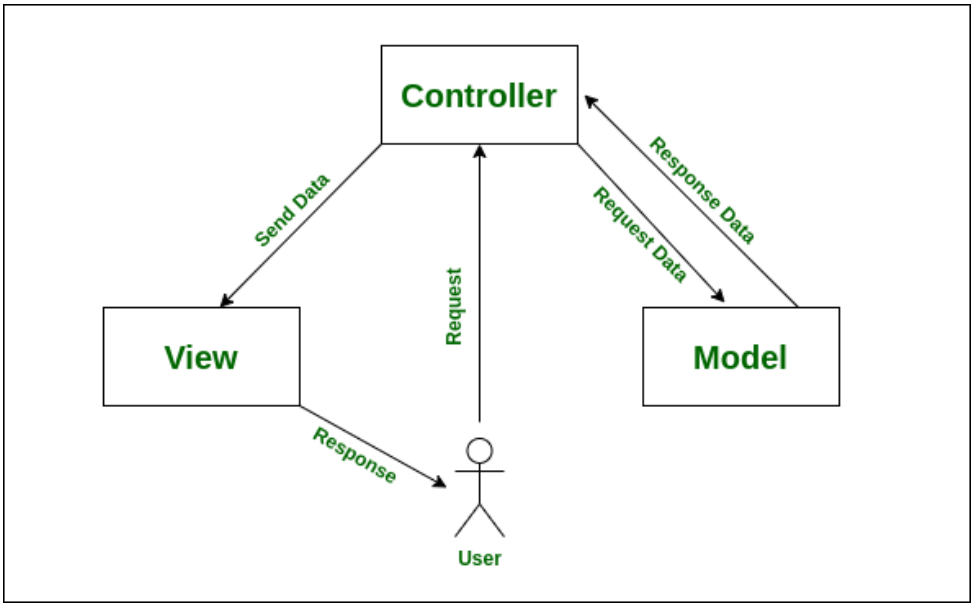

Fig. 5.0 – Modelul arhitectural MVC

Concluzionând, aplicația a fost construită folosindu-se: limbajul de programare orientat pe obiecte Java îmbinat cu modelul arhitectural MVC pentru partea de backend, framework-ul JavaFX pentru realizarea interfeței grafice și SQL Server pentru stocarea datelor.

6. Implementarea Software

6.1. Arhitectura bazei de date

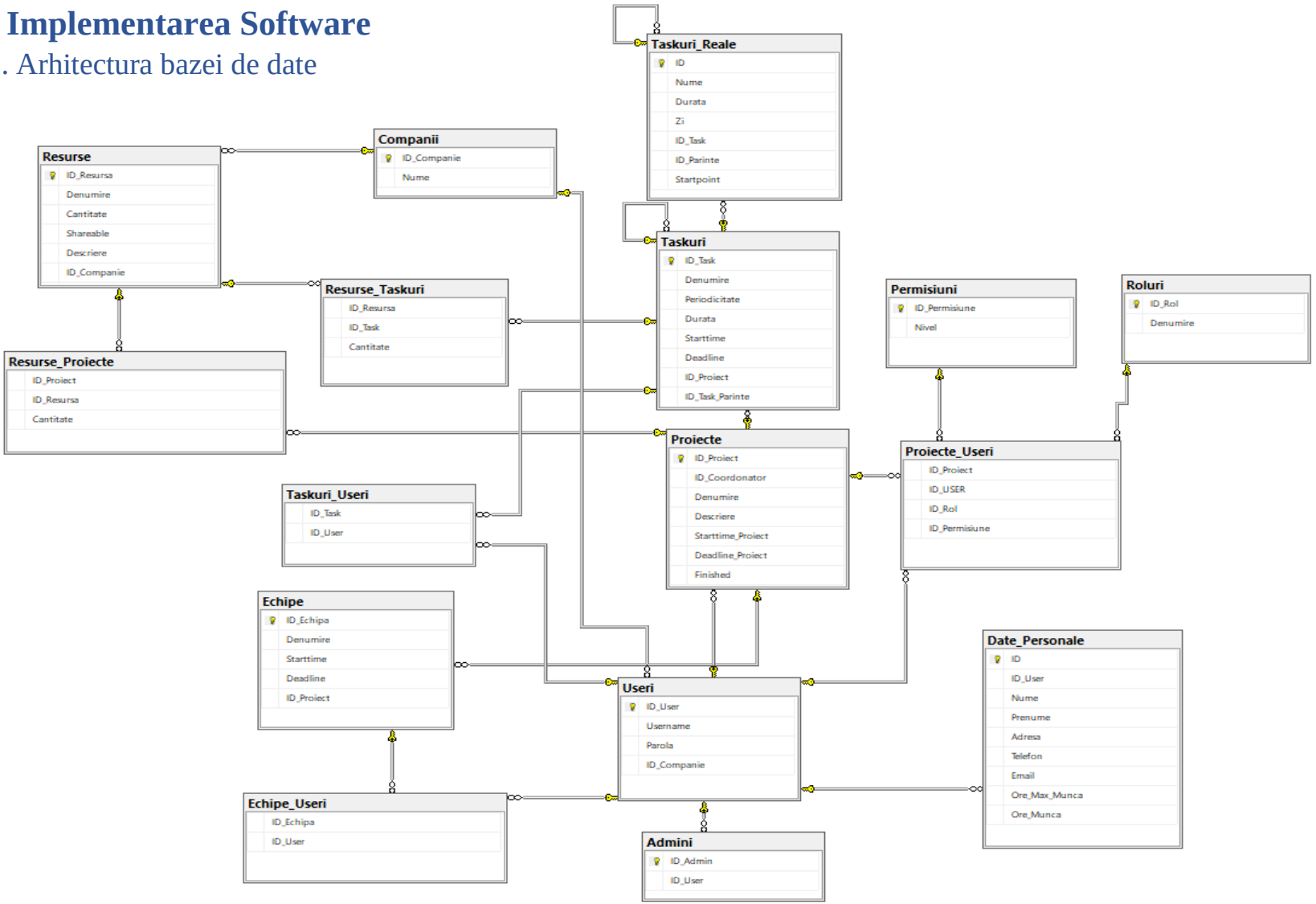

Fig. 6.1 – Arhitectura bazei de date

Pornind de la caracteristicile deținute de task-uri, cunoscute din teoria sistemelor de operare și prezentate în capitolele anterioare, s-a început conturarea aplicației prin realizarea bazei de date. Pentru stocarea informațiilor necesare aplicației s-a decis utilizarea SQL Server. Arhitectura bazei de date poate fi vizualizată în Fig. 6.1 .

Se poate observa faptul că tabela Taskuri conține toate caracteristicile provenite din cadrul sistemelor de operare: start time, deadline și durata. Pe lângă acestea, a fost nevoie de adăugarea unor caracteristici suplimentare. Printre acestea menționăm: relații de precedență definite prin intermediul unei coloane Foreign key către Primary key-ul aceleiași tabele, asignarea acestora către angajați prin intermediul unei tabele many-to-many ce face legătura între tabelele Taskuri și Useri. De asemenea, coloana „Periodicitate” va fi folosită în momentul în care inserăm valori în tabela Taskuri_Reale. Practic, tabela din urmă menționată va conține activitățile efective ce vor fi afișate în cadrul aplicației, iar tabela Taskuri va conține activitățile generale. De exemplu, daca vom defini o activitate ce va avea periodicitatea setata ca „Daily” pentru o perioada de zece zile, în tabela Taskuri vom avea o singură intrare, iar tabela Taskuri_Reale va conține zece intrări specifice fiecărei activități, urmând ca apoi să putem face modificări pe fiecare în parte.

Această parte a fost cea specifică paralelei cu sistemele de operare. În continuare, pentru a realiza gestionarea unei organizații am avut nevoie de mai multe tabele. În primul rând, pentru a realiza autentificarea în cadrul aplicației, am avut nevoie de o tabelă în care să stocăm conturile (Useri). În cazul în care contul este deținut de un administrator al companiei, ID-ul acestuia se va afla și în tabela Admini. Pentru că vorbim despre organizații reale, fiecare angajat va avea mai multe caracteristici personale: adresă, telefon, email etc. O practică des utilizată în realizarea bazelor de date este ca informațiile cu caracter personal să fie ținute separat de informațiile specifice contului, astfel, după cum se poate observa și în diagrama prezentată anterior, există o tabelă specifică informațiilor cu caracter personal. Un alt element principal al bazei de date, pe lângă activități, este reprezentat de resursele de care dispun companiile. Astfel, pentru definirea acestora a fost creată o tabelă în care sunt reținute detaliile despre acestea precum: denumire, descriere, cantitate. Legăturile dintre resurse și proiecte/activități sunt realizate prin intermediul a două tabele de tip many-to-many.

