Fisa documentatie proiect

# Ce ne propunem?

Ne propunem sa realizam o aplicatie cu un anumit set de functionalitati, urmand specificatiile de mai jos.

O imagine care conține text

Descriere generată automat

# Functionalitate necesara

Asa cum indica cerintele, functionalitatea va fi impartita in 6 categorii, dupa cum urmeaza:

1. Adauga cheltuiala
2. Stergere
3. Cautari
4. Rapoarte
5. Filtrare
6. Undo

Acestor functionalitati li se alatura operatiile de interactiune cu utilizatorul, considerate fundamentale pentru a avea o aplicatie folositoare.

# Structura/ Arhitecura proiect

Pentru realizarea proiectului, vom folosi limbajul Python si vom crea o interfata tip consola pentru a interactiona cu utilizatorul si a primi datele necesare de la acesta pentru prelucrare.

Proiectul va fi impartit pe module, fiecare cu propria sarcina, astfel incat organizarea sa fie ideala si codul cat mai usor de gestionat. Fiecare functionalitate va detine un modul separat, functiile fiind impartite in aceste module in functie de tematica scopului lor.

Ideologia de lucru urmarita va fi cea incrementala, bazata pe functionalitati, astfel ca orice linie de cod scrisa are drept scop final o noua functionalitate. Pentru a reusi acest lucru, vor fi efectuate teste pentru a verifica fiecare functie noua. Urmarim sa validam codul prin teste pentru fiecare operatie intermediara.

# Scenarii de rulare Imaginate

|  |  |  |
| --- | --- | --- |
| Intentie utilizator | Rezultat program | Descriere limbaj natural |
| Deschidere program | UI | Afisare mesaje corespunzatoare si meniu principal |
| Adauga cheltuiala | UI-1 | Afisare meniu func. 1; introducere de date |
| Introducere date | Operatii interne+UI | Datele sunt adaugate si validate; se revine la afisare meniu |
| Afisare date | UI | Datele sunt afisate prin intermediul interfatei construite |

|  |  |  |
| --- | --- | --- |
| Intentie utilizator | Rezultat program | Descriere limbaj natural |
| Stergere date | UI | Afisarea meniului de stergere date |
| Selectare date pentru sters | Operatii interne + UI | Efectuarea modificarilor, confirmarea si testarea reusitei + afisarea meniului pentru feedback |
| Cautare informatii | UI | Afisarea meniului corespunzator cautarii |
| Introducere date cerute | Operatii interne + UI | Cautarea datelor cerute + Afisarea lor, prin interfata consola |

# Cazuri de testare

Aceasta categorie ilustreaza situatii posibile ale datelor de intrare, unele valide, altele invalide, aratand si modul in care aplicatia va interpreta astfel de scenariu.

|  |  |  |
| --- | --- | --- |
| Date introduse | Operatia intentionata | Rezultat |
| Zi:1 Suma:1 Tip:mancare | Adauga cheltuiala | Cheltuiala adaugata in baza de date |
| Zi:-1 Suma:1 Tip:mancare | Adauga cheltuiala | Eroare si intoarcerea la meniul de adaugare |
| Zi:1 Suma:1 Tip:imbracaminte | Actualizeaza cheltuiala | Se actualizeaza cheltuiala, dupa ce a fost identificata |
| Zi: 13 | Sterge cheltuieli pentru o zi data | Toate cheltuielile din acea zi sunt sterse permanent |
| Zi\_start:1 Zi\_sfarsit:31 | Sterge cheltuieli pentru interval dat | Toate cheltuielile, din toate zilele, sunt sterse |
| Tip: mancare | Sterge cheltuieli de un anumit tip | Toate cheltuielile de tip “mancare” vor fi sterse |
| Tip:fmaf | Sterge cheltuieli de un anumit tip | Validarea input esueaza si utilizatorul se intoarce la meniu stergeri |
| Suma: 3 | Tipareste cheltuieli criteriu suma | Toate cheltuielile care se incadreaza in criteriu sunt tiparite |
| Zi: 2, Suma: 34 | Tipareste cheltuieli criteriu criteriu zi si suma | Toate cheltuielile dinainte de ziua 2 si cu suma mai mica decat 34 vor fi afisate |
| Tip: intretinere | Tipareste cheltuieli criteriu tip | Toate cheltuielile referitoare la intretinere sunt sterse |
| - | Undo | Se va reveni la ultima stare cunoscuta |
| Suma: 5.0 | Raport: gasire cheltuieli dupa suma | Se vor gasi si apoi tipari toate cheltuielile care au o anumita suma |

# Planificare

|  |  |
| --- | --- |
| **Iteratie** | **Functionalitati planificate** |
| I1 | Adaugare, Stergere, Cautare |
| I2 | Rapoarte, Filtrare |
| I3 | Undo |

