**UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA**
**FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ**
**SPECIALIZAREA Informatică**

# LUCRARE DE LICENŢĂ

# Optimizarea și organizarea timpului personal folosind Algoritmi Genetici

**Conducător științific**
**Lect. Dr. Adriana Mihaela COROIU**

*Absolvent*
*Munteanu Ecaterina*

2024

ABSTRACT

This bachelor thesis presents a study on Genetic Algorithms used in optimizing our personal time and their appliance in a mobile application. The purpose of the app is to automatise the process of scheduling our daily activities, by using our past patterns and routines to generate our daily plan. In this paper, there are studied two versions of a GA and the difference between them is how the space of time is represented: one considers the activities done in periods of the day and takes into account the location too, while the other organises activities at specific time intervals, but without considering the location. Also, at the end of this thesis there are presented the results of the experiments of the algorithm and their analysis.

The mobile application integrates the version of genetic algorithm which returns better results, offering to the user an AI tool to generate the schedule of his daily activities. The UI is intuitive and easy to use, allowing the user login/logout/register, to view his list of activities per each day, mark activities as completed or not and to delete, edit or add one. When using the AI tool, the user has a preview of the generated schedule and can choose either to keep it, retry or cancel and no changes are made to his list of activities.

I declare that this paper is the result of my own work and research and does not contain any plagiarism. It respects the Romanian legislation and the international conventions corresponding to copyright. All bibliographic sources, including data, figures and tables, are cited in the text of the thesis and in the reference section.

# Cuprins


**1 Introducere** **1**

1.1 Utilizarea instrumentelor IA generative . . . . . 2

**2 Aspecte teoretice despre algoritmii genetici (GA)** **3**

2.1 Ce este un GA? . . . . . 3

2.2 Componentele de bază . . . . . 4

2.2.1 Inițializarea populației de cromozomi . . . . . 4

2.2.2 Funcția de fitness . . . . . 5

2.2.3 Operatorii genetici . . . . . 5

2.2.4 Fenotip și genotip . . . . . 7

2.2.5 Convergența . . . . . 7

2.2.6 Generațiile . . . . . 7

2.3 Aplicații ale algoritmilor genetici . . . . . 8

2.4 GA în planificarea automatizată . . . . . 8

**3 Lucrări similare despre organizarea timpului personal** **10**

3.1 Organizarea zilnică a timpului sub forma unui joc . . . . . 10

3.2 Algoritm genetic în planificarea zilnică . . . . . 11

3.3 TaskDo: Sistem ce recomandă obiective zilnice . . . . . 12

3.4 Concluzii . . . . . 13

**4 Aplicația de organizare a timpului personal** **15**

4.1 Cazuri de utilizare . . . . . 15

4.2 Interfața aplicației și modul de utilizare al aplicației . . . . . 16

4.3 Detalii de implementare a aplicației . . . . . 21

4.3.1 Dezvoltarea algoritmilor genetici . . . . . 21

4.3.2 Backend . . . . . 26

4.3.3 Frontend . . . . . 29

4.4 Tehnologii utilizate . . . . . 31

4.4.1 React . . . . . 31

4.4.2 Flask . . . . . 32

4.4.3 MongoDB . . . . . 32
4.5 Comparare cu aplicații din literatura de specialitate . . . . . 33

**5 Experimente** **34**
5.1 Seturi de date . . . . . 34
5.1.1 ”Ms-Latte” . . . . . 34
5.1.2 ”Life tracking” . . . . . 36
5.2 Rezultate experimentale . . . . . 39
5.2.1 GA cu spațiul temporal sub formă de perioade ale zilei . . . . 39
5.2.2 GA cu spațiu temporal precizat și exact . . . . . 41

**6 Concluzii și direcții viitoare de dezvoltare** **44**

**Bibliografie** **46**

# Capitolul 1

# Introducere

Una dintre cele mai mari probleme din viața activă a adulților este organizarea și planificarea activităților corespunzător astfel încât toate scopurile sunt atinse. Ca oameni, diferiți factori externi și interni ne afectează în fiecare moment al zilei. Aceștia pot interfera cu deciziile noastre și obținerea obiectivelor finale. Majoritatea oamenilor aleg să mențină o rutină zilnică datorită faptului că micșorează stresul și presiunea de a face decizii zilnice. În schimb, totul în jurul nostru este imprevizibil și putem întâmpina oricând situații în care este necesară rezolvarea unor probleme. Cum ne setăm obiective realistice care pot fi realizate într-o zi? Cum decidem importanța lor și prioritățile?

Fiind studentă, mă întâlnesc cu aceste întrebări zilnic. Fiecare zi arată diferit din cauza orarului, nevoile diferă de la o zi la alta, preocuparea cu facultatea și atingerea tuturor deadlineurilor, toate aceste lucruri ne pun în dificultate în organizarea noastră astfel încât să le putem face pe toate într-un timp eficient. Acesta este unul dintre motivele pentru care am ales să implementez o aplicație mobilă care ne poate ajuta să scăpăm de un stres zilnic și de sentimentul constant de presiune.

Majoritatea oamenilor au un stil de viață activ și doresc să-și rezolve problemele și nevoile zilnice cât mai eficient, atât din punct de vedere al timpului, dar și al costului. Existența diferitor aplicații de planificare s-au dovedit a fi eficiente in organizarea timpului pentru majoritatea persoanelor. Multe dintre ele au format de calendar sau listă, reamintesc utilizatorului de planurile acestuia, fie prin notificări sau alarme, însă problema utilizatorului de a se organiza cât mai eficient rămâne nerezolvată.

Scopul acestei lucrări este de a oferi o soluție pentru optimizarea și organizarea timpului personal folosind algoritmi genetici pentru planificarea automată. Astfel se ușurează toată munca utilizatorului de a se gândi cum să aranjeze toate activitățile într-un mod eficient. Cum organizarea personală diferă de la persoană la persoană, algoritmul ține cont de rutina din trecut ale utilizatorului și de preferințele acestuia.

Lucrarea este structurată pe mai multe capitole, începând cu partea teoretică pentru a prezenta algoritmii genetici, după care se continuă cu partea practică, unde este discutată implementarea aplicației, și se finalizează cu capitolul de experimente, în care se prezintă seturile de date și rezultatele experimentale.

Capitolele lucrării sunt împărțite astfel:

2. *Aspecte teoretice despre algoritmii genetici (GA)* explică cum algoritmii genetici funcționează și conceptele de bază despre aceștia, după care se discută aplicabilitatea lor în viața reală. De asemenea, se discută mai în detaliu cum sunt utilizați în planificarea automată.

3. *Lucrări similare despre organizarea timpului personal* este un capitol în care se prezintă trei lucrări similare, pe tematica de organizare a timpului personal, care conțin și aplicații implementate. Pentru fiecare se evidențiază avantajele și dezavantajele, împreună cu îmbunătățirile pe care aplicația ce susține această lucrare, poate să le facă.

4. *Aplicația de organizare a timpului personal* reprezintă capitolul practic ce prezintă structura aplicației, modul de utilizare a acesteia, detalii despre implementare în care se descriu algoritmii și părțile de backend și frontend, iar în final, se prezintă tehnologiile alese și motivul.

5. *Experimente* se concentrează pe descrierea fiecărui set de date utilizat în crearea algoritmilor genetici și cum acestea au fost procesate și modificate. După care, o secțiune este dedicată pentru prezentarea și analiza rezultatelor experimentale.

În final sunt prezentate concluziile studiului efectuat pe parcursul scrierii acestei lucrări și îmbunătățirile ce pot fi efectuate în urma continuării implementării aplicației mobile ce susține această lucrare de licență.

## 1.1 Utilizarea instrumentelor IA generative

Declar că nu am folosit instrumente IA generative în elaborarea acestei lucrări. Lucrarea este scrisă pe cont propriu pe baza studiului personal asupra temei, iar fiecare sursă este menționată în secțiunea *Bibliografie*.

# Capitolul 2

# Aspecte teoretice despre algoritmii genetici (GA)

## 2.1 Ce este un GA?

Algoritmii genetici(GA) sunt algoritmi evolutivi inspirați din selecția biologică și evoluția naturală, reprezentând un instrument pentru rezolvarea problemelor de căutare și optimizare [Car14]. Aceștia sunt utilizați în general atunci când spațiul problemei este vast și soluția exactă este dificil de calculat într-un timp optim. Ca și în genetică, procesele acestui algoritm sunt aleatoare, însa prezintă un nivel de control în căutarea soluției, ceea ce determină un algoritm cu mult mai puternic și eficient decât unul de căutare aleator.

Dar care este diferența dintre un algoritm genetic și diferite metode de căutare și optimizare? [Fan95] oferă răspunsul la această întrebare prin menționarea următoarelor aspecte despre GA:

- Algoritmii genetici lucrează cu o codare a setului de parametri, ci nu cu parametrii în sine
- Se caută într-o populație întreagă de puncte de vedere, nu doar unul singur
- Folosesc o funcție obiectivă de evaluare a performanței fiecărei soluții, fără a necesita informații suplimentare
- Se bazează pe reguli probabilistice, ci nu reguli deterministe
- Mutație aleatoare pentru cromozomii unei generații noi

Componentele pentru construirea unui GA ce utilizează principiile de bază din genetică sunt:

- O funcție de evaluare pentru optimizare

- O populație de cromozomi
- Selecția cromozomilor pentru reproducere
- Încrucișare pentru producerea următoarei generații de cromozomi
- Mutație aleatoare pentru cromozomii unei generații noi

Pentru înțelegerea mai clară, am realizat o figură 2.1 cu ordinea fiecărui proces și cum poate fi vizualizat acesta.

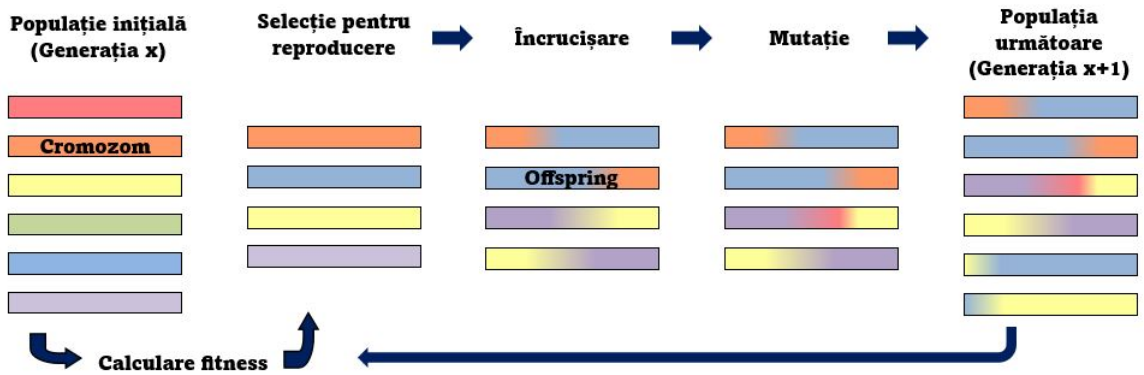



Figura 2.1: Componentele de bază ale unui Algoritm Genetic

## 2.2 Componentele de bază

### 2.2.1 Inițializarea populației de cromozomi

Primul pas este de a inițializa populația de cromozomi, iar dimensiunea acesteia depinde de problemă. Spațiul de căutare este un factor care decide dimensiunea populației, aceasta având în general valori cuprinse între 40-100 de indivizi. Pentru problemele mai complexe, sunt recomandate populații de dimensiunea 400 sau mai mari [Cho09].

Un cromozom este compus din mai multe gene, reprezentând o soluție candidat pentru problema care algoritmul încearcă să o rezolve. Fiecare cromozom este codat în funcție de logica programatorului, dar în general sunt reprezentați sub formă binară (secvență de '0' sau '1') sau vector de variabile. Dacă problema are dimensiunea N, de obicei cromozomul va fi codat drept un vector de N variabile

$$cromozom = [p_1, p_2, ..., p_N]$$

unde $p_i$ reprezintă valoarea parametrului $i$. Mai nou, reprezentările pot fi și sub formă permutări, numere reale și multe alte tipuri de obiecte [Car14].

O mulțime de cromozomi reprezintă o populație, iar când algoritmul începe, aceasta este inițializată cu cromozomi aleși aleator, creând prima generație (populația inițială).

### 2.2.2 Funcția de fitness

După inițializarea populației, fiecare cromozom este evaluat prin funcția de fitness. Funcția de evaluare este cheia unui algoritm genetic, deoarece influențează ce cromozomi vor fi selectați pentru reproducere. Aceasta evaluează fiecare cromozom și îi asignează un 'fitness', o valoare care reprezintă cât de potrivită este soluția curentă în rezolvarea problemei. Pentru crearea unei funcții de fitness se iau în considerare toți parametrii ce afectează alegerea cromozomilor și se alege o formulă astfel încât să satisfacă toate condițiile necesare ce vor duce la găsirea celei mai bune soluții.

### 2.2.3 Operatorii genetici

În faza de reproducere a unui GA, sunt selectați indivizi (soluții potențiale) din populație și recombinați [BBM93], astfel producând 'offsprings' (urmași) ce vor compune următoarea generație. Părinții sunt selectați aleator din populație, favorizând cei cu fitness mai bun. Indivizii mai potriviți pentru soluție vor fi aleși de mai multe ori, iar cei mai slabi, rareori sau chiar deloc. După ce cei doi părinți sunt selectați, procesul de reproducere este compus din funcțiile de încrucișare (crossover) și mutație.

**Încrucișarea**, cunoscută și sub numele de 'Crossover', este cel mai important operator în algoritmii genetici [Fan95] deoarece produce recombinarea prin interschimbarea unor segmente într-o pereche de cromozomi. Se disting patru tipuri de încrucișări:

- *One-point crossover*: Încrucișare într-un singur punct necesită doi indivizi pe care îi va segmenta în două părți, într-un punct ales aleator pentru amândoi. Se interschimbă primul segment de la primul individ, cu primul de la al doilea individ și se procedează la fel și pentru cel de-al doilea segment.

  În figura 2.2 este un exemplu de încrucișare pentru doi cromozomi reprezentați în format binar în punctul ales aleator înainte de ultimele două gene.

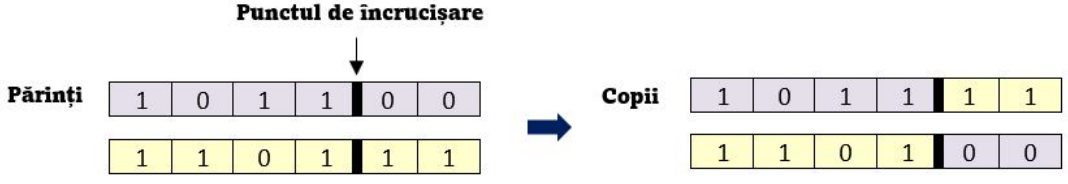



Figura 2.2: Încrucișarea într-un singur punct (Single-point crossover)

- *Two-point crossover*: Încrucișarea în două puncte este asemănătoare cu cea într-un punct, în schimb se aleg două puncte aleator. Se interschimbă doar genele prezente între cele două poziții. Această metodă oferă posibilitatea de a păstra părțile de început și final al cromozomilor și să se interschimbe doar mijlocul, ceea ce nu este posibil cu încrucișarea într-un singur punct [Fan95].

- *N-point crossover*: Această încrucișare este similară cu cele două tipuri de crossover prezentate anterior, cu excepția că se aleg $n$ poziții aleator și se interschimbă doar segmentele aflate între o poziție impară și una pară. Cele dintre un punct par și impar nu se modifică.

- *Uniform crossover*: Pentru a avea o încrucișare uniformă, fiecare genă din primul părinte are o șansă de 50% de a fi schimbată cu gena corespunzătoare celui de-al doilea părinte.