6.2. Comunicația Client-Server

Așa cum a fost menționat și la începutul acestui capitol, conexiunea dintre cele doua entități Client-Server se va realiza prin intermediul socket-urilor. Pentru a se putea realiza schimbul de mesaje între cele două entități, fiecare cerere trimisă de

către Client spre Server va fi transmisă sub forma unui șir de caractere ce respectă formatul JSON. Fiecare mesaj transmis va avea următoarea formă:

```
{
 "Type":"Cererea clientului",
 "Argv":"Parametrii necesari realizării cererii"
}
```

În continuare, după ce Serverul analizează solicitarea primită de la Client va urma să execute interogările necesare pe baza de date și să întoarcă un răspuns ce va respecta același format JSON. O dificultate întâmpinată inițial a fost reprezentată de modalitatea în care se vor serializa obiectele ce urmează a fi transmise. Având în vedere faptul că urma să transmitem obiecte ce posedă pe lângă membrii de bază (int, string etc.) și alte obiecte, a fost nevoie de folosirea unei biblioteci ce poate converti un obiect în formatul său JSON. Pentru a realiza această operațiune, a fost aleasă biblioteca open-source GSON. De asemenea, aceasta poate fi folosită și în sens invers, pentru a converti stringuri JSON în echivalentul obiectelor Java. Un lucru important de menționat este reprezentat de faptul că toți membrii claselor de pe ambele componente trebuie obligatoriu să aibă aceeași denumire (ex. clasa Proiect situată pe Server va conține exact aceiași membri precum clasa Proiect situată pe Client). În caz contrar, operația de serializare/deserializare nu se poate concretiza.

Deși această abordare a micșorat codul ce ar fi fost necesar în cazul unei serializări membru cu membru, am întâmpinat o problemă în cazul în care obiectul ce se dorea a fi serializat avea în componența sa un membru de tip LocalDate deoarece biblioteca Gson nu poate realiza serializarea acestui tip de date. Astfel, a fost nevoie de crearea a două Adaptere speciale care să indice modul în care ar trebui serializat acest tip de date. Codul necesar creării unui astfel de Adaptor este următorul:

```
public class LocalDateSerializer implements JsonSerializer<LocalDate> {
    private static final DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("d-MMM-yyyy");

    @Override
    public JsonElement serialize(LocalDate localDate, Type srcType,
JsonSerializationContext context) {
        return new JsonPrimitive(formatter.format(localDate));
    }
}
```

Se poate observa că o variabilă de tip LocalDate va fi transformată după următorul format: „d-MMM-yyyy”. Pentru Adaptorul ce realizează deserializarea codul este asemănător:

```
public class LocalDateDeserializer implements JsonDeserializer<LocalDate> {
    @Override
    public LocalDate deserialize(JsonElement json, Type typeOfT,
JsonDeserializationContext context)
            throws JsonParseException {
        return LocalDate.parse(json.getAsString(),
                DateTimeFormatter.ofPattern("d-MMM-
yyyy").withLocale(Locale.ENGLISH));
    }
}
```

Integrarea acestor Adaptere se realizează în urma configurării unui obiect de tip GsonBuilder pe baza căruia, ulterior, vom inițializa un obiect de tip Gson. Setarea Adapterelor prezentate anterior se realizează în următorul mod:

```
GsonBuilder gsonBuilder = new GsonBuilder();
gsonBuilder.registerTypeAdapter(LocalDate.class, new LocalDateSerializer());
gsonBuilder.registerTypeAdapter(LocalDate.class, new
LocalDateDeserializer());
Gson gson = gsonBuilder.setPrettyPrinting().create();
```

Un alt aspect ce a creat inconveniente pe partea de comunicare a fost produs de durata ridicată de timp necesară Serverului pentru a răspunde la anumite cereri primite. Astfel, utilizatorului i se dădea impresia că aplicația se blochează și revine ulterior. Pentru a rezolva această problemă a fost nevoie de implementarea unui mecanism de multithreading. Prin urmare, în momentul în care utilizatorul realizează o astfel de acțiune (ex. vizualizarea unui proiect), un fir de execuție secundar realizează request-ul, iar pe firul principal de execuție se încarcă o animație sugestivă de loading. Pentru a porni acest thread secundar, a fost nevoie de folosirea unui obiect de tip Service (Anexa C). Un exemplu ce ilustrează această rezolvare este prezentat în figura următoare:

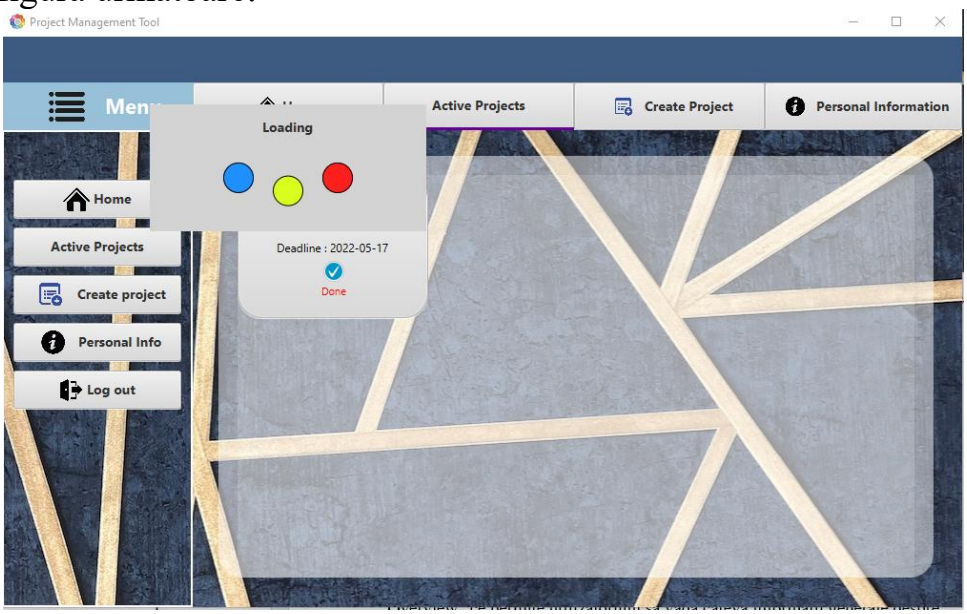

Fig. 6.2 – Ilustrarea unei animații de „loading...”

Totodată, din motive de securitate, s-a dorit ca mesajele să nu fie transmise în clar. Astfel, soluția aleasă a fost ca între cele două entități să existe o conexiune TLS (Transport Layer Security).

Deoarece cele două entități pot comunica între ele cu sau fără TLS, este necesar pentru client să ceară server-ului stabilirea unei conexiune TLS. Odată ce clientul și serverul au stabilit folosirea TLS-ului, aceștia negociază o conexiune stateful folosind procedura de handshake. Protocoalele folosesc un handshake împreună cu o criptare asimetrică pentru a stabili și o cheie de sesiune ce va fi folosită mai apoi pentru a cripta comunicația folosind o criptare simetrică.

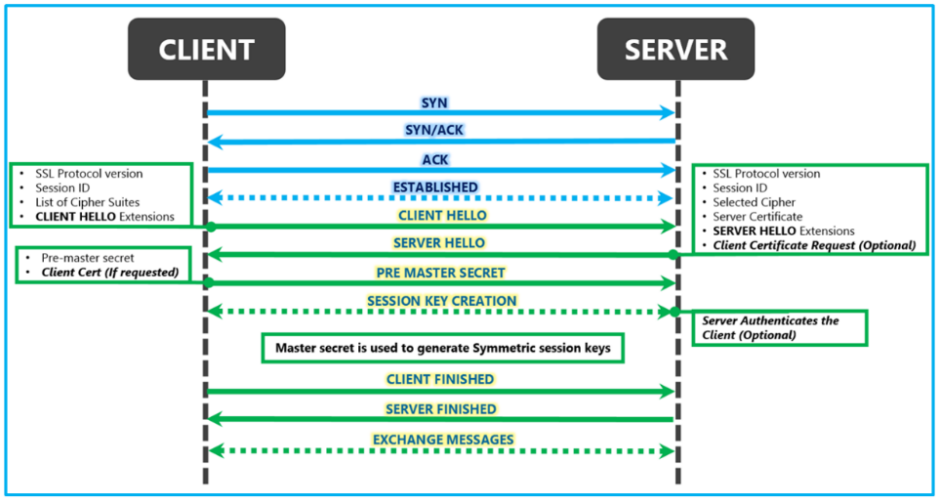

Fig 6.3 - TLS Exchange

Pentru a implementa TLS avem nevoie de două fișiere: Truststore și Keystore. Primul este folosit pentru a stoca certificatul prezentat de Server pentru a se autentifica într-o conexiune SSL, în timp ce al doilea este utilizat pentru a stoca cheia privată și certificatele pe care un program ar trebui să le prezinte pentru verificare. În figura următoare este ilustrat rolul pe care cele două fișiere îl au în cadrul realizării conexiunii TLS.

