UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Smart Planner

Pavel Catalin
Luca Gunter
Malan Andreea
Zimbru Denis

1. Descrierea proiectului

SmartPlanner este o aplicatie web pentru planificarea activitatilor zilnice, care ajuta utilizatorii sa isi organizeze timpul intr-un mod eficient si automatizat. Aplicatia permite utilizatorilor sa isi creeze taskuri, sa le prioritizeze si sa primeasca sugestii si notificari pe baza comportamentului lor.

Sistemul nu se limiteaza la stocarea taskurilor, ci analizeaza modul in care utilizatorul isi organizeaza activitatea si reactioneaza prin rearanjarea taskurilor si prin trimiterea de notificari.

2. Limbaje si tehnologii utilizate

Aplicatia este realizata folosind urmatoarele tehnologii:

2.1. Python

Este limbajul principal folosit pentru dezvoltarea backend-ului aplicatiei. In Python sunt implementate logica de business, regulile de planificare, autentificarea, notificatiile si comunicarea cu baza de date.

2.2. Flask

Este framework-ul web folosit pentru construirea aplicatiei. Flask gestioneaza rutele (URL-urile), request-urile utilizatorilor si legatura dintre backend si interfata grafica.

2.3. SQLAlchemy

Este folosit ca ORM (Object Relational Mapper). El permite lucrul cu baza de date folosind clase Python in loc de interogari SQL directe. Este utilizat pentru definirea modelelor User, Task, Notification, Friendship etc.

2.4. RailwaySql

Este baza de date a aplicatiei. Sunt stocate toate datele legate de utilizatori, taskuri, prieteni si notificari.

2.5. Flask-Login

Este utilizat pentru gestionarea autentificarii utilizatorilor. El se ocupa de mentinerea sesiunilor, de verificarea daca un utilizator este logat si de protejarea rutelor care necesita autentificare.

2.6. Bcrypt

Este folosit pentru criptarea parolelor. Parolele nu sunt salvate in clar in baza de date, ci sub forma de hash-uri securizate.

2.7. HTML, CSS si Jinja2

Sunt folosite pentru interfata grafica:

- HTML defineste structura paginilor

- CSS este folosit pentru stilizare
- Jinja2 este motorul de template-uri care permite afisarea dinamica a datelor din backend in pagini (taskuri, notificari, prieteni etc.)

Arhitectura aplicatiei este de tip MVC (Model - View - Controller), organizata prin Blueprints.

- Model: clasele SQLAlchemy (User, Task, Notification etc.)
- View: fisierele HTML/Jinja2
- Controller: rutele Flask organizate in Blueprints (auth, tasks, friends, admin, home)

Aceasta arhitectura face aplicatia mai usor de inteles, modificat si extins.

3. Cerinte functionale

Pentru utilizatori:

- Crearea unui cont si autentificare
- Adaugarea, stergerea si vizualizarea taskurilor
- Alegerea importantei unui task (low, medium, high)
- Vizualizarea taskurilor zilnice si viitoare
- Primirea de notificari
- Adaugarea si gestionarea prietenilor

Pentru sistem:

- Reordonarea automata a taskurilor in caz de conflict
- Mutarea taskurilor pe zilele urmatoare cand nu exista spatiu
- Generarea de notificari pe baza regulilor
- Analiza activitatii utilizatorului

Pentru administrator:

- Vizualizarea utilizatorilor
- Restrictionarea sau stergerea conturilor

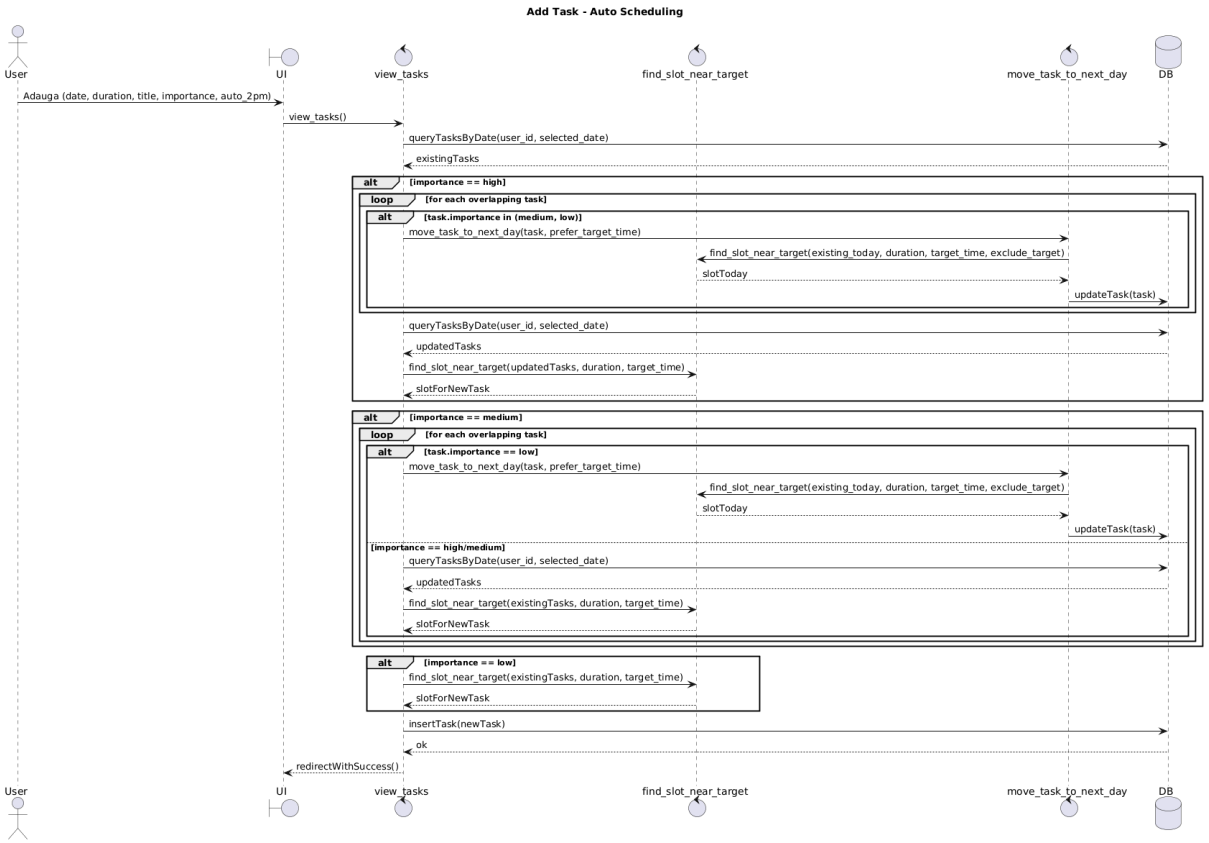
4. Cerinte non-functionale

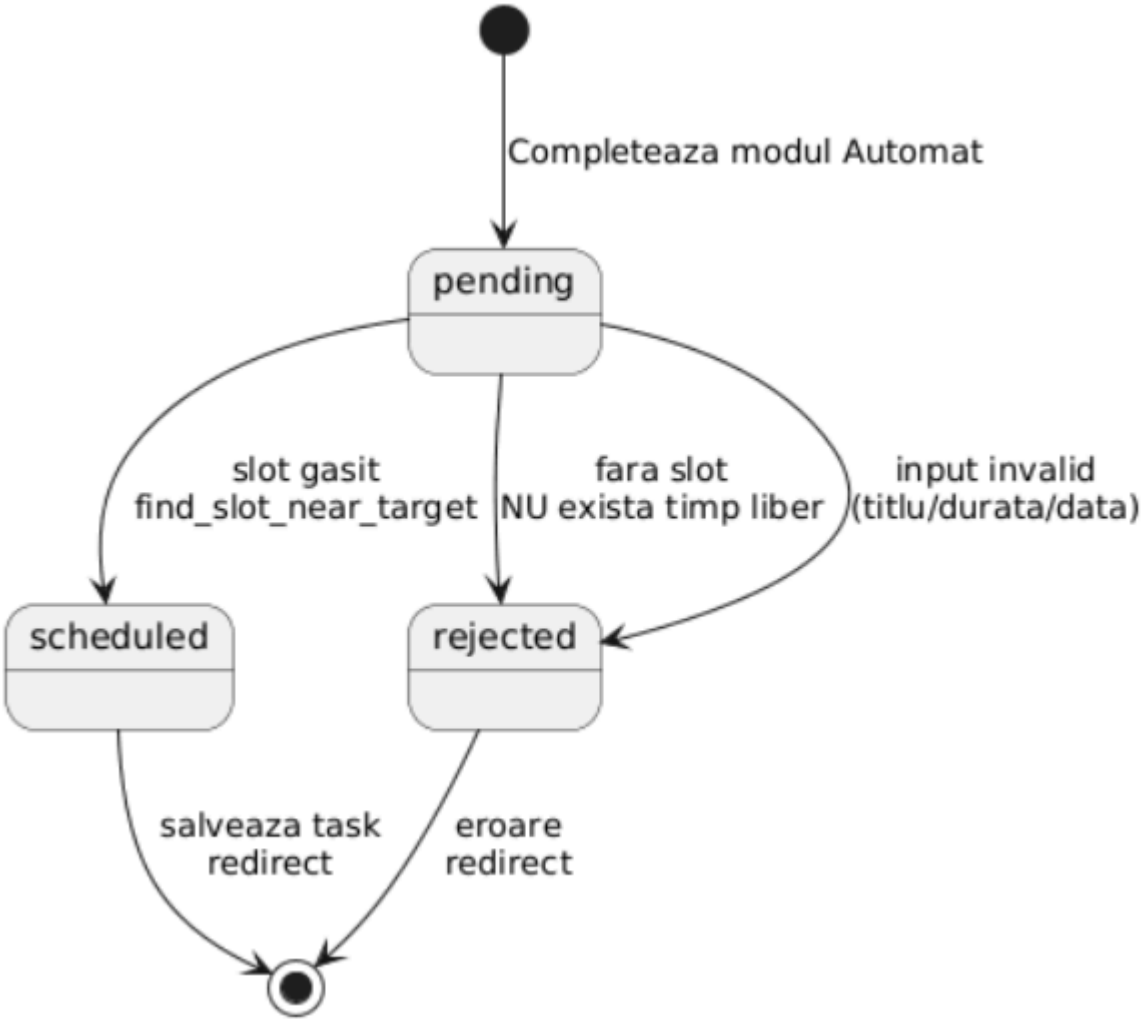
- Securitate - parolele sunt criptate
- Fiabilitate - datele sunt salvate in baza de date
- Performanta - se folosesc mecanisme de cache si proxy
- Scalabilitate - aplicatia este modulara
- Usurinta in utilizare - interfata este simpla si usor de inteles

