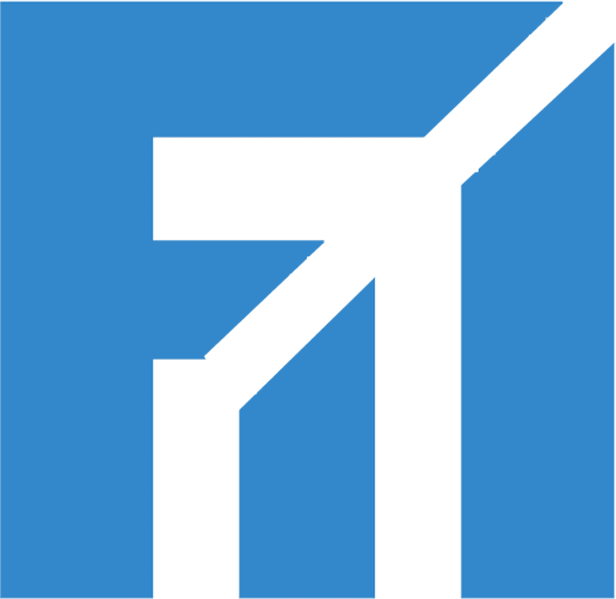
UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI

**Facultatea de Informatică din Iași**



LUCRARE DE LICENȚĂ

**SMART FINANCIAL ADVISOR**

**propusă de**

*MANOLIU VICTOR-CODRIN*

**Sesiunea:** *Iulie, 2019*

**Coordonator științific**

**Prof.Colab. Florin Olariu**

UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI

**Facultatea de Informatică din Iași**

**SMART FINANCIAL ADVISOR**

*MANOLIU VICTOR-CODRIN*

**Sesiunea:** *Iulie, 2019*

**Coordonator științific**

**Prof.Colab. Florin Olariu**

**Cuprins**

[**Introducere** 2](#_Toc12367887)

[**Motivația** 3](#_Toc12367888)

[**Noutate** 4](#_Toc12367889)

[**Obiective** 4](#_Toc12367890)

[**Descrierea sumară** 4](#_Toc12367891)

[**Structura Lucrării** 4](#_Toc12367892)

[**Contribuții** 5](#_Toc12367893)

[**1.** **Descrierea problemei** 6](#_Toc12367894)

[**2.** **Abordări anterioare** 6](#_Toc12367895)

[**3.** **Descrierea soluției** 6](#_Toc12367896)

[**3.1** **Arhitectura Proiectului** 12](#_Toc12367907)

[**3.2** **Arhitectura Bazei de Date** 16](#_Toc12367908)

[**3.3** **Arhitectura modelului** 17](#_Toc12367909)

[**3.3.1** **Antrenarea modelului** 18](#_Toc12367910)

[**3.4 Pașii Clientului** 20](#_Toc12367911)

[**Concluziile lucrării** 27](#_Toc12367912)

[**Bibliografie** 33](#_Toc12367918)

# **Introducere**

O aplicație de tip advisor în orice domeniu, nu doar cel economic este definită ca fiind un software care pe baza unui anumit input oferit de către utilizator, poate crea un model folosind datele respective și poate oferi soluții în vederea îmbunătățirii rezultatelor extrase din modelul menționat anterior, soluții ce pot varia în funcție de domeniul de activitate al aplicației respective.

Scopul acestei lucrări este de a construi o aplicație inteligenta de consiliere financiara, cât mai ușor de folosit pentru utilizator și care să ascundă pe cât posibil detaliile complicate din punct de vedere financiar. În momentul actual, mai există produse similare pe piața doar ca acestea au un cerc de utilizatori restrânși, iar utilizatorul final care ar folosi aplicația sunt angajați ai sucursalelor de bănci fie oameni cu o experiență economică anterioară ridicata, lucru dat de gradul încărcării aplicației cu detalii sau formule financiare greu de înțeles pentru un utilizator ce nu are aceste cunoștințe în prealabil.

În cadrul acestei lucrări mi-am produs să dezvolt o aplicație capabilă să realizeze un profil financiar al clientului, bazat pe datele sale personale precum: orașul de domiciliu, orașul de reședința, domeniul în care își desfășoară meseria, veniturile sale care pot proveni fie dintr-o sursă directă fie din proprietăți sau companii iar pe baza acestui profil financiar îi pot estima probabilitatea îndeplinirii unor obiective economice setate de acesta folosind modelul statistic de regresie liniară multiplă[1]. Elementul de inovație al acestei aplicații, este așa cum am menționat anterior, simplitatea în utilizare și ascunderea detaliilor complicate care pentru un utilizator normal nu ar fi adus nicio valoare din punct de vedre al experienței acestuia, putem chiar afirma în acest sens că aplicația este o implementare a design patternului de Façade.

Proiectul înglobează 3 servicii de backend decuplate și conectate la baza de date, fiecare expunându-și funcționalitatea prin implementarea paradigmei REST și a design pattern-ului MVC, că arhitectură folosindu-se pentru toate arhitectură de tip 3-tier[[1]](#footnote-1) , alegerea arhitecturii fiind determinată de dorința de a avea flexibilitatea modificării și adăugării de noi nivele și posibilitatea de refolosire a nivelelor deja existente, crescând astfel scalabilitatea aplicației per total. Serviciul de front-end va realiza requesturi HTTP la cele 3 servicii menționate anterior pentru obținerea sau introducerea de date în baza de date. Motivul pentru care am ales 3 servicii diferite pentru partea de back-end este de a scoate în evidență deschiderea aplicației pentru adăugare de noi funcționalități, cele 3 servicii fiind fiecare scris într-un limbaj de programare diferit, frameworkuri și/sau tehnologii diferite, alegerea limbajului și a tehnologiilor folosite bazându-se pe motive de performanță, ușurință în implementarea anumitor funcționalități, de exemplu: Python oferă foarte multe librării pe partea de statistică și Machine Learning în timp ce Java oferă framework-ul de Spring Boot ce ne pune la dispoziție posibilitatea de a scrie relativ ușor o aplicație web.

### **Motivația**

Motivația din spatele alegerii proiectului de față este dată de gradul de complexitate al aplicațiilor de acest tip din punctul de vedere al utilizatorului final, restrictitia grupului de utilizatori ce ar avea accesul la o asemenea aplicație.

### **Noutate**

În urmă unor cercetări amănunțite în domeniul aplicațiilor cu specific economic/bancar, proiectul de față aduce elemente de noutate chiar și față de cel mai apropiat competitor menționat în cadrul capitolului de “Abordări Anterioare”. Elemente de noutate aduse sunt : posibilitatea ascunderii detaliilor financiare de către utilizator, posibilitatea estimării probabilității unor obiective financiare introduse direct de către utilizator.

### **Obiective**

Obiectivul proiectului, în momentul actual este a de construi o aplicație Web, accesibilă oricărui utilizator indiferent de cunoștințele sale în domeniul economic/bancar ce poate îi conferi acestuia posibilitatea de a primi o estimare statistică cu un grad de acuratețe cât se poate de ridicat în momentul de față asupra obiectivelor financiare care acesta dorește să și le seteze. Estimarea menționată va fi realizată în momentul în care utilizatorul va introduce toate datele necesare atât pentru realizarea unui profil financiar cât și pentru antrenarea modelului de regresie liniară multiplă, date precum: orașul de domiciliu, orașul de reședința, domeniul în care își desfășoară meseria, veniturile sale care pot proveni fie dintr-o sursă directă fie din proprietăți sau companii.

### **Descrierea sumară**

Soluția propusă în contextul de față este constituită din o aplicație de tip Web, care la rândul ei este formată din 3 servicii Web, ce comunica prin intermediul requesturilor de tip AJAX.

### **Structura Lucrării**

În această secțiune vom prezența structuura acestei lucrări alături de componentă fiecărui capitol și legăturile dintre capitole.

Primul capitol, intitulat: “Descrerea problemei” are rolul de a introduce cititorul în cadrul problemei ce proiectul de față încearcă să o rezolve.

Cel de al doilea capitol, intitulat: “Abordări anterioare” va prezența rezultatul cercetării autorului legate de produse similare pe piață în momentul actual ce au că și scop rezolvarea aceleași probleme.

În cel de al treilea capitol, intitulat: “Descrierea soluției” vom discuta la nivel detaliat din punct de vedere tehnic despre solutiea găsită de autor. Capitolul respectiv va prezența atât arhitectură generală a proiectului, modul de comunicare și cum se realizează această efectiv între componentele aplicației, arhitectură bazei de date alături de componentă fiecărei tabele, arhitectură și antrenarea modelului de regresie liniară multiplă, oferindu-se chiar un exemplu cu un rezultat obținut în urmă folosirii modelului și tehnologiile folosite în proiect alături de o scurtă motivație pentru fiecare alegere realizată.

# **Contribuții**

În cadrul acestei secțiuni vom prezenta contribuțiile aduse în rezolvarea problemei menționate în secțiunea anterioară de: “Introducere”.

Am ales ascunderea detaliilor financiare de către utilizator pentru a oferi o experiență cât mai prietenoasă, putând afirmă faptul că aplicația respectă design patternul Façade. De asemenea, în materie de tehnologii am încercat să realizez alegeri cât mai atractive și care să reducă pe cât posibil de mult gradul de dificultate în implementarea cerințelor necesare, alegerile făcute fiind determinate și de dorința construiții unor direcții de viitor ale aplicației. Mai multe detalii tehnice vor fi prezentate în capitolele ce urmează.

# **Descrierea problemei**

Dându-se o mulțime de utilizatori ce nu posedă cunoștințe din mediul economic sau bancar și care doresc să afle o estimare probabilistică asupra obiectivelor financiare setate, să se construiască o aplicație care să poată deservi acest scop într-o manieră cât mai plăcută și prietenoasă din punctul de vedere al experienței utilizatorului final. Utilizatorul final nu trebuie să fie clientul unei anumite bănci în particular, sau să fie nevoit să dețină o suma minimă de bani.