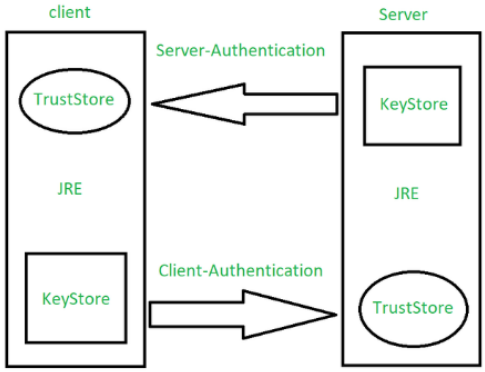

Fig 6.4 – Utilizarea fișierelor Keystore și Truststore

Pentru crearea acestor fișiere a fost utilizat tool-ul *keytool* oferit de Java. Comenzile ce au fost date pentru realizarea acestora sunt:

- Pentru crearea perechii de chei și a certificatului self-signed:

Keytool -genkeypair -alias mykey -keyalg RSA -keystore keystore

- Pentru exportarea certificatului self-signed:

Keytool – export -alias mykey -keystore keystore -file mycert.cer

- Importarea certificatului într-un TrustStore

Keytool -import -alias mycert -file mycert.cer -keystore truststore

Pentru a confirma faptul că implementarea a fost corectă, am folosit utilitarul Wireshark pentru a analiza traficului de pe stația locală unde se poate observa folosirea TLSv1.2 în cadrul comunicației dintre cele două entități.

tls

No.	Time	Source	Destination	Protocol	Length	Info
45	2.165461	127.0.0.1	127.0.0.1	TLSv1.2	339	Client Hello
47	2.250943	127.0.0.1	127.0.0.1	TLSv1.2	138	Server Hello
49	2.252147	127.0.0.1	127.0.0.1	TLSv1.2	946	Certificate
51	2.289828	127.0.0.1	127.0.0.1	TLSv1.2	349	Server Key Exchange
53	2.290188	127.0.0.1	127.0.0.1	TLSv1.2	53	Server Hello Done
55	2.307942	127.0.0.1	127.0.0.1	TLSv1.2	86	Client Key Exchange
57	2.321434	127.0.0.1	127.0.0.1	TLSv1.2	50	Change Cipher Spec
59	2.323965	127.0.0.1	127.0.0.1	TLSv1.2	89	Encrypted Handshake Message
61	2.330672	127.0.0.1	127.0.0.1	TLSv1.2	1203	New Session Ticket
63	2.331727	127.0.0.1	127.0.0.1	TLSv1.2	50	Change Cipher Spec
65	2.332167	127.0.0.1	127.0.0.1	TLSv1.2	89	Encrypted Handshake Message

Fig. 6.5 – Captura de trafic TLS

6.3. Tipurile de utilizatori

În funcție de rolul pe care o persoană o are în cadrul organizației din care face parte, au fost definite două roluri posibile: administrator sau angajat normal. Astfel, aplicația nu oferă posibilitatea creării unui nou cont de angajat, ci acestea vor putea fi create doar de către administrator. S-a ales această modalitate deoarece s-a luat în calcul faptul că o persoană ar putea crea conturi false în cadrul diverselor companii. Așadar, opțiunea de register a aplicației va oferi doar posibilitatea creării unui cont de administrator împreună cu organizația acestuia. În continuare, administratorul va putea defini conturile de utilizator pentru fiecare angajat in parte. O altă funcție pe care doar administratorul o deține este reprezentată de posibilitatea definirii resurselor companiei. Restul acțiunilor pe care un administrator le mai poate face sunt doar de vizualizare a datelor privind: angajații, proiectele si resursele. Diagrama din figura următoare are rolul de a clarifica acțiunile pe care un administrator le poate face:

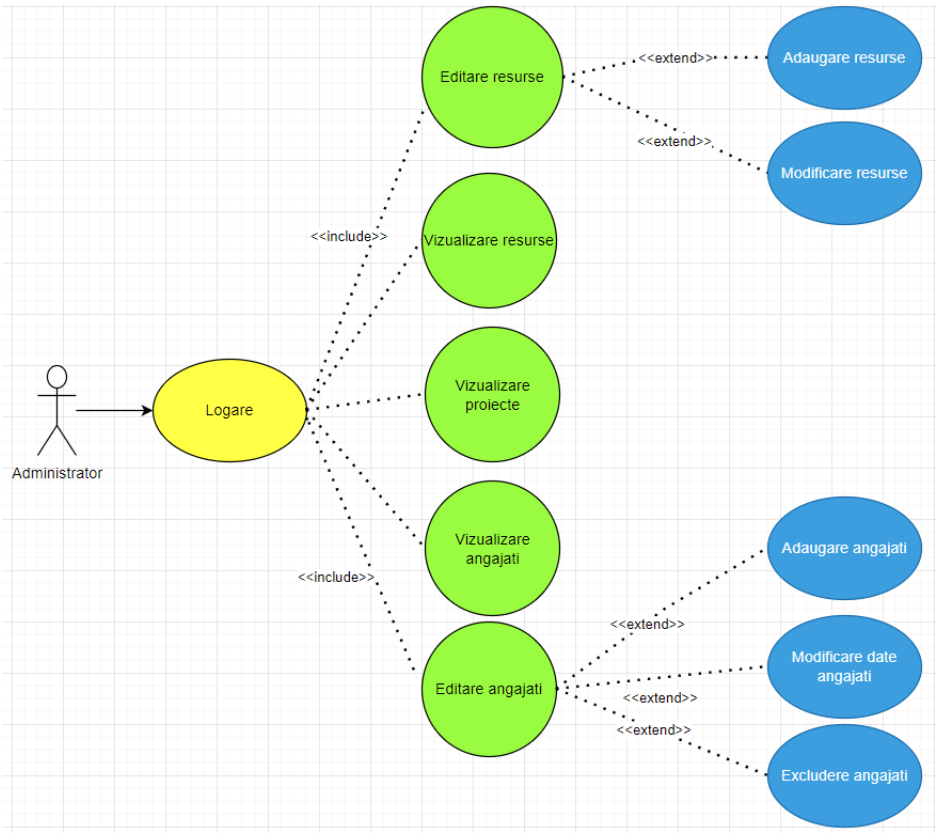

Fig. 6.6 – Diagrama cazurilor de utilizare pentru modul administrator

Activități mai complexe sunt definite pentru modul de angajat normal deoarece prin intermediul acestor conturi se vor putea crea noi proiecte și studia modul de ordonare al activităților în cazul în care există realocări de resurse. Astfel, pe lângă acțiunile standard de vizualizare, un angajat poate inițializa un nou proiect. Inițializarea unui proiect va presupune trei pași consecutivi pe care angajatul desemnat ca manager de proiect îi are de parcurs. În primul rând, utilizatorul va trebui să completeze informațiile de bază despre proiect precum: denumire, intervalul de timp în care se va realiza, descrierea și asignarea personalului. După ce sunt completate aceste câmpuri, managerul de proiect va desemna rolul și nivelul de prioritate al fiecărui angajat din cadrul proiectului (rolul va permite sau va bloca pe viitor încercarea angajaților de a realiza modificări în cadrul proiectului), va crea echipe (acestea au rol informativ și nu vor influența ordonarea activităților) și va asigna resursele necesare executării acestuia. Ultimul pas pe care managerul îl are de făcut este reprezentat de crearea activităților ce vor trebui executate pe parcursul desfășurării proiectului. Ulterior se vor mai putea adăuga activități suplimentare dacă situația o impune. În momentul în care un proiect este creat, se realizează un request către Server care va realiza și ordonarea activaților.