În urmarea acestui proces rezultă doi cromozomi noi, amândoi moștenind o parte de la ambii părinți. În general, probabilitatea de încrucișare variază între 0.6 și 1.0, astfel fiecare individ având șansa de a-și transmite genele următoarei generații. Conform teoriei din genetică, dacă părinții sunt puternici din punct de vedere al scorului de evaluare, atunci urmașii lor (offspring) vor fi și mai buni. Folosind această logică, prin procesul de încrucișare se asigură găsirea unei soluții mai bune.

**Mutația** are scopul de a perturba indivizi și de a explora noi soluții posibile pentru rezolvarea problemei. Aceasta este aplicată fiecărui individ după încrucișare, cu o probabilitate foarte mică aleasă (de obicei 0.001). O genă aleasă aleator este alterată cu o altă valoare respectivă parametrului. Deoarece mutația nu reprezintă o componentă esențială într-un algoritm genetic, se poate evita utilizarea ei în schimbul utilizării mai multor populații inițializate aleator. Includerea operatorului de mutație în GA nu garantează eficientizarea și găsirea unei soluții mai bune, dar nici nu are un efect negativ deoarece întotdeauna indivizii cei mai puternici vor fi selectați pentru reproducere[Cho09]. Astfel, în cazul în care prin mutație se ajunge la un individ mai slab, acesta riscă să nu se mai reproducă.

Păstrând exemplul utilizat anterior pentru încrucișare, în figura 2.3 este ilustrată mutația pe un cromozom binar.


Figura 2.3: Mutație cromozom

**Inversarea** este cel de-al treilea operator care afectează copii în a fi diferiți față de părinți. Acesta acționează pe un singur cromozom, ca o tehnică de reordonare. Se aleg aleator două poziții din interiorul cromozomului și se inversează genele dintre

cele două puncte. Deși a fost studiat acest operator, nu s-a dovedit a fi o componentă cea poate îmbunătăți rezultatele unui algoritm genetic [Fan95].

### 2.2.4 Fenotip și genotip

În algoritmii genetici, obiectele codificate sunt numite genotip, iar obiectele în sine se află sub numele de fenotip[Fan95]. În genetică, genotipul conține informația necesară pentru construirea unui organism, adică fenotipul[BBM93].

### 2.2.5 Convergența

Convergența reprezintă momentul în care fitness-ul celui mai bun individ și a celor care se află în medie se îndreaptă spre un nivel global optim după ce populația a evoluat prin generații succesive. O genă se consideră conversă atunci când 95% din populație are aceeași valoare de fitness. Astfel, o populație este conversă atunci când toate genele sunt converse[BBM93]. În figura 2.4 se ilustrează cum prin rularea unui algoritm genetic, odată cu convergența populației, scorul de evaluare a unui individ în medie îl atinge pe cel al individului cel mai bun. Convergența este un alt factor care influențează dimensiunea populației, care poate să conveargă mai curând sau mai târziu, depinzând de problemă.

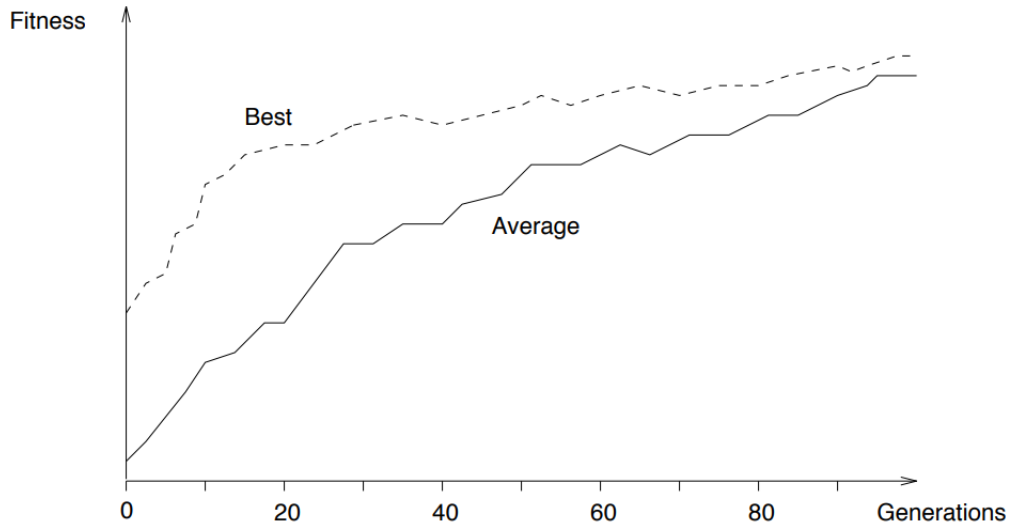



Figura 2.4: Ilustrarea rulării unui GA în general [BBM93]

### 2.2.6 Generațiile

După parcurgerea tuturor pașilor prezentați anterior, se reiau din nou, începând cu evaluarea cromozomilor din noua generație. Fiecare generație reprezintă populația

alcătuita din cromozomii rezultați din reproducerea cromozomilor părinți din populația anterioară. Nu există o metodă strictă pentru determinarea numărului necesar de rulări al unui GA. De obicei, problemele simple pot converge la soluții bune din 20-40 de generații, iar cele mai complexe, pot depăși 400 de generații [Cho09].

## 2.3 Aplicații ale algoritmilor genetici

Pentru ilustrarea flexibilității algoritmilor genetici, [BBM93] listează câteva aplicații în utilizarea lor, unele care au fost puse în practică și unele care au rămas doar ca temă de cercetare.

**Optimizarea funcției numerice**: Mare parte al studiului tradițional al unui GA se focusează în această zonă, deoarece algoritmii genetici s-au dovedit că depășesc și sunt mult mai puternici decât tehnicile de optimizare convenționale pentru funcțiile dificile, discontinue, multimodale și cu variații aleatoare[BBM93].

**Procesarea imaginilor**: Un exemplu interesant menționat de [BBM93] este utilizarea neobișnuită a unui GA în criminalistică. Algoritmul va genera imagini cu criminali suspecți, iar martorul selectează două care i se par similare cu înfățișarea criminalului. Aceste două imagini vor fi utilizate pentru generarea altui set de chipuri, care reprezintă următoarea generație. Astfel, martorul acționează drept funcția de evaluare și el va fi cel care va controla convergența algoritmului spre găsirea imaginii corecte.

**Optimizarea combinatorică**: Cea mai studiată de problemă de acest fel este 'travelling salesperson' care constă în găsirea celui mai scurt drum pentru vizitarea unui grup de orașe.

## 2.4 GA în planificarea automatizată

Algoritmii genetici au fost aplicați în diferite domenii datorită abilității lor de a rezolva probleme NP și NP-H [AF18]. În planificarea automatizată, mare parte din algoritmii genetici rezolvă problema planificării unui drum din cauza existenței multor situații reale ce necesită o soluție la o astfel de problemă.

Printre exemple se află planificarea drumului pentru vehicule aeriene fără pilot, unde algoritmul genetic caută cea mai bună cale de a ajuta un avion să aterizeze în cazul unei situații de urgență [dSAdSATW15]. Un alt exemplu este găsirea celui mai bun drum într-un supermarket pentru a reduce timpul de cumpărături pentru cumpărători [CLH15] sau utilizarea GA în industria de producție unde ar putea planifica funcționarea liniilor de producție și controla capacitatea lor [KGA14].

Aplicabilitatea algoritmilor genetici în organizarea timpului personal și a plani-

ficării activităților zilnice nu are foarte multe exemple din cauza dificultății de a reprezenta obiectivele drept genele unui cromozom [AF18]. De asemenea, sunt foarte multe perspective din care poate fi privită problema, iar soluțiile pot fi diferite de la un algoritm la altul.

# Capitolul 3

# Lucrări similare despre organizarea timpului personal

Fiecare persoană în ziua de astăzi se confruntă cu problema organizării timpului personal și își propune să realizeze anumite activități zilnice. Cu toții ne setăm niște așteptări și scopuri pe care vrem să le îndeplinim, iar acestea depind în totalitate de noi. Având în vedere că tehnologia este îndeajuns de avansată în prezent, un mod de a rezolva problema planificării obiectivelor zilnice este prin automatizarea acesteia bazată pe rutina fiecărui utilizator.

Câteva studii importante care se concentrează pe optimizarea timpului personal sunt „Deconstructing 'Gamified' Task-Management Applications" [KJN13], „Using Genetic Algorithm to Plan Individuals Temporal and nonTemporal Daily Activities" [AF18], "TaskDo: A Daily Task Recommender System" [KG19]. Fiecare lucrare abordează aceeași problemă din puncte de vedere diferite, oferind informații esențiale și diferite perspective în găsirea unei metode cât mai eficiente de optimizare a timpului personal.

## 3.1 Organizarea zilnică a timpului sub forma unui joc

În lucrarea „Deconstructing 'Gamified' Task-Management Applications" [KJN13] se discută despre o abordare diferită asupra aplicațiilor de organizare a trebuințelor și activităților din viața de zi cu zi, aceasta fiind sub formă de joc. Nefiind destule studii despre eficiența integrării jocurilor în aplicațiile de organizare zilnică, odată cu dezvoltarea acestei lucrări, autorii au fost interesați în testarea a două jocuri în acest format pe un număr limitat de persoane. Cele două aplicații folosite în acest studiu diferă ușor prin gradul de dificultate a utilizării și a motivării în realizarea task-urilor. Scopul a fost să se observe cum răspunde publicul atunci când utilizează aplicațiile.

Prin această abordare, autorii cred ca este încurajată productivitatea, captivarea și concentrarea în realizarea scopurilor personale, de asemenea și satisfacția după ceea ce ducem la final ceea ce ne-am propus să facem.

"Task Hammer" și „Epic Win", cele două jocuri din studiu, au în comun următoarele funcționalități: alegerea unui personaj/avatar, personalizarea acestuia, câștigul de bonusuri cu cât utilizatorul își amintește să realizeze mai multe task-uri. Acestea ajută la motivarea utilizatorului, prin dorința de obținere de bonusuri, cu care acesta își poate personaliza caracterele.

După testarea jocului pe un număr mic de persoane, dar diferite din punct de vedere al sexului, vârstei și cunoștințelor și abilităților în jocuri, și pe diferite platforme(IOS/Android) rezultatele sunt cu mult diferite față de așteptările autorilor. Deși cele doua aplicații sunt baza unei idei promițătoare, s-au dovedit a fi greu de înțeles pentru public și nu se vede o diferență clară în îmbunătățirea productivității în comparație cu aplicațiile simple pentru organizare deja existente (Microsoft Outlook calendar, Google Calendar). Din acest motiv, sunt de părere că implementarea unei aplicații care promovează o interfață ușor de utilizat și funcționalități care ajută utilizatorii în planificarea scopurilor personale, este mult mai compatibilă pentru motivarea și productivitatea utilizatorilor.

## 3.2 Algoritm genetic în planificarea zilnică

„Using Genetic Algorithm to Plan Individuals Temporal and nonTemporal Daily Activities" [AF18] prezintă o problemă similară, din perspectiva vârstnicilor sau a persoanelor cu dizabilități ce pot întâmpina dificultăți în organizarea a unei săptămâni întregi cu activități personale. Utilizatorii pot avea anumite cerințe din partea unui agent extern cum ar fi medicul sau fizioterapeutul, iar efortul zilnic de a memora toate acestea poate fi destul de intens pentru unii. Scopul lucrării este de a proiecta algoritmul astfel încât utilizatorul să aibă cât mai multe zile libere de odihnă și planificarea unor obiective să nu se suprapună cu altele.

În lucrare [AF18] sunt prezentați doi algoritmi genetici, o versiune care nu depinde de niciun spațiu temporal și una care prezintă limitări de intervale de timp. Problema P este reprezentată sub forma unui tuplu P = (p, X, A, C), unde p este planul generat,

$$X = (x_1, x_2, ..., x_n)$$

este un set de variabile care ia valorile dintr-un set de acțiuni

$$A = (a_1, a_2, ..., a_n)$$

ce respectă condițiile

$$C = (c_1, c_2, ..., c_n)$$

În prima versiune, activitățile sunt reprezentate folosind operatori de tip STRIPS, iar în figura 3.1 se poate vizualiza un exemplu.

**Operator {**
**Name : Cook**
**Precondition :**
**(CleanHands)**
**Effect : (Dinner)}**

Figura 3.1: Operator de tip STRIPS [AF18]

Cea de-a doua versiune lucrează cu două tipuri de activități: dependente (conțin precondiții ce trebuie respectate, efecte care au loc în funcție de activitățile anterioare) și independente (nu au nicio relație cu activitățile anterioare). Algoritmul se va opri din iterat fie după ce a parcurs numărul de iterații, fie după ce găsește 3 iterații cu aceeași valoare pentru funcția de evaluare, ceea ce înseamnă că o soluție mai bună nu poate fi găsită [AF18].

Pentru operatorul de selecție se utilizează un scor asociat fiecărei planificări, iar în generațiile următoare se vor lua doar cromozomii cu cel mai mare scor. Selecția are loc după mutație și după ce fiecare plan al săptămânii este evaluat de funcția de fitness. Fiecare versiune a algoritmului are o funcție specifică de încrucișare, care va afecta ordinea activităților în cazul planificării non-temporale sau ziua și timpul activităților sunt afectate dacă planurile se află într-un spațiu temporal.

Fiecare algoritm va genera o planificare a săptămânii, iar utilizatorul are posibilitatea de a refuza acest plan. În acest caz, din cauză că se memorează planificările cu cele mai bune scoruri, următorul va fi recomandat [AF18]. Această abordare oferă flexibilitate utilizatorului și posibilitatea de a planifica în funcție de preferințele sale, iar în plus, planifică săptămânal, nu zilnic. Însă, odată ce utilizatorul face anumite alegeri, ideal ar fi să se țină cont de preferințele acestuia pentru a îmbunătăți acuratețea organizării săptămânale pe viitor.

## 3.3 TaskDo: Sistem ce recomandă obiective zilnice

*TaskDo* este aplicația care susține lucrarea [KG19] în care se consideră că factorul care ne influențează cel mai mult în nerealizarea obiectivelor zilnice propuse este planificarea ineficientă. Existența multor aplicații de planificare nu rezolvă această

problemă, deoarece toate dintre acestea nu ajută utilizatorul cu o organizare eficientă sau cu anumite recomandări potrivite. În schimb, *TaskDo* sugerează ideea de a genera sugestii zilnice de activități pe care utilizatorul s-ar putea să le plănuiască, bazate pe istoricul acestuia de obiective, preferințe și diferiți factori din momentul respectiv.

În timpul procesului de creare aplicației, autorul [KG19] a realizat că există o gamă largă de activități ce pot fi efectuate și a decis să le organizeze pe diferite categorii: fitness și sănătate, social, treburi casnice, spiritual, sarcini, intelectual, fizic. Pe lângă acestea, există și câțiva factori ce pot influența productivitatea utilizatorului și pot ajuta în recomandarea mult mai exactă a activităților: tipul zilei, perioada din timpul zilei. La crearea unui obiectiv, este necesar ca utilizatorul să introducă toate subcategoriile specifice, iar după completarea acestuia se va colecta o recenzie legată de satisfacția în finalizarea sarcinii.

Toate informațiile colectate sunt salvate într-o bază de date, iar în funcție de acestea se calculează un scor pentru fiecare activitate. Cele 2 obiective cu cel mai mare scor, vor constitui recomandările pentru fiecare zi a săptămânii. Alte variabile ce ar putea influența productivitatea utilizatorului sunt: vremea, locația activității, durata etc.

Deși această idee [KG19] este diferită față de lucrările anterioare, utilizatorul beneficiază de ajutorul aplicației în organizarea timpului personal, având în vedere că aceasta se bazează pe preferințele acestuia și modul de viață. În schimb, făcând doar câteva recomandări zilnice, utilizatorul nu este ajutat în deplin cu planificarea activităților în cazul în care acesta are o zi mai ocupată și un număr mai mare de sarcini. De asemenea, nu se ține cont de cazul în care sunt anumite obiective cu o perioadă de timp fixată, ceea ce poate influența sugestiile zilnice.