5. Diagrame

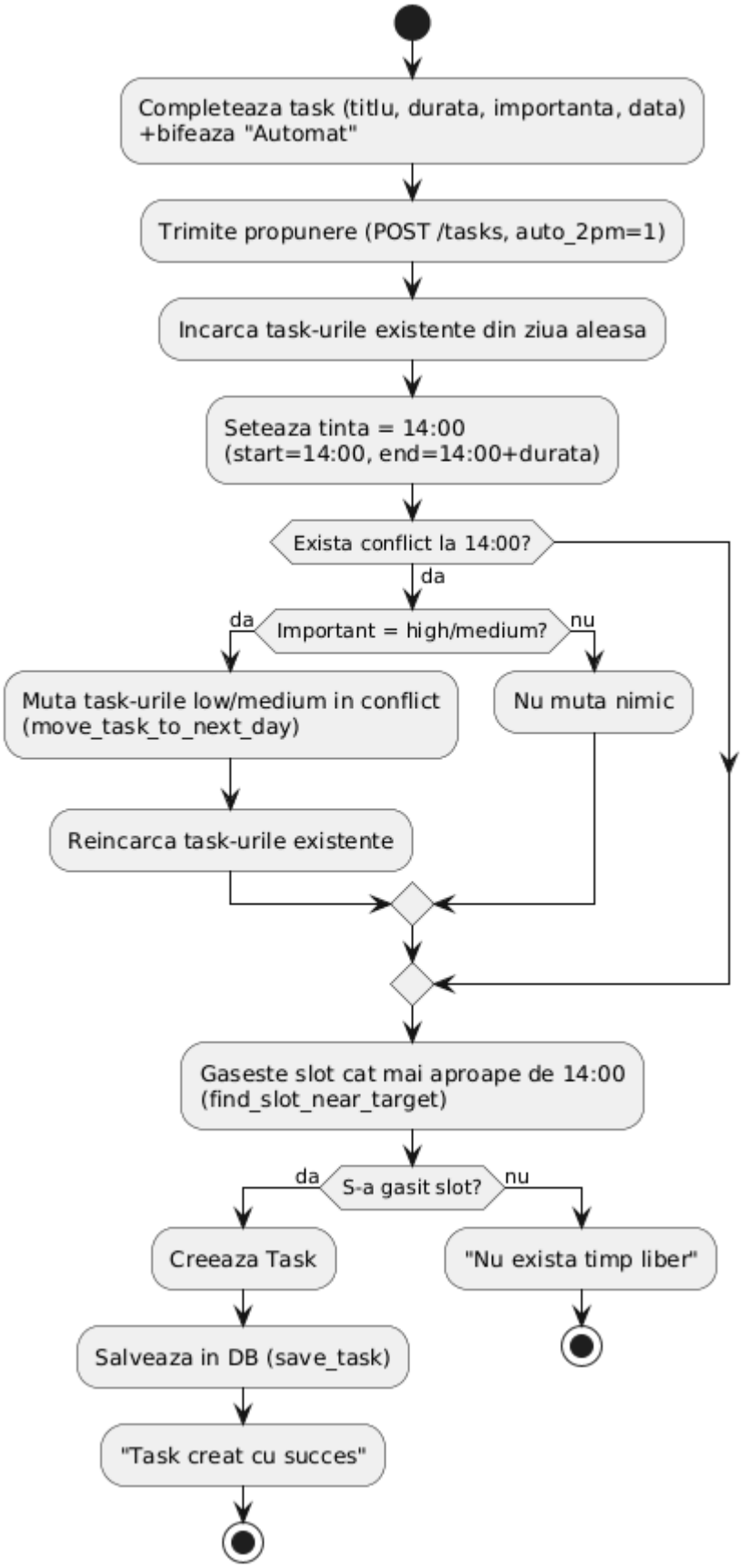
Zimbru Denis:

Sequence Diagram

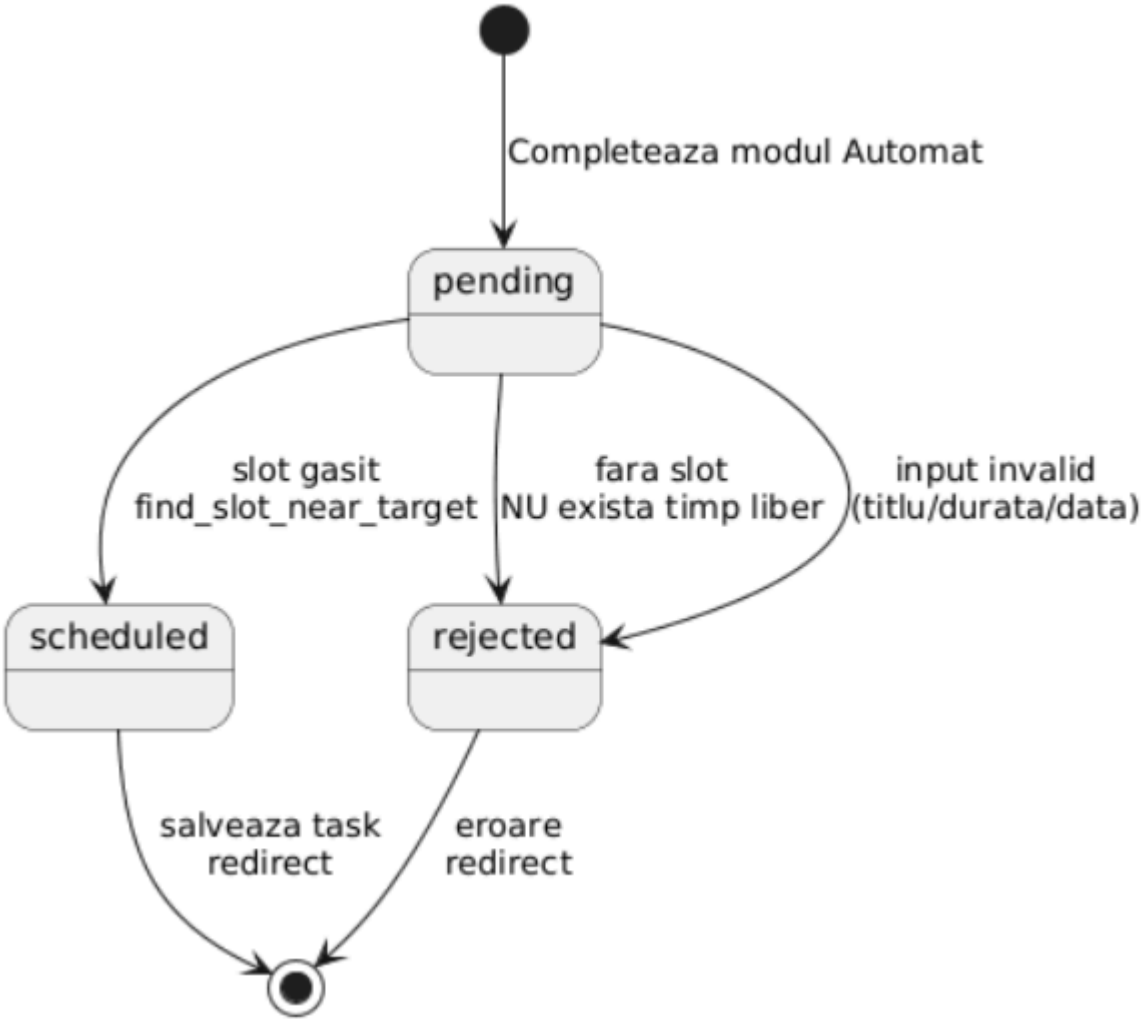




Activity Diagram