# **Abordări anterioare**

În momentul actual nu mai există alte abordări anterioare în care aplicația să își deservească scopul având că utilizatori finali oamenii de rând. Singurul competitor direct la momentul actual poate fi considerat aplicația “Revolut”, dar care în momentul actual nici acesta nu oferă posibilitatea utilizatorilor de a primi estimări de probabilitate asupra unor obiective financiare ci doar o analiză la nivel statistic a cheltuielilor realizate în ultimele luni. Pentru o detailare mai succintă, cititorul este rugat să vizioneze secțiune intitulată “Introducere”.

# **Descrierea soluției**

Soluția problemei constă într-o aplicație web formată din 3 servicii web decuplate ce interacționează unul cu celălalt prin intermediul requesturilor HTTP [1]. Unul din cele 3 servicii conține un model de tip regresie liniară multiplă antrenat pe un set de date produs direct din aplicație și apoi corectate pentru îmbunătățirea gradului de acuratețe. În această secțiune vom prezența pe scurt tehnologiile folosite, motivația alegerii realizate dar și unde a fost această utilizată.

### **Java**

Java [6] este un limbaj de programare orientat-obiect, puternic tipizat, conceput de către James Gosling la Sun Microsystems (acum filiala Oracle) la începutul anilor 90, fiind lansat în 1995. Cele mai multe aplicații distribuite sunt scrise în Java, iar noile evoluții tehnologice permit utilizarea să și pe dispozitive mobile precum un telefon cu sistemul de operare Android sau o agenda electronică, în acest fel creându-se o platforma unică, la nivelul programatorului, deasupra unui mediu eterogen extrem de diversificat. Acesta este utilizat în prezent la o scala mondială cu succes în programarea aplicațiilor atât destinate intranet-ului cât și extra-netului.

În proiectul de față, Java a fost folosit în programarea serviciului de backend destinat management-ului informațiilor despre client precum veniturile, datoriile acestuia dar și realizarea profilului financiar.

Motivația alegerii acestui limbaj este experiență anterioară a autorului cu acest limbaj câștigate în cadrul materiilor din cadrul ciclului de licență.

### **Spring Boot**

Spring Boot [7] este un framework cu sursa deschisa folosit ca alternative la modelul Enterprise JavaBeans.

În implementarea proiectului am folosit Spring Boot în realizarea serviciului web ce realizeaza management-ul datelor clientului punându-se la dispoziție posibilitatea de a integra în serviciul respectiv arhitectura MVC[[2]](#footnote-2) și sa expunem toate funcționalitățile sub forma unui API ce implementează paradigma REST si care poate fi refolosit atât în implementări viitoare ale altor aplicații cu un specific similar cat și în adăugarea de funcționalități pentru proiectului curent.

Motivul alegerii acestei tehnologii în pofida modelului Enterprise JavaBeans este documentația vasta și bine realizata, experiența anterioara a autorului cat și varietatea alegerii de tehnologii pe partea de Persistance si ORM[[3]](#footnote-3).

### **Spring Data**

Spring Data [8] este un framework cu sursă deschisă ce face parte din familia Spring și abstractizează JPA[[4]](#footnote-4) venind ca o alternativa a modelului JDBC[[5]](#footnote-5).

În implementare, Spring Dată a fost folosit în cadrul realizării serviciului web de management a datelor clienților, mai precis la nivelul datelor. Ni se pune la dispoziție, posibilitatea de a prelua și sorta date din baza de date într-un mod relativ mai ușor față de metodă clasică de a scrie query-uri astfel reducând numărul de linii de cod redundante, ușurință în respectarea principiilor ACID și eliminarea posibilelor metode ce ar putea apărea într-o implementare clasică, spre exemplu: deschiderea și închiderea unei tranzacții.

Motivul pentru care am ales Spring Dată în detrimentul altor tehnologii similare performanță și experiență anterioară a autorului.

### **C#**

C# [9] este un limbaj de programare orientat-obiect ce înglobează principii precum programarea funcțională, programare imperativă și declarativă, conceput de către Microsoft în jurul anului 2000 în cadrul inițiative .NET, limbaj ce mai târziu a fost aprobat că și standard de către ECMA (ECMA-334) și ISO (ISO/IEC 23270) [10].

În cadrul acestui proiect, C# a fost folosit pentru realizarea serviciului de autentificare a utilizatorilor în aplicație și înregistrare a noilor utilizatori ce doresc să folosească această aplicație. Motivația alegerii este dată atât de experiență autorului cu acest limbaj de programare cât și dovedirea ideii de scalabilitate a acestei aplicații pe mai multe limbaje de programare.

### **ASP.NET Web API**

În proiect, ASP.NET Web API [11] a fost folosit pentru realizarea serviciului de autentificare a utilizatorilor în aplicație inclusiv generarea token-ului unic pentru fiecare utilizator, ce va fi folosit mai târziu că validare inițială a oricărui request.

Am ales folosirea acestui framework deoarece ni se pune la dispoziție posibilitatea de a expune funcționalitățile de autentificare și înregistrare sub formă unui API construit pe arhitectură MVC ce respectă paradigmă REST [2] relativ ușor întărind astfel ideea de scalabilitate a fiecărui serviciu și a aplicației per total.

### **Entity Framework**

Entity Framework [12] este un framework cu sursă deschisă ce are rolul de abstractiza nivelul de date al aplicației unde este folosită.

În implementarea proiectului, Entity Framework a fost folosit în nivelul de date al serviciului ce realizează autentificarea clienților, mai precis în maparea tabelelor din baza de date la obiecte .NET, și scrierea query-uri în mod mai eficient folosind LINQ.

Am ales această tehnologie deoarece ni se pune la dispoziție posibilitatea de a reduce numărul de linii de cod redundante oferite de metoda clasică de a scrie query-uri în sensul obținerii și adăugării de date în baza de date, reducerea problemelor ce ar fi putut apărea cu metodă clasică și reducerea efortului implementării în mod corespunzător principiilor ACID.

### **Python**

Python [13] este un limbaj de scripting, mulți-paradigmă. Motivul alegerii acestui limbaj pentru realizarea serviciului de prezicere a probabilității de realizare a unui obiectiv este sintaxa ușoară cât și suportul mare pentru biblioteci și frameworkuri de statistică, atât că documentație cât și că implementări efective. În această secțiune vom numi succint modulele folosite:

1. **Sklearn** [4]: pentru realizarea modelului de regresie liniara multipla, antrenarea acestuia și pentru prezicerea efectiva a coeficientului.
2. **Numpy** [14]: pentru reprezentarea inputului modelului ca și lista dublu-dimensionata
3. **Pymsql:** pentru conectarea la baza de date, preluarea datelor de antrenament și inserarea unui obiectiv.

### **Flask**

Flask [15] este un micro-framework[[6]](#footnote-6) cu sursa deschisa realizat de Armin Ronacher of Pocoo, un grup internațional de programatori entuziasmați ai limbajului Python și reprezinta o alternativa a serverului CGI[[7]](#footnote-7) [16].

Am folosit această tehnologie în realizarea serviciului de prezicere al obiectivelor financiare, mai precis în expunerea acestei funcționalități prin intermediului unui API ce respectă paradigmă REST și implementează design pattern-ul MVC.

Am ales Flask pentru îndeplinirea acestui task în detrimentul serverului de tip CGI, în ciuda absenței unor funcționalități cu un grad ridicat de importantă în realizarea oricărei aplicații web din motivele de documentație foarte bine realizată și gradul scăzut de dificultate în familiarizare și maparea requesturilor HTTP în implementare.

### **Angular**

Angular [17] este un framework de dezvoltare web cu sursă deschisă bazat pe limbajul de programare TypeScript. Proiectul a fost realizat și este în continuare dezvoltare de către Echipa Angular de la Google și de comunitatea de utilizatori individuali și companii, fiind o rescriere completă a framework-ului AngularJS. Scopul principal al acestei tehnologii este de a simplifică dezvoltarea și testarea aplicațiilor web încurajând refolosirea componentelor, astfel încărcarea DOM-ului se face într-un mod dinamic.

În realizarea acestui proiect, am folosit Angular în realizarea serviciului web pe partea de client ce va expune utilizatorului interfață aplicației alături de datele din această. Motivele alegerii acestui framework sunt: fost suportul mare pentru module de tip third-party cât și interne, sintaxa ușoară și experiență anterioară a autorului.

În aceasta secțiune vom enumera succint câteva din modulele folosite:

* Angular-Bootstrap: pentru realizarea interfeței ce va fi expusa utilizatorului, într-un mod cat mai intuitiv.
* Ngx-Cookie-Service: pentru adăugarea, ștergerea și verificarea existentei unei cookie pe partea de client.
* Router: pentru maparea diferitelor route la pagini efective.
* HttpClient: pentru realizarea de request-uri HTTP la endpoint-uri externe.

### **SonarQube**

SonarQube este o platforma cu sursă deschisă dezvoltată de către echipa Sonar. Platforma are rolul de a realiza o analiză continuă asupra codul oricărui proiect în scopul detectării bug-urilor, vulnerabilităților de securitate.

În cadrul acestui proiect am folosit SonarQube pentru a analiză în mod continuu codul necesar realizării serviciilor web de autentificare și de management a datelor clientului.

Motivația alegerii acestei tehnologii este dorința de a scrie un cod cât mai curat, asigurarea respectării principiilor de ingineria programării și posibilitatea de întreținere și continuare a dezvoltării proiectului cu un timp minim de înțelegere a codului.

Rezultatele obținute în urma analizării codului serviciilor web pot fi vizionate în **Anexa 1**.

## **Arhitectura Proiectului**

Proiectul complet este compus din 4 componente:

* Un serviciu web bazat pe .NET Core Web API. Rolul acestui serviciu este de a realiza autentificarea și înregistrarea utilizatorilor pentru înregistrarea sau aprobarea accesului în aplicație. De asemenea, la logare generează un token unic pentru fiecare utilizator folosind algoritmul SHA256[[8]](#footnote-8) pe adresa de email pentru fiecare utilizator ce va fi setat ca și cookie pentru validarea viitoarelor requesturi ale utilizatorului.
* Un serviciu web bazat pe SpringBoot. Rolul acestei componente este de a realiza management-ul datelor clientului, precum veniturile sau datoriile acestuia. De asemenea, mediatizeaza requesturile intre interfața web și serviciul ce înglobează modelul de regresie liniara în scopul validării datelor.
* Un serviciu web bazat pe Flask. Acest serviciu va îngloba modelul de regresie liniara multiplă și antrenarea acestuia, ambele fiind realizate folosind librăriile: sklearn și numpy. De asemenea, acest serviciu va expune și rezultatul prezicerii realizate de model prin intermediul unui endpoint.
* Un serviciu web bazat pe Angular 6. Acest serviciu va valida fiecare request al utilizatorului, va trimite datele introduse de către client către celelalte servicii, va întoarce răspunsurile acestora și de asemenea va livra utilizatorului codul interfeței web al aplicației.