Cea mai importantă pagină din cadrul proiectului, cea care face și obiectul cercetării noastre, este reprezentată de pagina de vizualizare a proiectelor. În cadrul acesteia, angajații ce dețin un nivel de prioritate necesar modificării datelor curente pot realiza următoarele acțiuni: vizualizarea modului în care algoritmul a alocat resursele, adăugarea unei noi activități, modificarea numărului de resurse atribuit proiectului și acceptarea unor modificări propuse de aplicație, oferite pentru activitățile ce nu au putut fi incluse în timpul de lucru inițial. În figura următoare este reprezentată diagrama cazurilor de utilizare pentru un utilizator ce nu are rol de administrator:

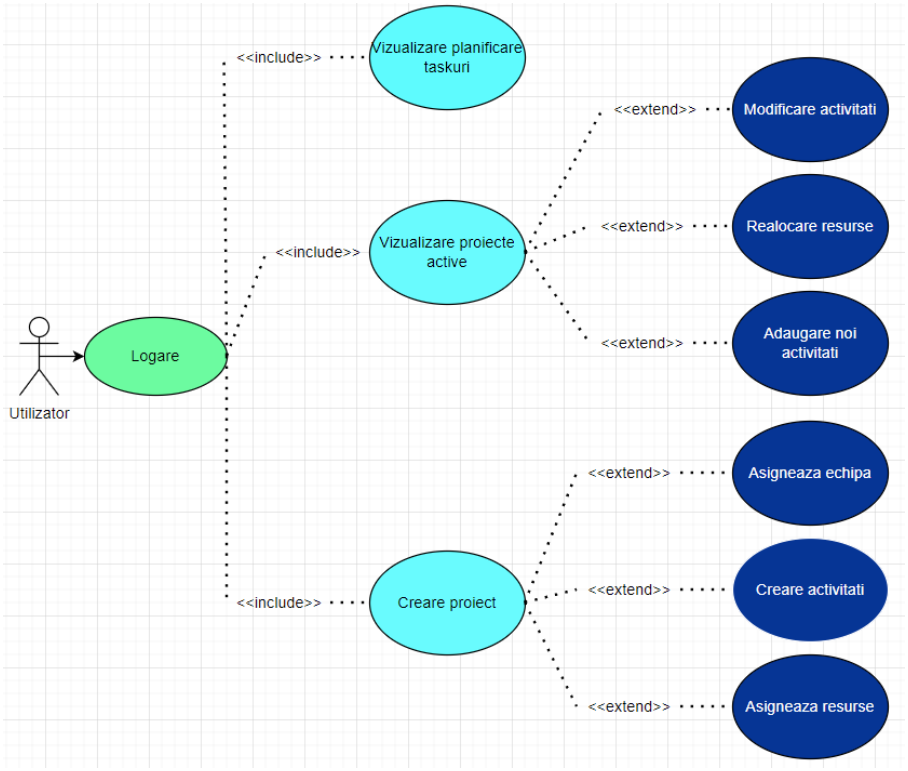

Fig. 6.7 Diagrama cazurilor de utilizare pentru utilizatori fără rol de administrator

6.4. Algoritmul de ordonare a activităților

Principala funcționalitate a aplicației este reprezentată de funcția ce permite o ordonare automată a activităților în cadrul proiectelor. Astfel, s-a decis că algoritmul de la care vom pleca să fie cel propus de Christian Artigues, Sophie Demassey și Emmanuel Neron în lucrarea [2]. Notațiile pe care le vom vedea în următorul pseudocod sunt aceleași notații ce au fost prezentate in Capitolul 4.1. :

```
1: L = {C_j | A_j ∈ V}
2: sort L in increasing values
3: for t ∈ L do
4:    for R_k ∈ R do
5:       o ← 0, F ← ∅
6:       for A_j ∈ A do
7:          if S_j < t, C_j ≥ t and b_jk > 0 then
8:             o ← o + b_jk, F ← F ∪ A_j
9:          end if
10:         if o > B_k then
11:            S is not resource-feasible because activities of F are executed in parallel
12:            return false
13:         end if
14:      end for
15:   end for
16: end for
17: return true
```

Fig. 6.8 – Pseudocod al algoritmului ce testează fezabilitatea unui scenariu de task-uri

Practic, acest algoritm nu ne oferă și o ordonare a activităților, așa cum am fi predispuși să credem, ci doar testează fezabilitatea scenariului dat. De la acest algoritm a trebuit să plecăm în procesul de implementare, iar primul pas pe care l-am avut de implementat a fost setarea inițială a momentului de timp la care fiecare activitate ar trebui să își înceapă execuția. Activitățile ce nu aveau relații de precedență definite față de alte activități vor avea acest punct de plecare setat cu valoarea 0, iar restul vor avea inițial acest punct la :

$$S_j = S_i + p_i \tag{6.0}$$

, unde S_i și p_i reprezintă punctul de început, respectiv durata activității față de care există relația de precedență. Următorul pas a presupus ca în momentul în care intrăm pe ramura din care rezultă că setul dat nu are o ordonare fezabilă, să incrementăm punctul de start al primei activități, apoi să recalculăm toate celelalte puncte de start pentru că, din cauza relațiilor de precedență, se pot modifica mai multe activități simultan. În final, vom relua execuția algoritmului până când toate activitățile pot fi ordonate.

Este important de menționat și faptul că până la începerea execuției algoritmului, se va realiza o translatare a angajaților în resurse, astfel încât unui angajat să nu i se poată atribui mai mult de o activitate la un moment de timp. Pentru a reține aceste informații, fiecărei activități i-a fost atribuită un HashMap în care am reținut ID-ul resursei și cantitatea necesară execuției. Algoritmul este prezentat in

Anexa A. Dificultatea întâmpinată a fost reprezentată de translatarea personalului ce reprezintă resursa umană a organizației în resurse pentru a putea fi luați în calcul în momentul în care algoritmul realizează ordonarea activităților. S-a decis ca pentru memorarea numărului de resurse necesar unei activități să se folosească un HashMap în care sunt reținute perechi de forma:

<ID resursa, Cantitate necesara>

Problema a apărut în momentul în care a trebuit să introducem și angajații în această listă deoarece s-ar fi putut ajunge la un caz de ambiguitate în care o resursă ar fi avut un ID identic cu angajatul ce avea de executat task-ul respectiv. Astfel, s-a decis ca în HashMap-ul respectiv ID-ul angajaților va fi memorat cu semn negativ. Acest lucru elimină cazurile de ambiguitate deoarece atât resursele cât și angajații au un ID unic.

Ulterior execuției algoritmului, în cadrul bazei de date, vor fi introduse valorile actuale de start pentru fiecare activitate în parte. Pentru a realiza o vizualizare asemănătoare cu cea prezentată în carte (Fig. 6.9), a fost folosit un algoritm de backtracking pentru a reuși să poziționăm fiecare activitate. Nu ne interesează numărul resursei pe care o activitate o folosește, astfel, prima soluție oferită de algoritm va fi si cea aleasă pentru vizualizare. Algoritmul de backtracking este prezentat în Anexa B.

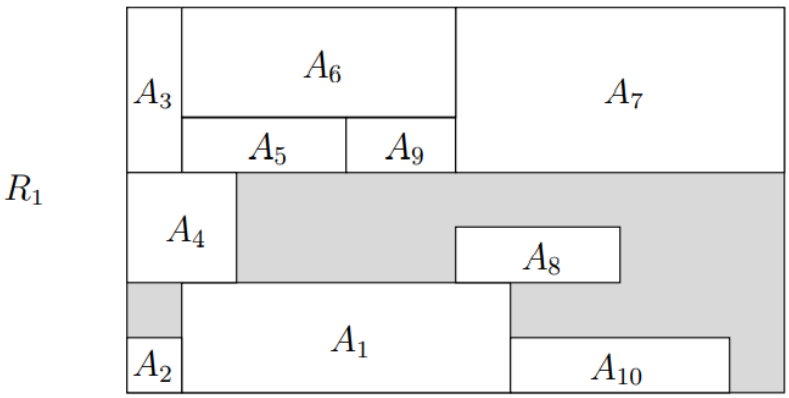

Fig. 6.9 – Exemplu de vizualizare a utilizării unei resurse R1 pe un scenariu cu 10 activități [2]

În concluzie, aplicația a fost implementată ținându-se cont de noțiunile teoretice descrise în Capitolele 2 și 3 prin care activitățile sunt definite pe baza caracteristicilor task-urilor din *Sistemele de Operare,* iar proiectele sunt definite ca având un scop clar, o perioadă finită de timp și un număr restrâns de resurse disponibile. Pentru a realiza ordonarea activităților ținându-se cont de constrângerile resurselor a fost folosită o variantă modificată a algoritmului prezentat în lucrarea [2].

