

UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IAȘI
FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

Aplicație de monitorizare și optimizare a investițiilor:

Investing Wizard

propusă de
Apostol Horia-Andrei

Sesiunea: iulie 2024

Coordonator științific
Olariu Florin

Cuprins

Abstract	8
Introducere	9
Motivație	9
Contribuții proprii	9
Structura lucrării	10
Capitolul 1: Studiu de piață	11
1.1 Track Your Dividends	11
1.2 Dividend Watch	13
1.3 Yahoo Finance	14
1.4 Bursa de Valori București	15
1.5 Concluzii	16
Capitolul 2: Noțiuni teoretice și legale	17
2.1 Impozitul pe venit	17
2.2 Impozitul pe dividende	18
2.4 Concluzii	19
Capitolul 3: Tehnologii utilizate	20
3.1 .NET	20
3.1.1 ASP.NET Core	20
3.1.2 ASP.NET Core Identity	21
3.2 Blazor	21
3.3 PostgreSQL	23
3.4 Entity Framework Core	23
3.5 AutoMapper	23
3.6 MediatR	24
3.7 Serilog	24
3.8 Quartz.NET	24
3.9 Twilio SendGrid	24
3.10 EODHD API	25
3.10.1 Exchanges API	25
3.10.2 Fundamental Data API	25
3.10.3 End Of Day Data API	25
3.10.4 Live Stock Prices API	26
3.11 OpenAI API	26
3.12 Alpha Intelligence API	26
3.12.1 Market News & Sentiments API	27
3.13 Syncfusion	27

3.14 Concluzii	27
Capitolul 4: Arhitectura aplicației și detalii de implementare	28
4.1 Arhitectura generală	28
4.1.1 Clean Architecture	28
4.1.2 Structura soluției	29
4.1.3 Clasa Result	30
4.2 Domeniul aplicației	30
4.3 Comenzi și interogări	32
4.4 Maparea entităților	33
4.5 Servicii și infrastructură	34
4.5.1 Persistența datelor	35
4.5.2 Implementarea serviciilor	35
4.5.3 Programarea joburilor	36
4.5.4 Integrarea de logging structurat	36
4.6 Autentificare și autorizare	38
4.7 API	40
4.8 Interfața Web	41
4.8.1 Structura proiectului Blazor	41
4.8.3 Utilizarea Syncfusion pentru componente avansate	42
4.9 Modulul Trading Assistant	43
4.10 Stock News Assistant	43
4.11 Concluzii	44
Capitolul 5: Funcționalități principale și scenarii de utilizare	45
5.1 Gestionarea utilizatorilor	45
5.1.1 Managementul profilului	46
5.2 Administrarea datelor	47
5.3 Vizualizarea structurată a entităților	47
5.4 Portofolii și liste de urmarire	51
5.4.1 Optimizarea tranzacțiilor	54
5.5 Accesul rapid la știri importante	57
5.6 Concluzii	57
Concluzii	58
Direcții viitoare	59
Bibliografie	61

Abstract

Folosirea unui șablon comun pentru a analiza companii din întreaga lume poate contribui la descoperirea de oportunități noi de investiții. Pentru investitorii români, o modalitate de face o comparație între companiile domestice și cele străine înseamnă un mare avantaj. În urma studiului pieței de aplicații financiare constatăm că nu există o aplicație care să prezinte unui utilizator informații în formatul dorit. Investing Wizard este o soluție care pe baza celor mai noi tehnologii, împreună cu elemente de inteligență artificială și algoritmi avansați, propune o organizare mai bună a datelor și o îmbunătățire a strategiei tax-loss harvesting ajustată după normele legale din România, oferindu-i investitorului șansa de a avea o performanță cât mai bună, dacă stabilește să investească în bursa românească. Rădăcinile acestei lucrări pornesc de la studii ce vizează patternuri de design, arhitecturi scalabile și normele legale curente. Rezultatul este o aplicație web, dezvoltată pe baza principiilor Clean Architecture împreună cu pattern-uri ca Mediator, Repository, Unit of Work și Factory Method, cu ajutorul framework-ului ASP.NET Core, care pe lângă componente avansate, folosește o simplă structurare a datelor într-un mod inteligent, cu scopul de a îmbina funcționalitățile excepționale ale aplicațiilor deja existente într-o singură platformă, de a și de a oferi metode noi inteligente de a aborda taxele din România.

Cuvinte cheie: AI, ASP.NET Core, Clean Architecture, Domain-Driven Design, tax-loss harvesting, wash-sale rule, Unit of Work, Factory Method, Repository Pattern

Introducere

Motivație

Unul dintre cele mai importante aspecte în dezvoltarea personală este puterea de a economisi și ulterior investi diverse sume de bani. Pentru un român universul de investiții este destul de larg, deoarece există multe piețe în lume care au la bază o economie mai dezvoltată decât cea românească. Totuși, o problemă apare atunci când diversificarea activelor se lovește de complexitate ridicată. De exemplu, pentru a realiza analize financiare, un investitor român trebuie să caute informații în mai multe surse, care în general sunt structurate diferit, făcând compararea valorilor mobiliare între ele cât se poate de dificilă. Pe de altă parte, lipsa resurselor educaționale și nevoia de a deschide mai multe conturi la diverși brokeri, fiecare cu norme legale diferite, plus experiența necesară gestionării profiturilor și a taxelor fac acest demers intimidant pentru un investitor la început de drum.

Aplicația Investing Wizard propune o soluție simplă care să preia date atât pentru piața financiară din România cât și pentru piețele financiare străine și să le structureze și afișeze într-un mod unic, pe aceeași platformă. De asemenea, o funcționalitate foarte utilă ar fi gestionarea tuturor tranzacțiilor într-o singură monedă. Astfel, utilizatorul poate vedea suma estimată pe care o va primi în dividende, în lei și totodată profitul sau pierderea totală netă, indiferent de moneda în care este realizată tranzacția, aplicația folosind schimbul valutar din ziua respectivă pentru conversie. Alte caracteristici ale unei aplicații ce aduce valoare în acest domeniu este folosirea inteligenței artificiale sau a algoritmilor pentru a automatiza diverse activități, pentru ca investitorul să își poată pune în valoare mai mult creativitatea.

Contribuții proprii

O primă contribuție proprie este proiectarea arhitecturii aplicației. În acest stadiu am creat pe baza mai multor principii un model care să reflecte cât mai bine entitățile, regulile și

proprietățile pe care le-am considerat relevante pentru un investitor. La nivel de logică de afaceri am proiectat codul să fie cât mai curat și independent de orice serviciu extern. Apoi, conform datelor pe care le-am putut prelua din sursele externe, am ales un stack de tehnologii care să poată gestiona eficient acest volum de date și am creat un mapper personalizat pentru a aduce în forma dorită valorile preluate consumând API-urile externe. Tot aici am utilizat diferite tehnologii moderne pentru a folosi eficient memoria cache și pentru a programa background joburi. Am adus și o oarecare îmbunătățire modulului de Identity, implementând interfețele predefinite cu ajutorul unor servicii populare. În afară de acestea, am configurat asistentul pentru știri, bazat pe inteligență artificială, care folosește două servicii externe.

Cea mai importantă contribuție proprie, este algoritmul de optimizare a tranzacțiilor. Acest algoritm este proiectat de mine folosind diverse tehnici, gândite normal pentru piața americană, pe care le-am îmbunătățit în favoarea investitorului, deoarece legislația din România permite acest lucru.

Structura lucrării

Am ales să prezint aplicația Investing Wizard în cinci părți. Pentru început am motivat valoarea pe care consider că o aduce un astfel de proiect, apoi am făcut un studiu de piață, cu exemple concrete, afișând și conținut grafic, pentru a prezenta în ce măsură există deja aplicații asemănătoare. După acest studiu am prezentat diverse noțiuni pentru a putea fi mai explicit în capitolele următoare, împreună cu detalii privind legile, pentru a face mai ușor de înțeles algoritmul de optimizare. În următorul capitol am descris tehnologiile utilizate, motivul folosirii acestora și ce aduce special fiecare. Apoi, am continuat cu arhitectura aplicației și detaliile implementării. M-am concentrat să explic ideile pe care le-am folosit, cum am combinat diverse tehnologii și am oferit imagini cu structurarea componentelor și fragmente de cod relevante, care formează într-un final un flow întreg de la serviciul de date, până la domeniul aplicației. Pe urmă, am prezentat funcționalitățile principale ale aplicației, cu exemple grafice și explicații. În încheiere, am concluzionat reușitele aplicației și mi-am expus părerea în legătură cu posibilele dezvoltări pe viitor.

Capitolul 1: Studiu de piață

În acest capitol voi da exemple de câteva aplicații care sunt folosite de investitori în scopul analizelor financiare, punând accent pe funcționalitățile care se pot atât îmbunătăți, cât și îmbina, creând astfel o singură aplicație care le conține pe toate.

1.1 Track Your Dividends

Track Your Dividends¹ este o aplicație ce permite utilizatorilor monitorizarea portofoliilor de investiții și a dividendelor. Una dintre funcționalitățile principale este posibilitatea creării unui portofoliu și popularea acestuia cu proprietățile pe care le deține utilizatorul, aici fiind vizate valorile mobiliare. În **Figura 1.1.1** putem observa cum arată o pagină de acest gen, luând ca exemplu acțiuni la câteva companii americane.