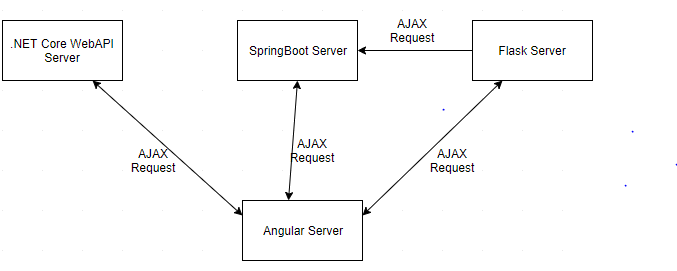


Figura 1: Arhitectura generala a proiectului

În această diagramă putem observă din punct de vedere schematic atât componentele aplicației cât și comunicarea dintre acestea. În implementare am ales requesturi AJAX pentru comunicarea între module din motive de ușurință în implementare, diferența de performanță între AJAX și WebSockets, alternativa cea mai populară este neglijabilă.

Legăturile dintre module:

* **.NET Core WebAPI Server – Angular Server :** reprezintă prima etapa din ciclul de viață al aplicației și anume: autentificarea utilizatorului. Comunicarea arată astfel:
  + Angular Server-ul va prelua datele de autentificare ale utilizatorului, va crea un obiect de tip *JSON* și va realiza un request AJAX [2] către serverul .NET.
  + Serverul va primi datele prin intermediul unui endpoint HTTP, va prelua datele din obiectul *JSON*, și va realiza un query în baza de date iar dacă sunt găsite rezultate valide se va genera token-ul pentru utilizatorul respectiv și se va crea un obiect JSON ce va conține codul de răspuns HTTP alături de id-ul utilizatorului și tokenul generat îl va trimite către serverul Angular.
  + Serverul Angular va primi obiectul prin intermediul unui endpoint iar dacă codul de răspuns este 200[[9]](#footnote-9) atunci va redirecționa utilizatorul către pagină principala de administrare și va crea cookie-uri ce conțin: id-ul utilizatorului autentificat și tokenul.
* **SpringBoot Server - Angular Server:** reprezintă etapa principala a aplicației în care se va realiza management-ul informațiilor necesare atât pentru crearea profilului financiar al utilizatorului cât și pentru informațiile necesare construirii modelului precum veniturile și datoriile utilizatorului. Schimbul de informații realizându-se prin intermediul requesturilor de tip AJAX [2] la diferite endpoint-uri HTTP, informațiile interschimbate fiind împachetate în obiecte de tip JSON. Fiecare request este validat de către serverul Angular, validarea constând în verificarea existenței token-ului asignat utilizatorului în prima etapă înainte de executarea efectivă a requestului, indiferent de natură acestuia
* **SpringBoot Server – Flask Server:** reprezintă ultima etapa a aplicației în care se va crea modelul de regresie liniară multiplă în vederea prezicerii probabilității de realizare a unor obiective financiare introduse de către utilizator pe o anumită durata de ani. Comunicarea se realizează astfel:
  + Serverul de Spring trimite prin intermediul unui request AJAX [2] un obiect de tip JSON ce conține:
    - Id-ul utilizatorului logat în momentul actual;
    - Numele Obiectivului pentru care se va realiza estimarea probabilității de realizare;
    - Valoarea Obiectivului, valoarea monetara ce se dorește a fi economisită;
    - Anii, durata de timp în care utilizatorul dorește sa își îndeplinească obiectivul.
  + Serverul de Flask va primi obiectul prin intermediul unui endpoint, va prelua datele de antrenament ce sunt reprezentate de obiective altor utilizatori și va crea modelul ce va fi antrenat folosind datele menționate anterior iar apoi va realiza prezicerea pentru datele utilizatorului curent. Va stoca obiectivul în baza de date și va trimite la SpringBoot Server un obiect de tip JSON ce conține codul HTTP corespunzător rezultatului execuției.
  + SpringBoot Server va primi răspunsul de la Serverul de Flask și va face un refresh la pagină curentă astfel încât noul obiectiv introdus în baza de date să fie disponibil și în interfață Web.

## **Arhitectura Bazei de Date**

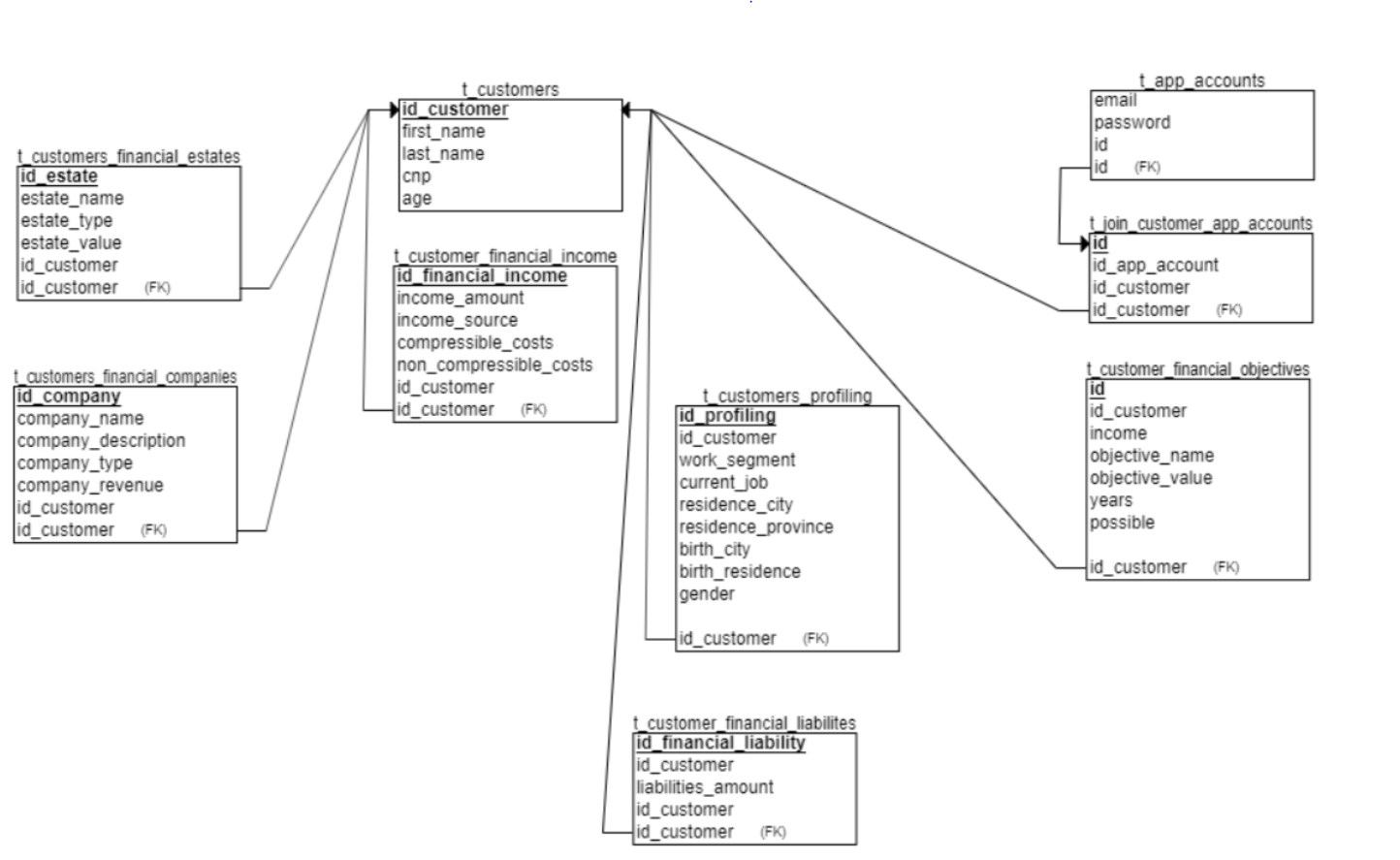


Figura 2 : Arhitectura bazei de date

În continuare descriem în detaliu rolul fiecărei tabele:

* **T\_customers**: conține principalele date despre un utilizator, date ce sunt introduse la înregistrare.
* **T\_app\_accounts**: conține informațiile de autentificare ale unui utilizator.
* **T\_join\_customer\_accounts**: este o tabela de legătură intre cele 2 tabele menționate anterior, am ales crearea acestei tabele în scopul respectării normei 4NF[[10]](#footnote-10).
* **T\_customers\_financial\_companies**: conține informațiile necesare din punct de vedere economic și legal pentru o companie deținută.
* **T\_customers\_financial\_estates**: conține informațiile pentru stocarea unei proprietăți de către un utilizator.
* **T\_customer\_financial\_income:** conține informațiile pentru stocarea unui venit de către utilizator. Este obligatoriu să existe o înregistrare pentru utilizatorul curent.
* **T\_customer\_financial\_liabilities:** conține informațiile pentru stocarea unei datorii achitată sau nu de către utilizator.
* **T\_customer\_financial\_profiling:** conține informațiile necesare realizării unui profil financiar utilizatorului curent, informații ce sunt necesare și din punct de vedere legal întrucât ele stau la baza generării codului fiscal alocat fiecărei persoane.
* **T\_customer\_objectives:**  în această tabela vor fi stocate obiectivele financiare setate de utilizator.

## **Arhitectura modelului**

Modelul statistic folosit este de tip “Regresie Liniara Multipla” [3]. Pentru construirea modelului am

folosit în spate librăria: “sklearn” [4] ce abstractizează în acest fel detaliile de implementare si e optimizat pentru crearea de modele de acest fel.