7. Testarea

Pentru a putea confirma faptul că algoritmul implementat de către noi oferă rezultate corecte, a fost creat un proiect pentru care se putea verifica ordinea în care sunt ordonate activitățile.

Astfel, pentru acest caz vom considera un scenariu de activități în care avem un număr de $n = 10$ activități si $R = 2$ resurse având următoarele disponibilități $B_1 = 7$ & $B_2 = 4$. În următorul tabel putem observa durata și cantitățile de resurse necesare executării fiecărei activități în parte.

A_i	A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}
p_i	0	6	1	1	2	3	5	6	3	2	4	0
b_{i1}	0	2	1	3	2	1	2	3	1	1	1	0
b_{i2}	0	1	0	1	0	1	1	0	2	2	1	0

În următoarea imagine se poate observa graficul ce ilustrează relațiile de precedență definite între activități:

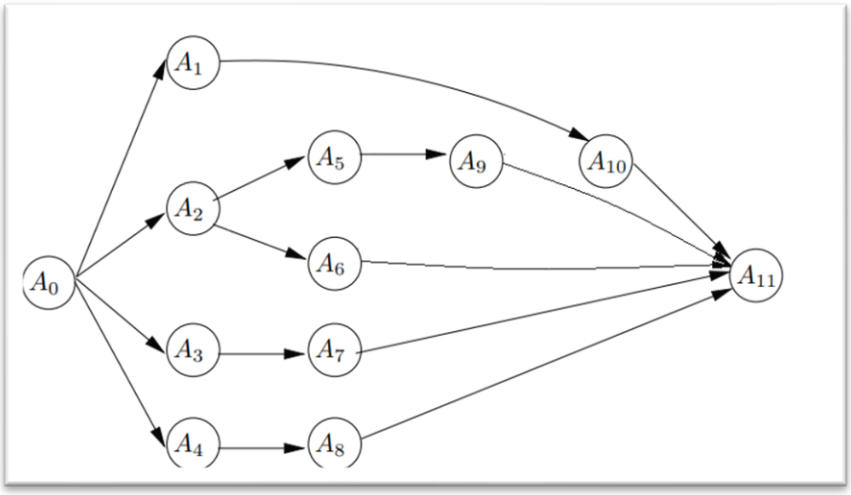

Fig. 7.0 – Grafic ce ilustrează relațiile de precedență

În figura următoare este prezentată ordonarea activităților pentru scenariul dat, unde se poate observa faptul că există o ordonare fezabilă, iar setul de activități poate fi planificat în 12 unități de timp:

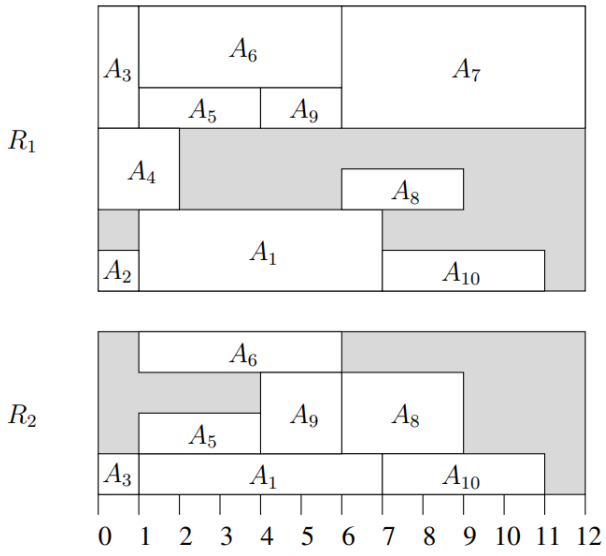

Fig. 7.1 – Ordonarea dată pentru scenariul dat

După ce datele au fost introduse în aplicație și proiectul a fost creat, algoritmul implementat a realizat ordonarea activităților. Rezultatul este prezentat în următoarea figură:

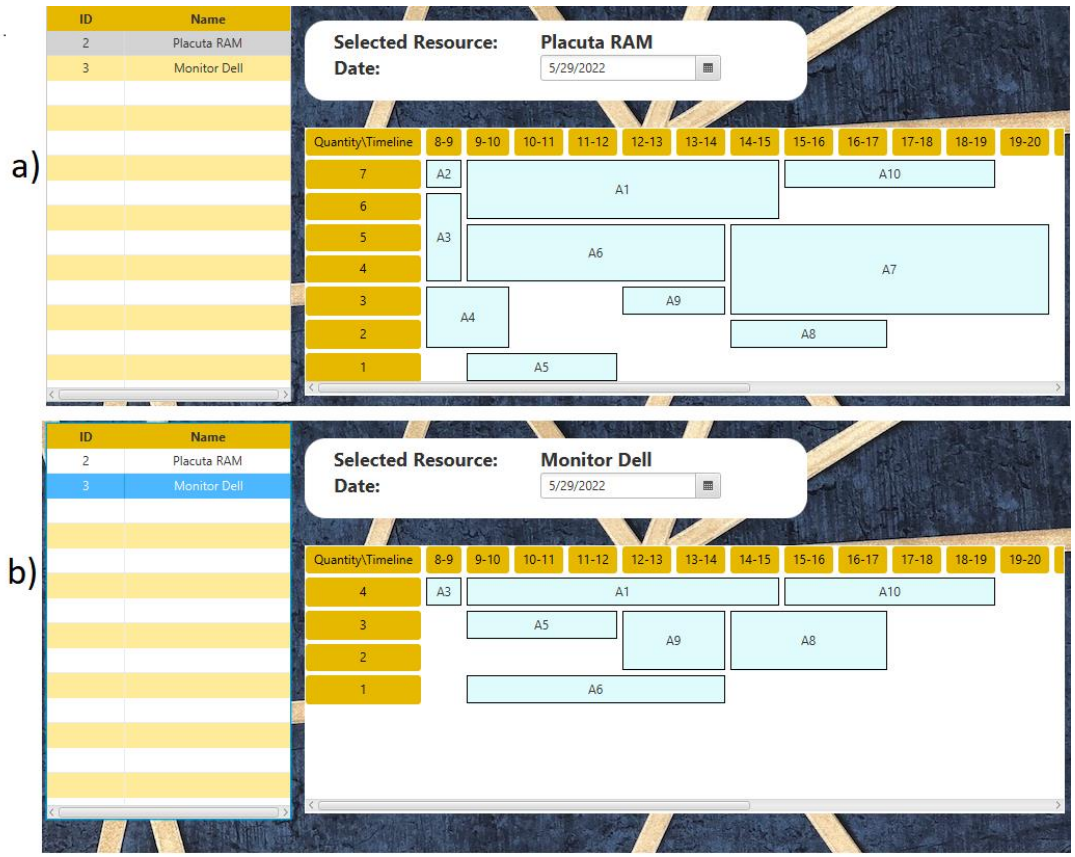

Fig. 7.2 – Ordonarea oferită de aplicație

În fereastra de vizualizare a alocării resurselor se poate observa cantitatea și timpul pentru care o resursă este alocată unei activități anume. În cazul nostru, resursele folosite pentru exemplul dat au fost: „Plăcuță RAM” și „Monitor Dell”. În partea dreaptă a imaginilor se poate observa cantitățile deținute de proiect iar în partea de sus se poate observa o axă a timpului în care unitatea de timp este reprezentată de 1h.

Deși la prima vedere ar putea părea că cele două rezultate diferă, dacă observăm punctele de început al fiecărei activități în parte, vom observa faptul că rezultatele sunt identice. Această diferență de afișare apare deoarece, după cum am menționat și în capitolul 6.4, numărul resursei folosite nu contează pentru algoritm, iar afișarea este realizată prin intermediul unui algoritm de backtracking, astfel prima soluție în care blocurile corespunzătoare activităților nu se suprapun este și soluția aleasă pentru afișare.

7.1. Testare practică

Pentru a observa funcționalitatea aplicației în cazul unor situații reale, am creat un scenariu care modelează un anumit tip de organizație, în speță o universitate. De-a lungul testării am realizat mai multe realocări de resurse sau adăugări suplimentare de activități pentru a observa comportamentul aplicației. De asemenea, am avut parte și de anumite activități ce nu au putut conduce la o ordonare fezabilă din două motive: fie acestea solicitau o cantitate mai mare de resurse decât cea disponibilă, fie prin ordonarea acestora s-ar fi depășit limita impusă de 24h pentru planificare. În subcapitolul următor este prezentat în detaliu întregul scenariu de activități împreună cu modificările aferente.