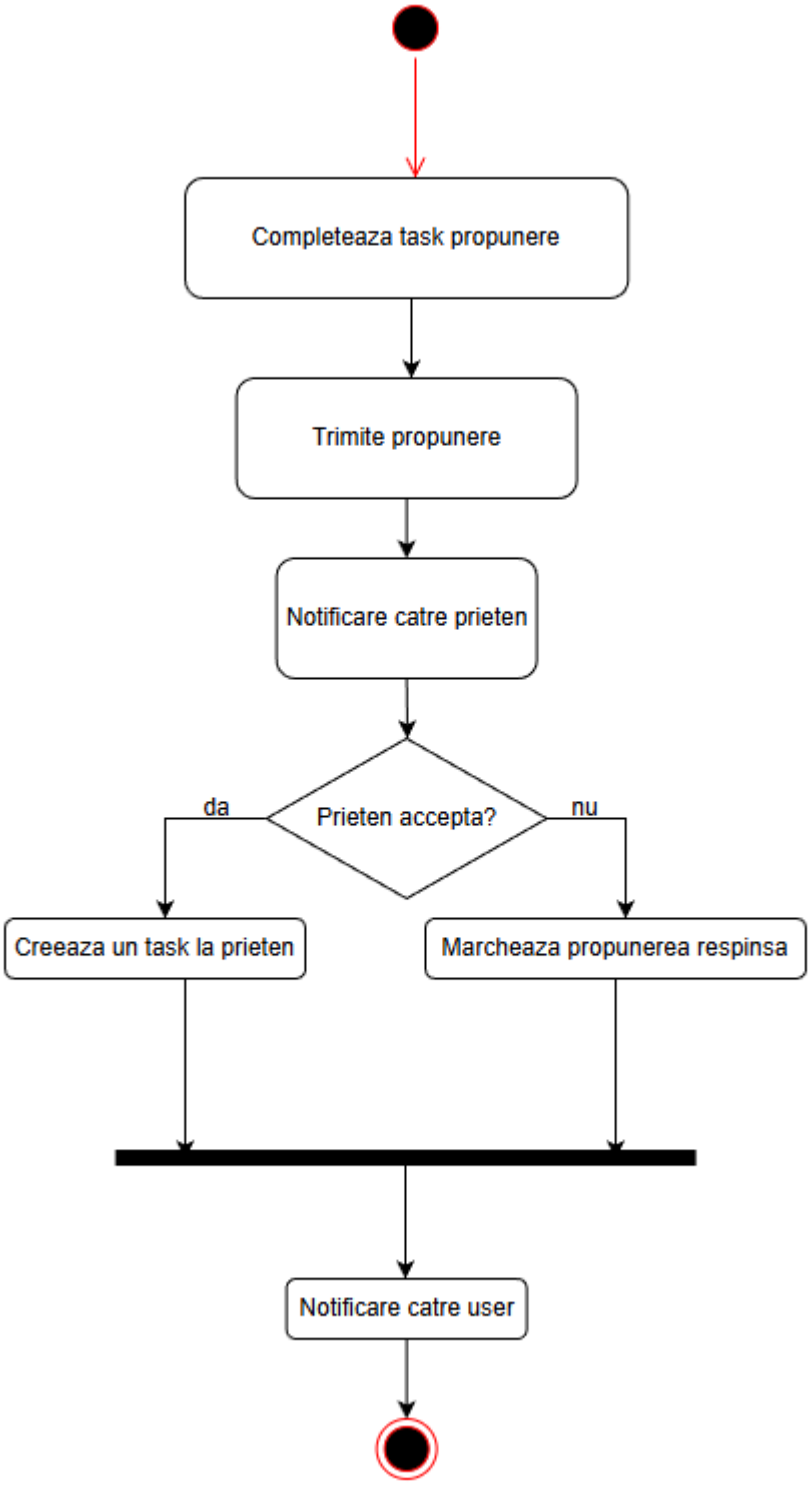


State Diagram

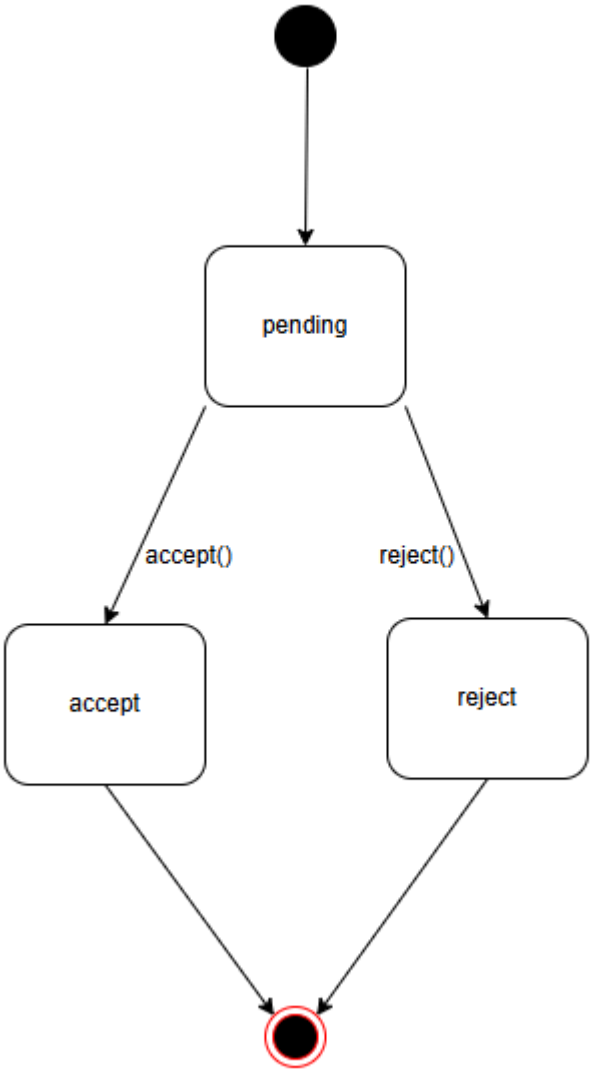


Luca Gunter:

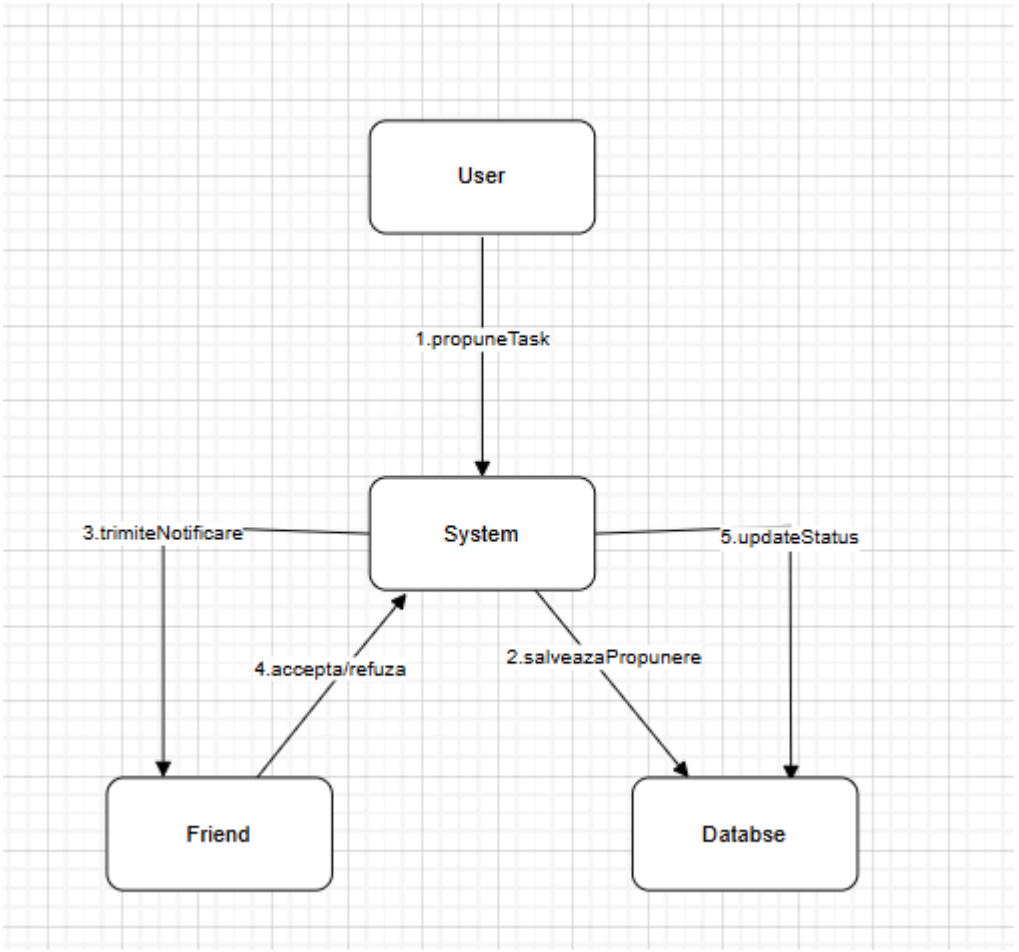
Activity Diagram



State Diagram

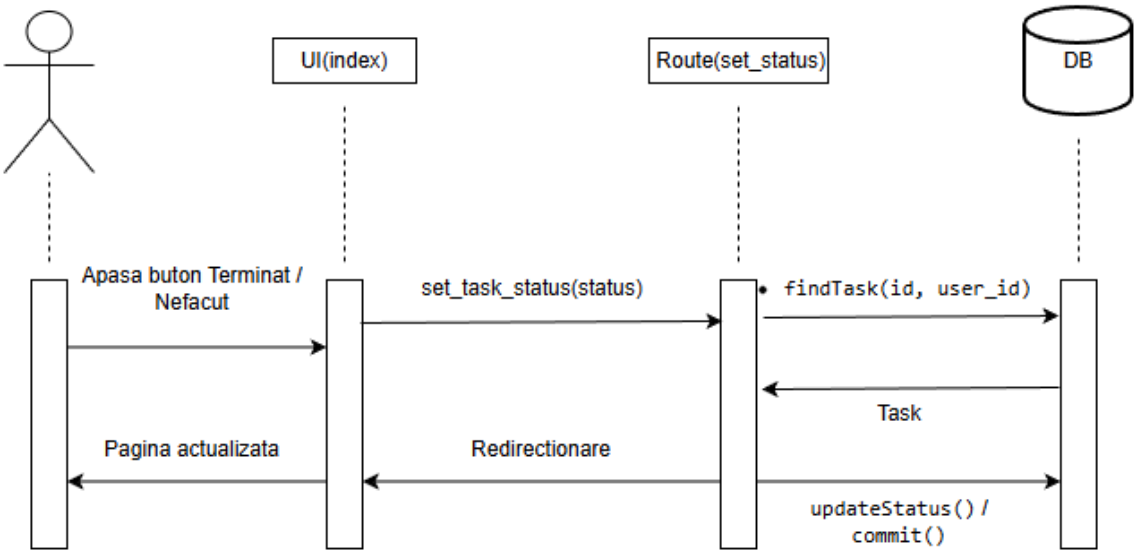


Communication Diagram

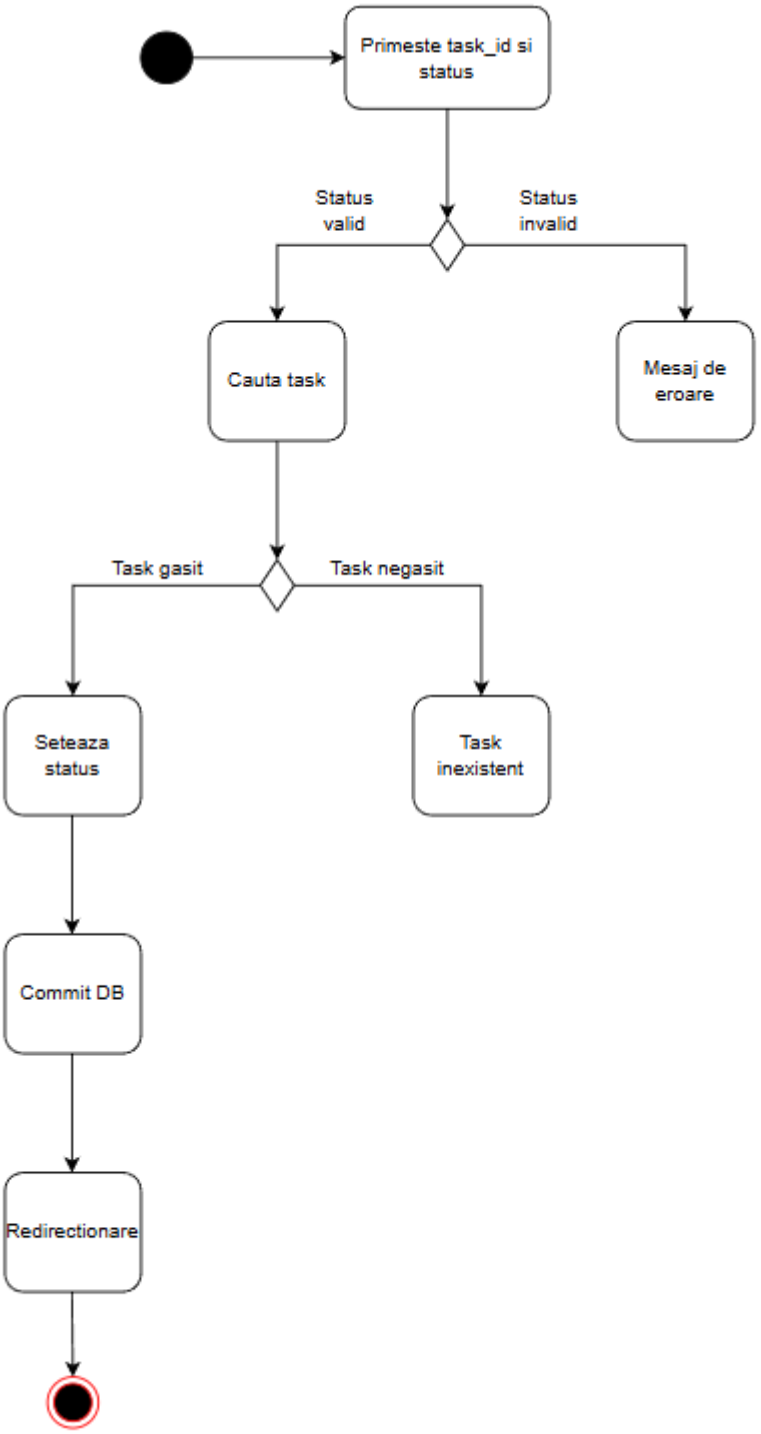


Malan Andreea:

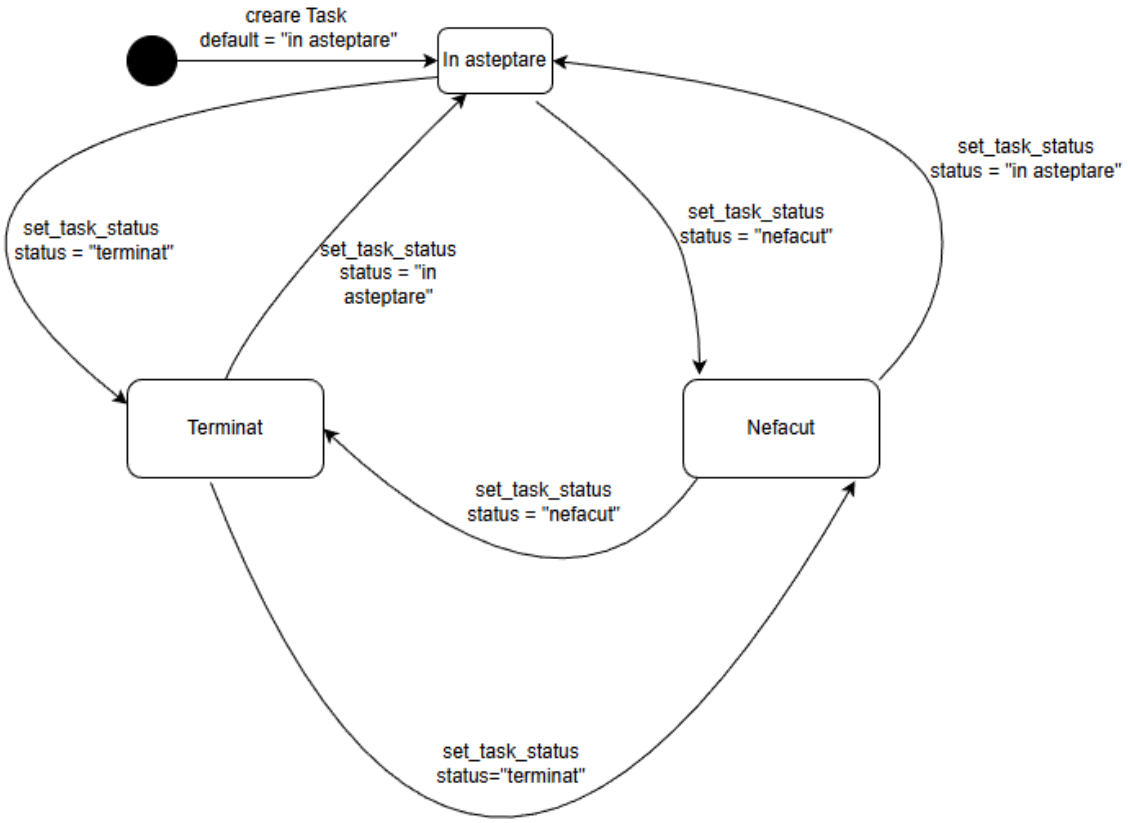
Sequence Diagram



Activity Diagram

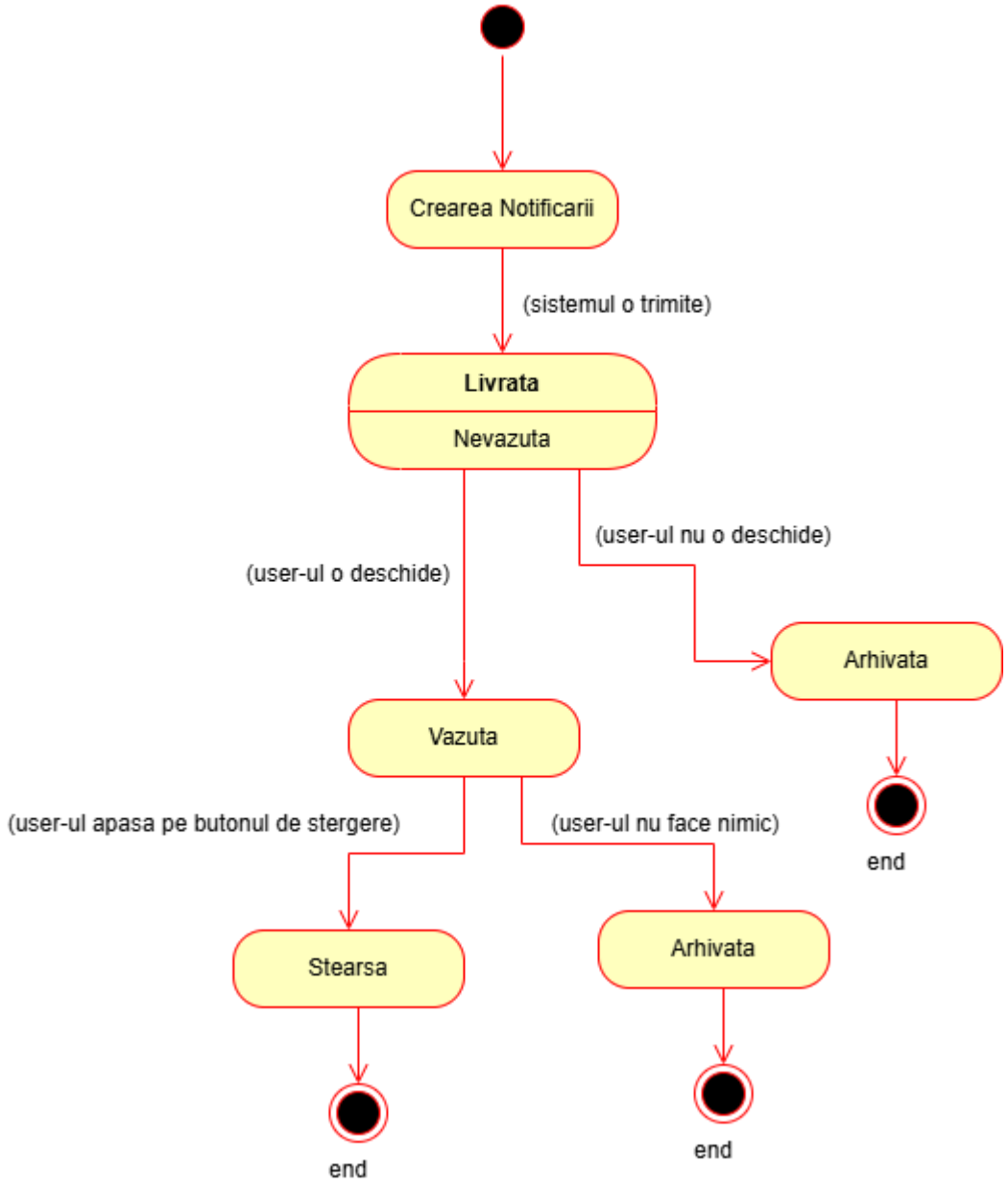


State Diagram

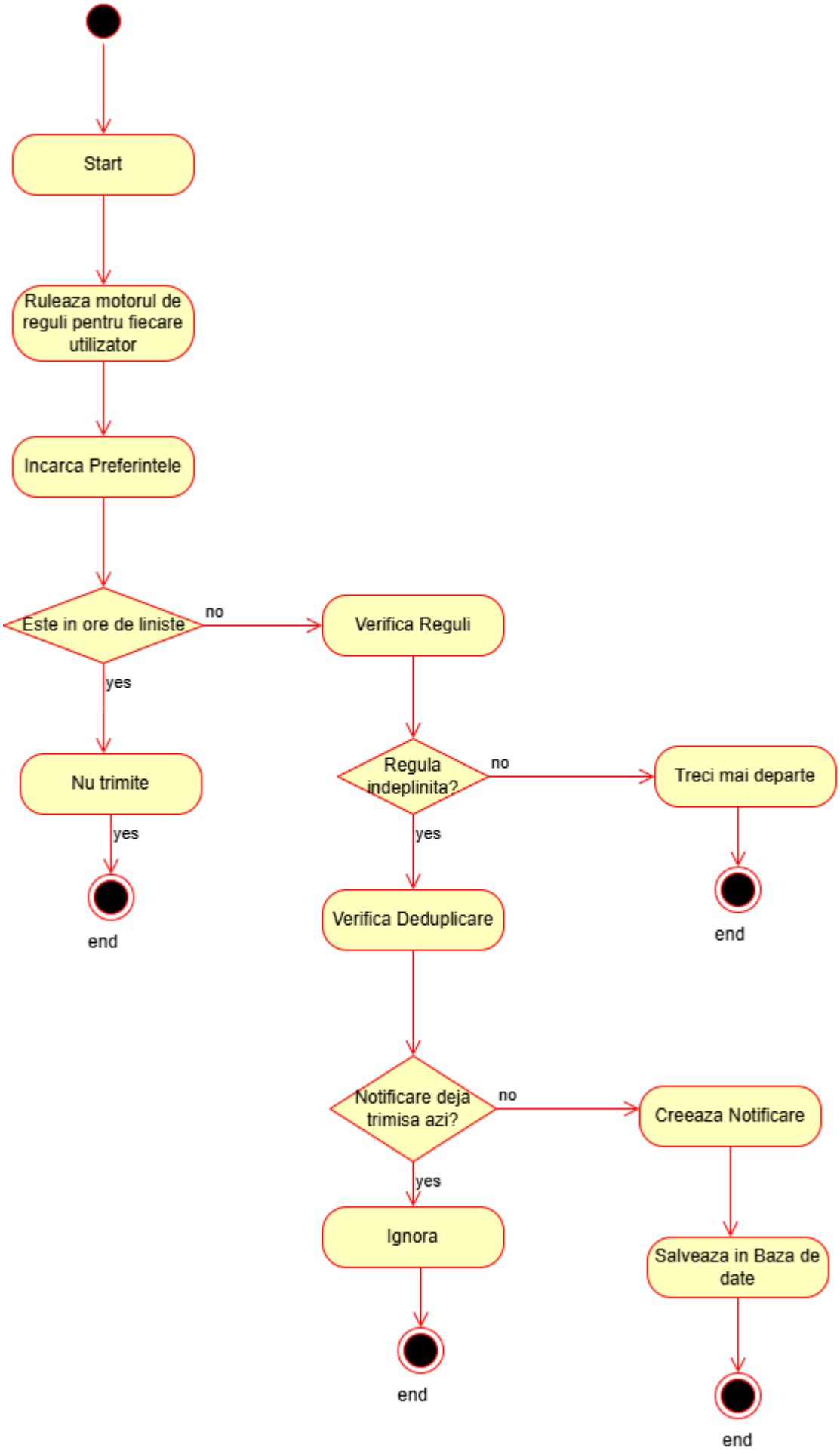


<u>Pavel Catalin:</u>

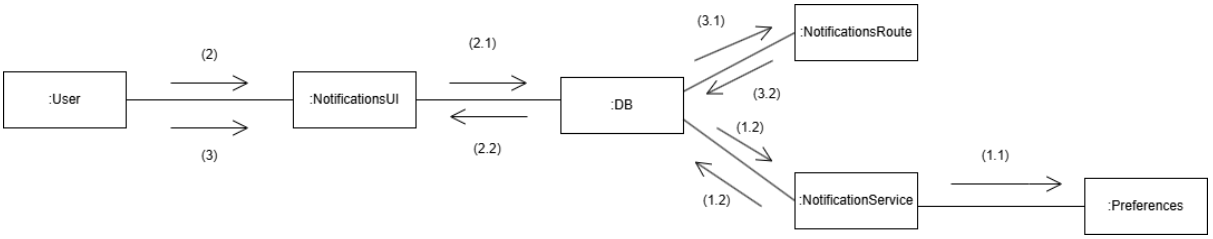
State Diagram



Activity Diagram


Start
Ruleaza motorul de reguli pentru fiecare utilizator
Incarca Preferintele
Este in ore de liniste
no
yes
Nu trimite
yes
end
Verifica Reguli
Regula indeplinita?
no
yes
Treci mai departe
end
Verifica Deduplicare
Notificare deja trimisa azi?
no
yes
Ignora
end
Creeaza Notificare
Salveaza in Baza de date
end

Communication Diagram



1) – apare un eveniment sau o regula (de exemplu o invitatie la un task sau o regula din sistem) care declanseaza crearea unei notificari.

(1.1) – serviciul de notificari verifica setarile utilizatorului (daca are activat acest tip de notificare, daca nu este in quiet hours).

(1.2) – notificarea este salvata in baza de date.

(2) – utilizatorul deschide aplicatia si vrea sa vada notificarile.

(2.1) – interfata cere din baza de date notificarile care nu au fost vazute.

(2.2) – baza de date trimite lista de notificari catre interfata.

(3) – utilizatorul apasa pe butonul de „mark as seen” sau „clear”.

(3.2) – notificarile sunt actualizate in baza de date ca fiind vazute.

6. Design Patterns

Denis - Singleton Pattern

Singleton este folosit pentru a pastra toate setarile si resursele importante ale aplicatiei intr-un singur loc. Acest lucru asigura faptul ca exista o singura instanta a configuratiei si a conexiunii la baza de date pe tot parcursul rularii aplicatiei.

In SmartPlanner, acest pattern este utilizat pentru configurarea aplicatiei si pentru conexiunea la baza de date, astfel incat toate componentele sa foloseasca aceleasi setari. Un exemplu de functionalitate care poate fi controlata prin Singleton este modul Dark Mode sau alte setari globale.

Gunter - Observer Pattern

Observer Pattern este folosit pentru a permite actualizarea automata a mai multor componente atunci cand apare o modificare in sistem.