## 3.4 Concluzii

Lucrările și aplicațiile specifice descrise mai sus vin cu idei și perspective noi în rezolvarea problemei optimizării timpului folosind aplicații sau algoritmi genetici. Însă, fiecare prezintă avantaje și dezavantaje. [KJN13] prezintă una dintre cele mai originale idei, însă prin testarea jocului pe un anumit număr de persoane, s-a dovedit că jucătorii prezintă dificultăți în utilizarea jocului pentru a planifica obiective personale, ceea ce nu rezultă în îmbunătățirea clară a productivității.

Cea de-a doua lucrare [AF18] prezintă ca și soluție utilizarea algoritmilor genetici pentru planificarea unei săptămâni întregi, privind problema atât în cazul unui spațiu non-temporal, dar și în cazul unui spațiu temporal în care există condiții ce trebuie respectate. Algoritmul este proiectat mai mult pentru utilizatorii ce fac parte din categoria vârstnicilor sau a persoanelor cu dizabilități deoarece se concentrează

pe alocarea cât mai multor zile libere și pe precondițiile sarcinilor. Însă, alegerile și preferințele utilizatorului nu sunt luate în considerare atunci când se generează planificările săptămânilor, ceea ce scade din acuratețe pentru fiecare utilizator în parte.

În schimb, [KG19] se concentrează pe preferințele și stilul de viață al utilizatorului pentru a face sugestii de planificare. Chiar dacă recomandările prezintă o acuratețe ridicată, dezavantajul este faptul că nu se consideră obiectivele cu intervale de timp fixate, iar în cazul unei zile ocupate cu multe sarcini, numărul mic de recomandări nu ajută în organizarea sarcinilor.

În concluzie, după studiul avantajelor și dezavantajelor fiecărei soluții studiate în fiecare lucrare, soluția problemei de organizare a timpului personal prezentată în această lucrare îmbină avantaje din studiile discutate anterior cu rezolvări la dezavantajele acestora. Algoritmul genetic este proiectat în două versiuni, pentru date din spațiul non-temporal, dar și pentru cele cu constrângeri de timp. Se utilizează două seturi de date care joacă rolul de istoric al utilizatorului, iar acesta va beneficia de o interfață ușor de utilizat și intuitivă.

# Capitolul 4

# Aplicația de organizare a timpului personal

În acest capitol este descrisă structura aplicației și procesul de dezvoltare a acesteia, începând cu cazurile de utilizare, apoi se descrie modul de utilizare acesteia, după, se intră în detalii despre algoritmul genetic și părțile de frontend și backend.

Aplicația a fost dezvoltată în special pentru persoanele care întâmpină dificultăți în organizarea personală, cum ar fi vârstnicii, sau cele care sunt mult prea ocupate în viața de zi cu zi și le-ar fi de ajutor un instrument pentru organizarea sarcinilor. După ce utilizatorul realizează cu succes autentificarea, fiind necesară o singură dată deoarece datele de conectare sunt salvate local, acesta poate vedea activitățile prezente din ziua curentă sau din alte zile și poate efectua anumite funcționalități ce vor fi descrise detaliat în următorul subcapitol.

## 4.1 Cazuri de utilizare

În cadrul aplicației, utilizatorul poate efectua diverse funcționalități, după ce acesta se autentifică. Acestea descriu cazurile de utilizare ale aplicației, care se pot observa și în figura 4.1 :

- *Login*: Utilizatorul introduce credențialele de autentificare, iar dacă acestea sunt validate de server, autentificarea se realizează cu succes
- *Vizualizare activități*: Odată ce autentificarea este realizată cu succes, utilizatorul poate vizualiza activitățile din data selectată
- *Selectare dată activități*: Utilizatorul poate modifica data de filtrare a activităților
- *Adăugare activitate*: Din pagina principală a aplicației, utilizatorul poate adăuga o nouă activitate

- *Utilizare instrument AI*: Pentru organizarea automată a activităților, fără menționarea timpului, utilizatorul poate utiliza instrumentul de planificare automată
- *Logout*: Pentru a șterge datele de autentificare salvate local, utilizatorul se poate deloga din aplicație
- *Setare status completare activitate*: Utilizatorul poate marca o activitate ca fiind efectuată sau nu, prin bifarea unei căsuțe
- *Editare activitate*: Fiecare activitate poate fi modificată
- *Ștergere activitate*: Orice sarcină se poate șterge de către utilizator

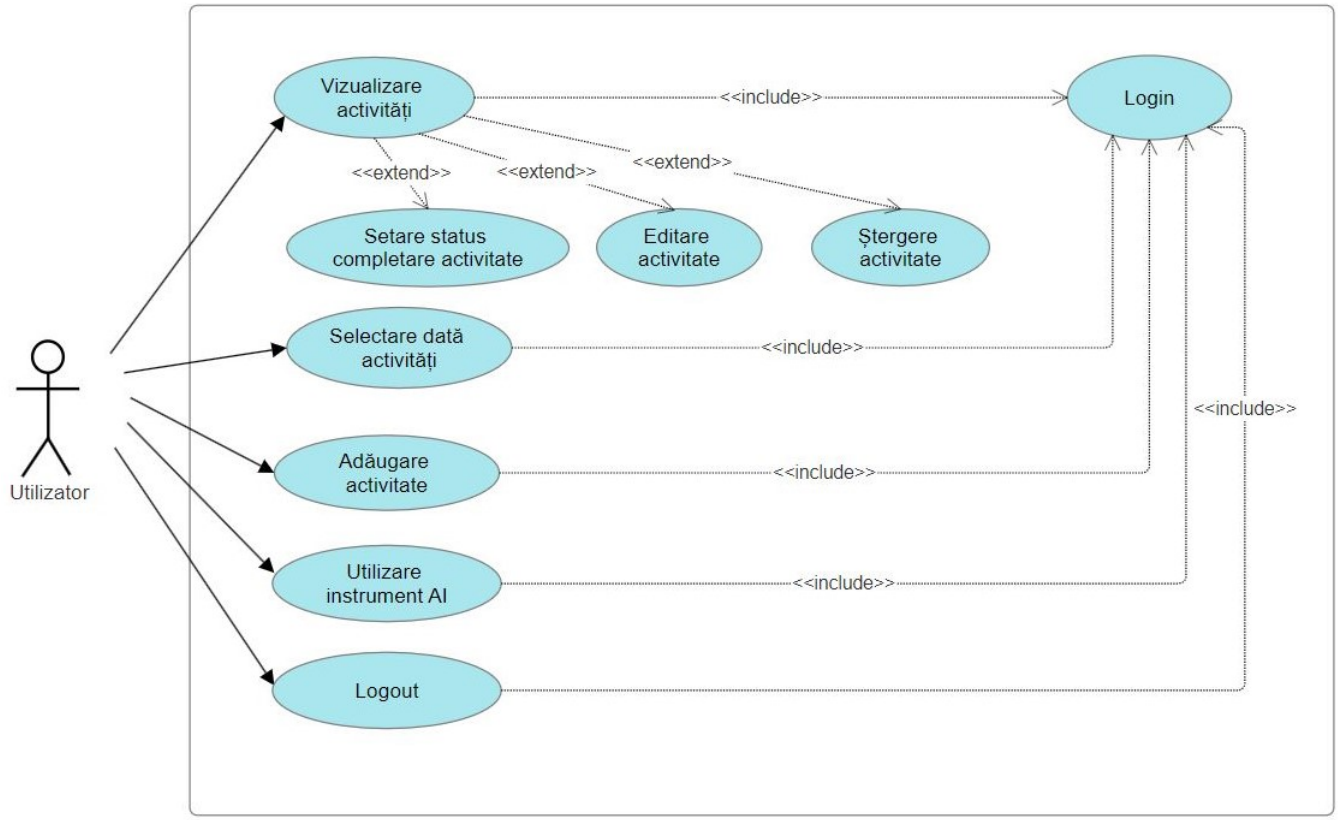



Figura 4.1: Diagrama cazurilor de utilizare

## 4.2 Interfața aplicației și modul de utilizare al aplicației

Pentru a face aplicația cât mai accesibilă pentru utilizator, am căutat aplicații asemănătoare de planificare a obiectivelor zilnice și colectat informații sau studii despre experiența și preferințele utilizatorilor. Unul dintre studii [KVGG20] prezintă răspunsurile la un chestionar adresat utilizatorilor în legătură cu ce și-ar dori de la o aplicație de planificare personală zilnică. După cum se poate observa în figura 4.2, cele mai

cerute funcționalități sunt de a reaminti utilizatorilor de planurile lor și de a avea flexibilitatea de a le accesa de oriunde.

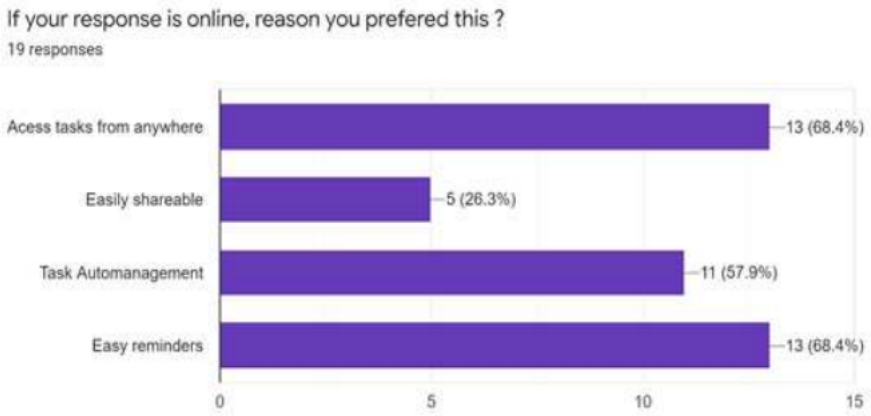



Figura 4.2: Preferințe utilizatori pentru o aplicație de planificare [KVGG20]

Având în vedere acest chestionar, am încercat să dezvolt aplicația de organizare a timpului personal astfel încât majoritatea utilizatorilor să fie cât mai mulțumiți de experiența utilizării acesteia. Fiind o aplicație mobilă, aceasta poate fi ușor de accesat în orice moment de oriunde. De asemenea, algoritmul genetic este un instrument ce utilizatorii îl pot folosi în organizarea automată a activităților. Designul aplicației este minimal, intuitiv, având un număr minim de interacțiuni necesare, cu o tematică de culori simplă astfel încât utilizatorul să nu fie copleșit în utilizarea acesteia.

Pagina de pornire 4.3 conține două câmpuri de input, unul pentru numele de utilizator și unul pentru parolă, care ascunde caracterele cu o bulină pentru securitatea utilizatorului. În urma acționării butonului *Login*, aplicația trimite o cerere la server pentru validarea credențialelor criptate. Dacă acestea se potrivesc cu cele prezente în baza de date, utilizatorul este autentificat cu succes și redirecționat la pagina principală. Prin acționarea butonului *Register* se creează un cont nou de utilizator, prin trimiterea unei cereri la server cu numele de utilizator și parola. În cazul în care numele de utilizator este unic, se adaugă în baza de date și se va afișa un pop-up cu un mesaj de succes. În caz contrar, utilizatorul primește un pop-up cu un mesaj de eroare.

Pagina principală 4.4 se deschide pentru ziua curentă, afișând activitățile corespunzătoare acesteia. În partea de sus este prezent un DatePicker, care, dacă este apăsat de utilizator, deschide un calendar pentru a alege o zi specifică. Pentru navigarea mai rapidă între zilele apropiate de cea curentă, tot în partea de sus, sunt prezente două săgeți. Prin acționarea acestora se modifică data selectată cu o zi înapoi sau cu o zi înainte. Următoarea componentă principală este lista de activități, fiecare activitate având formatul unui card, care poate fi acționat în 3 feluri pentru a

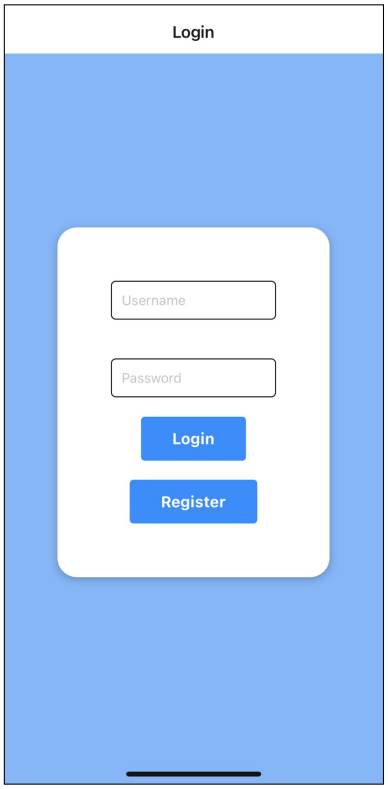



Figura 4.3: Pagina de pornire


Figura 4.4: Pagina principală

efectua diferite funcții:

- *Bifare/debifare checkbox*: Prin bifarea sau debifarea acestei căsuțe, activitatea se marchează ca efectuată, respectiv neefectuată.
- *Glisare card la stânga*: Prin glisarea la stânga a unei activități, aceasta se poate șterge. Se afișează un mesaj pentru a se asigura că utilizatorul intenționează ștergerea activității și se acționează butonul delete pentru a finaliza această operație. În cazul în care utilizatorul dorește să meargă cu un pas înapoi după glisarea la stânga pentru ștergere, acesta poate face swipe înapoi la dreapta pentru funcționalitatea de undo.
- *Click card activitate*: Prin apăsarea unui card, utilizatorul este redirecționat la pagina de editarea a unei activități 4.7.

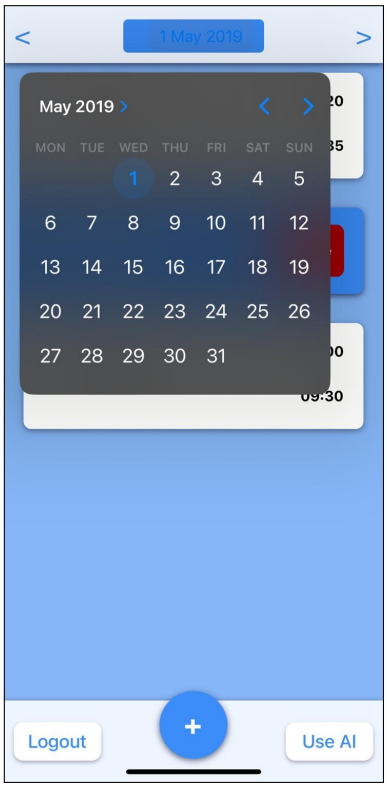



Figura 4.5: Calendar pentru selectarea zilei dorite pentru vizualizarea obiectivelor

Pentru schimbarea datei selectate, utilizatorul fie utilizează săgețile laterale pentru a merge cu o zi înainte sau înapoi, fie apasă pe data din partea de sus care este un buton și va deschide un modal calendar ca în figura 4.5 ce oferă selectarea anului, lunii și a zilei. În urma selecției unei zile, lista de activități vizibilă pe pagina principală se actualizează automat instant.