Modelul reprezintă o implementare de tip “Least Squares”. Este o implementare standard în analiză regresivă pentru aproximarea soluției unui sistem supradeterminat, prin minimalizarea sumei pătratelor reziduale, mai precis minimalizarea erorilor dintre valorile ce au fost returnate de model și valorile observate.

Structura inputului modelului este o lista de liste de conține datele obiectivului ce se dorește a îi fi estimată probabilitatea de realizare iar outputul este reprezentat de o lista cu un singur element ce constă în coeficientul de probabilitate a realizării obiectivului financiar respective.

### **Antrenarea modelului**

Modelul ce generează coeficientul de probabilitate a fost antrenat folosind metoda “Ordinary Least Squares” [5], fiind cea mai comună și ușor de implementat metodă din familia de implementări de tip “Least Squares” pe un set de input de 50 de elemente, datele de antrenament încercându-se să fie cât mai reale din punct de vedere al majorității utilizatorilor țintă a acestei aplicații.

Mai precis:

1. **Setul de antrenament** este de forma (X, Y) unde :
   1. X este de forma *X = ((x1, x2, x3…), (x4, x5, x6…), (x7, x8, x9... ))* unde :
      1. *(x1, x2, x3)* este o lista formata din numere reale strict pozitive ce reprezintă veniturile prezente în baza de date în momentul actual.
      2. *(x4, x5, x6)* este o lista formata din numerele reale strict pozitive ce reprezintă toate valorile obiectivelor prezente.
      3. *(x7, x8, x9)* sunt numerele reale strict pozitive ce reprezintă toate duratele dorite de realizare a obiectivelor prezente.
   2. Y este de forma *Y = (y1, y2, y3 … )* unde : y1, y2, y3 sunt numere ce reprezintă coeficienții de probabilitate ale obiectivelor deja estimate.
2. **Outputul modelului** este de forma:

*Output=(output1)* unde output1 reprezintă coeficientul de probabilitate, rotunjit superior[[11]](#footnote-11) ce va fi atribuit obiectivului respectiv și apoi salvat în baza de date.

Spre exemplu, pentru următorul input:

X = [800.0, 870.0, 1]

Folosind datele de antrenament:

X = [[800.0, 100.0, 1], [800.0, 900.0, 2], [800.0, 800.0, 1], [800.0, 90600.0, 1], [800.0, 90600.0, 1], [800.0, 91800.0, 2], [800.0, 9700.0, 1], [800.0, 9650.0, 1], [800.0, 9750.0, 1], [1300.0, 600.0, 1], [1300.0, 9000.0, 2], [800.0, 1000.0, 1], [800.0, 2500.0, 1], [800.0, 1500.0, 2], [800.0, 870.0, 1]]

Y = [1, 1, 1, -1, -1, -1, 0, 0, -1, 1, 1, 1, 1, 1, 1, 1]

Outputul produs va fi următorul:

Output = 0.77436724

## **3.4 Pașii Clientului**

În această secțiune voi prezența o diagramă ce exemplifica exact, la nivel vizual pașii pe care orice client trebuie și poate să ii realizeze în interiorul aplicației la un moment dat de timp. Am ales exemplificarea acestui lucru prin intermediul unui diagrame funcționale UML în detrimentul unei liste sau a unui paragraf text din motivul gradului scăzut de înțelegere, citire și memorare a pașilor necesari menționați anterior.

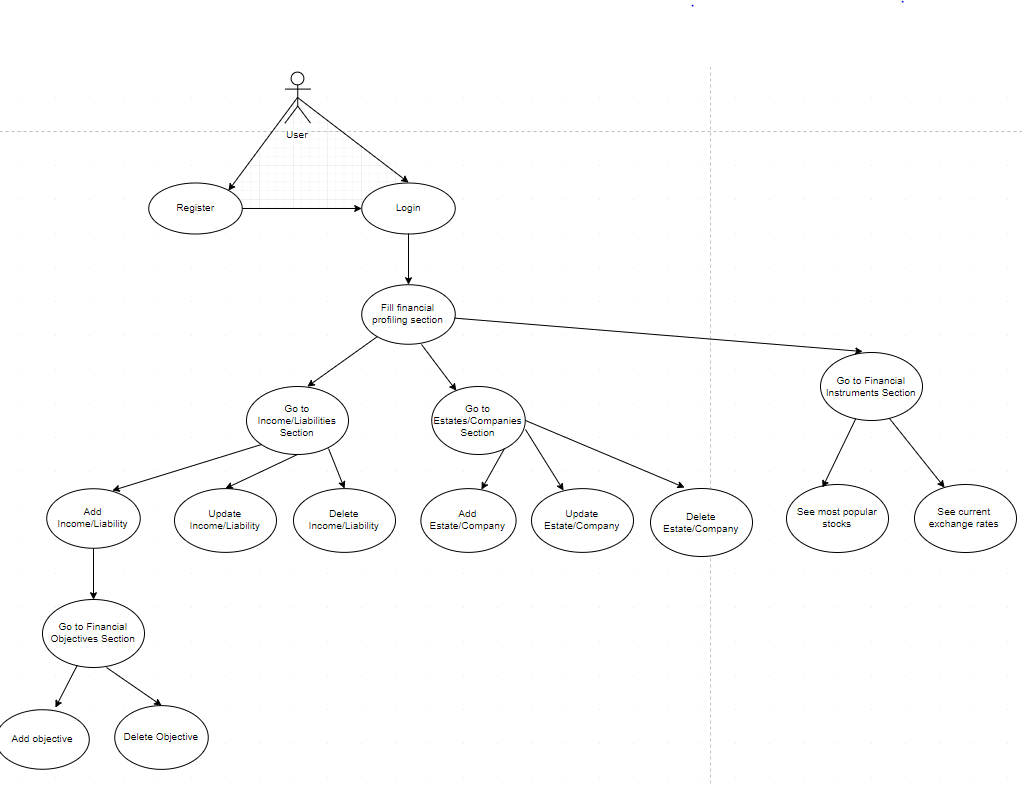


Figura 3: Diagrama funcțională a proiectului

Practic, odată ce utilizatorul va accesa pentru prima dată aplicația și deci nu va avea un cont creat acesta va fi rugat să își creeze unul folosind formularul de înregistrare:

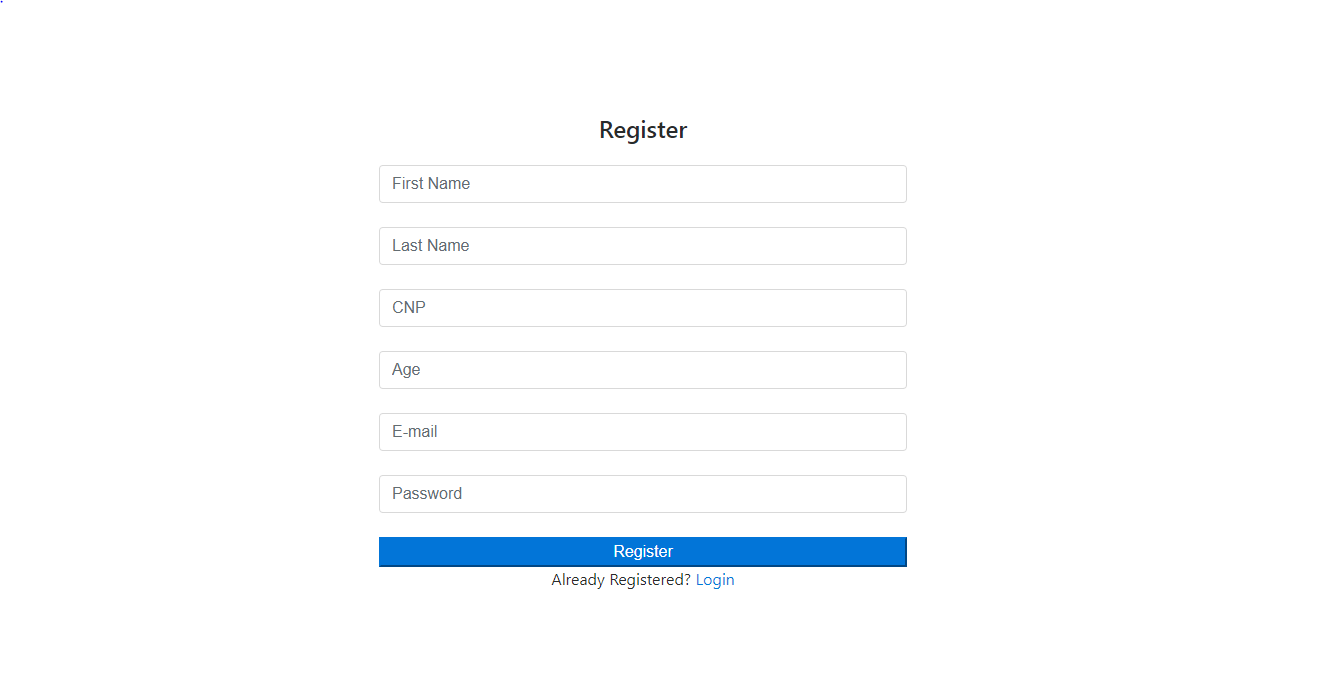


Figura 4 : Formularul de înregistrare a unui cont în aplicație

Dacă toate datele introduse sunt valide iar utilizatorul nu mai are deja un cont creeat acesta va fi redirecționat către pagină de autentificare:

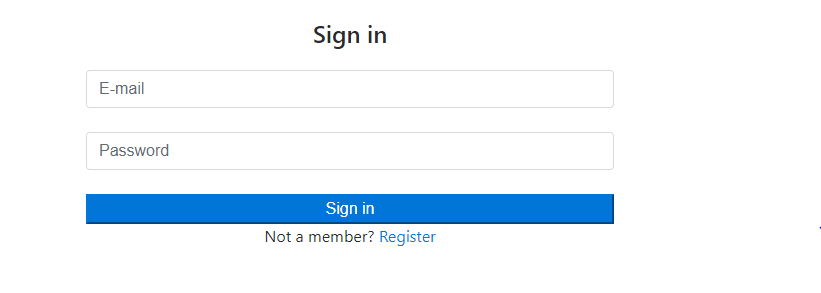


Figura 5 : Pagina de autentificare

Dacă datele introduse de către utilizator constituie un cont valid și existent atunci acesta va fi redirecționat către pagină principală:

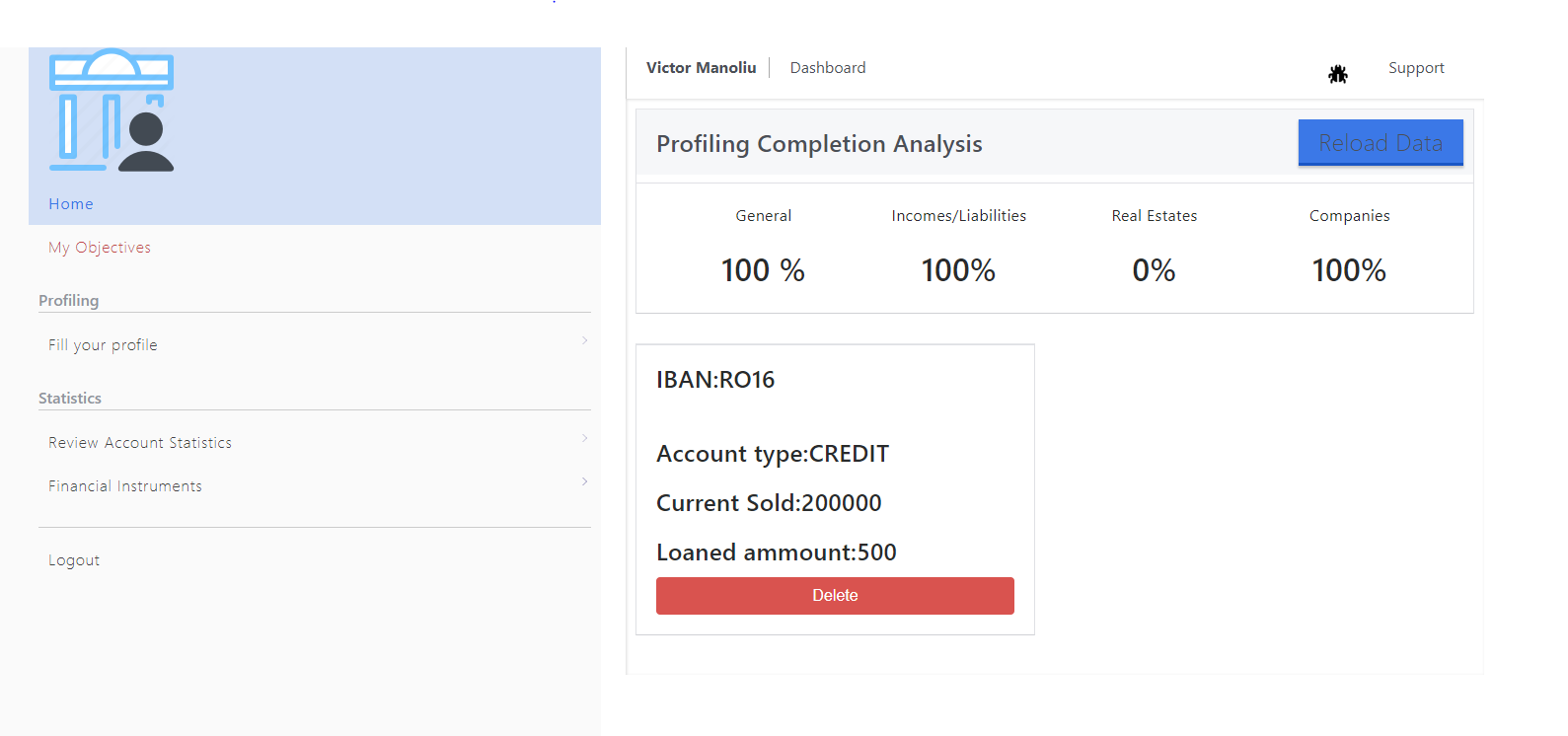


Figura 6 : Pagina principală a aplicației

În această pagină a aplicației, utilizatorul își poate vedea conturile bancare deschise în momentul actual cu posibilitatea de a șterge contul respectiv prin apăsarea butonului „Delete” , statusul completării fiecărei secțiuni în aplicație în partea de sus, iar în partea din stânga poate accesa fiecare secțiune a aplicației. Este foarte important de menționat faptul că, secțiunile : „General” și „Income/Liabilities” trebuie completate obligatoriu și mai important în ordinea aceasta. Motivul necesității completării secțiunilor respective în ordinea menționată este faptul că în secțiunea „General” utilizatorul va fi rugat să își introducă datele sale cu caracter personal ce atestă existența să într-un registru financiar iar seciunea de „Income/Liabilities” va stă la baza antrenarii serviciului statistic ce va returna coeficientul de probabilitate a realizării obiectivelor financiare ce vor fi setate după.

Așa cum am menționat anterior, în cadrul secțiunii de „General” utilizatorul va fi rugat să introducă date cu caracter personal precum: Meseria, Domeniul de Activitate, Orașul de reședința, etc pentru a asigura existența acestuia într-un registru financiar.

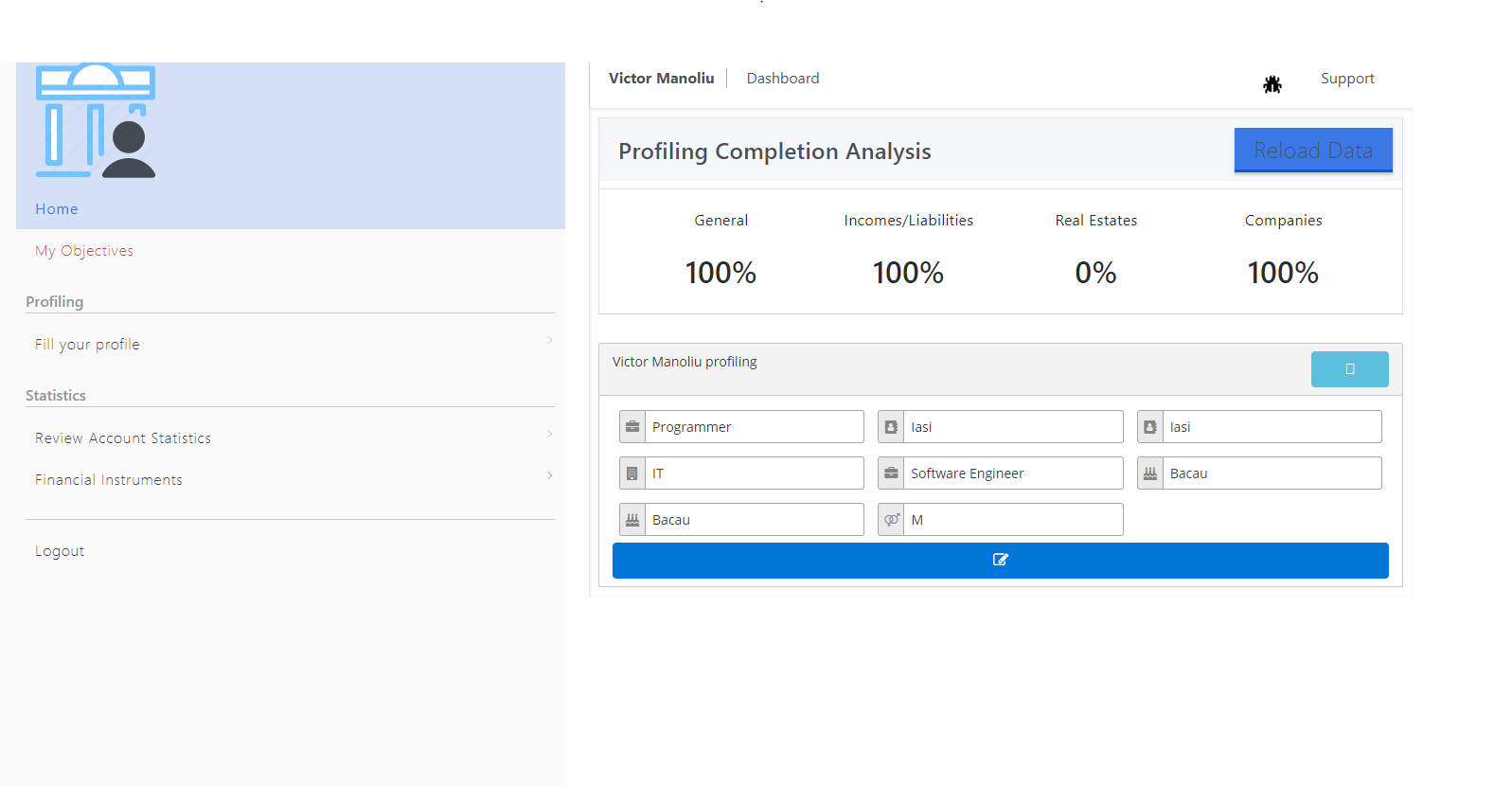


Figura 7 : Pagina de Profiling a utilizatorului

În continuare, vom prezenta secțiunea de „Income/Liabilities” în care utilizatorul își poate vizualiza veniturile și/sau datoriile introduse în aplicație, poate edita, poate șterge sau poate adaugă un venit sau o datorie.