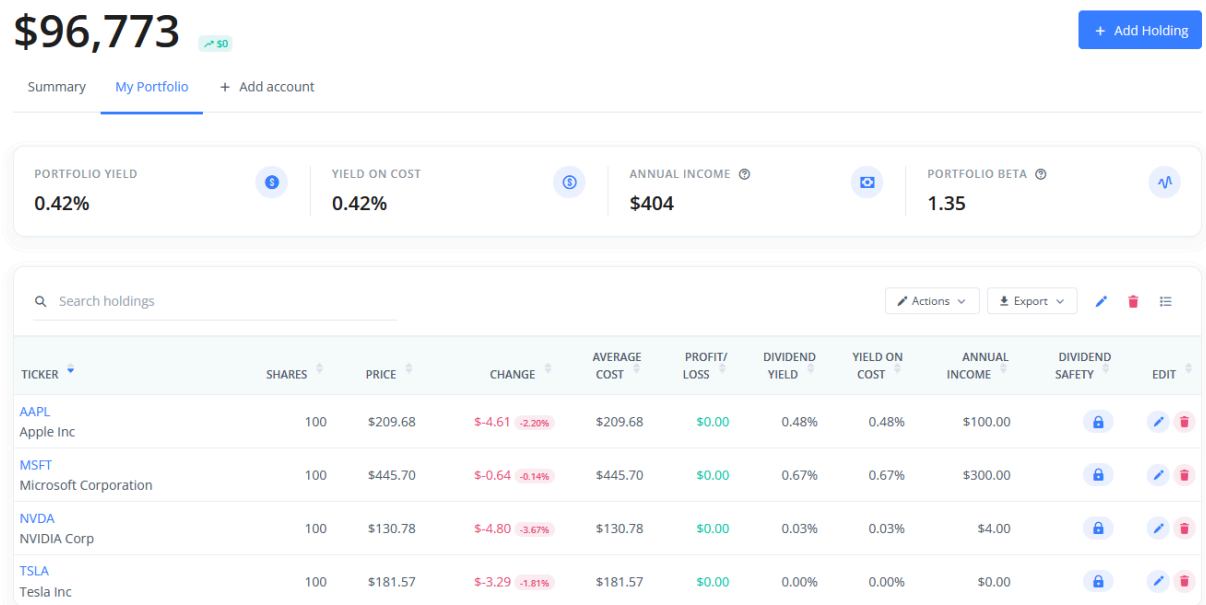
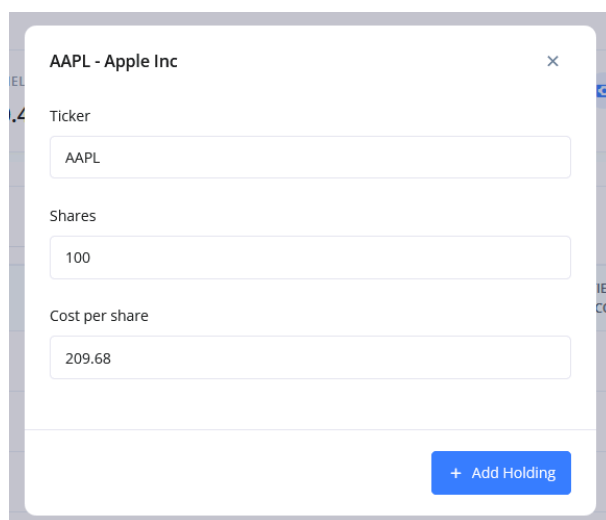


Figura 1.1.1: Pagină de portofoliu din aplicația Track Your Dividends.

¹ <https://trackyourdividends.com/>

Înregistrările sunt grupate după numele companiilor la care fac referire și oferă numărul de unități totale, costul total de cumpărare, prețul curent și venitul anual din dividende, împreună cu procente aferente. Pentru a adăuga o înregistrare de acest gen utilizatorul trebuie să aleagă din lista de tickere ale companiilor valoarea pe care dorește, să adauge prețul curent și numărul de unități. Un exemplu grafic este descris în **Figura 1.1.2**, unde se pregătește adăugarea a 100 de acțiuni la compania Apple Inc., prețul unei unități fiind de \$209.68.



The image shows a modal window titled "AAPL - Apple Inc" with a close button (X) in the top right corner. Inside the modal, there are three input fields: "Ticker" containing "AAPL", "Shares" containing "100", and "Cost per share" containing "209.68". At the bottom right of the modal is a blue button with a plus sign and the text "+ Add Holding".

Figura 1.1.2: Adăugarea unei înregistrări în aplicația Track Your Dividends.

Abordarea aceasta sugerează că aplicația este concepută special pentru a gestiona dividendele, întrucât nu ține cont de prețul de cumpărare al fiecărei tranzacții în parte, ci doar oferă un preț mediu pe care utilizatorul îl poate modifica.

1.2 Dividend Watch

Dividend Watch² este o aplicație asemănătoare cu Track Your Dividends dar care face diferența dintre proprietăți și tranzacții oferind utilizatorilor două tab-uri separate pentru un portofoliu, unde se face distingerea dintre cele două.


Group by Customize Actions									
Ticker	Name	Price	Shares	Value	Dividend yield	Yield on cost	Annual income	Day change	Total return
 MSFT	Microsoft Corporation	\$445.70	100	\$44,570.00	0.66%	0.66%	\$293.00	▼ 0.14%	\$0
 AAPL	Apple Inc	\$209.68	100	\$20,968.00	0.46%	0.46%	\$97.00	▼ 2.15%	\$0
 TSLA	Tesla, Inc.	\$181.57	100	\$18,157.00	—	—	—	▼ 1.78%	\$0
 NVDA	NVIDIA Corporation	\$130.78	100	\$13,078.00	0.02%	0.02%	\$2.60	▼ 3.54%	\$0

Figura 1.2.1: Proprietățile utilizatorului în aplicația Dividend Watch.

În **Figura 1.2.1** este afișat tab-ul în care apar proprietățile utilizatorului, pentru același exemplu de companii pe care l-am folosit și cu aplicația Track Your Dividends. Tranzacțiile corespunzătoare sunt prezentate în **Figura 1.2.2**. Modul de adăugare a acestora este la fel ca la aplicația precedentă, cu precizarea că aici vom adăuga și data specifică fiecărei tranzacții în parte.

Selected transactions: 0								Delete ▾	Export .CSV ▾	Filter transactions ▾
Type	Date	Ticker	Name	Amount	Price per share	Total	Actions			
↑ BUY	06/21/2024	NVDA-NASDAQ	NVIDIA Corporation	100	\$130.78	\$13,078.00	✍ □			
↑ BUY	06/21/2024	TSLA-NASDAQ	Tesla, Inc.	100	\$181.57	\$18,157.00	✍ □			
↑ BUY	06/21/2024	AAPL-NASDAQ	Apple Inc	100	\$209.68	\$20,968.00	✍ □			
↑ BUY	06/21/2024	MSFT-NASDAQ	Microsoft Corporation	100	\$445.70	\$44,570.00	✍ □			

Figura 1.2.2: Tranzacțiile utilizatorului în aplicația Dividend Watch.

Dividend Watch de asemenea oferă utilizatorilor posibilitatea de a vedea profilul fiecărei companii, alături de câteva date despre acestea.

² <https://dividend.watch>

1.3 Yahoo Finance

Yahoo Finance³ este o aplicație complexă ce oferă o gamă mai largă de funcționalități. În primul rând se remarcă prin punerea la dispoziție de volume mari de date financiare, având o pagină ca cea din **Figura 1.3.1** dedicată fiecărei companii.

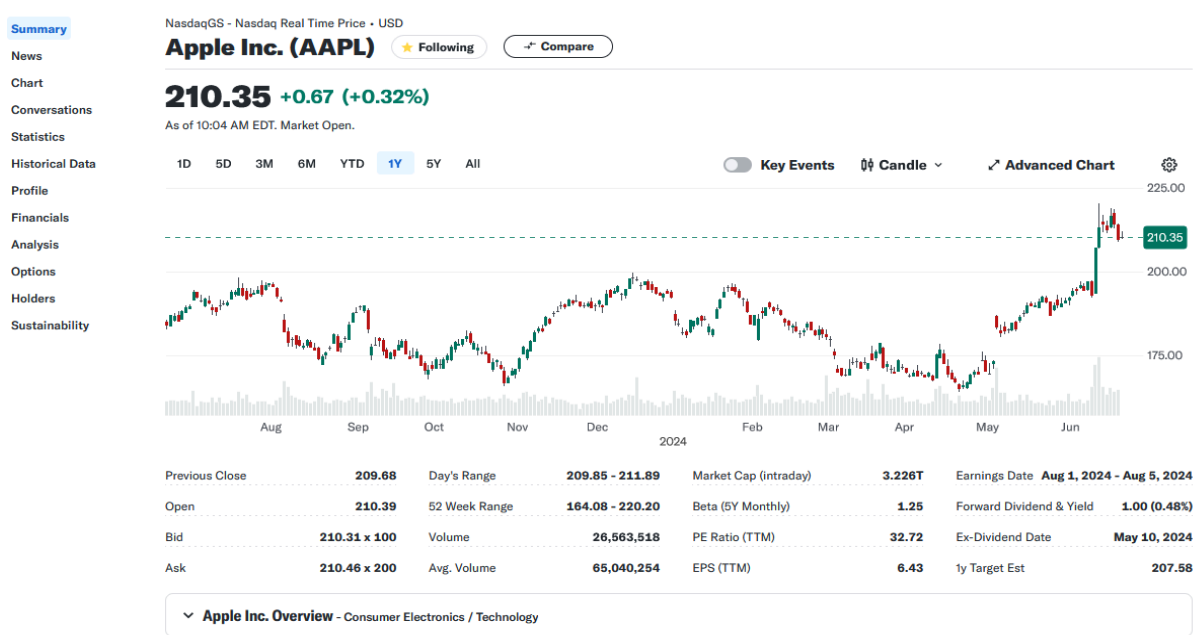


Figura 1.3.1: Pagina unei companii în aplicația Yahoo Finance.

Pe lângă acest sumar, pagina conține și statistici, date financiare istorice, analiză financiară, informații despre acționari și ultimele știri importante referitoare la compania respectivă. Funcționalitățile de gestionare a portofoliilor și a listelor de urmărire sunt oferite utilizatorilor în același mod ca în prima aplicație descrisă, Track Your Dividends, ce nu oferă monitorizarea tranzacțiilor.

Aplicația Yahoo Finance este singura dintre cele prezentate până acum care oferă o parte dintre aceste informații și pentru companii listate la Bursa de Valori București, piața financiară din România.

³ <https://finance.yahoo.com/>

1.4 Bursa de Valori București

Site-ul Bursei de Valori București⁴ oferă informații financiare și știri despre companiile listate pe piața românească într-o manieră asemănătoare cu cea a aplicației Yahoo Finance, așa cum putem vedea în **Figura 1.4.1**.