Ștergerea unei activități din lista prezentă, se face prin glisarea de la dreapta la stânga a cardului respectiv activității. Pentru a ne asigura că această acțiune a fost

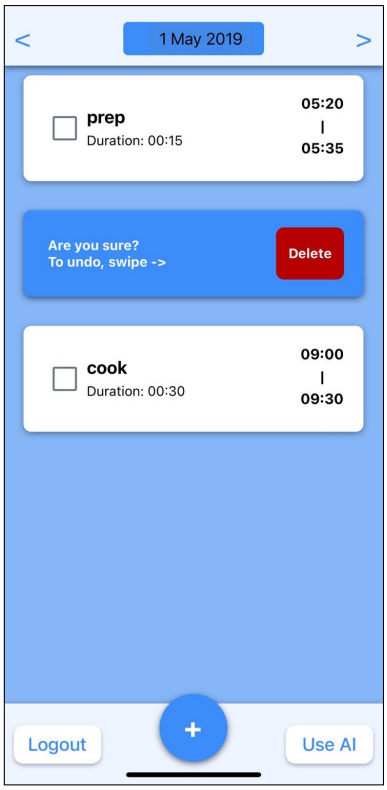
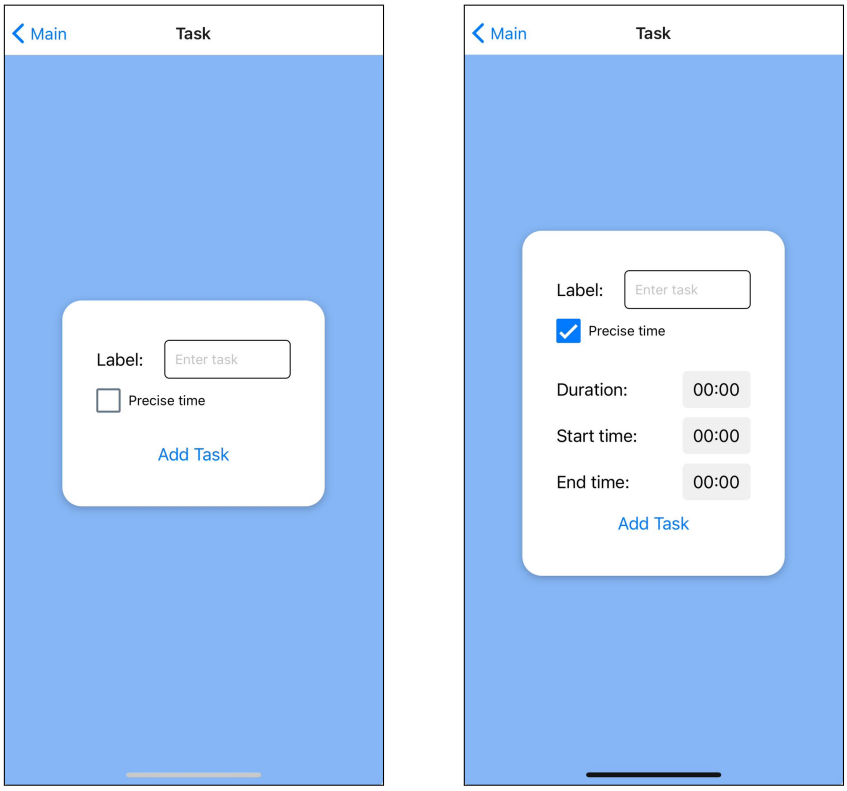



Figura 4.6: Ștergerea unei activități

realizată cu intenție, și nu din greșeală, va apărea în locul activității un mesaj în legătură cu ștergerea și butonul respectiv, după cum se poate observa în figura 4.6. Pentru undo, utilizatorul doar trebuie să gliseze înapoi de la stânga la dreapta.

Pagina de adăugare a unei activități 4.7 este reutilizată și pentru editarea acesteia. În cazul adăugării, câmpurile sunt goale sau inițializate cu 0, iar în cazul editării acestea sunt completate cu detaliile activității ce urmează să fie modificată, pentru a minimiza numărul de interacțiuni necesare ale utilizatorului. Opțiunea *Precise time* este prezentă deoarece dacă utilizatorul creează o nouă activitate doar cu numele acesteia 4.7a, algoritmul va genera intervalul de timp pentru aceasta. În schimb, prin bifarea acestei opțiuni 4.7b, câmpurile pentru precizarea intervalului de timp vor deveni vizibile. Odată ce utilizatorul a setat ora de start și durata, timpul de final este automat calculat, însa poate fi modificat. În cazul setării orelor de start și final, durata este automat calculată, aceasta putând fi modificată ulterior.

Prin acționarea butonului *Use AI*, data selectată va fi trimisă la server, iar algoritmul va genera o planificare cu activitățile prezente în ziua curentă. Cele ce nu au intervalul de timp precizat, îl vor primi după generare, iar cele ce au intervalul de timp schimbat rămân nefixate, însă se țin cont de ele pentru a nu suprapune restul activităților peste acestea. Pagina principală va arăta diferit, iar după terminarea algoritmului utilizatorul poate previzualiza planificarea și decide dacă dorește să îl păstreze, să folosească din nou instrumentul de AI sau să renunțe la planificare.




(a) Neextins (b) Extins

Figura 4.7: Pagina de adăugare/editare a unei activități

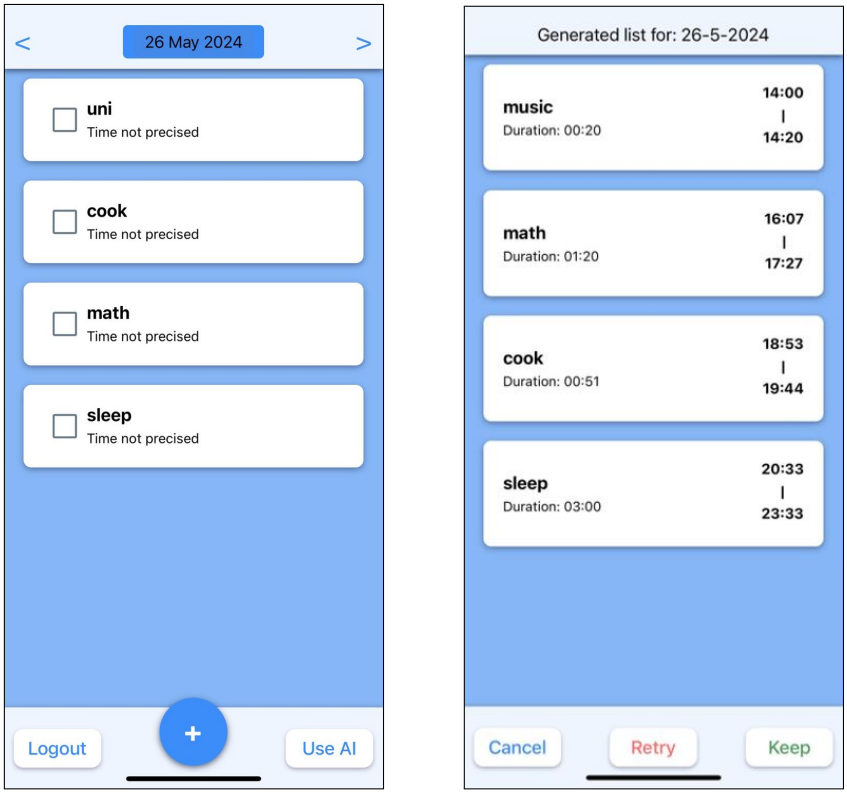
De exemplu, în figura 4.8a se observă pagina principală cu activități ce nu prezintă intervalul de timp menționat, iar în figura 4.8b este previzualizarea planificării și butoanele de decizie a utilizatorului. *Cancel* întoarce utilizatorul la pagina principală fără a face modificări, *Retry* rulează din nou algoritmul, generând o nouă lista de previzualizat, iar *Keep* modifică lista activităților curente și trimite serverului datele pentru actualizarea activităților în baza de date.

## 4.3 Detalii de implementare a aplicației

### 4.3.1 Dezvoltarea algoritmilor genetici

Având două seturi de date diferite pentru experimente, este necesară implementarea a două versiuni de algoritmi genetici. Diferența dintre cele două seturi de date este modul de reprezentare a spațiului temporal și prezența locației activității. Unul dintre ele conține locațiile fiecărei activități și perioadele zilei în care se desfășoară, iar celălalt doar timpul de start, durata și data.

Pentru început am creat o clasă generică pentru algoritmul genetic 4.9 care poate fi utilizat pentru ambele versiuni de implementare. Metodele dezvoltate reprezintă componentele de bază ale unui algoritm genetic: inițializarea populației, evalua-




(a) Înainte de utilizarea instrumentului AI

(b) După utilizarea instrumentului AI

Figura 4.8: Utilizarea instrumentului AI

rea cromozomilor, selecția, reproducerea pentru următoarele generații. Parametrii clasei conțin funcția de fitness, parametri specifici problemei și parametri specifici algoritmului genetic (dimensiunea populației, numărul de generații, procentul de mutație).

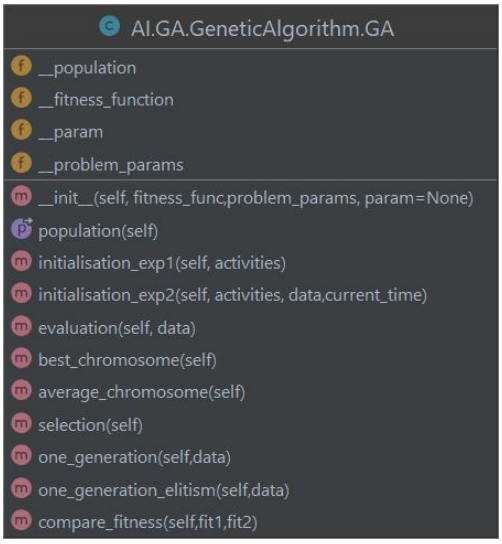



Figura 4.9: Clasa algoritmului genetic

Deoarece un cromozom va reprezenta o posibilă planificare a zilei, o reprezentare potrivită a acestuia este o listă de perechi. Utilizatorul va menționa ca date de intrare activitățile pe care dorește să le facă, ce pot sau nu să aibă menționate un interval sau perioadă de timp fixată. Astfel, fiecare pereche este specifică unei activități, indexul acesteia fiind același cu indexul activității din lista de intrare, și conține pentru prima versiune a algoritmului perioada zilei și locația, iar pentru cea de-a doua, timpul de început și durata.

Prezența a două funcții similare de inițializare se datorează parametrilor necesari diferiți și a claselor de cromozomi diferite. **Funcția de evaluare** este transmisă ca parametru acestei clase deoarece este specifică fiecărei versiuni, depinzând de structura cromozomilor și de rezultatul dorit. Este utilizată o metodă 'compare_fitness' deoarece pentru prima versiune a algoritmului sunt considerați cromozomi cu un scor bun cei care au cele mai mari valori, iar în cea de-a doua, cromozomii cu cele mai mici valori. Este esențială această informație pentru algoritm pentru a selecta corect cromozomii.

Pentru **funcția de selecție** 4.1 am ales de a selecta din primii 50% cei mai buni cromozomi de a fi transmiși în următoarele generații, deoarece atunci când există o varietate și un volum mare de date, algoritmul ajunge la convergență într-un timp mai bun, dar nu prematur. Astfel, se focusează pe cei mai buni cromozomi, incorporând și o parte mai puțin performantă, ceea ce ajută în păstrarea diversității și explorarea soluțiilor potențial mai bune.

```
    def selection(self):
        #select from 50% fittest generation
        half_pop = int((50 * self.__param['popSize'])/100)
        pos_1 = randint(0,half_pop)
        pos_2 = randint(0, half_pop)
        if self.compare_fitness(self.__population[pos_1].fitness,self.
    __population[pos_2].fitness):
            return pos_1
        else:
            return pos_2
```

Funcția de selecție

În structura clasei algoritmului genetic 4.9 se observa două metode de reproducere pentru crearea următoarelor generații, implementate pentru experimentare și analiza performanței. Diferența este că în timp ce 'one_generation' creează noua generație după reproducerea cromozomilor, 'one_generation_elitism' se asigură că un procent de 10% din cei mai buni cromozomi fac parte din următoarea generație, iar restul generației va fi creată prin același proces ca în 'one_generation'.

În ambele versiuni ale algoritmului genetic am ales să utilizez aceiași **operatori genetici** , încrucișarea într-un singur punct 4.2 și mutația 4.3, deoarece cromozomii fiind sub aceeași structură, nu există o diferență semnificativă între ei. Existând o diversitate mare în seturile de date, cromozomii vor fi inițializați într-o varietate mare, ceea ce nu necesită o modificare majoră a acestora cu ajutorul operatorilor genetici pentru a experimenta diferite soluții. Din acest motiv folosesc încrucișarea într-un singur punct, iar mutația care se realizează cu o anumită probabilitate, interschimbând două gene, aleasă în urma experimentelor.

```
def crossover(self, other):
    cut = random.randint(0, len(self.__representation)-1)
    new_repr = [None] * len(self.__representation)
    for i in range(cut):
        new_repr[i] = self.__representation[i]
    for i in range(cut, len(self.__representation)):
        new_repr[i] = other.__representation[i]
    offspring = MyChromosome()
    offspring.__representation = new_repr
    return offspring
```

Funcția de încrucișare

```
def mutation(self, mutation_rate):
    rnd_chance = random.randint(0,100)
    if rnd_chance < mutation_rate:
        poz_1 = random.randint(0, len(self.__representation)-1)
        poz_2 = random.randint(0, len(self.__representation)-1)
```

```
6         self.__representation[poz_1], self.__representation[poz_2] = \
7             self.__representation[poz_2], self.__representation[poz_1]
```

Funcția de mutație

**Funcțiile de evaluare** sunt cele care fac cea mai mare diferență, deoarece, fiind date diferite, prioritățile și evaluarea diferă în funcție de acestea. Pentru prima variantă de algoritm genetic, probabilitatea timpului de realizare al activităților din setul de date este împărțit pe perioade ale zilei și dacă sunt realizate în timpul săptămânii sau în weekend. Locațiile sunt împărțite în 3 categorii: acasă, public sau loc de muncă. Cum cromozomul reprezintă o planificare a unei zile, analizăm atât datele fiecărei activități în parte, dar și tot planul în ansamblu. Se consideră cel mai bun cromozom, cel care are cel mai mare scor.

Astfel, în funcția de evaluare 4.1 prioritatea este mai mare în găsirea unei perioade de timp pentru fiecare sarcină care se potrivește cel mai mult cu preferințele personale ale utilizatorului de realizare a fiecărei activități. Următoarea prioritate este reprezentată de locație, iar în funcția de evaluare se ia în considerare probabilitatea activității de a fi realizată într-un loc respectiv. De asemenea, pentru a echilibra activitățile pe parcursul zilei, ținem cont și de câte obiective sunt prezente în fiecare parte a zilei. În cazul în care utilizatorul vrea să realizez o activitate ce nu există în setul de date, aceasta va fi planificată aleator.

$$fitness_{V1} = fitness_{V1} + factor_{timp} * 0.6 + factor_{locatie} * 0.3 + factor_{echilibru} * 0.1 \quad (4.1)$$

4.1 reprezintă ecuația funcției de evaluare, unde:

- $factor_{timp}$: factorul de timp cu o pondere de 60%, reprezentând probabilitatea activității de a fi realizată în perioada de timp asignată
- $factor_{locatie}$: factorul locației cu o pondere de 30%, reprezentând procentul de preferință al utilizatorului de a realiza sarcina în locația asignată.
- $factor_{echilibru}$: factorul de echilibru cu o pondere de 10%, reprezentând cât de echilibrat sunt planificate activitățile peste zi

În cea de-a doua variantă lipsesc detaliile despre locațiile activităților, însă se cunoaște data exactă, timpul de început și durata, ceea ce oferă șansa unei analize mai detaliate a preferințelor și rutinei utilizatorului. Pe fiecare zi din săptămână, pentru fiecare activitate, am calculat ponderea de realizare a acesteia în diferite intervale de timp, astfel acuratețea planificării este mult mai mare. Cel mai important lucru în realizarea unui plan corect este de a nu suprapune intervalele de timp respective activităților și planificarea acestora începând cu ora curentă a începerii

rulării algoritmului. Cum cel mai bun cromozom este reprezentat de cel mai mic scor, în cazul unuia care suprapune doua sarcini, penalizarea este de 12 ore. De asemenea, se prioritizează găsirea unei soluții ce conține intervale libere de timp între obiective cât mai mici.

$$fitness_{V2} = fitness_{V2} + intervalLiber - factor_{intervalTimp} \tag{4.2}$$

Deși pare simplă funcția de evaluare 4.2, aceasta este foarte eficientă în găsirea unei soluții apropiate de preferințele utilizatorului.