In SmartPlanner, acest pattern este aplicat in sistemul de notificari. Atunci cand un task este creat, modificat, finalizat sau cand un prieten trimite o cerere, utilizatorii implicati sunt notificati automat. Acest lucru permite sincronizarea interfetei, a notificarilor si a prietenilor fara ca aceste componente sa fie legate direct intre ele.

Andreea - Builder Pattern

Builder Pattern este folosit pentru a separa procesul de creare a unui obiect complex de reprezentarea sa finala, permitand construirea acestuia pas cu pas, intr-un mod controlat si flexibil.

In SmartPlanner, acest pattern este aplicat in sistemul de creare a task-urilor. In loc ca obiectele Task sa fie construite direct in rute cu multi parametri, aplicatia foloseste TaskBuilder pentru a construi un task pornind de la datele primite de la utilizator. Builder-ul se ocupa de validarea datelor (de exemplu, ca ora de final sa fie dupa ora de inceput) si de configurarea corecta a tuturor campurilor necesare.

Catalin - Proxy Pattern

Proxy Pattern este folosit ca un intermediar intre utilizator si functionalitatile aplicatiei.

In SmartPlanner, proxy-urile controleaza accesul la anumite rute si operatii, verificand drepturile utilizatorilor inainte ca actiunea sa fie executata. De exemplu:

- utilizatorii restrictionati nu pot crea taskuri
- doar adminii pot accesa panoul de administrare
- accesul la anumite functii este monitorizat si validat

Acest pattern permite adaugarea de reguli, verificari si optimizari fara a modifica codul principal al functionalitatilor.

Concluzie

SmartPlanner este o aplicatie care combina organizarea taskurilor cu colaborarea si un sistem inteligent de notificari. Proiectul ofera utilizatorilor control mai bun asupra

timpului si o experienta mai interactiva. Prin integrarea mai multor module si design patterns, aplicatia este usor de extins si intretinut. Sistemul rezultat este practic, flexibil si potrivit pentru utilizarea zilnica.