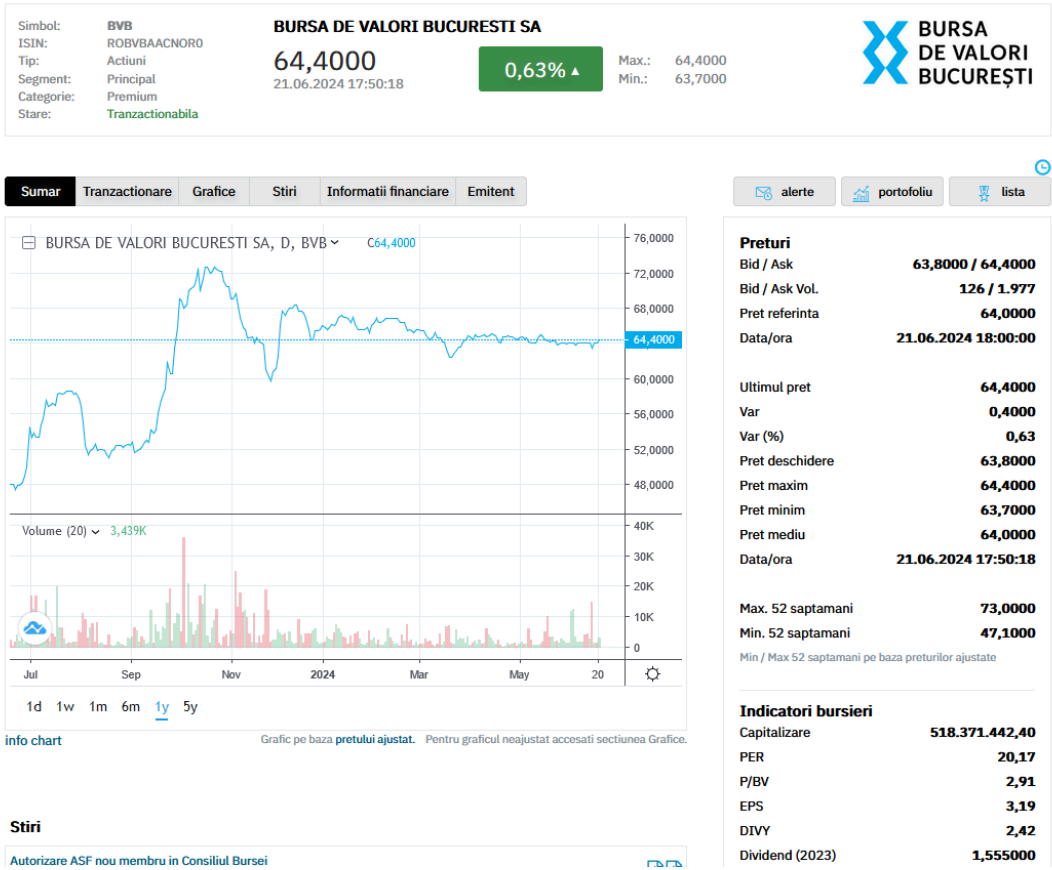


Figura 1.4.1: Pagina unei companii pe site-ul Bursei de Valori București.

⁴ <https://bvb.ro/>

1.5 Concluzii

În urma studiului de piață am remarcat mai multe funcționalități comune în diverse aplicații pe care un investitor le poate folosi pentru a-și ușura munca. În primul rând o modalitate de a prezenta companiile într-un mod structurat, evidențiind informațiile mai importante, ca de exemplu valoarea dividendului, prețul curent, graficul cu prețuri istorice și date necesare analizei cantitative. Referitor la analiza calitativă, o aplicație de acest gen ar trebui să dispună de un procedeu prin care să obțină știrile recente despre o companie, sau atitudinea generală a investitorilor față de aceasta. De asemenea, portofoliile ar trebui să fie gândite în așa fel încât să țină cont și de tranzacțiile pe care le face utilizatorul, nu doar de proprietățile acestuia, pentru a putea calcula ulterior profitul sau pierderea netă pe care o realizează.

Capitolul 2: Noțiuni teoretice și legale

Pentru a putea păstra consistența în aplicație am folosit câteva cuvinte cheie. În primul rând, când mă refer la „security” în denumirile proprietăților claselor mă refer la „valoare mobilă”, ceea ce în contextul aplicației reprezintă acțiunile companiilor la care facem referire și perechile forex pentru schimb valutar.

Termenul de „proprietate” este folosit în contextul codului reprezentând membri ai claselor dar în contextul funcționalităților ca posesiunile utilizatorului, termen descris și ca „holding”.

Când vorbim despre o „piață de capital”, acest termen împreună cu „market” și „exchange” sunt interschimbabili.

Un alt aspect important este diferența dintre „code” și „ticker”. Pentru o companie, ca de exemplu Apple Inc., ticker-ul este valoarea după care aceasta este listată pe bursă, în cazul acesta „AAPL”, iar code-ul este compus din ticker și codul pieței de capital aferent, separate prin punct. Pentru exemplul nostru code-ul va fi „AAPL.US”. Acesta este folosit ca un identificator natural al unei companii sau a unei piețe deoarece este unic prin lege.

Pe lângă aceste detalii, pentru a înțelege rolul unor îmbunătățiri trebuie să facem o clasificare bazată pe ultimele modificări legale.

2.1 Impozitul pe venit

Pentru a putea calcula impozitul pe venit vom face referire la **Legea nr. 142/2022**⁵ și vom împărți brokerii la care utilizatorul în persoană fizică are deschis contul în două categorii:

- a.) broker rezident fiscal român sau nerezident care are în România un sediu permanent.
- b.) broker care nu este rezident fiscal român și nu are un sediu permanent în România.

⁵ <https://legislatie.just.ro/Public/DetaliiDocument/255605>

Pentru prima categorie obligația calculării, declarării și plății impozitului îi revine brokerului, se reține automat la sursă și se calculează astfel:

1. prin aplicarea unei cote de 1% asupra fiecărui câștig din transferul titlurilor de valoare care au fost dobândite și înstrăinate într-o perioadă mai mare de 365 de zile, inclusiv, de la data dobândirii;
2. prin aplicarea unei cote de 3% asupra fiecărui câștig din transferul titlurilor de valoare care au fost dobândite și înstrăinate într-o perioadă mai mică de 365 de zile de la data dobândirii;

Un alt aspect important este faptul că pierderile sunt independente de câștiguri în cadrul acestei categorii.

Pentru a doua categorie impozitul este fix, în valoare de 10% din câștigul net, nu se reține la sursă și obligația calculării, declarării și plății impozitului îi revine utilizatorului care trebuie să completeze declarația unică anuală privind impozitul pe venit și contribuțiile sociale datorate de persoanele fizice. Referitor la pierderi, avantajul aici este că acestea se pot deduce din câștigurile obținute în următorii ani, conform **Legii nr. 227/2015**⁶.

2.2 Impozitul pe dividende

Tot pe baza **Legii nr. 227/2015** împărțim companiile în două categorii:

- a.) Companiile care au sediul social înregistrat în România.
- b.) Companiile care au sediul social înregistrat în afara României.

⁶ https://static.anaf.ro/static/10/Anaf/legislatie/Cod_fiscal_norme_2023.htm

La fel ca la impozitul pe venit, pentru prima categorie, obligațiile îi revin brokerului și impozitul se reține la sursă, iar pentru a doua, obligațiile îi revin utilizatorului care trebuie să completeze declarația unică la care am făcut referire. Impozitul pe dividende se calculează luând în considerare mai mulți factori, dar care sunt irelevanți în cadrul aplicației, deoarece rolul acesteia este doar clasificarea în funcție de sediul social, pentru a reduce timpul de cercetare al utilizatorului.

2.4 Concluzii

În concluzie, stabilirea unor aspecte de terminologie, împreună cu înțelegerea unor norme legale, ajută la identificarea posibilelor îmbunătățiri și ghidează crearea unor constrângeri clare. Totodată, aceste lucruri ajută și la gândirea logicii aplicației într-un mod în care să descrie cât mai bine evenimente reale.

Capitolul 3: Tehnologii utilizate

3.1 .NET

.NET⁷ este o platformă open-source concepută pentru dezvoltarea mai multor tipuri de aplicații software, creată de Microsoft, care suportă limbajele C#, F# și Visual Basic, primul fiind cel mai popular dintre acestea.

Lansată inițial ca .NET Framework, destinată special pentru Windows și Windows Server, astăzi avem o variantă modernă cross-platform numită .NET Core care este regândită pentru era cloud-ului și suportată la activ pentru Linux, Mac și Windows, foarte robustă în ceea ce privește dezvoltarea aplicațiilor web, desktop, mobile și altele.

Mentenanța este în principal făcută de Microsoft, cu contribuții din partea comunității, actualizând anual platforma. În prezent, cea mai recentă versiune stabilă este .NET 8, folosită în aplicația Investing Wizard, urmând ca spre sfârșitul acestui an să fie lansată a noua versiune, împreună cu C# 13. Un aspect care simplifică dezvoltarea în .NET este utilizarea mecanismului de partajare a codului, numit NuGet⁸, cu ajutorul căruia dezvoltatorii pot crea diverse pachete care conțin cod compilat ce poate fi reutilizat.

3.1.1 ASP.NET Core

În vederea dezvoltării aplicațiilor web în cadrul ecosistemului .NET se utilizează în general framework-ul ASP.NET Core⁹, un framework care tinde a fi modular, este cross-platform și permite dezvoltarea rapidă a web API-urilor și a interfețelor de utilizator. De asemenea, este proiectat pentru a facilita testarea și pentru a integra ușor elemente de securitate.

⁷ <https://learn.microsoft.com/en-us/dotnet/core/introduction>

⁸ <https://learn.microsoft.com/en-us/nuget/what-is-nuget>

⁹ <https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-8.0>

3.1.2 ASP.NET Core Identity

De multe ori aplicațiile moderne folosesc un serviciu terț în vederea autentificării și autorizării pentru a spori securitatea datelor. În ASP.NET Core există un API numit ASP.NET Core Identity¹⁰ care pune la dispoziție un mecanism pentru autentificarea și persistența utilizatorilor într-un mod practic și sigur, împreună cu o autorizare bazată pe roluri. Odată cu migrarea la .NET 8 a fost dezvoltată o serie de API-uri minimaliste pentru a gestiona mai bine aceste funcționalități. Un scenariu practic de utilizare a acestora sunt într-adevăr aplicațiile de tip Single Page Application¹¹. Totuși, cum expunerea în acest mod este nouă și încă în dezvoltare, iar arhitectura aplicației Investing Wizard este în principiu stratificată, cu o separare clară a responsabilităților, am ales o abordare care vizează testarea funcționalităților folosind aceste endpoint-uri dar care adoptă integrarea izolată în paginile razor specifice modulului Identity, pentru a nu avea concomitent atât comunicare prin API-uri minimaliste cât și prin API-uri clasice.

3.2 Blazor

Blazor¹² este un framework web de frontend, dezvoltat de Microsoft, folosit în cadrul aplicațiilor .NET care permite crearea de interfețe de utilizator interactive folosind C# și afișarea acestora ca HTML și CSS, dezvoltatorul având astfel control atât pe partea de server cât și partea de client într-un singur model.

Este conceput pentru a realiza aplicații de tipul Single Page Application dar totuși diferit de alte tehnologii prin simplul fapt că folosește C#, ceea ce simplifică semnificativ legăturile dintre date și integrarea cu backend-ul.

¹⁰ <https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-8.0&tabs=visual-studio>

¹¹ <https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity-api-authorization?view=aspnetcore-8.0>

¹² <https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-8.0>

De obicei, o aplicație dezvoltată astfel poate fi găzduită ori pe modelul Blazor WebAssembly¹³, ceea ce presupune că se va permite rularea codului C# direct în browser, oferind o experiență completă pe partea de client, fără a fi nevoie de a menține o conexiune la internet, ori pe modelul Blazor Server, care folosește o conexiune SignalR cu WebSocket-uri pentru a asigura o experiență interactivă fără a fi nevoie să reîncărcăm pagina.