- *intervalLiber*: timpul liber dintre activități, iar dacă acestea se suprapun, parametrul primește valoarea 720 ( 12 ore transformate în minute)
- *intervalTimp*: ponderea intervalului de timp, pentru o activitate într-o zi a săptămânii

### 4.3.2 Backend

Partea de backend este realizată în Python și este responsabilă de procesarea datelor din baza de date împreună cu administrarea serverului și a modelului de AI. Structura este organizată folosind arhitectură stratificată, fiecare strat ocupându-se de o componentă independentă din backend-ul aplicației. Straturile prezente, cum se poate observa în figura 4.10, sunt următoarele: Domain, Repository, Service, Server, modelul AI fiind separat de acestea și integrat în stratul Service.

Domain conține clasele pentru activitate și utilizator, iar pentru fiecare există o componentă repository ce implementează metodele CRUD, deoarece baza de date le gestionează separat. În MongoDB am creat 2 colecții, una ce gestionează utilizatorii, fiecare document având câmpuri pentru numele de utilizator și parolă, și una pentru activități. Pentru securitate, parola este criptată în baza de date folosind biblioteca Bcrypt [Bcr], ce utilizează o funcție de hash și un string numit salt(eng.) pentru criptare. Service pune la un loc cele două repository-uri și conține metodele necesare apelurilor din Server: operațiile CRUD pentru activități și funcția de autentificare a utilizatorului. În acest strat datele de la server sunt validate și transmise la repository pentru a fi procesate în baza de date.

Ultimul strat, partea principală din backend, este reprezentat de server. Pentru crearea serverului am utilizat frameworkul Flask [flab] deoarece este minimalist, foarte ușor de integrat în cod și suportă CORS (Cross-Origin Resource Sharing). Cum aplicația de frontend folosește ExpoGo și rulează pe portul 8081, iar serverul Flask pe 5000, înseamnă că se conectează două platforme de origini diferite, ce necesită prezent în configurarea serverului un mecanism pentru asigurarea conexiunii. În plus, pentru a ne asigura că alte dispozitive se pot conecta, hostul este configurat

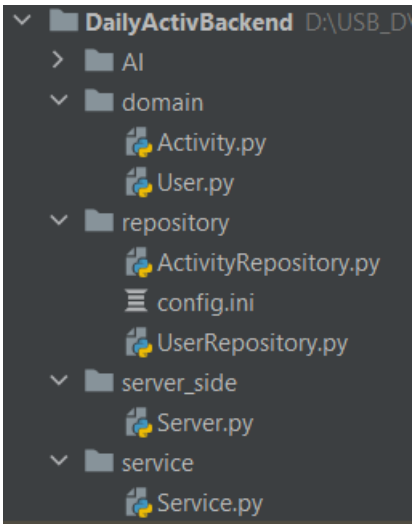



Figura 4.10: Arhitectura stratificată a aplicației

pe adresa IP 0.0.0.0, astfel ascultând pe orice adresă valabilă. Serverul conține endpointurile necesare pentru trimiterea și procesarea informației la componenta de frontend a aplicației, iar acestea sunt:

- $/tasks/\{username\}$ : folosind metoda GET, acest endpoint furnizează un dicționar cu toate activitățile realizate de un anumit utilizator, usernameul fiind transmis prin URL
- $/tasks/\{username\}$ : prin metoda POST, se adaugă o nouă activitate în baza de date, bodyul cererii conținând activitatea respectivă, iar ca răspuns se trimite activitatea actualizată
- $/tasks/\{id\}$ : folosind metoda DELETE, prin această cerere se șterge o activitate din baza de date după identificator. Ca răspuns, se trimite doar un mesaj și statusul, 200 dacă ștergerea se efectuează cu succes, sau 400 în caz contrar.
- $/login$ : acest endpoint verifică credențialele criptate ale utilizatorului prezente în bodyul cererii prin metoda POST cu cele din baza de date, iar dacă autentificarea se poate efectua cu succes, se trimite un răspuns ce conține numele de utilizator și statusul 200. În caz contrar, se trimite un mesaj de eroare împreună cu statusul 400.
- $/register$: această cerere prin metoda POST adaugă un utilizator nou în baza de date după ce se verifica ca acesta să nu fie deja existent. Dacă s-a creat contul cu succes, serverul returnează ca răspuns statusul 200, iar în caz contrar statusul 400.
- $/usealgorithm/\{username\}$: prin metoda POST, se trimite la server data selectată pentru care se rulează algoritmul, iar după rularea acestuia se trimit datele

despre noua planificare la client.

În figura 4.11 este ilustrată diagrama de clase a backendului, ce reprezintă logica din spatele aplicației de organizare a timpului personal. La rularea componentei backend, se inițializează baza de date și fiecăre strat necesar arhitecturii stratificate. Apoi, din componenta Main, începe rularea serverului, care va asculta cereri de la componenta de frontend și va răspunde acestora corespunzător. Procesarea datelor sau orice altă acțiune asupra acestora se întâmplă în backend, astfel, frontendul doar afișează datele sau rezultatele în urma unor acțiuni ale utilizatorului. Orice acțiune asupra unei activități se procesează în spate în ActivityRepository, cele legate de utilizator în UserRepository, iar algoritmul genetic, din MainGA, unde se inițializează și se setează parametrii pentru acesta.

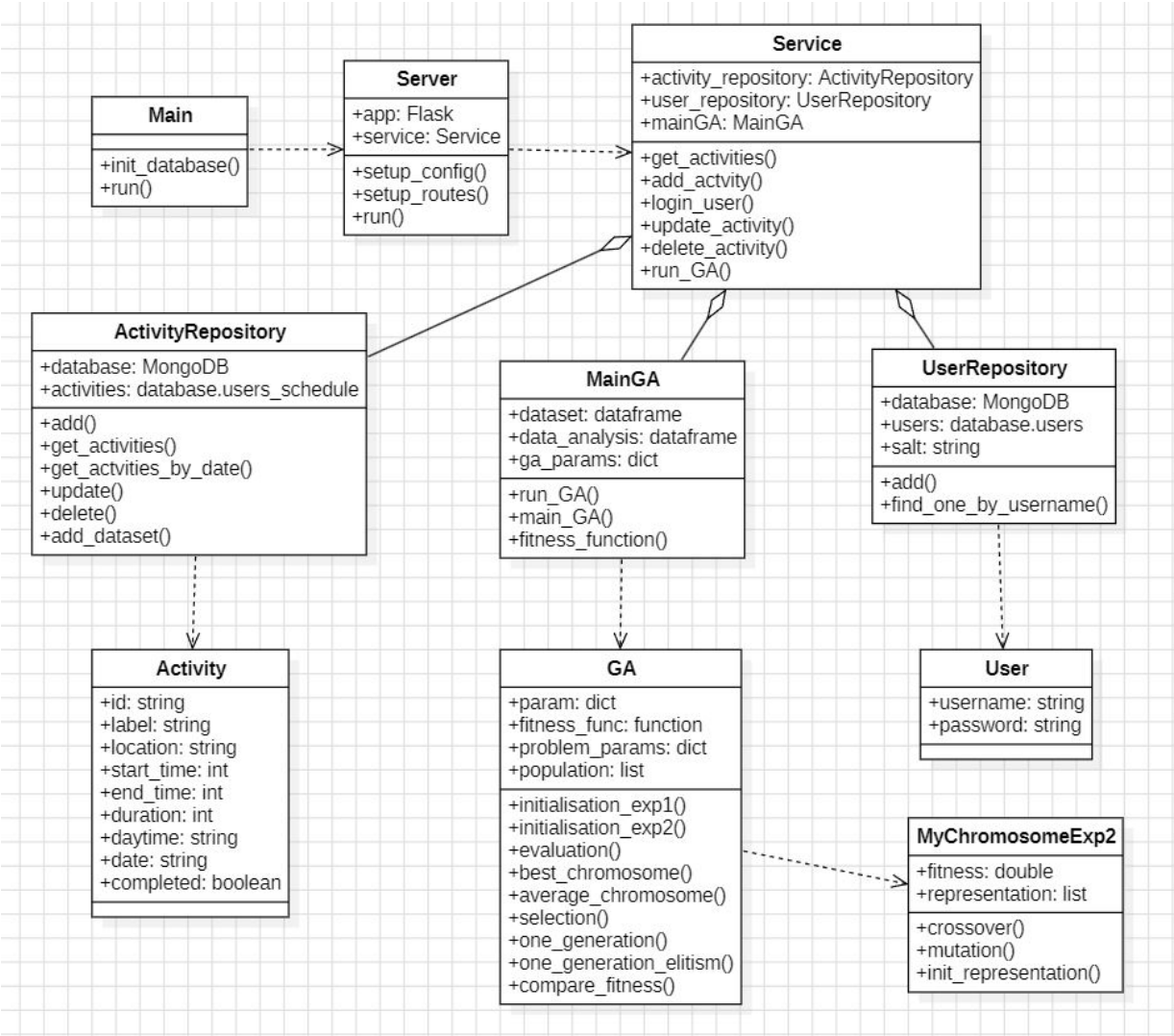



Figura 4.11: Diagrama de clase backend

Librăriile utilizate pe partea de backend sunt:

- *pandas* pentru procesarea datelor pentru algoritmul genetic și utilizarea lor sub formă de dataframe
- *matplotlib* pentru vizualizarea datelor și a rezultatelor algoritmului genetic
- *numpy* utilizat pentru pregătirea datelor, prezente sub formă de array, pentru a fi afișate cu librăria matplotlib

### 4.3.3 Frontend

Partea de frontend a fost realizată în TypeScript folosind librăria React [rea] și frameworkul open-source Expo [exp]. Aceasta este responsabilă de partea de client a aplicației, fiind componenta ce face legătura dintre utilizator și partea de backend. În figura 4.13 se poate observa diagrama de clase a frontendului, unde App este componenta principală. În aceasta, se creează o stivă pentru navigare între ecranele interfeței grafice 4.12 și se inițializează componentele pentru gestionarea autentificării și a activităților.

```
<NavigationContainer>
  <Stack.Navigator initialRouteName="LoginMain">
    <Stack.Screen name="LoginMain" component={LoginMain} options={{title: "Login"}}/>
    <Stack.Screen name="Main" component={TaskList}  options={{ headerShown: false }}/>
    <Stack.Screen name="EditTask" component={EditTask} options={{title: "Task"}}/>
  </Stack.Navigator>
</NavigationContainer>
```

Figura 4.12: Rutele componentelor Frontend

AuthProvider conține o componentă AuthState pentru a păstra constant informații despre starea utilizatorului autentificat și metodele necesare de autentificare sau delogare. În plus, gestionează metodele și apelează funcțiile necesare din LoginApi, care vor face un request la server. Orice schimbare din AuthProvider este oglindită și în componenta copil LoginMain care se ocupă de interfață.

TaskProvider este componenta ce se ocupă de gestionarea activităților pe partea de frontend și de a sincroniza schimbările ce apar în lista de activități astfel încât fiecare obiect este actualizat în interfață. Se folosește de TaskState pentru starea obiectului, iar când aceasta se schimbă, componentele se rerandează. Aceasta conține metodele principale prezente în aplicație, iar prin intermediul clasei TaskApi se pot face apeluri la server pentru a cere datele necesare sau efectua anumite funcții din partea de backend. Pentru partea de interfață, există două clase principale, pentru a diferenția ecranul principal de cel pentru adăugare sau editare a unei activități. TaskList este componenta pentru pagina principală, ce prezintă majoritatea funcționalităților și lista de activități, iar EditTask este utilizat fie pentru adăugarea unei sarcini, fie pentru editarea acesteia. După cum se observă și în diagrama de clase 4.13, atât TaskList, cât și EditTask depind de TaskProvider deoarece

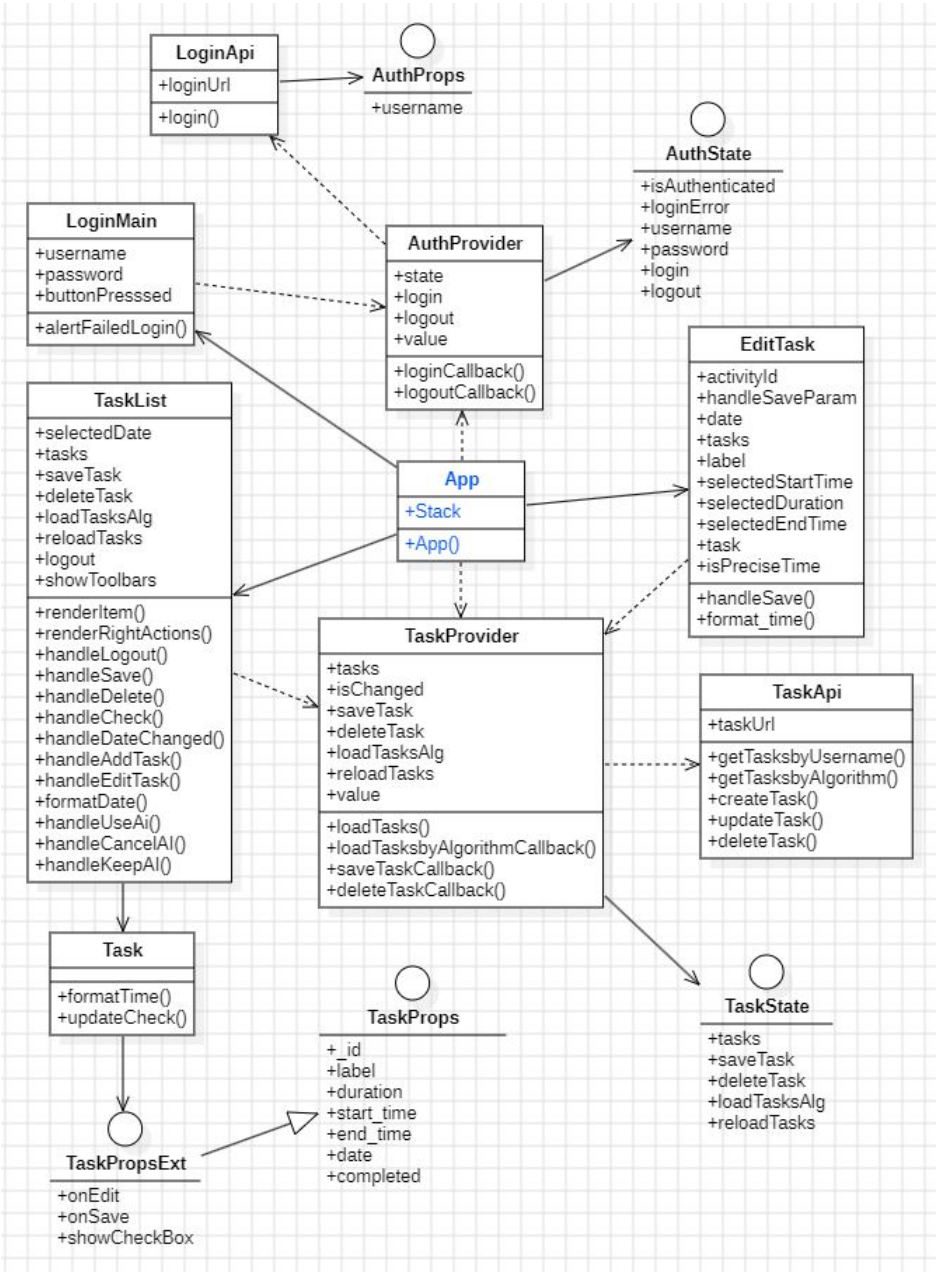



Figura 4.13: Diagrama de clase frontend

orice schimbare ce apare în această clasă se reflectă în toate componentele sale copil,astfel interfața fiind mereu actualizată cu datele prezente.