7.1.1. Scenariul de activități

În acest caz am luat spre analiză orarul unei universități. Vom avea un număr de n=4 angajați cu rolul de profesor:

Nr. Crt.	Nume	Rol
1.	Mihai Mihai	Profesor
2.	Valentin Badea	Profesor
3.	Cristian Mitoi	Profesor
4.	Petrică Nancu	Profesor

Tabel 7.1.1 - Resursa umană a organizației testate

De asemenea, în cadrul prezentei organizații există și un anumit număr de resurse fizice ce sunt utilizate având scopul executării cu succes al activităților. Aceste resurse sunt definite în următorul tabel după cum urmează:

Nr. Crt.	Denumire	Cantitate
1.	Amfiteatru A	5
2.	Switch Ethernet	5
3.	PC Asus	10
4.	Sala L1	5

Tabel 7.1.2 - Resursele materiale ale organizației

În continuare, vom defini un număr de n=11 activități periodice sau neperiodice pentru care dorim să obținem o ordonare care să nu încalce constrângerile de resurse sau relațiile de precedență pe care le definim. În următorul tabel putem vizualiza detaliile fiecărei activități definite:

Nr. crt.	Denumire	Periodicitate	Durata	Predecesor	Executant
1.	Curs POO C114	Daily	2	Null	Mihai Mihai
2.	Laborator POO C114	No Periodicity	2	Curs POO C114	Valentin Badea
3.	Curs Rețele	Daily	2	Null	Cristian Mitoi
4.	Laborator Rețele	Daily	2	Curs Retele	Petrica Nancu
5.	Consultații Rețele	Daily	1	Null	Mihai Mihai
6.	Practica	Daily	3	Null	Mihai Mihai
7.	Mentenanta Echipamente	Daily	2	Null	Mihai Mihai
8.	Curs AI	No Periodicity	2	Null	Cristian Mitoi
9.	Laborator AI	Daily	2	Curs AI	Petrica Nancu
10.	Curs Sisteme de Operare	No Periodicity	3	Null	Cristian Mitoi
11.	CCNA	Daily	2	Null	Valentin Badea

Tabel 7.1.3 - Tabel cu activitățile definite

În următorul tabel sunt definite cantitățile necesare fiecărei activități pentru a putea fi executată cu succes:

Activitate/Resursă	Amfiteatru A	Switch Ethernet	Sala L1	PC Asus
Curs POO C112	1	0	0	0
Laborator POO C114	0	0	1	8
Curs Rețele	0	2	1	0
Laborator Rețele	0	5	2	0

Consultații Rețele	0	5	1	5
Practica	1	0	0	0
Mentenanta echipamente	0	2	0	5
Curs AI	2	0	0	0
Laborator AI	0	0	2	3
Curs Sisteme de Operare	2	0	0	1
CCNA	2	2	0	0

Tabel 7.1.4 - Alocarea resurselor necesare executării activităților

După ce am introdus toate aceste date în aplicație și ne-am creat proiectul, putem deja să vizualizăm modul în care algoritmul a ordonat activitățile. În continuare, vom prezenta alocarea resurselor pe o anumită zi și, totodată, vom comenta rezultatele obținute.

Fig. 7.3 – Alocarea resursei "Amfiteatru A"

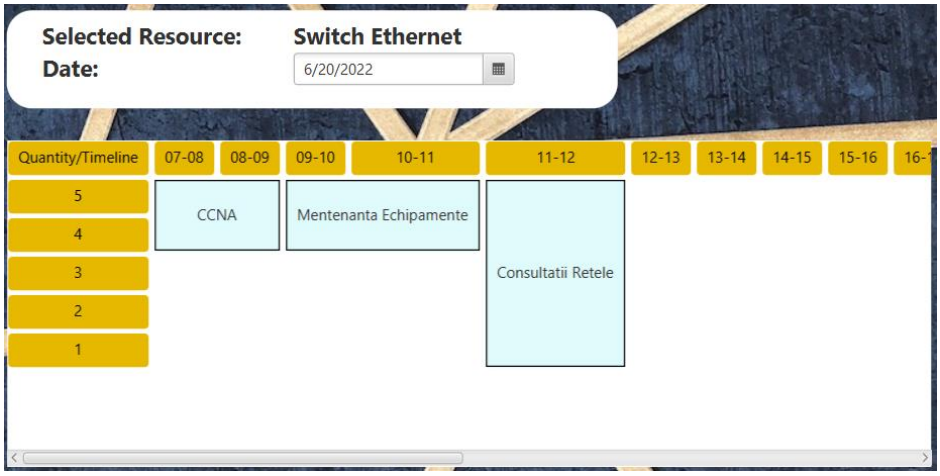

Fig. 7.4 – Alocarea resursei "Switch Ethernet"

Din analiza primelor două resurse putem observa deja faptul că există o activitate ce are nevoie de o anumită cantitate din ambele resurse definite. În cazul de față, este vorba despre activitatea *"CCNA"* ce solicită, conform tabelului 7.1.4, câte două cantități din ambele resurse. Se poate constata astfel, urmărind axa orizontală ce reprezintă timpul din ambele imagini, că nu există ambiguități în ordonare deoarece în ambele cazuri aceasta este programată în primele două unități de timp.

Din prima imagine se poate observa golul lăsat timp de trei unități de timp cauzat de neutilizarea niciunei cantități din prezenta resursă. Vom analiza, așadar, cauza ce produce această zonă liberă. Pentru a putea face comentariile necesare vom avea nevoie să prezentăm și alocarea celorlalte două resurse. Acestea sunt prezentate în Fig. 7.5 și Fig. 7.6 .

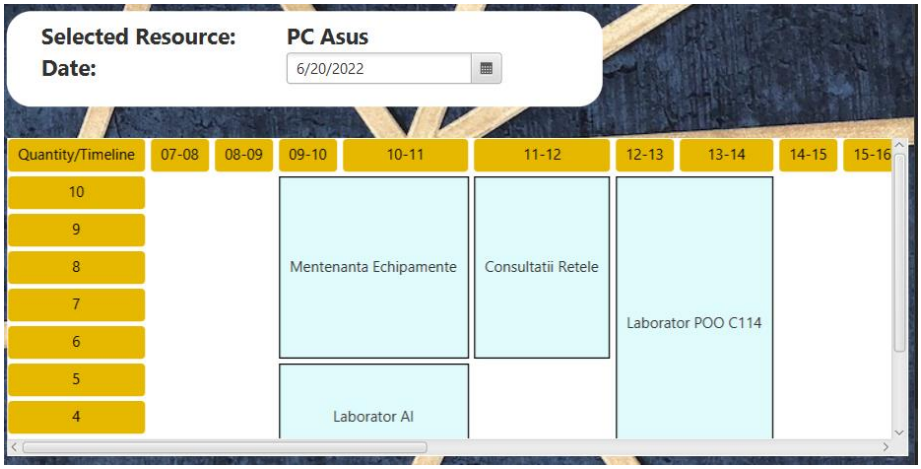

Fig. 7.5 – Alocarea resursei "PC Asus"

Fig. 7.6 – Alocarea resursei "Sala L1"

Astfel, motivul pentru care prima resursă nu este utilizată pentru acel interval de timp este reprezentat de indisponibilitatea resursei umane. Urmărind graficul (Fig. 7.7) specific utilizatorului “Mihai Mihai” se poate observa această continuitate în desfășurarea activitităților propuse:

Fig.7.7 – Organizarea activităților utilizatorului

7.1.2. Realocarea unei resurse

Pe lângă această caracteristică de ordonare a activităților, aplicația oferă și posibilitatea realocării resurselor. Pentru exemplul anterior vom presupune că în cadrul organizației se modifică disponibilitatea resursei “PC Asus” de la $B_i=10$ la $B_i=7$. O astfel de realocare a unei resurse va produce schimbări în întregul proiect.