Ultima versiune de ASP.NET a adus îmbunătățiri și schimbări în acest context. Acum se poate alege dezvoltarea unei variante hibrid, ce combină părțile pozitive ale celor două modele. Acest lucru conferă o gamă mai largă de opțiuni în cazul în care se dorește realizarea unei aplicații complexe ce trebuie să funcționeze offline, sau a unei aplicații moderne de tipul Progressive Web App¹⁴.

De asemenea, s-a introdus și o funcționalitate prin care putem alege modul de încărcare și afișare. Aplicația Investing Wizard are la bază o tehnică de generare interactivă a componentelor, cu preîncărcare pe server, cunoscută mai nou în ASP.NET 8 ca Interactive Server rendering. O aplicație Blazor este ușor întreținută, datorită noilor tehnologii și a modelului său bazat pe componente.

¹³ <https://learn.microsoft.com/en-us/aspnet/core/blazor/hosting-models?view=aspnetcore-8.0>

¹⁴ <https://learn.microsoft.com/en-us/aspnet/core/release-notes/aspnetcore-8.0?view=aspnetcore-8.0>

3.3 PostgreSQL

PostgreSQL¹⁵ este un sistem open-source de baze de date relaționale care folosește limbajul SQL în combinație cu diverse alte funcționalități, cu rolul de a asigura persistența, integritatea și scalabilitatea diverselor seturi complexe de date necesare dezvoltării unei aplicații. PostgreSQL este utilizat în prezent în multe proiecte datorită simplității, siguranței și a costurilor reduse dar și a compatibilității sale cu proprietățile ACID.

În contextul aplicației Investing Wizard am folosit acest tip de sistem deoarece pune la dispoziție toate instrumentele necesare realizării acesteia în forma sa actuală, asigurând totodată integritatea și securitatea, permițând astfel concentrarea resurselor pe rapiditatea dezvoltării aplicației.

3.4 Entity Framework Core

În vederea realizării unei soluții simple de modelare a datelor, alături de PostgreSQL am utilizat și Entity Framework Core,¹⁶ o tehnologie folosită în calitate de mapper relațional de obiecte, care permite dezvoltatorului să configureze structura bazei de date și să stabilească legăturile dintre entități direct din codul C# și are capacitatea de a genera și modifica modelele de date. Cu ajutorul acesteia se poate folosi o abordare de tipul code first, ceea ce implică proiectarea entităților relevante în cod și crearea ulterioară a bazei de date.

3.5 AutoMapper

AutoMapper¹⁷ este o bibliotecă simplă care are rolul de a converti un obiect de intrare de un anumit tip într-un obiect de ieșire de alt tip. Configurarea și utilizarea acesteia sunt ușor realizabile dacă se respectă anumite convenții legate de tipurile și denumirile proprietăților. În cadrul aplicației folosesc AutoMapper pentru a transforma ușor entitățile de domeniu în Data Transfer Objects.

¹⁵ <https://www.postgresql.org/about/>

¹⁶ <https://learn.microsoft.com/en-us/ef/core/>

¹⁷ <https://docs.automapper.org/en/stable/Getting-started.html#what-is-automapper>

3.6 MediatR

MediatR¹⁸ este o bibliotecă de mici dimensiuni, inventată de creatorul AutoMapper-ului, Jimmy Bogard, ce are ca scop implementarea patternului Mediator [2] într-un program .NET. În general, este folosită în aplicații pentru a trimite cereri simple și pentru a procesa răspunsuri într-un mod structurat. Pe de altă parte, poate fi folosită și pentru a controla comportamentul unor astfel de cereri în cadrul aplicației.

3.7 Serilog

Serilog¹⁹ este o bibliotecă .NET care oferă funcționalități de logging structurat și de diagnosticare a aplicației. Este ușor de utilizat și portabilă între toate versiunile recente de .NET, ceea ce face mentenanța să fie destul de ușoară. Poate fi configurată să afișeze informații sau erori în consolă, fișiere sau chiar să fie găzduite pe server, fiind astfel utilă pentru aplicații de diverse dimensiuni. Biblioteca este concepută pentru a gestiona evenimente, ceea ce o face compatibilă cu MediatR.

3.8 Quartz.NET

Quartz.NET²⁰ este un sistem de programare, administrare și monitorizare a sarcinilor pentru aplicațiile .NET, folosit atât în aplicații mici cât și în producție. Acesta poate rula integrat într-o aplicație sau ca un grup de programe independente. Configurarea este atât intuitivă cât și ușor adaptabilă deoarece pune la dispoziție interfețe simple, construite pentru a reprezenta sarcinile, care ulterior pot fi declanșate automat, la un moment în timp stabilit.

3.9 Twilio SendGrid

Twilio SendGrid²¹ este o soluție folosită pentru livrarea e-mailurilor, ușor de integrat în aplicații cu ajutorul API-urilor pe care le pune la dispoziție. Pe lângă funcționalitatea de trimitere, tehnologia include o interfață modernă pentru monitorizarea primirii e-mailurilor și a reacțiilor utilizatorilor, fiind astfel adoptată de multe companii mari, la nivel internațional.

¹⁸ <https://www.jimmybogard.com/sharing-context-in-mediatr-pipelines/>

¹⁹ <https://serilog.net/>

²⁰ <https://www.quartz-scheduler.net/>

²¹ <https://sendgrid.com/en-us/solutions/email-api>

3.10 EODHD API

EODHD²² este un furnizor de date financiare, cum ar fi prețuri istorice, prețuri în timp real, date fundamentale despre companii, piețe, perechi forex și altele. Acestea sunt expuse către utilizator prin diverse API-uri. În vederea realizării aplicației Investing Wizard am folosit patru dintre acestea.

3.10.1 Exchanges API

Exchanges API²³ este folosit pentru a obține date specifice unui Exchange cum ar fi numele acestuia, moneda de tranzacționare, zilele săptămânii și orele între care piața este deschisă și zilele de sărbători legale ale fiecărei țări în care exchange-ul respectiv nu operează.

3.10.2 Fundamental Data API

Fundamental Data API²⁴ este cel mai popular API din cadrul serviciilor EODHD. Acesta oferă un sumar pentru fiecare companie disponibilă, informații generale, statistici, indicatori tehnici, date istorice despre dividende și divizări ale acțiunilor, raporturi, acționari și multe alte date fundamentale.

3.10.3 End Of Day Data API

API-ul End Of Day²⁵ este utilizat pentru a prelua prețurile istorice a diverse active financiare, acoperind toată piața americană, unele date fiind chiar de acum 50 de ani, și multe alte piețe din afara Americii, pentru majoritatea înregistrându-se prețuri din jurul anului 2000.

²² <https://eodhd.com/financial-apis/quick-start-with-our-financial-data-apis>

²³ <https://eodhd.com/financial-apis/exchanges-api-trading-hours-and-stock-market-holidays>

²⁴ <https://eodhd.com/financial-apis/stock-etfs-fundamental-data-feeds>

²⁵ <https://eodhd.com/financial-apis/api-for-historical-data-and-volumes>

3.10.4 Live Stock Prices API

Live Stock Prices API²⁶ oferă acces la prețurile diverselor active, perechi forex și valori mobiliare listate pe diverse piețe financiare. Deși numele sugerează că prețurile primite sunt în timp real, acestea au totuși o întârziere de 15-20 de minute pentru active și 1 minut pentru forex. Acest lucru apare din cauza neconcordanțelor legislative care apar în diferite țări, unele impunând de asemenea și prețuri foarte mari pentru date bursiere în timp real. Aplicația Investing Wizard nu este o aplicație gândită special pentru tranzacționare, ci mai degrabă pentru analiză financiară, deci întârzierea nu afectează într-un mod concret modul în care lucrează. EODHD propune și o variantă în timp real, cu întârziere de doar 50 ms, bazată pe protocolul WebSocket, dar doar pentru piața americană și forex. Problema folosirii acestora ar fi fost inconsistența prețurilor companiilor listate în țări diferite și creșterea complexității aplicației pentru un câștig relativ nesemnificativ.

3.11 OpenAI API

API-ul OpenAI²⁷ furnizează o interfață simplă pentru dezvoltatori în vederea integrării unui strat de inteligență artificială în diverse aplicații. Acestea funcționează pe principiul intrare/ieșire, ceea ce înseamnă că un asistent care are la bază un model OpenAI va primi un text și va întoarce un răspuns după un anumit set de reguli. În aplicația Investing Wizard acest API este folosit pentru a împărți eficient anumite seturi de date și pentru a oferi utilizatorului un răspuns care să fie cât mai asemănător cu unul oferit de un om.

3.12 Alpha Intelligence API

API-urile Alpha Intelligence formează un set de API-uri oferite de Alpha Vantage²⁸ construite pe baza expertizei lor în inteligență artificială și analiză financiară cu scopul de a duce aplicațiile la un nivel mai înalt.

²⁶ <https://eodhd.com/financial-apis/live-realtime-stocks-api>

²⁷ <https://platform.openai.com/docs/quickstart>

²⁸ <https://www.alphavantage.co/documentation/>

3.12.1 Market News & Sentiments API

În cadrul aplicației Investing Wizard am folosit API-ul specializat pe știri de piață și tendințele curente, din cadrul setului Alpha Intelligence. Acest API oferă date despre cele mai recente știri dintr-o selecție mare de surse din întreaga lume, referitoare la companii specifice și alte tipuri de active.

3.13 Syncfusion

Syncfusion²⁹ este o companie care oferă un ecosistem de componente pentru interfețe grafice pentru web, desktop și mobile cu scopul de a face aplicațiile mai ușor de dezvoltat și personalizat. Componentele Syncfusion sunt folosite la scara largă, de companii precum Apple, IBM, Intel și Visa dar și de dezvoltatori individuali, pentru aplicații de complexitate mică.

3.14 Concluzii

Aplicația Investing Wizard este proiectată cu ajutorul celor mai noi tehnologii din sfera ASP.NET și aspiră să fie ușor de migrat la următoarele versiuni. De asemenea, un aspect important pe care l-am luat în considerare atunci când am ales aceste tehnologii a fost compatibilitatea dintre ele. Fiecare dintre acestea joacă un rol important în cadrul aplicației, fiind folosite în implementarea pragmatică a funcționalităților. Utilizând aceste tehnologii moderne și populare, activ suportate, validate prin implementări cu succes în diverse proiecte de producție, asigur că aplicația va putea fi întreținută pe termen lung.

²⁹ <https://www.syncfusion.com/company/about-us>

Capitolul 4: Arhitectura aplicației și detalii de implementare

4.1 Arhitectura generală

Alături de tehnologiile moderne, arhitectura este un alt aspect care are ca scop asigurarea unui ciclu de viață complet al aplicațiilor. Pe de altă parte, și modul în care este structurat codul are importanța sa. Acesta ar trebui să fie scris în așa fel încât să poată fi înțeles atunci când este citit și ușor de schimbat în cazul în care este nevoie.