Pentru această parte de frontend au fost utilizate următoarele librării:

- *axios* [axi] care este utilizat pentru a face apeluri HTTP de pe componenta de client, iar în figura 4.14 se poate observa un astfel de exemplu
- *react* pentru a construi o interfață nativă, îmbunătățind performanța reîncărcând doar elementele ce necesită o randare în plus

```
export const createTask: (username:string, task: TaskProps) => Promise<TaskProps[]> = (username, task) => {
    return withLogs(axios.post(`${taskUrl}/${username}`,task), 'createTask');
}
```

Figura 4.14: Exemplu apel HTTP pentru crearea unei activități

În plus, pentru stilizarea elementelor de interfață din Expo, acesta permite utilizarea CSS (Cascading Style Sheets), ceea ce oferă posibilitatea de a crea o interfață cu un aspect mai plăcut pentru utilizator și de asemenea libertatea de a crea frontendul dorit pentru aplicație.

## 4.4 Tehnologii utilizate

### 4.4.1 React

React este o bibliotecă JavaScript open-source, creată de Facebook, care are scopul de a simplifica procesul de construire a interfețelor interactive. Lucrul care evidențiază acest framework este faptul că aplicațiile sunt construite din componente reutilizabile și independente, ceea ce ușurează scrierea și gestionarea codului [Whab].

Principalele avantaje în dezvoltarea unei interfețe cu React sunt rapiditatea, flexibilitatea, performanța, dezvoltarea aplicațiilor mobile și componentele reutilizabile. În comparație cu alte framework-uri, codul este mai ușor de menținut și flexibil din cauza structurii modulare. Performanța înaltă se datorează DOM-ului(Document Object Model) virtual deoarece actualizează și reîncarcă eficient doar componentele care au avut parte de schimbări, astfel nefiind nevoie de actualizarea întregii pagini web [Whab].

Motivul pentru care am ales React este flexibilitatea și ușurința scrierii codului. Aplicația mobilă poate fi utilizată atât pe Android, cât și IOS, iar aceași versiune a aplicației poatre fi accesată și web, oferind o varietate de opțiuni utilizatorilor. De asemenea, mă concentrez și pe aspectul interfeței și ușurința de utilizare a acesteia, iar faptul că există multe tehnologii și framework-uri care pot fi integrate cu ușurință, îmi oferă șansa de a alege instrumentele ce satisfac cerințele. Astfel, am

ales să utilizez Expo împreună cu React, deoarece dezvoltarea aplicației este foarte rapidă și eficientă datorită Expo Go. Aplicația poate fi testată fără a aștepta procesul de construire, iar utilizatorii o pot testa scanând un cod QR ce o va deschide pe dispozitivele lor mobile.

În plus, având o comunitate mare și activă de programatori ce contribuie constant la îmbunătățirea sa, React oferă numeroase documentații, tutoriale și resurse. Comunitatea unită și puternică se asigură că fiecare persoană primește ajutorul și soluțiile necesare. În concluzie, flexibilitatea, performanța, comunitatea și ușurința de integrare a altor instrumente sunt caracteristicile principale care m-au influențat să aleg React pentru dezvoltarea interfeței aplicației.

### 4.4.2 Flask

Flask este un framework web open-source micro în Python ce oferă instrumente și caracteristici ce ușurează crearea aplicațiilor web în Python. Oferă flexibilitate programatorilor și este mult mai accesibil pentru cei noi în domeniu deoarece se poate construi o aplicație web rapid folosind doar un fișier scris în Python. De asemenea, Flask este extensibil și nu necesită alte biblioteci sau o structură particulară pentru a începe o aplicație. Este ideal pentru proiectele mici sau de dimensiune medie [Whaa].

Printre principalele avantaje în folosirea acestui framework se numără flexibilitatea, numeroase surse de documentație și o comunitate activă, integrare ușoară cu alte tehnologii, dezvoltare rapidă și eficientă, scalabilitate. Flask oferă tehnici de rutare folosind decoratorul route(), care face legătura dintre URL și o funcție, dar suportă și protocoale HTTP pentru a prelua date de la URL-ul(Uniform Resource Locator) specificat [Flaa]: GET, HEAD, POST, PUT, DELETE.

Am ales Flask pentru aplicația de optimizare a timpului personal deoarece este ușor de folosit, necesită puțin cod scris, iar aplicația este destul de mică, fiind necesare simple operații CRUD (create-read-update-delete) pe partea de server. În plus, datele pot fi trimise ca obiecte JSON (JavaScript Object Notation), ceea ce micșorează complexitatea aplicației având în vedere că se utilizează MongoDB pentru baza de date, ce poate stoca datele în același format. Astfel, nu este necesară conversia datelor pe partea de server, dar nici pe client, deoarece este ușor de gestionat obiecte JSON în React cu diferite biblioteci.

### 4.4.3 MongoDB

MongoDB este o bază de date NoSQL open-source, utilizată pentru a stoca un volum mare de date, performând foarte rapid. NoSQL este utilizat ca o alternativă a bazelor de date relaționale tradiționale,iar în loc de folosirea tabelelor și rândurilor,

MongoDB are o arhitectură formată din colecții și documente. Utilizatorii au acces la un server pentru a crea o astfel bază de date, unde documentele sunt alcătuite din perechi de câmpuri și valori [GB].

Documentele sunt similare cu obiectele JSON, dar folosesc o variație numită BSON (Binary JSON) deoarece suportă mai multe tipuri de date. Câmpurile din aceste documente sunt coloanele din bazele de date relaționale. Valorile pot conține o varietate de tipuri de date, cum ar fi alte documente, vectori, vectori de documente etc. [GB] Ca și identificator unic, documentele incorporează o cheie primară, iar structura lor se modifică în urma unei adăugări sau ștergerii unui câmp. Colecțiile reprezintă un set de documente, fiind echivalentul tabelelor bazelor de date relaționale.

Beneficiile utilizării MongoDB sunt flexibilitatea deoarece nu necesită o structură prestabilită și poate stoca orice tip de dată, este orientată pe documente ceea ce nu necesită unirea bazelor de date și scalabilitatea. Motivul pentru care am ales MongoDB în locul unei baze de date relaționale este faptul că datele necesare aplicației se află într-un volum mare, care va fi în continuă creștere odată cu utilizarea ei, iar interogările au o structură simplă, lipsită de complexitate. În plus, utilizarea unei baze de date ce stochează datele într-o variație a JSON, având în vedere că datele dintre server și client vor fi trimise în acest format, eficientizează codul și complexitatea programului.

## 4.5 Comparare cu aplicații din literatura de specialitate

După cum am discutat și în capitolul 3 despre anumite aplicații și lucrări similare, scopul acestei aplicații este de a crea o versiune îmbunătățită a unei aplicații de organizare a timpului personal față de ce există deja implementat. Astfel, cel mai mare avantaj al aplicației mobile și elementul inovativ al lucrării este optimizarea timpului personal prin planificarea automată a activităților zilnice, scutind utilizatorul de efortul de a-și organiza fiecare zi. De asemenea, față de jocul prezent în capitolul 3.1, interfața aplicației mobile este intuitivă și foarte ușor de utilizat pentru utilizator, având un număr minim de interacțiuni necesare. Prin accesibilitatea aplicației, ne asigurăm că productivitatea utilizatorului este îmbunătățită fără ca acesta să fie copleșit de utilizarea acesteia. În lucrarea prezentată în capitolul 3.2 un aspect important lipsă care este implementat în aplicația de organizare a timpului este planificarea bazată pe preferințele userului care îmbunătățește experiența de folosire a acesteia. Iar automatizarea acesteia, reflectă un avantaj și un plus față de lucrarea din secțiunea 3.3.

# Capitolul 5

# Experimente

## 5.1 Seturi de date

Problema de organizare a timpului personal necesită un model AI care să prezică o secvență de activități organizate după preferințele utilizatorului. Acesta depinde în primul rând de datele culese despre modul utilizatorului de a realiza activitățile zilnice,mai exact de: durata activității, perioada peste zi/intervalul de timp de realizare, locația și ordinea. Pentru dezvoltarea modelelor de AI au fost utilizate doua seturi de date, care diferă prin câteva detalii.

### 5.1.1 "Ms-Latte"

Acest set de date [JCGW21] conține 10,101 activități zilnice, iar pentru fiecare sunt notate 3-4 locuri și 5 intervale ale zilei în care pot fi realizate. Dezavantajul acestui set de date este faptul că nu exista o prioritate între activități, ceea ce afectează modelul în predicția unei organizări corecte ale activităților zilnice. În schimb, diversitatea activităților poate fi considerat un plus. De asemenea, specificarea spațiului temporal sub formă de perioade ale zilei, de exemplu: dimineața, seara etc., în loc de intervale fixe de timp pentru fiecare activitate, oferă utilizatorului o flexibilitate în realizarea lor. Astfel, nu apare stresul în cazul în care acesta nu se încadrează în intervalul de timp și pot fi efectuate în ordinea preferată.

Setul de date este în format JSON, dar înainte de a fi folosit, a fost preprocesat și curățat. Pentru perioada zilei există o coloană numită "TimeJudgements" care conține 5 dicționare cu cheia "Known" având valoarea "Yes" dacă se știe această informație, "No" altfel, și cheia "Times" având valoarea o parte a zilei în următorul format: **WE**_ (weekend)/**WD**_ (în timpul săptamânii) la care se adaugă perioada zilei **morning**(dimineață), **afternoon**(după-amiază), **evening**(spre seara),**night**(noaptea), **anytime**(oricând). Pentru locație se urmărește o structură asemănătoare, o coloana LocJudgements ce conține 3-4 dicționare astfel: dacă cheia "Known" are valoarea

”Yes”, cheia ”Locations” va conține una dintre valorile ”home”(acasă), ”work”(locul de muncă), ”public”. În cazul în care valoarea este public, ultima cheie ”PublicLocations” conține numele general al locației publice, fiind posibile în jur de 69 denumiri unice în tot setul de date. În figura 5.1 se poate vizualiza structura setului de date inițial, luând primele 5 intrări din setul de date.

| | ID | TaskTitle | List... | LocJudgements | TimeJudgements |
|---|---|---|---|---|---|
| 0 | 3026964 | rearrange closet | home | [{'Known': 'yes', 'Locations': 'home', 'PublicLocations': ''}, {'Known': 'yes', 'Locations': 'home', 'PublicLocations': ''},... | [{'Known': 'yes', 'Times': 'WE-morning'}, {'Known': 'yes', 'Times': 'WE-morning'}, {'Known': 'yes', 'Times': 'WE-afternoo... |
| 1 | 9829911 | meeting tasks | work | [{'Known': 'yes', 'Locations': 'work', 'PublicLocations': ''}, {'Known': 'yes', 'Locations': 'work', 'PublicLocations': ''}, {'... | [{'Known': 'yes', 'Times': 'WD-afternoon'}, {'Known': 'yes', 'Times': 'WD-morning,WD-afternoon,WD-evening'}, {'Know... |
| 2 | 9099975 | taste of home | groceries | [{'Known': 'yes', 'Locations': 'home', 'PublicLocations': ''}, {'Known': 'yes', 'Locations': 'home', 'PublicLocations': ''},... | [{'Known': 'yes', 'Times': 'WE-afternoon,WD-evening'}, {'Known': 'yes', 'Times': 'WE-afternoon'}, {'Known': 'yes', 'Times... |
| 3 | 8120843 | bring book in | default list | [{'Known': 'yes', 'Locations': 'public', 'PublicLocations': 'bookstore'}, {'Known': 'yes', 'Locations': 'home,work', 'Pub... | [{'Known': 'yes', 'Times': 'WD-afternoon,WD-evening'}, {'Known': 'yes', 'Times': 'WD-morning,WD-evening'}, {'Known':... |
| 4 | 6533945 | sociology paper | today | [{'Known': 'yes', 'Locations': 'work', 'PublicLocations': ''}, {'Known': 'yes', 'Locations': 'work', 'PublicLocations': ''}, {'... | [{'Known': 'yes', 'Times': 'WD-morning,WD-afternoon'}, {'Known': 'yes', 'Times': 'WD-morning'}, {'Known': 'yes', 'Time... |
| 5 | 12477687 | paper hold | to do | [{'Known': 'yes', 'Locations': 'home', 'PublicLocations': ''}, {'Known': 'yes', 'Locations': 'home', 'PublicLocations': ''},... | [{'Known': 'yes', 'Times': 'WE-morning,WD-evening'}, {'Known': 'yes', 'Times': 'WD-afternoon,WE-afternoon,WD-eveni... |

Figura 5.1: Primele elemente din setul de date [JCGW21]

Având în vedere că multe coloane conțin dicționare, ceea ce nu este recomandat atunci când urmează să fie folosit pentru un algoritm sau model de inteligență artificială, setul de date trebuie normalizat. Înainte de acest pas, este curățat, eliminând valorile nule sau duplicate. Soluția pentru normalizarea coloanei ”TimeJudgements”, este calcularea probabilității unei activități de a se efectua în anumite intervale ale zilei, în comparație cu celelalte. Astfel, se elimină această coloană, iar în schimb se adaugă valorile posibile pentru cheia ”Times” și cu probabilitatea specifică. Analog, se procedează la fel pentru ”LocJudgements” si cheia ”Locations”. Ținând cont că există aproximativ 69 de denumiri unice de locații publice unde obiectivul a fost realizat, pentru coloana ”Public_Locations” am păstrat structura de dicționar. În cazul unui număr mai mic de locații, am fi putut crea câte o coloană pentru fiecare locație, care ar fi fost marcată în cazul în care o sarcină poate fi realizată în acest loc. În plus, pentru fiecare activitate sunt posibile în jur de trei locații publice, ceea ce înseamnă că am fi avut foarte multe valori nule sau cu valoare 0 în setul de date.

Structura setului de date după procesare cu explicații este următoarea:

(a) **ID** - identificatorul activității

(b) **TaskTitle** - titlul activității care este unic în setul de date

(c) **ListTitle** - fiind un dataset bazat pe un ”ToDo list”, reprezintă categoria din care face parte

(d) **Loc_Home** - probabilitatea locației activității să fie acasă

(e) **Loc_Public** - probabilitatea locației să fie într-un loc public

(f) **Loc_Work** - probabilitatea locației să fie la locul de muncă

(g) **Time_freq** - frecvența activității față de restul activităților

(h) **Public_locations** - în cazul în care 'Loc_Public' este mai mare decât 0, dicționarul este populat cu maxim 4 nume de locații publice

(i) Parametrii pentru probabilitatea activității de a fi realizată într-un anumit interval de timp al zilei are același format ca în setul de date original (exemplu: WE_morning)

În figurile 5.2 și 5.3 se poate vizualiza structura setului de date după normalizare și procesare. În comparație cu setul inițial, există mult mai multe coloane, însă este mai ușor de vizualizat și de lucrat cu el în cadrul unui algoritm, datele fiind clar separate.

| | ID | TaskTitle | ListTitle | Loc_Home | Loc_Public | Loc_Work | Time_freq | Public_locations | WE_morning | WE_afternoon |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3026964 | rearrange closet | home | 1.00000 | 0.00000 | 0.00000 | 0.05000 | {} | 0.60000 | 0.20000 |
| 1 | 9829911 | meeting tasks | work | 0.00000 | 0.00000 | 1.00000 | 0.08000 | {} | 0.00000 | 0.00000 |
| 3 | 8120843 | bring book in | default list | 0.20000 | 0.40000 | 0.40000 | 0.09000 | {'bookstore': 1, 'offi... | 0.00000 | 0.11111 |
| 4 | 6533945 | sociology paper | today | 0.00000 | 0.00000 | 1.00000 | 0.08000 | {} | 0.12500 | 0.00000 |
| 5 | 12477687 | paper hold | to do | 1.00000 | 0.00000 | 0.00000 | 0.10000 | {} | 0.10000 | 0.30000 |

Figura 5.2: Primele elemente din setul de date [JCGW21] după procesare, prima parte de coloane

| | rning | WE_afternoon | WE_evening | WE_night | WE_anytime | WD_morning | WD_afternoon | WD_evening | WD_night | WD_anytime |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | 0.20000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.20000 | 0.00000 | 0.00000 |
| 1 | | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.25000 | 0.62500 | 0.12500 | 0.00000 | 0.00000 |
| 3 | | 0.11111 | 0.00000 | 0.00000 | 0.00000 | 0.11111 | 0.22222 | 0.55556 | 0.00000 | 0.00000 |
| 4 | | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.50000 | 0.37500 | 0.00000 | 0.00000 | 0.00000 |
| 5 | | 0.30000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.10000 | 0.50000 | 0.00000 | 0.00000 |

Figura 5.3: Primele elemente din setul de date [JCGW21] după procesare, ultima parte de coloane

### 5.1.2 "Life tracking"

Pentru realizarea acestui set de date [Sch19], un student și-a monitorizat viața de zi cu zi timp de o lună. Scopul a fost găsirea unor tipare în obiceiurile personale și ce îmbunătățiri ar putea fi făcute în stilul zilnic de viață. În comparație cu setul de date anterior, [JCGW21], acesta conține mai puține date și mai puțin diversificate, în schimb pentru fiecare activitate există timpul de început și durata, cu ajutorul căreia se poate determina și timpul de oprire. Astfel, utilizatorul are mai puțină flexibilitate, dar toate activitățile vor fi plănuite într-un interval de timp exact și nu va mai fi nevoie de niciun efort de a plănui din partea persoanei ce utilizează aplicația.