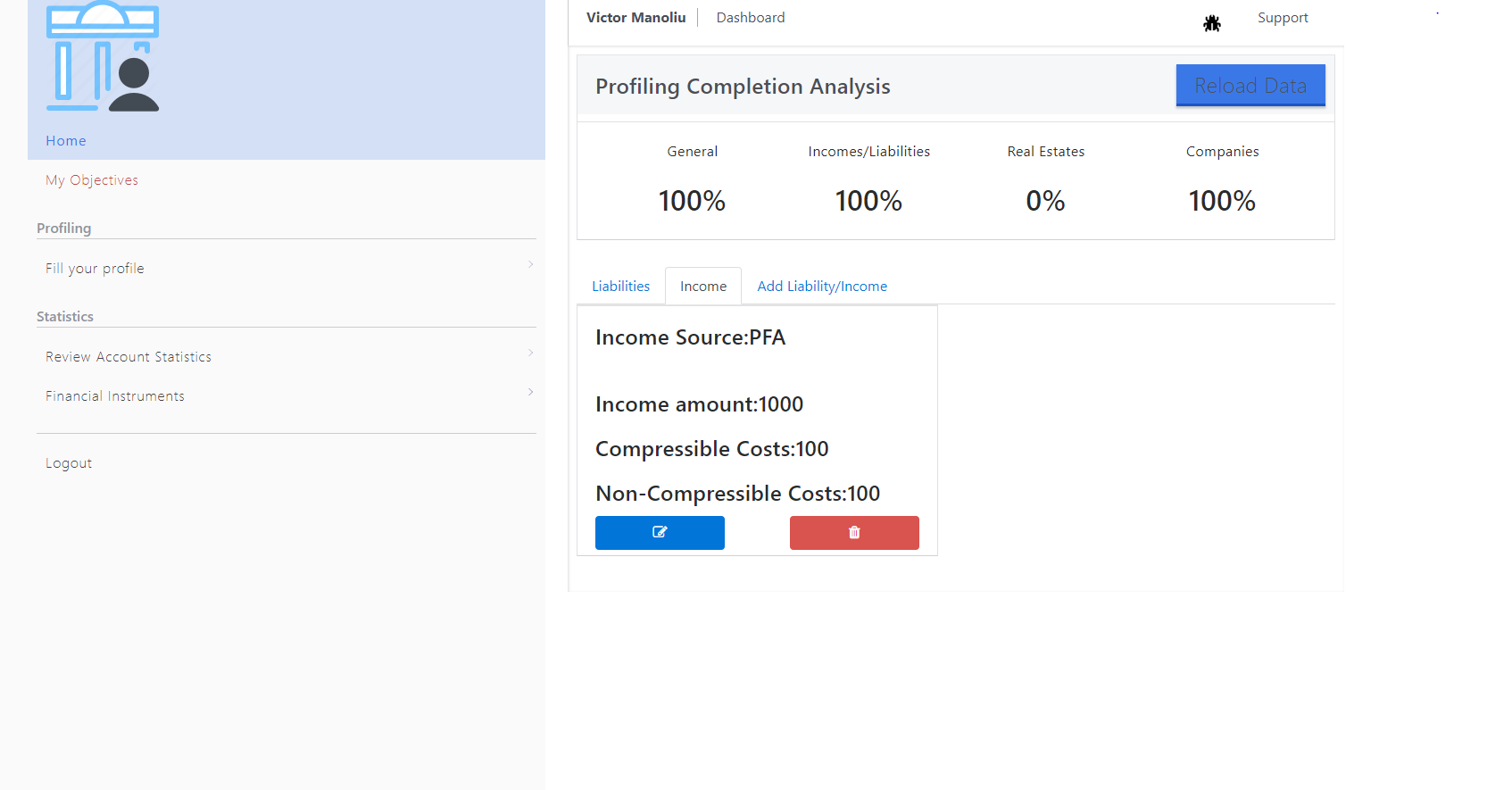


Figura 8 : Vizualizarea veniturilor/datoriilor

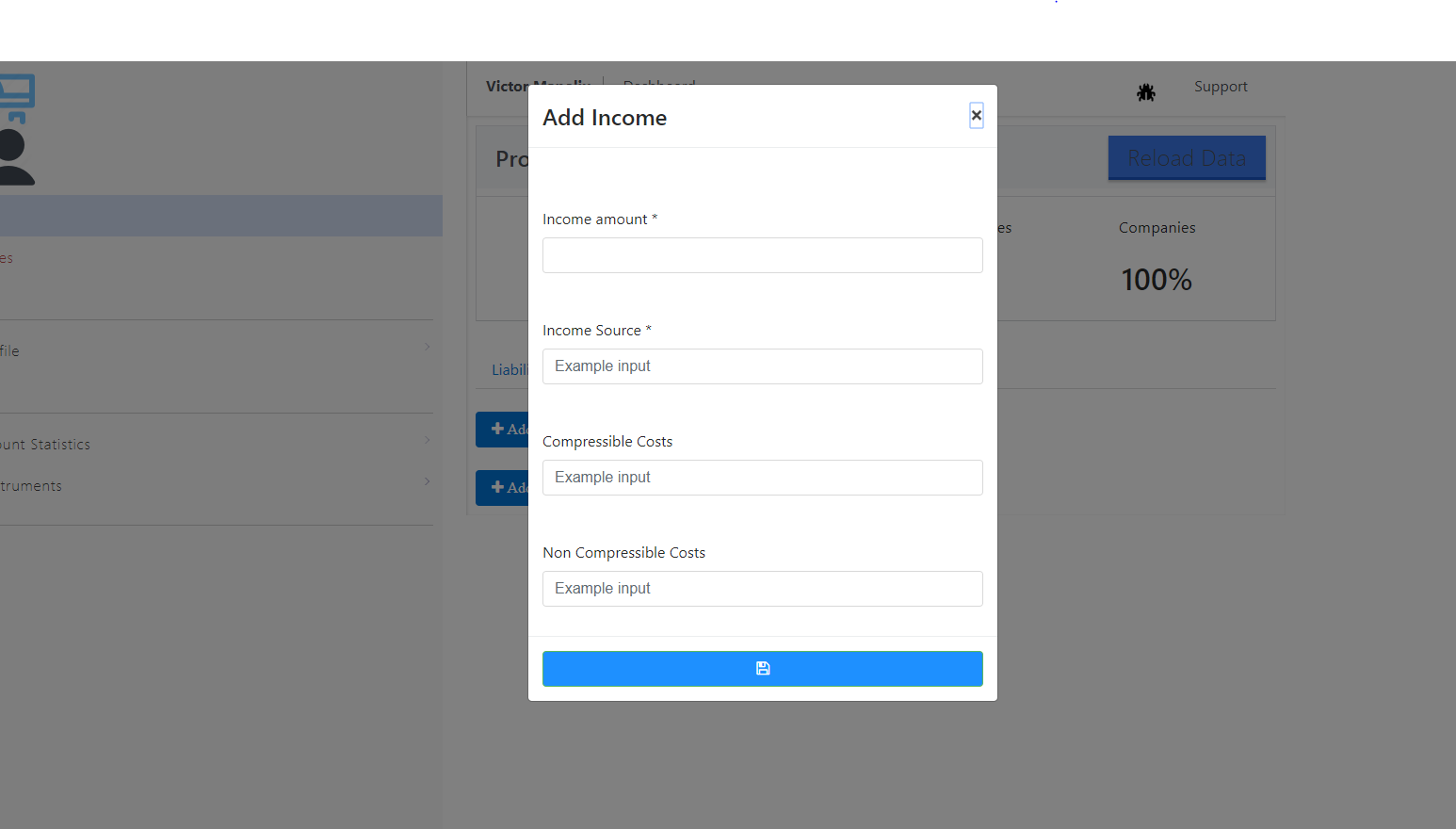


Figura 9 : Adăugarea unui venit/datorie

În cadrul acestui popup clientul își poate adauga un venit, acesta fiind rugat să specifice pe lângă sursa venitului și valoarea efectivă, o estimare personală asupra costurilor care se așteaptă să le aibă în fiecare luna precum cumpărături, etc și o estimare asupra costurilor neașteptate. Procesul este similar pentru vizualizarea și adăugarea de datorii.

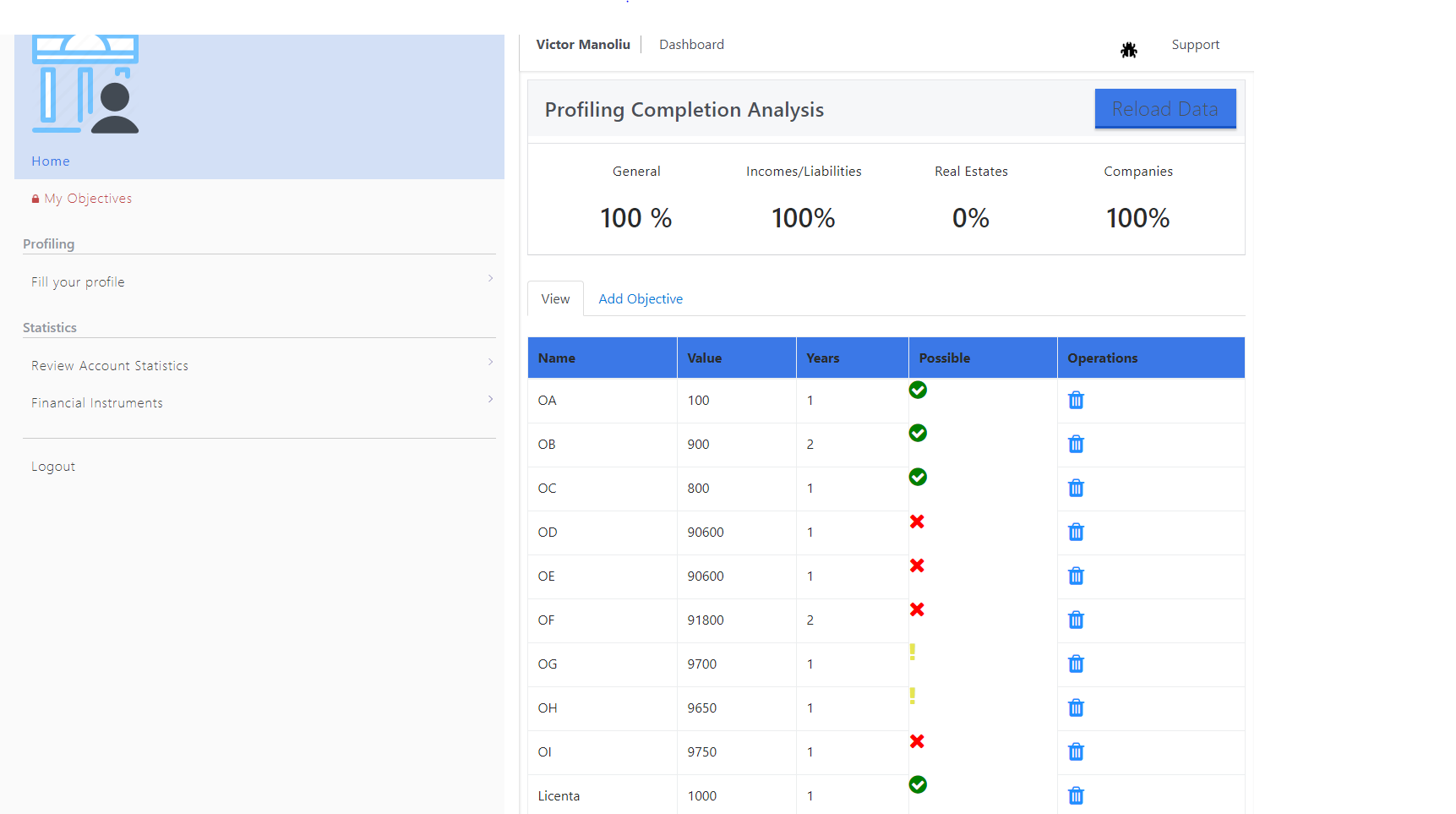


Figura 10 : Vizualizarea obiectivelor

În cadrul acestei pagini utilizatorul poate viziona obiectivele financiare deja setate într-un tabel în care coloanele sunt denumite într-un mod clar. În cadrul coloanei „Possible” este prezența o iconiță reprezentativă pentru cluster-ul în care a fost încadrat obiectivul respectiv, alături de o coloana denumită „Operations” în care utilizatorul poate șterge un obiectiv. Pentru mai multe detalii legate de clustere cititorul este rugat să consulte capitolul **„Arhitectura Modelului”**.