4.1.1 Clean Architecture

Clean Architecture [4] este un concept apărut în cartea cu același titlu, inventat de Robert C. Martin, cunoscut drept „Uncle Bob”, care subliniază o serie de principii comune întâlnite în arhitecturile aplicațiilor software de succes. Aceasta sugerează că în primul rând regulile de business ale unei aplicații trebuie să fie complet independente de framework-uri, baze de date, interfețe de utilizator și alte servicii externe. În al doilea rând apare regula dependențelor, care presupune că orice strat exterior are acces la interior dar nu și invers. **Figura 4.1.1** ilustrează o reprezentare simplificată a acestui concept.

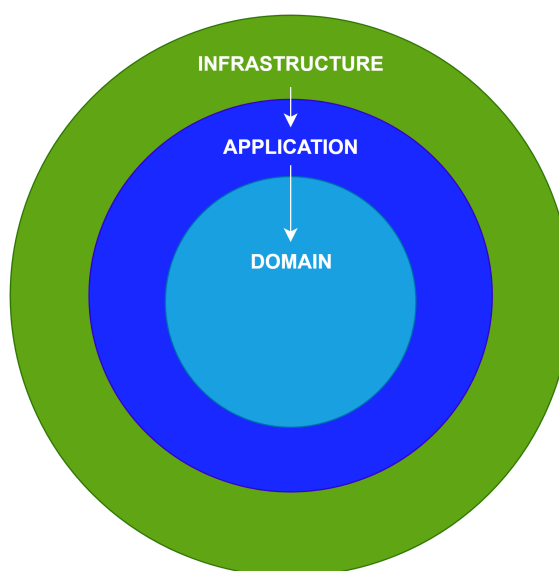


Figura 4.1.1 : Diagramă simplificată Clean Architecture

Această diagramă sugerează împărțirea proiectului în trei straturi de bază. Primul strat este cel de domeniu, unde în general definim entitățile și regulile de business. Al doilea, denumit Application, este responsabil pentru definirea cazurilor de utilizare. Aici stabilim cum sunt utilizate regulile de domeniu pentru a respecta cerințele aplicației. Ultimul strat, Infrastructure, cuprinde detaliile tehnice ale aplicației, framework-uri, persistența datelor, servicii externe și altele. Regula dependențelor este reprezentată prin săgeți, straturile exterioare au acces la straturile inferioare dar nu invers. Într-adevăr, o aplicație completă este formată din mai multe componente, dar am ales să reprezint momentan așa pentru a structura mai clar viziunea mea asupra Clean Architecture.

4.1.2 Structura soluției

Pornind de la diagrama descrisă anterior, am creat o primă componentă Core, structurată în proiectele Domain și Application. Apoi, o altă componentă numită Infrastructure unde avem implementări concrete, servicii de autentificare și autorizare, persistența datelor și orice alte framework-uri și servicii externe. Pentru partea de web avem două componente, respectiv două proiecte, numite WebUI și WebApi iar pentru a integra inteligența artificială în aplicație am făcut un proiect separat, deși nu era neapărat nevoie, numit TradingAssistant. Decizia de a face un proiect nou, are la bază ideea de a dezvolta, testa și găzdui partea de inteligență artificială fără a depinde de celelalte componente, deoarece tehnologia evoluează rapid în această zonă, ceea ce face izolarea un posibil avantaj în vederea integrării în versiuni viitoare. Pe lângă aceste componente am mai creat și un proiect comun Shared unde am implementat diverse clase care se împart între mai multe proiecte sau alte bucăți de cod reutilizabil. Structura soluției se poate observa în **Figura 4.1.2**.



Figura 4.1.2 : Structura soluției

4.1.3 Clasa Result

Un exemplu concret din acest proiect sunt clasele `Result` și `Result<T>` ce implementează natural ideea de Result Pattern³⁰. Folosirea patternului presupune returnarea unui rezultat ce poate lua valori de tip „success” sau „failure”, împreună cu valoarea în sine a acestuia și o eroare sau un mesaj adițional dacă este cazul. Întoarcerea unui rezultat de acest tip implică și o abordare cu gărzi de tipul if/return în loc de instrucțiuni if imbricate. Aceste idei au apărut inițial în programarea funcțională, fiind cunoscute ca Railway Oriented Programming³¹. În aplicațiile .NET în schimb, pot îmbunătăți performanța și ușura citirea codului, fără a încălca principiul fail fast, chiar dacă nu aruncăm excepții.

4.2 Domeniul aplicației

În partea de domeniu a componentei Core am definit entitățile folosind o abordare bazată pe agregate care presupune crearea unui nou folder pentru fiecare agregat, în care sunt incluse toate entitățile ce se leagă de acesta, împreună cu reguli de business și alte obiecte ajutătoare. În scopul realizării unui domeniu bogat am folosit diverse principii de încapsulare, am creat metode publice pentru accesarea proprietăților și am folosit patternul Factory Method [2] în loc să expun un constructor public. Să luăm ca exemplu clasa `Watchlist` din aplicația Investing Wizard. Dacă am fi lăsat un model anemic, aceasta ar fi arătat aproximativ ca în **Fragmentul 4.2.1**.

```
public class Watchlist(Guid userId, string name)
{
    public Guid Id { get; set; } = Guid.NewGuid();
    public Guid UserId { get; set; } = userId;
    public string Name { get; set; } = name;
    public List<string> SecurityCodes { get; set; } = [];
}
```

Fragmentul 4.2.1: Domeniu anemic.

³⁰ <https://www.milanjovanovic.tech/blog/functional-error-handling-in-dotnet-with-the-result-pattern>

³¹ <https://fsharpforfunandprofit.com/rop/>

Problema cu aceasta este faptul că putem modifica liber orice proprietate a unei instanțe ale acestei clase oriunde în proiect. Un astfel de comportament este în general nedorit în aplicații, de aceea, am creat un domeniu bogat. O variantă de implementare a unei clase de genul este reprezentată în **Fragmentul 4.2.2**.

```
public sealed class Watchlist
{
    public Guid Id { get; private set; }
    public Guid UserId { get; private set; }
    public string Name { get; private set; }
    public List<string> SecurityCodes { get; private set; }
    public static Watchlist Create(Guid userId, string name)
    {
        return new Watchlist(userId, name);
    }
    public void UpdateId(Guid id) => Id = id;
    public void UpdateName(string name) => Name = name;
    public void AddSecurityCode(string securityCode)
        => SecurityCodes.Add(securityCode);
    public void RemoveSecurityCode(string securityCode)
        => SecurityCodes.Remove(securityCode);
    private Watchlist(Guid userId, string name)
    {
        Id = Guid.NewGuid();
        UserId = userId;
        Name = name;
        SecurityCodes = [];
    }
}
```

Fragmentul 4.2.2: Domeniu bogat.

Pe lângă implementările de genul acesta, la nivelul domeniului am definit și interfețele de repository corespunzătoare fiecărui agregat. Acestea implică folosirea patternului Repository [3]. Un exemplu succint pentru două dintre funcționalitățile clasei descrise mai devreme poate fi observat în **Fragmentul 4.2.3**.

```
public interface IWatchlistRepository
{
    Task<Result> AddAsync(Watchlist entity);
    Task<Result<List<Watchlist>>> GetByUserIdAsync(Guid userId);
}
```

Fragmentul 4.2.3: Interfața unui repository.

Într-adevăr, în practică am folosit o interfață generică și apoi am extins-o cu proprietăți specifice pentru fiecare dintre aggregate, pentru a ușura scrierea. Abordarea aceasta de a crea un domeniu bogat îmbină principiile Domain Driven Design [1] cu scrierea de cod cât mai simplu și ușor de înțeles.

4.3 Comenzi și interogări

Următorul proiect din cadrul Core-ului este Application. În cadrul acestuia am structurat funcționalitățile întâi după entitatea pe care o vizează, apoi după tipul de operație pe care o execută. În acest context am ales patternul Command Query Responsibility Segregation³², pe scurt CQRS [5], care sugerează împărțirea clară a comenzilor și a interogărilor. Fiecare dintre aceste operații este numită apoi după convenții standard. Tot în acest proiect este important să stabilim și comportamentele acestora. Biblioteca potrivită pentru aceste sarcini este MediatR.

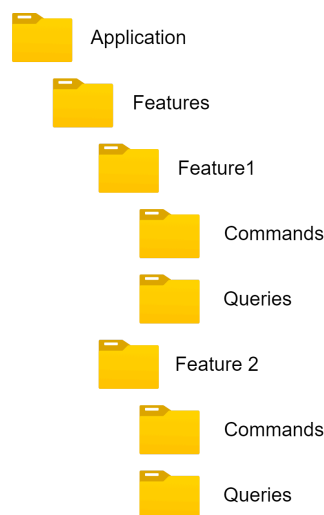


Figura 4.3.1 : Structura funcționalităților

Pentru a explora detaliile implementării voi merge pe același exemplu pe care l-am oferit în domeniul aplicației. Pe lângă numele sugestiv, operațiile trebuie să implementeze interfața `IRequest<TResponse>` din cadrul bibliotecii, așa cum am exemplificat în **Fragmentul 4.3.1**.

³² <https://martinfowler.com/bliki/CQRS.html>

```
public sealed record GetWatchlistByIdQuery(
    Guid Id) : IRequest<Result<WatchlistResponseDto>>;

public sealed record AddWatchlistCommand(
    Guid UserId, string Name) : IRequest<Result>;
```

Fragmentul 4.3.1: Exemplu de interogare și comandă.

După ce am înregistrat aceste clase ca fiind operații de tip request am implementat interfața `IPipelineBehavior<TRequest, TResponse>`, cu scopul de a controla comportamentul acestora, în cadrul căreia am expus și o interfață pentru logging, `ILoggingService`. Pentru a putea gestiona logica principală este necesar să implementăm și câte un handler pentru fiecare comandă sau interogare. Acest lucru se face după aceleași principii, dar destul de simplu dacă am făcut un domeniu bogat, după cum se poate vedea în **Fragmentul 4.3.2**.

```
internal sealed class AddWatchlistCommandHandler(
    IWatchlistRepository watchlistRepository)
    : IRequestHandler<AddWatchlistCommand, Result>
{
    private readonly IWatchlistRepository _watchlistRepository
        = watchlistRepository;
    public async Task<Result> Handle(
        AddWatchlistCommand request,
        CancellationToken cancellationToken)
    {
        var watchlist = Watchlist.Create(request.UserId, request.Name);
        return await _watchlistRepository.AddAsync(watchlist);
    }
}
```

Fragmentul 4.3.2: Exemplu de handler.