**Iteratia 1:** 13.10.2021 – 20.10.2021

Pentru perioada aceasta este planuita implementarea functionalitatilor 1-3

Pentru a incepe procesul, cream fundatia programului, peste care vom construi fiecare functionalitate. Pentru aceasta iteratie, blocurile de cod vor fi pastrate in acelasi fisier

**16.10.2021:**

* + Stabilire plan prin crearea documentatiei (prezentul document).
  + Fundatie
  + Implementare functionalitate 1
  + Teste

Structura “bazei de date” poate fi gandita drept o lista de dictionare, sau drept un dictionar de dictionare. A doua optiune permite incadrarea unui ID pentru identificarea fiecarei cheltuieli/tranzactii, astfel incat este cea la indemana dpdv logic. Dpdv al implementarii, lista de dictionare este mai usor de creat, insa este importanta organizarea si mentinerea pe termen lung, astfel ca va fi incercata metoda cu dictionar de dictionare. Structura aleasa este de tipul:

Baza de date = {

Id1: {date\_tranzactie1},

Id2: {date\_tranzactie2},

…

}

Unde

Date\_tranzactieI = {

“ziua”: integer,

“suma”: float,

“tipul”: string

}

Si Id\_i : string

Aceste date avand restrictii de validare conform specificatiilor

Fundatia programului este reprezentata de existenta meniurilor de optiuni in cascada si a posibilitatii de a alege diferitele functionalitati. Vom considera parte din aceasta fundatie si validarea de date si gestionarea cazurilor de date invalide de intrare; ne vom asigura deci ca programul nu va avea erori neprevazute ce pot duce la oprirea functionarii.

Functionalitatea 1 se considera implementata in momentul functionarii complete a subpunctelor mentionate in cerinta, impreuna cu testele de rigoare ce asigura corectitudinea operatiilor si validarea in continuare a rezultatelor.

Se va considera necesara si salvarea datelor pentru accesare ulterioara, astfel ca vom folosi tipul de fisier json pentru a salva datele in momentul modificarii in memorie si in momentul iesirii.

Intrarea in program va initia incarcarea in memorie a datelor, iar acestea vor ramane permanent salvate pe hard-disk.

Aceste operatiuni vor avea si ele o parte de testare.

La finalul zilei, functionalitatea 1 este implementata cu validare a datelor de intrare si protejare impotriva erorilor pe baza datelor de intrare eronate.

Lipsesc insa testele separate pentru testarea functionalitatii.

**17.10.2021:**

* Testare
* Functionalitate 2 + UI
* Functionalitate 3 + UI
* Testare + validare pentru functionalitatile noi

Efortul acestei zile incepe prin a crea functii de testare separata

Acum functiile de testare a functionalitatii 1 sunt complete, putem trece la implementarea func. 2

Dar inainte de asta vom implementa o functie de salvare a datelor atunci cand utilizatorul face schimbari sau cand iese din program

Acum functionalitatile 2,3 sunt implementate impreuna cu testarile corespunzatoare, si ca bonus o functie ce poate genera o baza de date pentru testare exista

Baza de date folosita de utilizator este salvata in memorie nevolatila la fiecare operatie ce o modifica, inclusiv la iesire.

La finalul zilei deci am incheiat iteratia 1

E de refactorizat mult cod pentru iteratia 2

**Iteratia 2 (3 de fapt):** 27.10.2021 – 3.11.2021

**27.10.2021:**

Nu exista inca un plan propriu-zis pentru aceasta iteratie. Nu a mai fost facuta munca pentru aplicatie, asa ca ramane de refactorizat mult cod si acum de asemenea de a ascunde implementarea pe cat posibil.

Trebuie deci sa regandesc structura si scopul fiecarei functie si cum comunica intre ele pentru a usura si liniile de cod si munca fiecarei functii si a face totul cat mai reutilizabil.

Pt saptamana viitoare trebuie facut modular

Ui

Domain

Infrastructura

**30.10.2021:**

Am completat refactorizarea codului existent, pentru a reduce functiile nenecesare si a maximiza refolosirea codului existent.

Raman de reimplementat testele si celelalte 2 functionalitati:

Rapoarte

Undo

**31.10.2021:**

Vor fi implementate rapoartele, apoi functia Undo, apoi refactorizate testele.

Implementarea astfel incat forma organizarii in memorie este obfuscata este dorita.

Din fericire, am facut trecerea la getteri si setteri astfel incat reorganizarea in memorie sa poata fi facuta cu modificare minima de cod.

Functionalitatea este acum complet implementata, functia de undo se bazeaza pe un fisier backup care tine minte toate versiunile prin care a trecut lista, astfel ca poate da undo de un nr de ori de cate ori a fost modificata lista (in limita scrierii intr-un fisier json)