În cele din urmă, utilizatorul își poate adaugă un obiectiv, la adăugare acesta este rugat să specifice:

* Numele Obiectivului;

·         Valoarea Obiectivului;

·         Numărul de ani în care acesta dorește să atingă valoarea respectivă

.

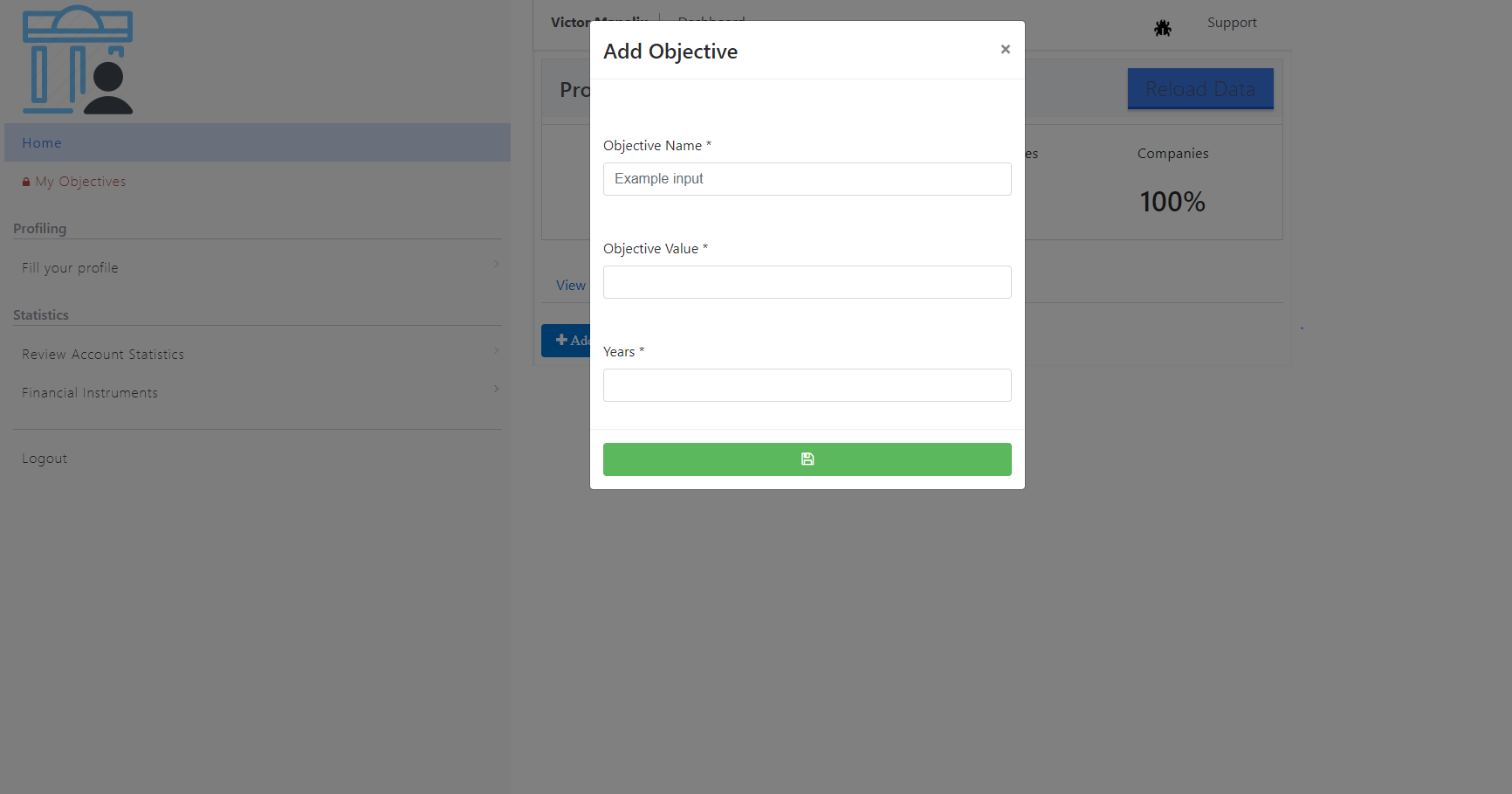


Figura 11 : Adăugarea unui obiectiv

# **Concluziile lucrării**

În momentul de față, soluția adusă aduce avantajele experienței prietenoase din punctul de vedere al utilizatorului final, ascunderea detaliilor financiare complicate pentru un utilizator ce nu are cunoștințe în domeniul economic și respectă pe cât posibil principiile de tip “best-practice” din ramură ingineriei programării și a bazelor de date dar încă sunt lucruri ce pot și ar trebui îmbunătățite. Lucrurile ce ar putea fi îmbunătățire la aceasta soluție vor fi menționate și detaliate succint în această secțiune.

## **Creșterea numărului datelor de antrenament**

Din păcate, în momentul de față modelul de regresie liniară multiplă antrenat, nu produce rezultate mulțumitoare, nefiind capabil să aibă o acuratețe de prezicere a rezultatelor mai mare de ~60%. O soluție a acestei probleme ar putea fi creșterea numărului datelor de antrenament, lucru ce ar putea rezultă într-o antrenare mai bună a modelului și deci o acuratețe mai crescută. Deployerea aplicației într-un mediu de producție, ar putea aduce și date de antrenament mult mai apropiate de lumea reală și deci antrenarea ar fi mai eficientă plus folosirea unui algoritm precum K-Means [22] pentru clusterizarea indexului de probabilitate ar rezultă în rezultate mulțumitoare din punct de vedere al procentului de acuratețe.

## **Construirea unui pipeline de CI/CD**

Un pipeline CI/CD (Continuos Integration/Continous Deployment) este o componentă ce ajută programatorii în automatizarea procesului de livrare de funcționalități. Această componentă este de foarte mare ajutor deoarece elimina riscul oricât de mare sau de mic a erorii umane.

Continous Integration este de asemenea și o tehnică în care fiecare programator își va pune schimbările pe propriul sau depozit de cod realizat prin clonarea depozitului de cod central iar apoi va realiza un “merge request” cu un depozitul de cod central, request ce pornește un build automat dar și teste automate. Acest build automat poate fi realizat prin tool-uri deja existente cu măr fi: Jenkins sau Travis, ambele suportând integrarea cu Github sau Gitab.

Continuous Delivery pe de altă parte este o componentă automatizată care o dată ce un request trece cu succes de componentă de build menționată anterior, acesta va fi deployata în mediul de producție folosind un script special, în cele mai multe cazuri fiind numit script de producție. Acest script de producție are rolul de a lansa pe un server, a codului deja existent, compilat și rulat și îl va face disponibil pentru utilizatori de oriunde.

## **Implementarea unui Load Balancer**

Este bine cunoscut faptul că majoritatea aplicațiilor de dimensiuni mari sau cele ce au un trafic mare de utilizatori dispun de această componentă de Load Balancer ce are rolul de a distribui conexiunile din partea clienților pe mai multe servere ce formează un cluster. Este sigur să afirmăm că această componentă optimizează timpul de așteptare pentru fiecare client, și conferă aplicației un risc mic de supraîncărcare a unei resurse, fie că vorbim de o resursă la nivel fizic, un server spre exemplu sau fie că vorbim de o resursă la nivel pur tehnologic.

În momentul actual, aplicația de față nu dispune de un trafic mare de utilizatori atât din pofida faptului că a fost realizat că și reprezentare practică al lucrării de față dar și din cauza lipsei unui build de producție și de deployment astfel încât utilizatorii să poată accesa aplicația. Dar, chiar și în situația de față trebuie să luăm în considerare și posibilitatea de dezvoltare continuă a aplicației și deci posibilitatea că aceasta sa ajungă la un număr mare de utilizatori autentificați și care sa realizeze acțiuni și requesturi în același timp și deci necesitatea unui load balancer ar fi esențială pentru a oferi cea mai buna experienta de folosire.

Există mai multe opțiuni de a implementa un load balancer, cele mai populare fiind: Round-Robin DNS [21] și DNS Delegation. Deși Round-Robin DNS este cea mai simplă că și implementare, nu necesitând echipamente hardware sau software adiționale este și cea mai pun eficientă, ea folosindu-se de algoritmul Round Robin. Pe de altă parte, metodă DNS Delegation conferă o eficientă mai crescută, deoarece da opțiunea de separare a requesturilor în zone DNS diferite ce le putem salva, copia sau distribui către alte severe DNS ce vor prelua requestul respectiv. Această abordare este des folosită în cadrul aplicațiilor ce dețin mai multe spații de nume sau dorim să adăugăm mai multe spații de nume sau domenii aplicației noastre în așa natură astfel încât utilizatorii să poată continuă să se bucure de o experiență plăcută și cât mai prietenoasă.