4.4 Maparea entităților

Din cele două exemple de operații de tip request, putem observa că în general comenzile returnează un obiect de tip `Result` iar interogările un obiect de tip `Result<T>` care să conțină și valoarea obiectului. O practică des întâlnită este crearea unui obiect de tip Response Data Transfer Object, cu rolul de a expune doar proprietățile de care avem nevoie. Pe parte internă am folosit biblioteca AutoMapper, în modul exemplificat în **Fragmentul 4.4.1**, pentru a transforma obiectele de domeniu primite de la repository în obiecte de transfer de date.

```
var watchlist = await _watchlistRepository.FindByIdAsync(request.Id);  
return _mapper.Map<WatchlistResponseDto>(watchlist.Value);
```

Fragmentul 4.4.1: Conversie folosind AutoMapper

În schimb, pentru datele primite de la API-urile externe am expus o interfață unde am supraîncărcat metoda `Map()` ca în exemplul din **Fragmentul 4.4.2**.

```
public interface IEntityMapper  
{  
    Exchange Map(string exchangeCode, ExchangeExternalDto exchangeData);  
    Price Map(string securityCode, PriceExternalDto priceData);  
}
```

Fragmentul 4.4.2: Conversie folosind Mapper-ul personalizat

Decizia de a crea o metodă de mapare în plus, dedicată exclusiv datelor preluate din surse externe, se datorează mai multor factori. În primul rând, complexitatea acestor date este ridicată și nu avem mereu posibilitatea de a garanta că structura acestora și denumirile proprietăților sunt la fel cu entitățile de domeniu, așa cum cere AutoMapper. În al doilea rând versiunile viitoare sau alte servicii auxiliare ar putea necesita mici schimbări, ceea ce ar fi destul de ușor de realizat dacă maparea a fost făcută manual.

Tot în cadrul proiectului Application am expus și interfețe pentru restul serviciilor externe de care avem nevoie. Astfel putem spune că respectăm principiul SOLID al inversării dependențelor [3]. Acest lucru se realizează la nivel de cod folosind mecanismul de Dependency Injection³³ cu ajutorul căruia putem injecta interfețele în comenzi sau interogări, codul devenind astfel decuplat, pe principiile Clean Architecture.

³³ <https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection>

4.5 Servicii și infrastructură

După cum am descris anterior, am alcătuit componenta Infrastructură în mare parte din implementările concrete ale interfețelor expuse în logica de business la care am adăugat unele funcționalități pentru a face aplicația mai eficientă și mai ușor de întreținut.

4.5.1 Persistența datelor

Pe partea de persistență a datelor am folosit Entity Framework Core pentru a configura și modela într-un mod facil baza de date PostgreSQL care are rolul de a stoca informațiile. În primul rând am creat o clasă care extinde clasa `DbContext`³⁴, care este o implementare a patternului Unit Of Work. Aceasta expune `DbSet`³⁵-uri, care reprezintă colecții de elemente de același tip dintr-un context. Apoi am configurat separat legăturile dintre agregatele și entitățile aplicației pentru a asigura crearea corectă a bazei de date. Pentru a implementa interfețele de tip `IRepository` am injectat clasa creată anterior într-o clasă generică `Repository`, apoi am extins această clasă pentru fiecare agregat în parte, implementând și contractele definite în domeniul aplicației, ce conțin operații personalizate. În acest fel, implementările repository-urilor constituie un wrapper peste `DbContext`, combinând patternurile Repository³⁶ și Unit Of Work.

4.5.2 Implementarea serviciilor

Cele mai importante servicii din aplicația Investing Wizard, care servesc la baza funcționalităților sunt: serviciul de API-uri externe, serviciul de actualizare a prețurilor curente din memoria cache, serviciul de preluare a prețurilor din memoria cache și serviciul care actualizează prețurile istorice în funcție de starea pieței și ultimele prețuri din memoria cache. Primul dintre acestea folosește `HttpClient`³⁷ pentru a face request-uri către API-urile EODHD cu scopul de a introduce în baza de date a aplicației informații despre companii și piețe financiare, împreună cu prețuri istorice ale valorilor mobiliare sau a perechilor forex.

³⁴ <https://learn.microsoft.com/en-us/dotnet/api/system.data.entity.dbcontext?view=entity-framework-6.2.0>

³⁵ <https://learn.microsoft.com/en-us/dotnet/api/system.data.entity.dbset-1?view=entity-framework-6.2.0>

³⁶

https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design?trk=article-ssr-frontend-pulse_little-text-block

³⁷ <https://learn.microsoft.com/en-us/dotnet/fundamentals/networking/http/httpclient>

Serviciul de actualizare a prețurilor curente din memoria cache preia din baza de date codurile companiilor existente, folosește API-ul EODHD care furnizează prețurile reale și introduce în memoria cache ultimul preț înregistrat, folosind interfața `IMemoryCache`³⁸. Acest lucru se realizează automat, la un interval de timp prestabilit de administrator. Al treilea serviciu are rolul de a prelua prețurile curente din memoria cache, folosind de asemenea `IMemoryCache`. Serviciul de actualizare a prețurilor istorice are ca rol salvarea în baza de date a ultimului preț legat de o valoare mobilă sau o pereche forex care se poate găsi salvat în cache. Pentru a asigura corectitudinea și consistența am folosit biblioteca `TimeZoneConverter`³⁹.

4.5.3 Programarea job-urilor

Pentru a putea asigura actualizarea corectă a prețurilor, am folosit sistemul Quartz care utilizează interfețe de tipul `IJob` în vederea programării job-urilor. Tot ce a trebuit să fac a fost să implementez interfața respectivă, să injectez service-ul care face update și să programez rularea acestuia în cazul în care o piață a fost deschisă la tranzacționare în ziua curentă dar acum este închisă. Acest lucru sugerează că putem să luăm ultimul preț salvat în cache de serviciul de actualizare și să îl introducem în baza de date ca preț istoric la data zilei curente.

4.5.4 Integrarea de logging structurat

Pentru a monitoriza aplicația am folosit Serilog, împărțind funcționalitatea în două părți. În primul rând am oferit o implementare concretă a interfeței `ILoggingService`, definită anterior în Application, în clasa destinată gestionării comportamentului operațiilor de tip request, mai exact comenzile și interogările. Cu ajutorul acestei implementări putem integra Serilog în aplicație. Am ales varianta de afișare în consolă pentru acest stadiu al aplicației.

³⁸ <https://learn.microsoft.com/en-us/aspnet/core/performance/caching/memory?view=aspnetcore-8.0>

³⁹ <https://www.nuget.org/packages/timezoneconverter/>

În **Figura 4.5.4** observăm un exemplu de monitorizare a comportamentelor din Application.

```
[17:47:40 INF] GetPortfolioEntriesByIdQuery handled successfully
[17:47:40 INF] Started to handle GetDividendByIdQuery
[17:47:40 INF] GetDividendByIdQuery handled successfully
[17:47:42 INF] Started to handle GetAllCompanyCodesQuery
[17:47:42 INF] GetAllCompanyCodesQuery handled successfully
[17:47:42 INF] Started to handle GetPortfolioTransactionsByIdQuery
[17:47:42 INF] GetPortfolioTransactionsByIdQuery handled successfully
[17:47:42 INF] Started to handle GetAllCompanyCodesQuery
[17:47:42 INF] GetAllCompanyCodesQuery handled successfully
[17:47:42 INF] Started to handle GetPortfolioTransactionsByIdQuery
[17:47:42 INF] GetPortfolioTransactionsByIdQuery handled successfully
[17:47:42 INF] Started to handle GetAllCompanyCodesQuery
[17:47:42 INF] GetAllCompanyCodesQuery handled successfully
[17:48:06 INF] Started to handle AddTransactionCommand
[17:48:06 WRN] Cache miss for CompanyCode
[17:48:06 ERR] AddTransactionCommand failed with error: Error { Code = UnexpectedNullValue, Description = Value is null. }
[17:48:06 INF] Started to handle GetPortfolioTransactionsByIdQuery
[17:48:06 INF] GetPortfolioTransactionsByIdQuery handled successfully
```

Figura 4.5.4: Logging pentru commands și queries

Ca soluția să fie cât mai avansată am integrat logging structurat și pentru request-urile http, prin expunerea unui middleware și atribuirea de identificatori de urmărire și corelare pentru requesturi. Acest lucru ne permite să vedem în detaliu răspunsurile primite de la server, de la API-urile externe și cum anume depind unele de celelalte, un detaliu important în complexitatea aplicației deoarece întoarcerea de obiecte de tip `Result` nu oferă de la sine urmărirea stivei, ca în cazul excepțiilor. În **Figura 4.5.5** de exemplu putem observa ce mesaj de eroare ar returna API-ul EODHD de date fundamentale dacă un administrator ar introduce un cod ce nu aparține niciunei companii. În cazul de față s-a trimis codul „RandomCode” către API. Această funcționalitate este folositoare și în cazul în care primim erori de la servicii externe ce nu sunt mereu definite în domeniu.

```
[18:11:28 INF]
Started to handle GetCompanyDataFromExternalApiQuery
{"CorrelationId": "0HN4H7B8QPVD:00000077", "ActionId": "d711c002-ee37-4a7f-a04d-e3fcff7c5b8c", "ActionName":
"InvestingWizard.WebUI.Controllers.CompaniesController.GetCompanyDataFromExternalApi (InvestingWizard.WebApi)"
, "RequestId": "0HN4H7B8QPVD:00000077", "RequestPath": "/api/companies/external/RandomCode", "ConnectionId":
"0HN4H7B8QPVD"}

[18:11:29 ERR]
GetCompanyDataFromExternalApiQuery failed with error: CustomError { Code = NotFound, Description = Error: NotF
ound, Content: Symbol not found }
{"CorrelationId": "0HN4H7B8QPVD:00000077", "ActionId": "d711c002-ee37-4a7f-a04d-e3fcff7c5b8c", "ActionName":
"InvestingWizard.WebUI.Controllers.CompaniesController.GetCompanyDataFromExternalApi (InvestingWizard.WebApi)"
, "RequestId": "0HN4H7B8QPVD:00000077", "RequestPath": "/api/companies/external/RandomCode", "ConnectionId":
"0HN4H7B8QPVD"}

[18:11:29 INF]
HTTP GET /api/companies/external/RandomCode responded 200 in 991.8358 ms
{"SourceContext": "Serilog.AspNetCore.RequestLoggingMiddleware", "RequestId": "0HN4H7B8QPVD:00000077", "Conne
ctionId": "0HN4H7B8QPVD"}
```

Figura 4.5.5: Logging structurat cu ajutorul unui middleware.