Activitățile din setul de date au fost desfășurate pe perioada 5/1/2019 - 5/31/2019, iar numărul de activități este 11: cook, eat, math, music, pause, prep, sleep, uni, meditation, special, work. Pentru vizualizarea structurii setului de date se poate observa figura 5.4, unde coloanele sunt următoarele:

(a) **date** - data în care a fost realizată activitatea

(b) **time** - durata de finalizare a unei sarcini

(c) **activity** - numele activității

(d) **start_time** - ora de început

| | date | time | activity | start_time |
|---|---|---|---|---|
| 0 | 2019-05-01 | 00:15 | prep | 2019-05-01 05:20:00 |
| 1 | 2019-05-01 | 01:00 | math | 2019-05-01 05:35:00 |
| 2 | 2019-05-01 | 00:10 | pause | 2019-05-01 06:35:00 |
| 3 | 2019-05-01 | 01:00 | math | 2019-05-01 06:45:00 |
| 4 | 2019-05-01 | 01:30 | music | 2019-05-01 07:45:00 |
| 5 | 2019-05-01 | 01:38 | uni | 2019-05-01 09:15:00 |

Figura 5.4: Primele elemente din setul de date [Sch19]

Având în vedere că pentru algoritmul genetic ne vom folosi de durata și timpul de început al activității în funcția de evaluare, formatul curent de dată și oră nu avantajează calculele și cresc complexitatea algoritmului. Astfel, durata din coloana ”time” va fi transformată în minute, la fel și datele din coloana ”start_time”, nefiind necesar să păstrăm data în aceeași coloană deoarece deja există în coloana ”date”. Setul de date este structurat simplu cu o cantitate mică de informație, ceea ce nu necesită foarte multe modificări sau prelucrări. Coloanele rămân neschimbate, însa toate formatele de timp sunt transformate în minute. În figura 5.5 se pot observa aceleași intrări din figura 5.4, dar cu modificările menționate anterior.

| | date | time | activity | start_time |
|---|---|---|---|---|
| 0 | 2019-05-01 | 15 | prep | 320 |
| 1 | 2019-05-01 | 60 | math | 335 |
| 2 | 2019-05-01 | 10 | pause | 395 |
| 3 | 2019-05-01 | 60 | math | 405 |
| 4 | 2019-05-01 | 90 | music | 465 |
| 5 | 2019-05-01 | 98 | uni | 555 |

Figura 5.5: Primele elemente din setul de date [Sch19], după prelucrare

Din analiza autorului a setului de date, în figura 5.6 este reprezentat într-un grafic cât timp este acordat fiecărei activități, unde ”productive” reprezintă suma dintre timpul acordat pentru ”math”, ”uni” și ”productive”. Se observă că, pe lângă timpul de dormit care necesită cele mai multe ore, timpul de productivitate este cel mai mare împreună cu cel de pregătire care include rutina de dimineață, cumpărături, sport.

Din cauza lipsei locației din acest set de date, ne putem concentra mai mult pe analiza mai complexă a activităților utilizatorului. Pentru a îmbunătăți acuratețea

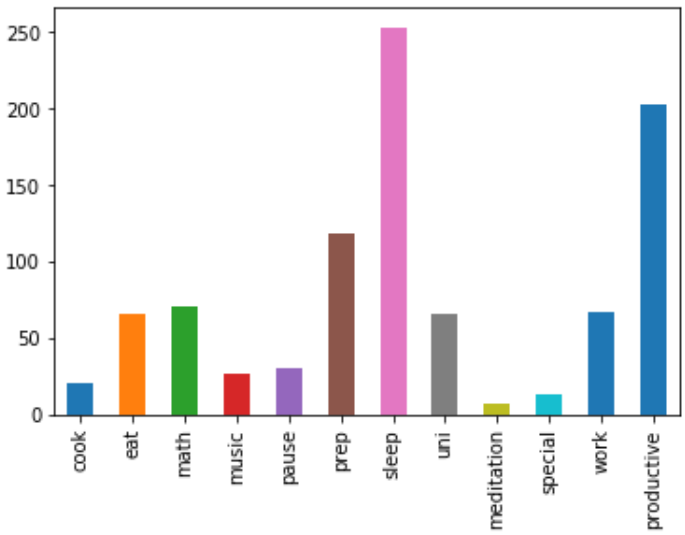



Figura 5.6: Vizualizarea timpului acordat pentru fiecare activitate [Sch19]

algoritmului genetic, pe baza datelor existente, am analizat frecvența medie a fiecărei activități în funcție de fiecare zi a săptămânii. De exemplu, deoarece autorul setului de date este un student, în timpul săptămânii este o probabilitate destul de mare de a se repeta aceleași activități în aceleași intervale orare. Chiar dacă utilizatorul nu este student, în general oamenii tind să aibă o rutină repetitivă. Această analiză 5.7 ajută algoritmul de a avea posibilitatea de a genera o planificare specifică planului utilizatorului în care anumite activități se pot repeta, nefiind necesar ca acesta să menționeze aceeași activitate de mai multe ori.

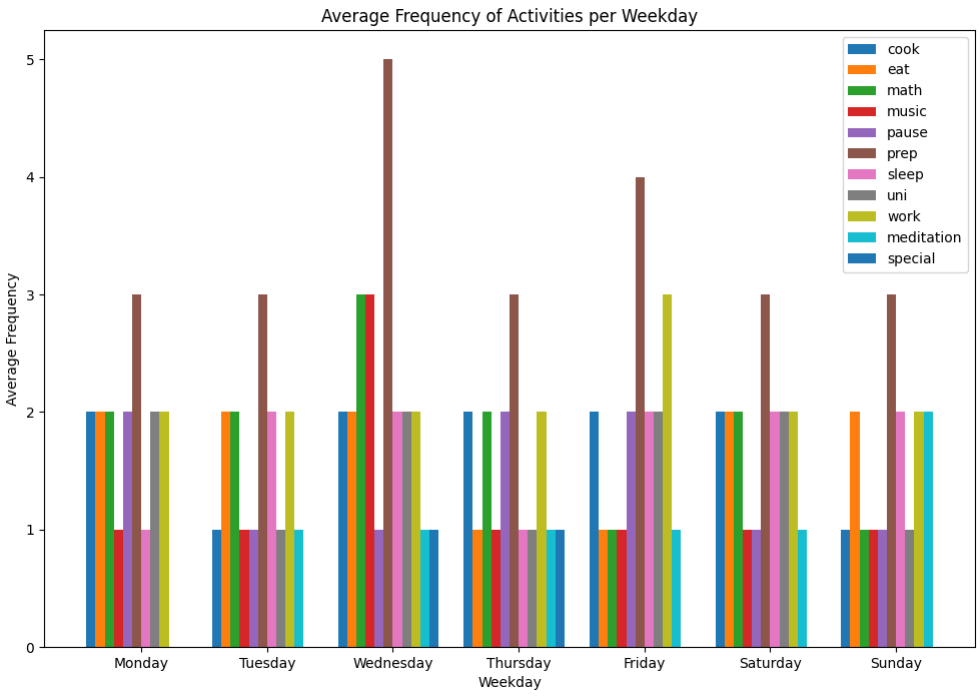



Figura 5.7: Frecvența medie a fiecărei activități specifică fiecărei zi din săptămână

În plus, pentru a observa când o activitate tinde să fie realizată într-o zi specifică a săptămânii, am împărțit cele 24 de ore ale zilei în 4 intervale orare de analiză: 00-06, 06-12, 12-18, 18-00. Acest lucru ajută la funcția de evaluare de a avea o planificare cât mai specifică pentru utilizator, încercând să potrivească sarcinile pe baza istoricului în intervalele de timp cele mai potrivite. În figurile 5.8, 5.9 și 5.7 este analiza pe intervale de timp a activităților în care autorul setului de date [Sch19] consideră că este productiv.

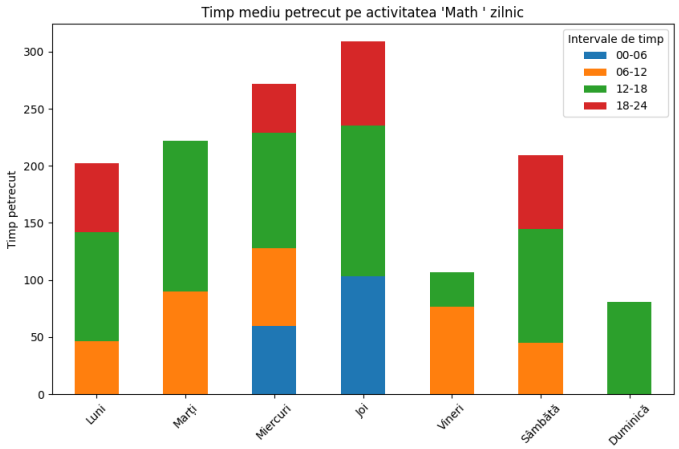



Figura 5.8: Timp mediu petrecut pentru ”math” în diferite intervale de timp ale zilei

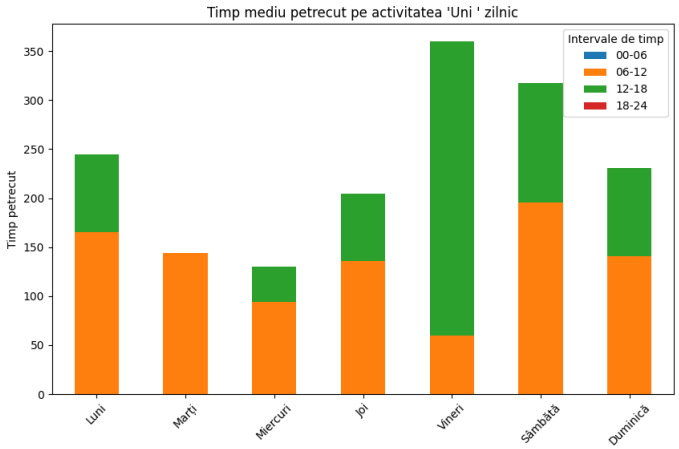



Figura 5.9: Timp mediu petrecut pentru ”uni” în diferite intervale de timp ale zilei

## 5.2 Rezultate experimentale

### 5.2.1 GA cu spațiul temporal sub formă de perioade ale zilei

Rezultatele experimentale ale algoritmilor genetici se bazează pe convergența acestora și parametrii utilizați pentru a-i aduce la o convergență cât mai bună. Pentru

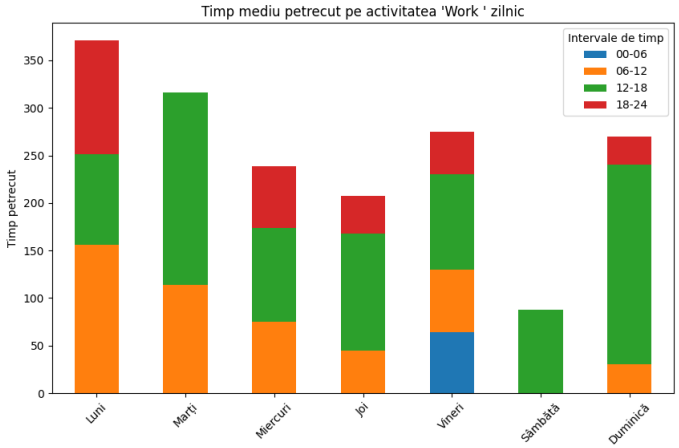
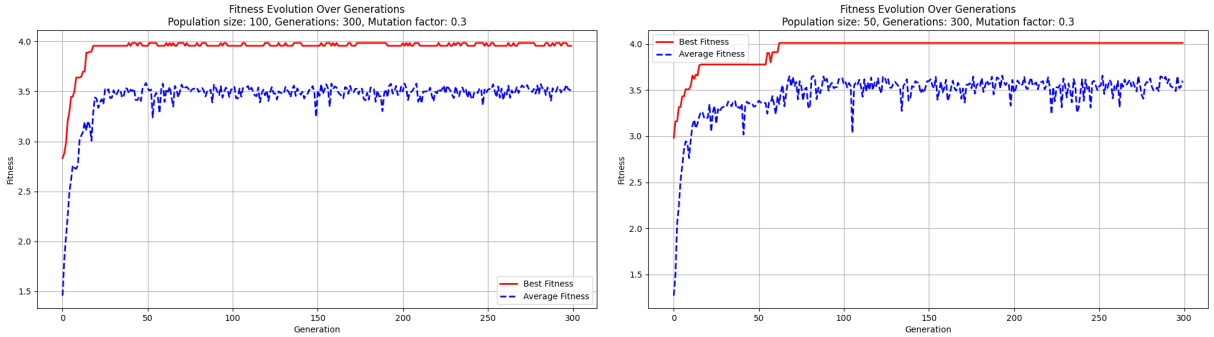



Figura 5.10: Timp mediu petrecut pentru ”work” în diferite intervale de timp ale zilei

primul algoritm genetic, cel care conține spațiul temporal reprezentat sub formă de perioade ale zilei, s-au utilizat ca date de intrare diferite activități generate din setul de date. Dintre acestea un procent de 25% prezintă locația specificată. Fiind un algoritm ce ar trebui utilizat într-o aplicație mobilă, scopul este de a găsi parametrii ce l fac să ajungă la o convergență bună într-un timp cât mai scurt. Astfel, utilizatorul nu așteaptă un timp îndelungat rezultatele și este satisfăcut de experiența utilizării aplicației.

Pentru început, fiind un set de date cu o diversitate foarte mare, am ales sa experimentez pe o populație de dimensiunea 100 și 300 de generații, iar scorul de mutație primește valori între 0.01 și 0.3. Rezultatele sunt ineficiente deoarece necesită un timp mare de rulare. În plus, pe parcursul fiecărei generații, fitnessul celui mai bun cromozom oscilează, la fel și media scorului, după cum se poate observa în graficul 5.11. Se observă că pentru o populație mai mică, valorile oscilează mai puțin, tind mult mai mult spre convergență și cel mai bun scor este îmbunătățit.