Fig. 7.8 – Noua ordonare rezultată în urma realocării resursei

7.1.3. Sugestii oferite pentru obținerea unui scenariu fezabil

Se poate observa în Fig. 7.8 faptul că în urma unei modificări o activitate nu mai poate fi ordonată, aceasta solicitând un număr de resurse mai mare decât cele disponibile. Aplicația va oferi utilizatorului o soluție de a reuși să ordoneze și această activitate. În cadrul acestei posibile soluții, aplicația va testa dacă respectiva activitate poate fi ordonată în cazul în care numărul de resurse solicitat poate fi micșorat sau dacă se poate realiza o ordonare în urma modificării duratei de execuție.

În cazul în care utilizatorul este de acord cu soluția propusă de aplicație, algoritmul va fi iarăși executat și vom putea observa nouă ordonare a activităților.

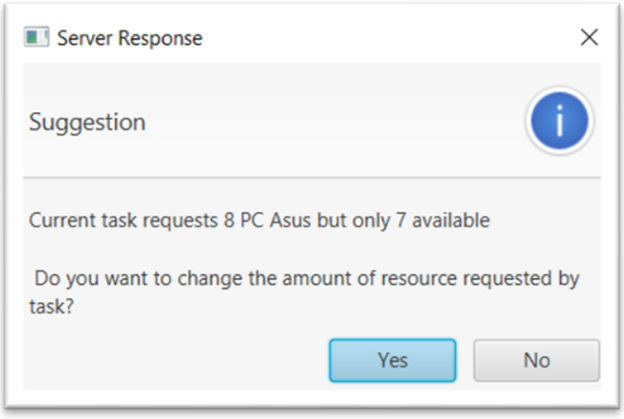

Fig. 7.9 – Sugestia oferită pentru readăugarea activității

În următoarea figură se poate constata ordonarea rezultată în urma modificării numărului de resurse solicitat. Se observă faptul că resursa selectată are în acest moment un număr de *B=7* unități disponibile iar activitatea a fost reintrodusă.

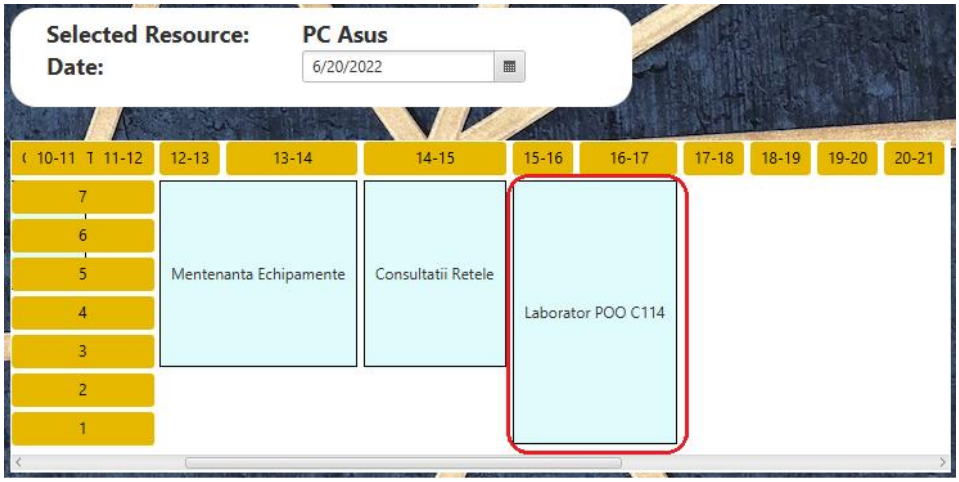

Fig. 7.10 – Readăugarea unei activități ce a suferit modificări

Pentru a putea prezenta a două variantă pe care aplicația o poate oferi este nevoie de introducerea unei noi activități ce va avea următoarele caracteristici:

Denumire	Periodicitate	Durata	Predecesor	Executant	Dată execuție
Proiect Software	No periodicity	17	Practica	Mihai Mihai	21/6/2022

Pe lângă caracteristicile de bază, această activitate va utiliza următoarele resurse conform tabelului:

Amfiteatru A	Switch Ethernet	Sala L1	PC Asus
0	0	1	2

În urma creării acestei activități, aceasta va apărea, precum cea anterioară, în interiorul tabelului ce conține activitățile imposibil de ordonat. În cazul în care dorim să vizualizăm varianta care ni se propune pentru a obține o ordonare fezabilă, vom obține următorul rezultat:

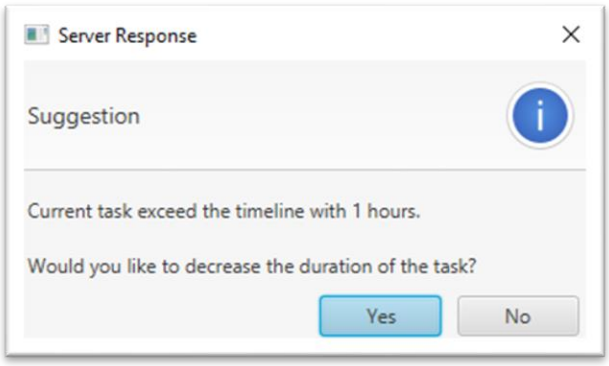

Fig. 7.11 – Sugestie pentru micșorarea execuției unei activități

În cazul afirmativ în care suntem de acord cu soluția propusă, vor fi executați aceeași pași ca și în cazul precedent. După ce Serverul va trimite răspunsul către Client, acesta va putea vizualiza cum a fost ordonată prezenta activitate. Cazul propus de către noi poate fi vizualizat în următoarea figură:

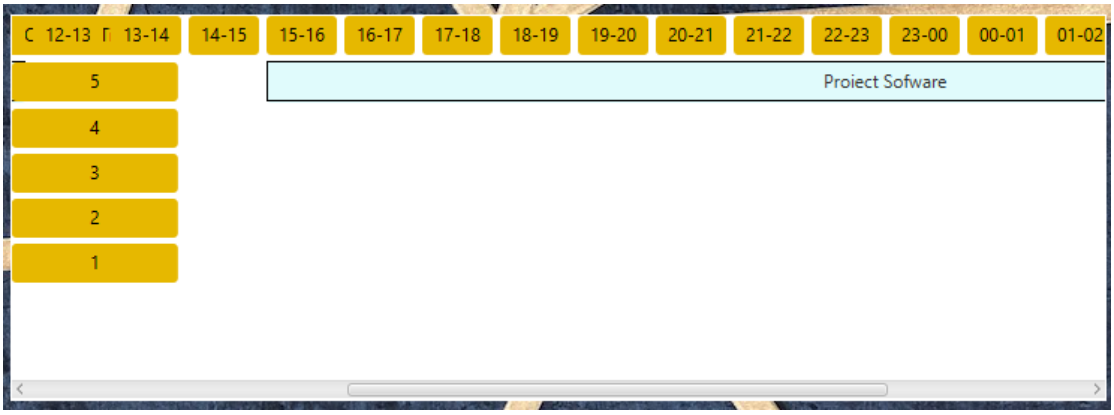

Fig. 7.12 – Ordonarea unei activități în urma micșorării duratei de execuție

Există, totuși, și cazuri pe care aplicația nu le poate modela. Unul dintre acestea este reprezentat de situația în care s-ar impune anumite constrângeri de timp de către angajații organizației. Spre exemplu, un angajat ar putea pune condiția ca toate activitățile atribuite să se încadreze într-un anumit interval de timp. În acest caz, algoritmul nu poate realiza o ordonare care să respecte cerințele impuse deoarece acesta nu este gândit pentru astfel de situații.

Totodată, după cum am menționat și în cadrul Capitolului 4 în care am detaliat problema RCPSP, algoritmul nu permite întreruperea activităților și nici schimbarea pe parcursul desfășurării activității a numărului de resurse utilizat.

În concluzie, testarea aplicației a fost în primă fază de tip unit testing, testându-se individual fiecare componentă în parte după ce aceasta a fost implementată. De asemenea, a fost nevoie și de o testare de tip integration testing pentru partea principală a aplicației, aceea de ordonare automată a activităților deoarece implica mai multe module.

8. Concluzii

Scopul aplicației a fost acela de a crea o soluție software de management de proiecte prin intermediul căreia să se poată oferi un exemplu de ordonare a activităților de care s-ar putea ține cont astfel încât să nu existe încălcări ale constrângerilor definite pentru resursele utilizate.

Aplicația își atinge cu succes scopul principal pentru diferite modele de taskuri. Pe lângă backend-ul avut, partea de frontend a aplicației se prezintă cu o interfață intuitivă pentru orice tip de utilizator. Deși se pot face optimizări atât pe partea de cod, cât și pe partea de interacțiune cu utilizatorul, aplicația reprezintă un bun punct de plecare pentru rezolvarea problemei propuse și pentru îmbunătățiri viitoare.