Utilizatorii aplicației pot fi de asemenea monitorizați pentru a depista din timp încercări de atacuri și pentru a putea oferi suport într-un mod mai practic. În **Figura 4.5.6** observăm modul în care putem vedea activitatea unui utilizator.

```
[18:17:15 INF]
User with id c56d0013-94d2-418c-8ca2-8ff677fe3963 logged in.
{"CorrelationId": "0HN4H7B8QPVD:00000093", "RequestId": "0HN4H7B8QPVD:00000093", "RequestPath": "/Account/Login", "ConnectionId": "0HN4H7B8QPVD"}

[18:17:15 INF]
HTTP POST /Account/Login responded 302 in 198.5859 ms
{"SourceContext": "Serilog.AspNetCore.RequestLoggingMiddleware", "RequestId": "0HN4H7B8QPVD:00000093", "ConnectionId": "0HN4H7B8QPVD"}

[18:18:11 INF]
User with ID 'c56d0013-94d2-418c-8ca2-8ff677fe3963' has enabled 2FA with an authenticator app.
{"CorrelationId": "0HN4H7B8QPVD:000000C5", "RequestId": "0HN4H7B8QPVD:000000C5", "RequestPath": "/Account/Manage/EnableAuthenticator", "ConnectionId": "0HN4H7B8QPVD"}

[18:18:24 INF]
User with ID 'c56d0013-94d2-418c-8ca2-8ff677fe3963' logged in with 2fa.
{"CorrelationId": "0HN4H7B8QPVD:00000107", "RequestId": "0HN4H7B8QPVD:00000107", "RequestPath": "/Account/LoginWith2fa", "ConnectionId": "0HN4H7B8QPVD"}
```

Figura 4.5.6: Monitorizare de utilizatori în cadru logging-ului.

4.6 Autentificare și autorizare

Autentificarea și autorizarea constituie una dintre cele mai importante părți ale unei aplicații. Folosind un serviciu testat, precum ASP.NET Identity, sporim securitatea aplicației și facem dezvoltarea mai ușoară. Decizia mea de a folosi un astfel de serviciu are la bază statisticile din perioada curentă referitoare la probleme de autentificare în aplicații. Odată cu adoptarea cât mai largă a soluțiilor de acest tip riscul de securitate a scăzut în această direcție, conform top-urilor OWASP⁴⁰ din anii 2017 și 2021. În primul rând, am implementat această funcționalitate la nivel de Infrastructură, separând tot ce ține de utilizatori într-o bază de date complet izolată.

Am repetat aceiași pași ca la persistența datelor aplicației dar am implementat interfețele puse la dispoziție de biblioteca Identity pentru a putea integra în soluția de autentificare un strat de comunicare cu baza de date. În legătură cu autorizarea, am pus la dispoziție două tipuri de utilizator: „user” respectiv „admin” și am reprezentat în cod rolurile aferente, apoi am implementat o metodă de inițializare a bazei de date. Această tehnică mai este cunoscută sub numele de Database Seeding⁴¹, adică popularea acesteia cu un set de date. În cazul aplicației Investing Wizard aceasta reprezintă concret inițializarea rolurilor și crearea unui utilizator cu rolul de „admin” ce se poate autentifica imediat în aplicație.

⁴⁰ <https://owasp.org/www-project-top-ten/>

⁴¹ <https://learn.microsoft.com/en-us/ef/core/modeling/data-seeding>

Modalitatea standard de autentificare în aceste tipuri de aplicații este bazată pe cookie-uri. Am decis ca în acest caz să folosesc middleware-ul pentru validarea tokenilor antiforgery⁴² adăugat odată cu .NET 8, pentru a securiza sesiunile.

Referitor la permisiuni, un utilizator trebuie să fie autorizat pentru a putea folosi oricare dintre funcționalitățile aplicației. Folosesc această abordare, împreună cu monitorizarea activităților, pentru a putea dezvolta cât mai ușor în continuare o metodă de apărare împotriva atacurilor de tip DoS și DDoS⁴³. Iau în considerare acest lucru din cauză că aplicația gestionează seturi de date destul de voluminoase în raport cu arhitectura sa.

Pe partea de funcționalități, ASP.NET Identity pune la dispoziție logica autentificării, pe care dezvoltatorii pot să o modifice ulterior după cerințele proiectului. În acest context am implementat prima dată o imagine QR, cu ajutorul unei biblioteci foarte simple, numită QRCode⁴⁴, și am integrat cu partea de autentificare cu doi factori pusă la dispoziție de Identity.

De asemenea, am folosit serviciul Twilio SendGrid pentru a implementa interfața `IEmailSender<ApplicationUser>` cu scopul de a trimite e-mailuri în vederea confirmării sau recuperării conturilor de utilizator.

⁴² <https://learn.microsoft.com/en-us/aspnet/core/security/anti-request-forgery?view=aspnetcore-8.0>

⁴³ <https://www.microsoft.com/ro-ro/security/business/security-101/what-is-a-ddos-attack>

⁴⁴ <https://www.nuget.org/packages/QRCode/>

4.7 API

Pentru a expune funcționalitățile aplicației către exterior am proiectat componenta WebApi, un proiect de tip API⁴⁵ în cadrul căruia am făcut câte un Controller corespunzător fiecărui agregat, folosind în continuare biblioteca MediatR.

Un exemplu simplificat de Controller, care să expună endpoint-ul pentru crearea unei noi liste de urmărire, injectând interfața `IMediator`, poate fi explorat în cadrul **Fragmentului 4.7.1**.

```
[ApiController]
[Route("api/watchlists")]
public class WatchlistsController(IMediator mediator)
    : ControllerBase
{
    private readonly IMediator _mediator = mediator;
    [HttpPost]
    public async Task<IActionResult> AddWatchlist(
        [FromBody] AddWatchlistRequestDto request)
    {
        var result = await _mediator.Send(
            new AddWatchlistCommand(
                request.UserId,
                request.Name)
            );
        return Ok(result);
    }
}
```

Fragmentul 4.7.1: Controller pentru watchlists.

⁴⁵ <https://learn.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-3.1>

4.8 Interfața Web

Această componentă este alcătuită dintr-un singur proiect, numit WebUI, după cum am precizat anterior. Rolul acesteia este atât comunicarea cu API-urile cât și expunerea interfeței de utilizator.

4.8.1 Structura proiectului Blazor

În vederea structurării proiectului am luat în considerare modul în care este proiectat Blazor, și anume modular. Astfel, am proiectat întâi componente ce reprezintă diverse șabloane, precum aspectul principal, aspectul cuprinsului paginii și meniul de navigare. Aceste trei pagini au rolul de a crea un punct de pornire inițial, o practică întâlnită în general în Single Page Applications.

Așadar, toate componentele aplicației sunt afișate în centrul paginii pe care o putem observa în **Figura 4.8.1**, acest lucru îmbunătățind în primul rând performanța, iar în al doilea rând aspectul, totodată facilitând o ușoară dezvoltare a aplicației pe viitor.

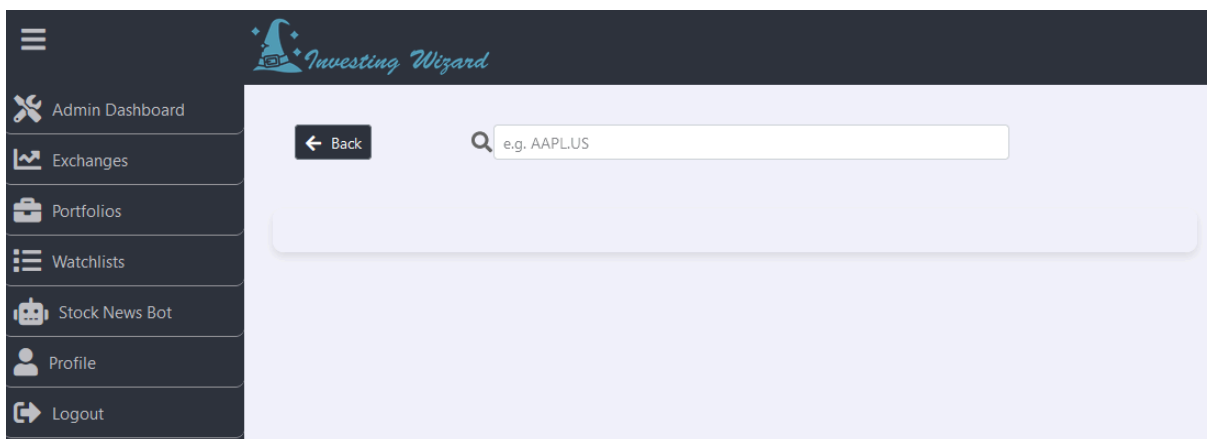


Figura 4.8.1: Aspectul principal.

După crearea acesteia am separat componentele în două categorii: cele care au legătură cu profilul și contul utilizatorului și cele care au legătură cu funcționalitățile. Pentru prima categorie am păstrat structura impusă de ASP.NET Core Identity pentru a folosi componentele în scopul în care au fost create, fără împachetarea acestora în alte clase.

Acest lucru oferă încă un strat de siguranță împotriva comportamentului neașteptat și asigură că principiile de securitate nu sunt încălcate. Pentru restul componentelor am creat câte un element de tip razor page unde am implementat atât partea de interfață grafică, cât și comunicarea prin API-uri. În scopul unei organizări mai bune am creat servicii de date pentru entități, după modelul **Fragmentului 4.8.1**.

```
public class WatchlistDataService(HttpClient httpClient) : IWatchlistDataService
{
    private readonly HttpClient _httpClient = httpClient;
    public async Task<Result<WatchlistModel>> AddWatchlistAsync(
        AddWatchlistRequestDto request)
    {
        var response = await _httpClient.PostAsJsonAsync(Url.AddWatchlist,
            request);
        return await response.Content.ReadFromJsonAsync<Result<WatchlistModel>>();
    }
}
```

Fragmentul 4.8.1: Serviciu de date pentru watchlist.

4.8.3 Utilizarea Syncfusion pentru componente avansate

Deoarece aplicația necesită elemente de control destul de complexe am folosit câteva dintre cele puse la dispoziție de Syncfusion. De exemplu, pentru afișarea primei pagini ale unei companii am folosit un `SfStockChart`, care vine la pachet cu funcționalități precum filtrarea perioadelor. De asemenea, am folosit pachetul compatibil Bootstrap⁴⁶ pentru a oferi elementelor o notă deosebită.