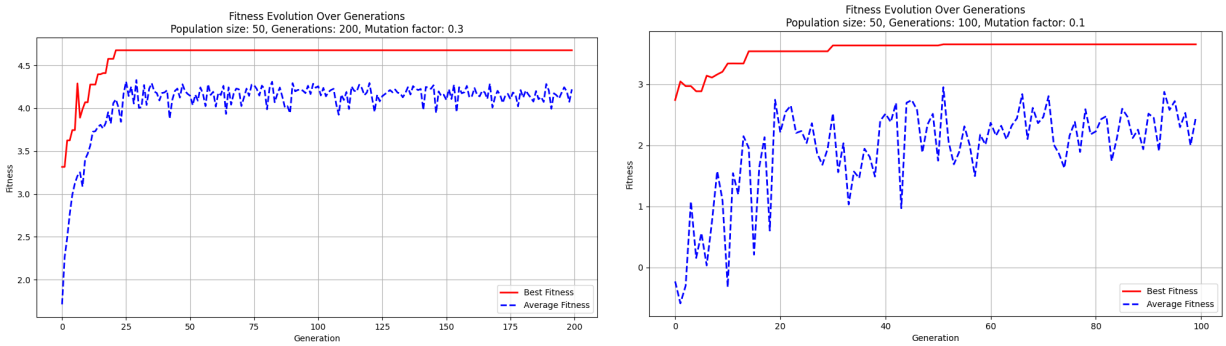

(a) Populație: 100

(b) Populație: 50

Figura 5.11: Comparare convergență cu dimensiune diferită a populației

Următoarele experimente sunt pentru o populație mai mică, dar număr diferit de generații și alți factori de mutație, având în vedere că aceștia afectează convergența. În figura 5.12a, observăm că pentru un număr mai mic de generații și o mutație de factor 0.1, algoritmul converge mult mai bine și valorile oscilează mai puțin. În plus, din rezultatele de până acum se observă că cel mai bun rezultat devine constant până în generația 100, iar stabilitatea acestuia ne indică faptul că soluția găsită este cea mai bună.

Având în vedere că încrucișarea se efectuează la fiecare reproducere, am experimentat și cu un factor de încrucișare 0.8 pentru a observa cum se comportă rezultatele algoritmului. În figura 5.12b se pot vizualiza rezultatele, care variază foarte mult, fiecare generație având mult prea mulți cromozomi de scoruri diferite și convergența nu este îmbunătățită.




(a) Factor încrucișare: 1

(b) Factor încrucișare: 0.8

Figura 5.12: Reprezentarea grafică a convergenței cu factori de încrucișare diferiți

În final, după experimentare, pentru acest algoritm cei mai buni parametri sunt dimensiunea populației de 30, 150 de generații, factor de mutație 0.1 și de încrucișare 1. În figura 5.13 se observă cum se obține cel mai bun scor, iar diferența dintre fitnessul mediu și cel mai bun este cea mai mică în comparație cu rezultatele obținute cu alți parametri.

### 5.2.2 GA cu spațiu temporal precizat și exact

În această variantă de algoritm, planificarea este precisă, fiind generat intervalul de timp exact al activități, bazat pe istoricul și preferințele utilizatorului. Față de versiunea anterioară, setul de date corespunzător este mai puțin diversificat, ceea ce optimizează cu mult timpul de rulare. Astfel, în cadrul aplicației, când utilizatorul dorește să utilizeze instrumentul AI, acesta primește rezultatele după rularea algoritmului aproape instant.

Experimentele vor fi realizate pe o listă de 12 activități în care se cunoaște doar eticheta acestora, iar rezultatul dorit este unul cu un scor cât mai mic, acesta fiind

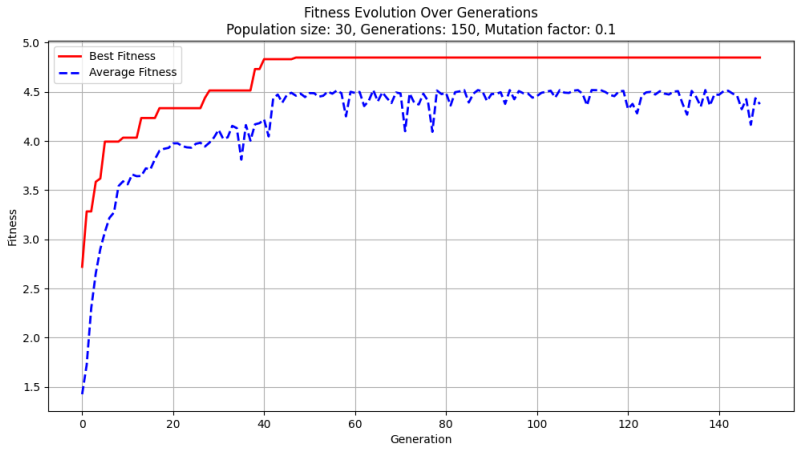
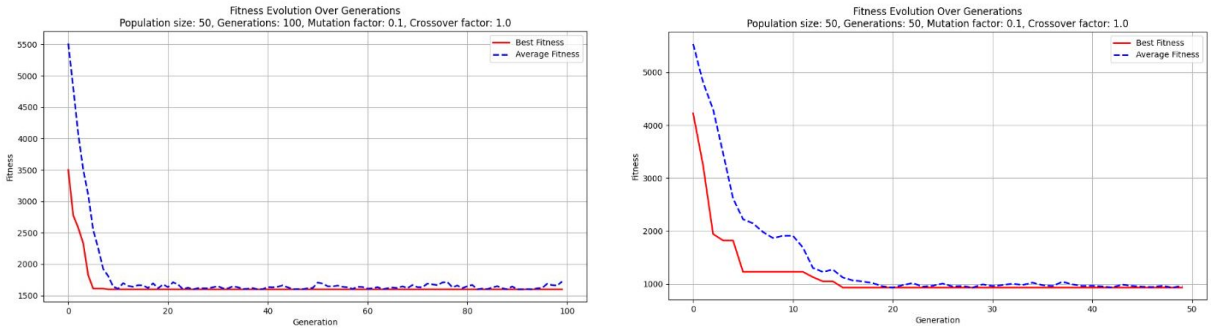



Figura 5.13: Reprezentarea grafică a convergenței cu parametrii finali

considerat cel mai bun în această versiune. De asemenea se urmărește ca activitățile să nu se suprapună iar organizarea să respecte cât mai mult preferințele utilizatorului. O primă încercare de combinație a parametrilor se poate vizualiza în figura 5.14a unde se observă că se atinge convergența în generațiile 20-40. Astfel, în figura 5.14b se poate vizualiza cum rulează alghoritmul cu mai puține generații.



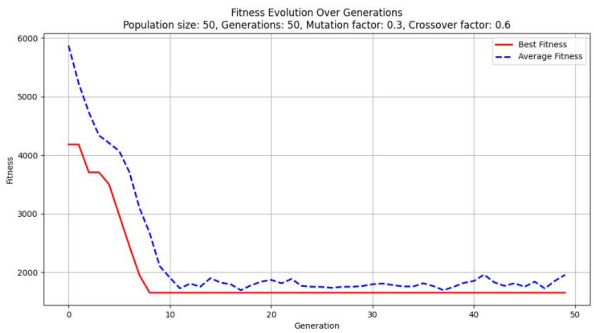

(a) Generații: 100 (b) Generații: 50

Figura 5.14: Reprezentarea grafică a convergenței cu număr de generații diferit

De asemenea, am experimentat și cu un factor de încrucișare mai mic sau de mutație mai mare pentru a vizualiza cum se schimbă convergența și scorul rezultatelor. În graficul din figura 5.15a se observă cum diferența dintre scorul mediu și cel mai bun este mai mare și oscilează mult mai mult, iar același lucru se întâmplă și cu un factor de mutație mai mare 5.15b.

După efectuarea și analiza experimentelor cu diferiți parametri, concluzia este că cei mai buni parametri sunt cei din figura 5.14b. O populație mai mică converge bine, dar scorul este mai rău, cum se vede și în figura 5.16, ceea ce rezultă ca o populație de 50 de cromozomi este potrivită. Numărul de generații, după cum se observă în figuri este perfect între 40-50, iar factorii de încrucișare și mutație sunt cei


(a) Factor de încrucișare 0.6 și de mutație 0.1




(b) Factor de încrucișare 0.6 și de mutație 0.3

Figura 5.15: Reprezentarea grafică a convergenței cu factori de încrucișare și mutație diferiți

mai potriviți cu valorile 1, respectiv 0.1.

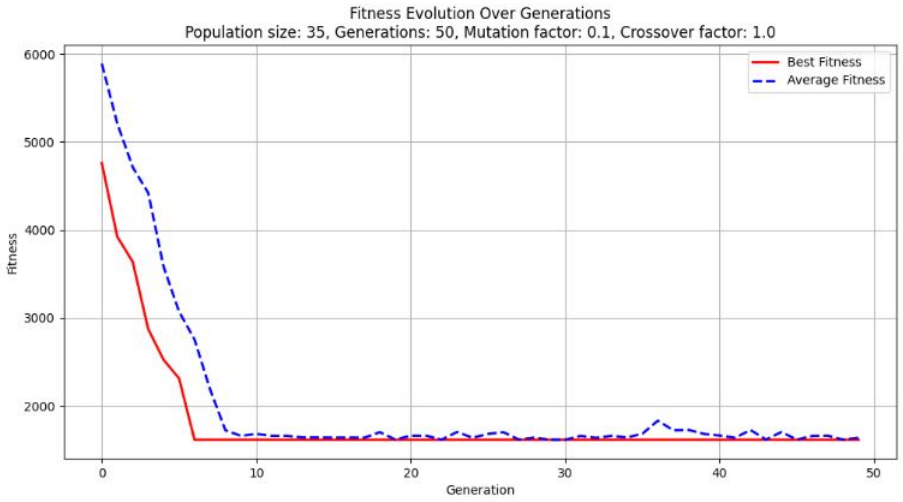



Figura 5.16: Reprezentarea grafică a convergenței cu o populație mică

# Capitolul 6

# Concluzii și direcții viitoare de dezvoltare

Oamenii din ziua de azi devin din ce în ce mai preocupați, concentrându-se pe setarea de noi obiective și de dezvoltarea personală. Însă, odată cu această dorință de a atinge cât mai multe obiective personale vine și problema timpului cu care toți ne confruntăm. Mulți dintre noi întâmpinăm probleme în a ne organiza și motiva să finalizăm anumite activități, iar aplicațiile existente nu sunt destul de eficiente în optimizarea timpului personal.

În această lucrare a fost prezentată soluția la această problemă, folosind algoritmi genetici pentru a crea un instrument de planificare automată a timpului personal. În acest mod, utilizatorul este scutit de planificarea fiecărei zile, deoarece aplicați se folosește de rutina din trecut și preferințele acestuia.

După descrierea teoretică a algoritmilor genetici și studiul lucrărilor similare pe tema organizării timpului personal și aplicațiile existente, s-a prezentat aplicația mobilă ce susține această lucrare de licență. Sunt două versiuni de algoritmi genetici implementați și studiați: una ce prezintă spațiul temporal sub formă de perioade ale zilei, iar cealaltă, sub formă de intervale de timp fixate. În urma experimentelor efectuate în capitolul 5, versiunea ce folosește intervale de timp fixate converge cel mai bine și rulează în cel mai bun timp, motiv pentru care această versiune a fost integrată în aplicația mobilă. De asemenea, experiența utilizatorului în urma utilizării aplicației este mult mai bună dacă fiecare obiectiv este planificat la o oră exactă, și nu pe perioade ale zilei.

Aplicația este împărțită în două părți principale: backend și frontend. Partea de backend conține algoritmul genetic integrat și serverul care ascultă requesturi de la partea de frontend. În urma primirii acestora, datele sunt procesate de Service, iar apoi de Repository, care se ocupă de operațiile CRUD în baza de date. Partea de frontend reprezintă componenta de legătură dintre utilizator și backend, adică interfața aplicației mobile. Aceasta primește cerințe de la utilizator, care sunt trans-

mise mai departe la server sub forma unui apel HTTP. Funcționalitățile principale ce pot fi efectuate de utilizator sunt: vizualizarea liste de activități pe zile, ștergerea, adăugarea sau modificarea unui obiectiv, autentificarea, crearea unui cont sau delogarea, completarea unei activități și utilizarea instrumentului de planificare automată.

Odată cu implementarea aplicației de optimizare a timpului personal, am identificat o serie de îmbunătățiri, atât asupra algoritmului, cât și a aplicației. O posibilă funcționalitate ce ar putea ajuta utilizatorul în planificarea personală este recomandarea de sarcini care se repetă în în general în rutina utilizatorului, astfel amintindu-i de anumite activități pe care le-ar putea face.

De asemenea, algoritmul și funcția de evaluare pot fi actualizate să țină cont și de locația în care o activitate este efectuată, iar atunci când se planifică o zi să se ia în considerare distanța dintre locațiile activităților. Pe partea de aplicație, ar putea fi implementată o hartă în care utilizatorul poate vedea sub formă de pinuri activitățile, sau chiar și sub forma unui traseu ce urmărește în ordine cronologică obiectivele.

O altă îmbunătățire pe partea de algoritm ar putea fi adăugarea unei proprietăți noi obiectului de activitate: prioritatea. Astfel, algoritmul va genera planificarea prioritizând activitățile mai importante și programându-le cât mai repede posibil. În acest mod, experiența utilizatorului este cu mult îmbunătățită, crescând acuratețea planificării față de preferințele utilizatorului.

# Bibliografie


[AF18] Feras Sameer Alhaijawy and Adina Magda Florea. Using genetic algorithm to plan individuals temporal and nontemporal daily activities. In *Mediterranean Conference on Pattern Recognition and Artificial Intelligence*, 2018.

[axi] Axios api. Accessed: 2024-04-28.

[BBM93] David Beasley, David R. Bull, and Ralph Robert Martin. An overview of genetic algorithms: Part 1. 1993.

[Bcr] Hashing passwords in python with bcrypt. Accessed: 2024-05-16.

[Car14] Jenna Carr. An introduction to genetic algorithms. 2014.

[Cho09] Ossam Chohan. University timetable scheduling using genetic algorithm. Master's thesis, Department of Computer Engineering, Dalarna University, Röda vägen 3, Sweden, 3 2009.

[CLH15] Xiaojia Chen, Ying Li, and Tao Hu. Solving the supermarket shopping route planning problem based on genetic algorithm. *2015 IEEE/ACIS 14th International Conference on Computer and Information Science (ICIS)*, pages 529–533, 2015.

[dSAdSATW15] Jesimar da Silva Arantes, Márcio da Silva Arantes, Claudio Fabiano Motta Toledo, and Brian Charles Williams. A multi-population genetic algorithm for uav path re-planning under critical situation. *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 486–493, 2015.

[exp] Expo docs. Accessed: 2024-04-01.

[Fan95] Hsiao-Lan Fang. Genetic algorithms in timetabling and scheduling. 1995.

[Flaa] Introduction to web development using flask. Accessed: 2024-05-06.

[flab] How to build a simple android app with flask backend? Accessed: 2024-05-01.

[GB] Alexander S. Gillis and Bridget Botelho. Mongodb. Accessed: 2024-05-06.

[JCGW21] Sujay Kumar Jauhar, Nirupama Chandrasekaran, Michael Gamon, and Ryen W. White. Ms-latte: A dataset of where and when to-do tasks are completed. *arXiv preprint 2111.06902*, 2021.

[KG19] Mohammad Amin Kuhail and Nikhil Sai Santosh Gurram. Taskdo: A daily task recommender system. *2019 International Conference on Computational Intelligence in Data Science (ICCIDS)*, pages 1–5, 2019.

[KGA14] Lev Kazakovtsev, Mikhail N. Gudyma, and Alexander Antamoshkin. Genetic algorithm with greedy heuristic for capacity planning. *2014 6th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, pages 607–613, 2014.

[KJN13] Dennis L. Kappen, Jens Johannsmeier, and Lennart E. Nacke. Deconstructing 'gamified' task-management applications. *Proceedings of the First International Conference on Gameful Design, Research, and Applications*, 2013.

[KVGG20] Suyash Kejriwal, Vaibhav Vishal, Aastha Gulati, and Gaurav Gambhir. A review of daily productivity growth using to-do manager. *International Research Journal of Modernization in Engineering Technology and Science*, 2:969–974, 2020.

[rea] React. Accessed: 2024-04-01.

[Sch19] Maxim Schmidt. Life tracking project dataset, 2019. Accessed: 2024-03-08.

[Whaa] What is flask? benefits and uses. Accessed: 2024-05-06.

[Whab] What is react.js? uses, examples, and more. Accessed: 2024-05-05.