Limitările aplicației sunt date de modul în care este definită problema standard RCPSP (Resource-Constrained Project Scheduling Problem) deoarece există anumite cazuri ce pot apărea în practică și nu sunt tratate prin modul în care aceasta este definită. Un caz concret este reprezentat de constrângeri de timp ce pot apărea pentru anumite activități, dar aplicația nu poate ține cont de acestea în momentul în care realizează ordonarea. Astfel, aplicația ar beneficia de un plus în cazul în care s-ar implementa și anumite variații ale problemei propuse. Modificări precum: definirea de constrângeri de timp pentru anumite activități sau posibilitatea întreruperii unei activități și reluarea acesteia într-un moment ulterior ar reprezenta un beneficiu important.

9. Bibliografie

[1] Carlos Montoya, Odile Bellenguez-Morineau, Eric Pinson & David Rivreau, “Handbook on Project Management and Scheduling Vol. 1”, Springer, 2014

[2] Christian Artigues, Sophie Demassey & Emmanuel Neron, “Resource-Constrained Project Scheduling: Models, Algorithms, Extension and Applications”, ISTE, 2008

[3] M. L. Dertouzos and A. K. Mok, "Multiprocessor online scheduling of hard-real-time tasks," in IEEE Transactions on Software Engineering, vol. 15, no. 12, Dec. 1989

[4] Sonke Hartmann, Dirk Briskorn, “An updated survey of variants and extensions of the resource-constrained project scheduling problem”, European Journal of Operational Research, 2021

[5] Sonke Hartmann, “Project scheduling with resource capacities and requests varying with time: a case study”, Springer, 2013

[6] Harold Kerzner, „Project Management: A Systems Approach to Planning, Scheduling, and Controlling”, Wiley, 2017

[7] M. Dertouzos, “Control robotics: The procedural control of physical processes”, Proc. IFIP Cong., 1974,

[8] https://www.pmi.org/about/learn-about-pmi/what-is-project-management

[9] Project Management Institute, „A Guide to the Project Management Body of Knowledge”, 2017

[10] https://trello.com/

[11] https://app.asana.com/

[12] https://www.atlassian.com/software/jira

[13] https://www.jetbrains.com/help/idea/maven-support.html

[14] Badea Mihai-Valentin, “Implications of Resource Reallocation to Feasibility of tasks’ scenarios”, Students’ International Conference “CERC”, București, 2022

10. Anexe

Anexa A

```
public boolean algorithm() {
    ArrayList<Integer> listaCompletion = new ArrayList<>();
    for (int i = 0; i < listaTaskuri.size(); i++) {
        listaCompletion.add(listaTaskuri.get(i).getCompletionTime());
    }
    Collections.sort(listaCompletion);
    /** Sort on starttime*/
    Collections.sort(listaTaskuri);

    for (TaskReal taskToBeDeleted : listaTaskuri) {
        for (Resource r : listaResurse) {
            if (taskToBeDeleted.getQuantityOfResourceRequest(r.getID()) >
r.getCantitate()) {
                listaTaskuriImposibleToSchedule.add(taskToBeDeleted);
            }
        }
    }
    for (TaskReal taskToBeDeleted : listaTaskuriImposibleToSchedule) {
        listaTaskuri.remove(taskToBeDeleted);
    }
    List<Integer> listWithoutDuplicates =
listaCompletion.stream().distinct().collect(Collectors.toList());

    int old_value = listaTaskuri.get(0).getStartTime();
    TaskReal tobeModified = listaTaskuri.get(0);

    for (int i = 0; i < listWithoutDuplicates.size(); i++) {

        int t = listWithoutDuplicates.get(i);
        old_value = listaTaskuri.get(0).getStartTime();
        tobeModified = listaTaskuri.get(0);

        for (int k = 0; k < listaResurse.size(); k++) {
            int o = 0;
            ArrayList<TaskReal> F = new ArrayList<>();

            for (int j = 0; j < listaTaskuri.size(); j++) {
                int start = listaTaskuri.get(j).getStartTime();
                int completion = listaTaskuri.get(j).getCompletionTime();
                int resRequest =
listaTaskuri.get(j).getQuantityOfResourceRequest(listaResurse.get(k).getID())
;

                if ((start < t) && (completion >= t) && (resRequest > 0)) {

                    o += resRequest;
                    F.add(listaTaskuri.get(j));

                    if (listaTaskuri.get(j).getStartTime() > old_value) {
                        tobeModified = listaTaskuri.get(j);
                        old_value = listaTaskuri.get(j).getStartTime();
```

```
                    } else {
                        int area = tobeModified.getCompletionTime() *
tobeModified.getQuantityOfResourceRequest(listaResurse.get(k).getID());
                        int curentArea =
listaTaskuri.get(j).getCompletionTime() *
listaTaskuri.get(j).getQuantityOfResourceRequest(listaResurse.get(k).getID())
;
                        if (curentArea > area) {
                            tobeModified = listaTaskuri.get(j);

                        }
                    }
                }

                if (o > listaResurse.get(k).getCantitate()) {
                    int currentStart = tobeModified.getStartTime();
                    currentStart++;
                    tobeModified.setStartTime(currentStart);
                    return false;
                }
            }
        }
    }
    return true;

}
```

Anexa B

```
public void RecursiveFunction(int timeslot, int placedTimeslot, String[][] m,
int noTaskPut, Resource res) {
        if (found == 1)
            return;
        if (noTaskPut == listaTaskuri.size()) {
            found = 1;
            return;
        }
        if (timeslot > getMakespan())
            return;
        if (getNoTimeslot(timeslot) == 0)
            RecursiveFunction(timeslot + 1, 0, m, noTaskPut, res);
        else if (placedTimeslot == getNoTimeslot(timeslot))
            RecursiveFunction(timeslot + 1, 0, m, noTaskPut, res);
        else {
            TaskReal task = getTaskNoTimeslot(timeslot, placedTimeslot);

            int nrResurse = task.getQuantityOfResourceRequest(res.getID());

            for (int rand = 0; rand < (res.getCantitate() - nrResurse + 1);
rand++) {
                int start = task.getStartTime();
                int durata = task.getDuration();

                boolean result = checkSpace(rand, start, nrResurse, durata,
m);
```

```
                if (result) {
                    for (int linie = rand; linie < rand + nrResurse; linie++)
{
                        for (int coloana = start; coloana < start + durata;
coloana++) {
                            m[linie][coloana] = task.getName();
                        }
                    }

                    RecursiveFunction(timeslot, placedTimeslot + 1, m,
noTaskPut + 1, res);

                    for (int linie = rand; linie < rand + nrResurse; linie++)
{
                        for (int coloana = start; coloana < start + durata;
coloana++) {
                            m[linie][coloana] = "0";
                        }
                    }
                }
            }
        }
    }
```

Anexa C

```
Service<ProjectModel> service = new Service<ProjectModel>() {
    @Override
    protected Task<ProjectModel> createTask() {
        return new Task<ProjectModel>() {
            @Override
            protected ProjectModel call() throws Exception {
                SenderText data = new SenderText();
                int id = Integer.parseInt(data.getData());

                Client client = Client.getInstance();
                JSONObject tosend = new JSONObject();

                tosend.put("Type", "Get Project");
                tosend.put("IDproject", id);

                GsonBuilder gsonBuilder = new GsonBuilder();
                gsonBuilder.registerTypeAdapter(LocalDate.class, new
LocalDateSerializer());
                gsonBuilder.registerTypeAdapter(LocalDate.class, new
LocalDateDeserializer());
                Gson gson = gsonBuilder.setPrettyPrinting().create();

                client.sendText(tosend.toString());
                String response = client.receiveText();


                ProjectModel project = gson.fromJson(response,
ProjectModel.class);
                projectLocal = project;
```

```
                JSONObject tosend3 = new JSONObject();
                tosend3.put("Type", "Get Lista Taskuri Reale Project");
                tosend3.put("IDproject", projectLocal.getID());
                client.sendText(tosend3.toString());
                String receive2 = client.receiveText();

                Type ArrayListRealTasks = new
TypeToken<ArrayList<TaskRealModel>>() {
                }.getType();
                ArrayList<TaskRealModel> listaTaskuriRealeProject =
gson.fromJson(receive2, ArrayListRealTasks);
                projectLocal.setListaTaskuriReale(listaTaskuriRealeProject);

                return project;
            }
        };
    }
};
```