## **Alegerea algoritmului de generare a token-ului**

Așa cum am menționat în secțiunea anterioară de arhitectură a proiectului, am specificat faptul că serviciul web ce se ocupă de autentificarea utilizatorilor în aplicație va genera un token pentru fiecare utilizator iar acel token va fi setat apoi că și cookie și verificat înainte de executarea fiecărui request din partea serviciului ce deservește interfață web. De asemenea, am menționat că generarea se face folosind algoritmul SHA256 pe adresa de email a utilizatorului ce încearcă să se logheze, lucru ce ar putea fi îmbunătățit alegând un algoritm de criptare mai puțin cunoscut, mai puternic sau unul nou realizat. Motivul acestei afirmații este unul de securitate, riscul folosirii unui algoritm cunoscut este acela că un utilizator rău-intenționat poate menține o sesiune infinită că și timp creându-și singur token-ul respectiv de validare și apoi setându-l că și cookie pentru validare. Un exemplu de astfel de algoritm ar putea fi chiar: BCrypt [20], care deși este mai ineficient din punct de vedere al timpului de execuție, oferă un grad crescut de siguranță împotriva atacurilor de tip “brute-force”.

## **Posibilitatea realizărilor plăților online**

În contextual de față, cel al unei aplicații de tip advisor în domeniul economic, posibilitatea de realiza plăti de tip online sau de tip transfer între conturile utilizatorului sau conturile altor utilizatori ar fi bine venită. Motivul principal ar fi posibilitatea utilizatorilor de a avea o vedere de ansamblu a tuturor cheltuielilor realizate în luna curentă prin intermediul unui desfășurător detaliat și nu doar a unei sume totale ce este luată în considerare pentru antrenarea modelului de prezicere, în momentul actual. Desigur că această funcționalitate ar ridică multe probleme de securitate cum ar fi: securitatea cardului sau contului bancar introdus în aplicație și posibilitatea unor utilizatori rău intenționați de a realiza fraude sau de a intercepta plățile altor utilizatori. Aceste probleme de securitate pot fi rezultate prin implementarea în aplicație a design pattern-ului Saga [19].

În continuare vom oferi câteva detalii despre Saga pattern, modul cum funcționează și cum ar fi putut fi acesta utilizat în aplicația noastră.

Saga (de la numele Saga Pattern) constă în o serie de tranzacții executate secvențial, unde fiecare tranzacție se updateaza în cadrul unui singur serviciu. Practic, prima tranzacție este inițiată de un request extern ce corespunde unei operații de sistem iar apoi fiecare pas intermediar este pornită de completarea unui pas anterior.

Implementarea design pattern-ului poate fi realizată prin 2 modalități:

* Evenimente: Nu există o componentă centrală care să coordoneze tranzacțiile și deci fiecare serviciu va produce și va asculta la alt serviciu și va decide dacă o anumită acțiune la un anumit timp trebuie realizată sau nu.
* Comanda: Este prezentă o componentă/serviciu central ce are rolul de a decide logica de business, ordinea executării și dacă o anumită tranzacție la un anumit moment de timp se va executa sau nu.

Readucem în atenție că această lucrare conține elemente de noutate nu mai existând în momentul actual niciun produs cu acest specific pe piață, și chiar și fata de cel mai apropiat competitor și anume aplicația “Revolut” am adus elemente de noutate precum: posibilitatea ascunderii detaliilor financiare de către utilizator, posibilitatea estimării probabilității unor obiective financiare introduse direct de către utilizator.

Îmbunătățirea gradului de acuratețe a coeficientului de probabilitate în scopul obținerii unor preziceri mai bune cât și restul aspectelor ce pot fi îmbunătățite și au fost menționate anterior reprezintă direcții de viitor ale aplicației, scopul final fiind acela de a crea o aplicație cu un coeficient de corectitudine de 100% ce poate fi folosită în mod sigur de un număr foarte mare de utilizatori.

Anexa 1

În această anexa vom prezența câteva imagini cu rezultatele analizelor realizate folosind SonarQube:

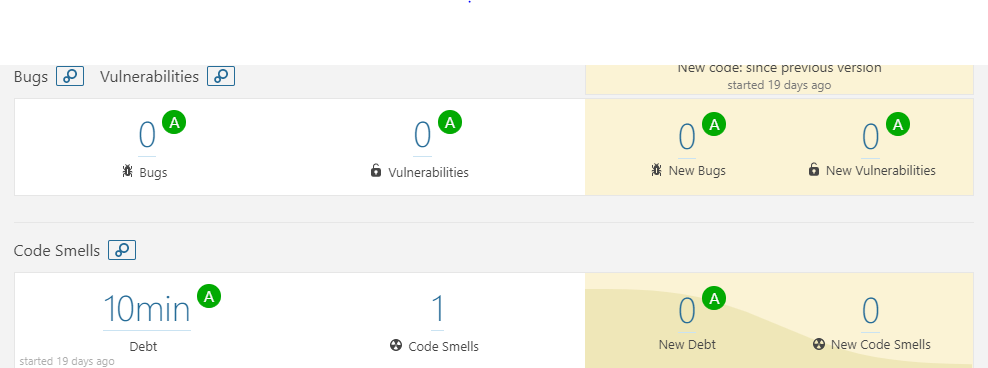


Figura 4: Rezultatul analizei serviciului de management a datelor client

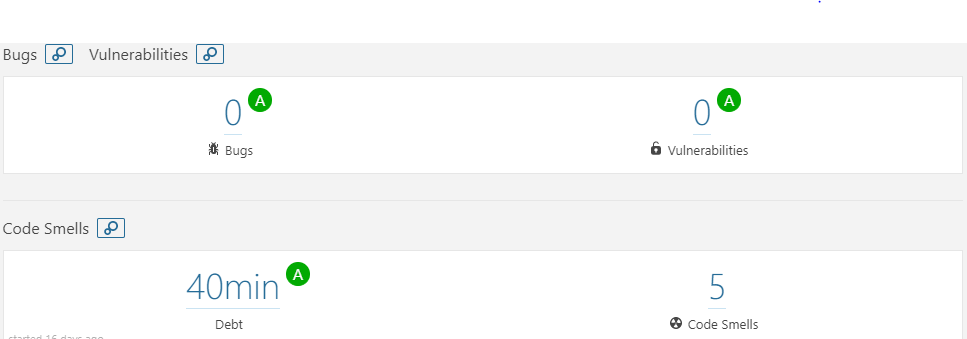


Figura 5: Rezultatul analizei serviciului de autentificare

# **Bibliografie**

|  |  |
| --- | --- |
| [1] | S.-C. Buraga, "Tehnologii Web," [Online]. Available: https://profs.info.uaic.ro/~busaco/teach/courses/web/presentations/web02ProgramareWeb-HTTP-CGI.pdf. |
| [2] | S.-C. Buraga, "Tehnologii Web," [Online]. Available: https://profs.info.uaic.ro/~busaco/teach/courses/web/presentations/web12InteractiuneWeb-Ajax-Mashups.pdf. |
| [3] | [Online]. Available: https://www.investopedia.com/terms/m/mlr.asp. |
| [4] | [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.LinearRegression.html. |
| [5] | M. S. Lewis-Beck, Applied Regression: An Introduction, Beverly Hills: CA: Sage, 1980. |
| [6] | Oracle. [Online]. Available: https://docs.oracle.com/en/java/. |
| [7] | [Online]. Available: https://spring.io/projects/spring-boot. |
| [8] | [Online]. Available: https://spring.io/projects/spring-data. |
| [9] | [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/. |
| [10] | [Online]. Available: https://en.wikipedia.org/wiki/C\_Sharp\_(programming\_language). |
| [11] | [Online]. Available: https://dotnet.microsoft.com/learn/web/aspnet-hello-world-tutorial/intro. |
| [12] | [Online]. Available: https://www.entityframeworktutorial.net. |
| [13] | [Online]. Available: https://www.python.org/doc/. |
| [14] | [Online]. Available: https://docs.scipy.org/doc/numpy/dev/. |
| [15] | [Online]. Available: http://flask.pocoo.org. |
| [16] | [Online]. Available: https://en.wikipedia.org/wiki/Flask\_(web\_framework). |
| [17] | [Online]. Available: https://angular.io/docs. |
| [18] | B. Beizer. |
| [19] | A. Said, "Transactions and Failover using Saga Pattern in Microservices Architecture," [Online]. Available: https://medium.com/@so3da/transactions-and-failover-using-saga-pattern-in-microservices-architecture-baf5a13111c9. |
| [20] | D. Arias, "Hashing in Action: Understanding bcrypt," [Online]. Available: https://auth0.com/blog/hashing-in-action-understanding-bcrypt/. |
| [21] | L. Alboaie, "Computer Networks," [Online]. Available: https://profs.info.uaic.ro/~computernetworks/files/7rc\_ProgramareaInReteaIII\_Ro.pdf. |
| [22] | L. Ciortuz, "Machine Learning," [Online]. Available: https://profs.info.uaic.ro/~ciortuz/SLIDES/cluster.pdf. |

1. Multitier Arhitecture : https://en.wikipedia.org/wiki/Multitier\_architecture [↑](#footnote-ref-1)
2. Design Pattern-ul Model-View-Controller [↑](#footnote-ref-2)
3. Tehnica Object-Relational Mapping [↑](#footnote-ref-3)
4. Java Persistance API [↑](#footnote-ref-4)
5. Java Database Connectivity [↑](#footnote-ref-5)
6. Un framework minimalistic ce se ocupa doar de maparea requesturilor HTTP și ii lipsesc funcționalități precum: validarea inputului sau abstractizarea bazei de date [↑](#footnote-ref-6)
7. Standardul Common Gateway Interface [↑](#footnote-ref-7)
8. S-a recurs la aceasta alegere datorita securitatii de care dispune facand parte din familia de algoritmi criptografici SHA (Secure Hash Algorithms) [↑](#footnote-ref-8)
9. Codul răspunsului HTTP de : OK [↑](#footnote-ref-9)
10. Forma de normalizare a bazei de date 4NF [↑](#footnote-ref-10)
11. Am recurs la rotunjire superioara pentru a putea clusteriza mai ușor output-ul in 3 clustere: very possible, possible, not possible [↑](#footnote-ref-11)