⁴⁶ <https://getbootstrap.com/docs/5.0/getting-started/introduction/>

4.9 Modulul Trading Assistant

Datorită structurării inteligente, algoritmul de tranzacționare este de impact dar simplu de implementat. Principiul de bază al acestuia este de a folosi pierderile utilizatorului în scopul de a diminua pe hârtie profitul total net anual, ceea ce duce la taxarea redusă sau chiar inexistentă. Ideea este aplicarea metodei tax-loss harvesting⁴⁷ pentru legislația din România, unde nu se aplică regula wash sale⁴⁸ pentru investiții străine. În practică, atunci când utilizatorul vrea să închidă o tranzacție algoritmul îi va sugera cele mai bune opțiuni pentru a reduce cât mai mult posibil taxele pe care ar trebui să le plătească la final de an. Utilizatorul are opțiunea de a accepta soluția dată de algoritm printr-un click, în acest caz logica aplicației se va ocupa de închiderea și deschiderea a multiple tranzacții astfel încât să satisfacă regulile impuse.

În aplicația Investing Wizard, arhitectura pe care am folosit-o permite doar utilizarea mai multor interogări, acolo unde este cazul, și o singură operație de tip command, în cadrul unei singure funcționalități, garantând astfel integritatea datelor. Pentru modulul de Trading Assistant, unde o singură sugestie este formată din închiderea și deschiderea simultană a diferite tranzacții financiare, am folosit un `TransactionScope`⁴⁹ cu rolul de a face commit dacă toate tranzacțiile financiare au fost executate corect sau rollback dacă oricare dintre acestea nu a reușit.

4.10 Stock News Assistant

Deoarece am considerat folosirea unui News API o metodă destul de complicată de a prelua conținut despre ultimele știri ale unei companii, am implementat ideea de a folosi două API-uri bazate pe inteligență artificială, unul foarte cunoscut, OpenAI și unul recunoscut mai puțin, API-ul de la Alpha Vantage. Rolul celui de-al doilea este să preia un ticker al unei companii și să returneze utilizatorului toate știrile pe care le găsește pe diferite site-uri, pentru care este îndreptățit să culeagă informațiile, împreună cu sentimentul general al pieței pentru fiecare dintre acestea.

⁴⁷ https://www.gsam.com/content/dam/gsam/pdfs/us/en/articles/2021/FAQ_Tax_Loss_Harvesting_Strategies.pdf?sa=n&rd=n

⁴⁸ https://apps.irs.gov/app/vita/content/10s/10_04_011.jsp?level=advanced

⁴⁹ <https://www.bytehide.com/blog/dotnet-transactions>

Rolul asistentului bazat pe ChatGPT este să preia aceste date structurate și să le redea în cadrul aplicației într-un format cât mai uman, astfel încât să fie puse în valoare ambele părți ale inteligenței artificiale: cantitatea de informații corecte oferită într-un timp scurt fără ajutor din partea omului și interpretarea sau filtrarea acestor date conform specificațiilor. Astfel, această funcționalitate reprezintă o soluție completă folosind inteligența artificială.

4.11 Concluzii

Am implementat logica aplicației conform principiilor DDD și Clean Architecture deoarece acestea se bazează pe separarea logicii de lumea externă, acest lucru fiind un plus major pentru scopul în care este gândită aplicația. Pentru început, modelarea unui domeniu bogat duce la posibilitatea de a avea mai mulți membri care să se ocupe de dezvoltarea aplicației, indiferent dacă fac parte din domeniul tehnic sau nu. În aceeași categorie se află și funcționalitățile descrise prin interfețe, menite să reprezinte use case-uri practice. Aceste concepte contribuie la crearea unui Ubiquitous Language [1], sau măcar duc dezvoltarea în acea direcție. Acest lucru este crucial deoarece Investing Wizard este o aplicație financiară, ce trebuie întreținută atent. Pe de altă parte, folosirea mecanismelor de injectare a serviciilor externe în logica de business permite migrarea la alți provideri de date sau alte tehnologii, într-o manieră cât se poate de simplă. Totodată dezvoltarea în sfera inteligenței artificiale este facilitată în acest fel. Un alt punct forte al aplicației constă în implementările cu ajutorul bibliotecilor cunoscute, ca de exemplu Quartz, astfel încât asigură suport și siguranță. În aceeași categorie intră și serviciul de Identity care este implementat folosind cele mai noi tehnologii de la Microsoft. În încheiere, consider că folosirea diverselor pattern-uri și practici uzuale bune fac dezvoltarea și mentenanța mult mai ușoară, reduc numărul de erori logice posibile și cresc rapiditatea de livrare a noilor proprietăți.

Capitolul 5: Funcționalități principale și scenarii de utilizare

5.1 Gestionarea utilizatorilor

Aplicația dispune de o fereastră simplă de register, modificată după șablonul oferit de serviciul integrat de Identity, după cum vedem în **Figura 5.1.1**, împreună cu o fereastră asemănătoare pentru login. Acestea păstrează validarea standard a datelor așa cum este oferită.

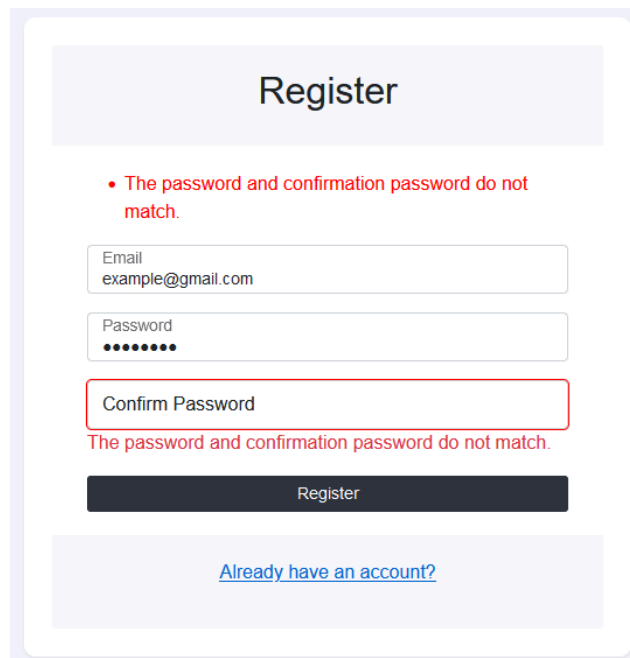
The image shows a web form titled "Register". It contains three input fields: "Email" with the value "example@gmail.com", "Password" with masked characters "••••••••", and "Confirm Password". A red border highlights the "Confirm Password" field. Above the fields, a red error message states: "• The password and confirmation password do not match." Below the "Confirm Password" field, another red error message states: "The password and confirmation password do not match." At the bottom of the form is a dark "Register" button and a link that says "Already have an account?".

Figura 5.1.1: Fereastră de register.

Referitor la implementarea serviciului Twilio SendGrid, descris anterior, în cazul înregistrării cu succes a utilizatorului, acesta primește un e-mail de genul celui afișat în **Figura 5.1.2**.

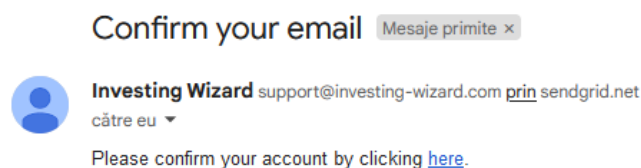


Figura 5.1.2 : E-mail trimis prin SendGrid.

Utilizatorul primește notificări pe ecranul principal atunci când acești pași sunt parcurși cu succes. Funcționalitățile de retrimiteră a link-ului de confirmare și de uitare a parolei sunt asemănătoare, realizate cu ajutorul aceluiași serviciu.

5.1.1 Managementul profilului

Utilizatorii aplicației au posibilitatea de a-și alege moneda preferată din meniul profilului, de a-și schimba email-ul sau parola, de a configura autentificarea cu doi factori și de a-și șterge contul. Toate acestea sunt oferite de soluția de identity, fiind posibilă modificarea, precum și definirea informațiilor adiționale pentru utilizatori și configurarea unei imagini de tip cod QR. În **Figura 5.1.3** putem vedea pașii de configurare, funcționalitate creată folosind template-ul furnizat de identity și biblioteca QRCode pentru imagine.

Configure authenticator app

To use an authenticator app go through the following steps:

1. Download a two-factor authenticator app like Microsoft Authenticator for [Android](#) and [iOS](#) or Google Authenticator for [Android](#) and [iOS](#).
2. Scan the QR Code or enter this key `cz3y cjas mte5 gfg3 szjl 5e2k mads 2v7u` into your two factor authenticator app. Spaces and casing do not matter.
3. Once you have scanned the QR code or input the key above, your two factor authentication app will provide you with a unique code. Enter the code in the confirmation box below.

Verify

Figura 5.1.3: Imagine cod QR cu QRCode.

5.2 Administrarea datelor

Pentru ca aplicația să poată fi întreținută ușor am creat o interfață grafică pentru utilizatorii cu rolul de „admin”. Acest tablou de bord este conceput să fie minimalist, să ofere utilizatorului posibilitatea de a adăuga și schimba înregistrări expunând doar trei carduri, după cum se observă în **Figura 5.2.1**.

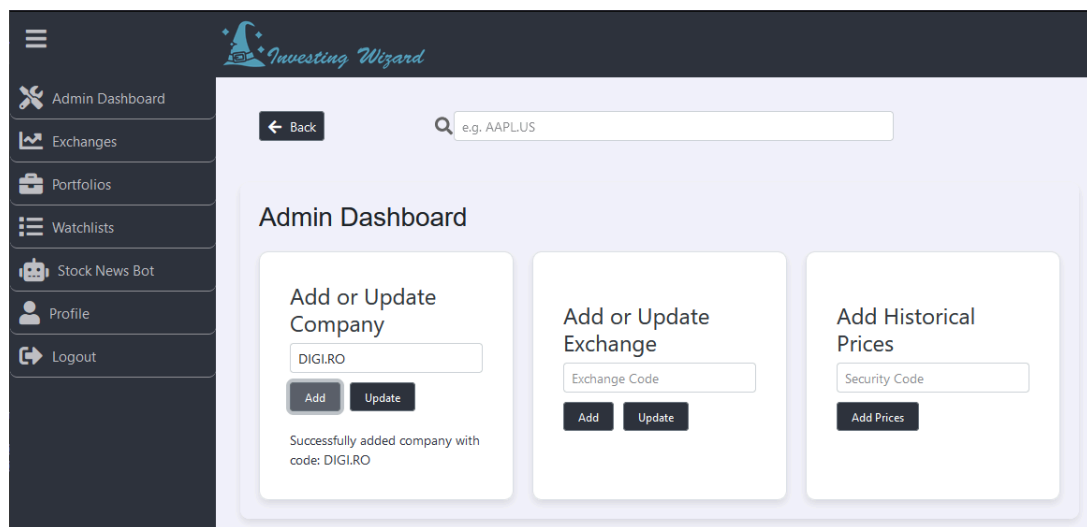


Figura 5.2.1: Tablou de bord pentru admini.

Pentru a adăuga sau modifica o înregistrare administratorul trebuie să cunoască și să furnizeze doar codul valorii mobiliare, al pieței financiare sau al perechii forex pe care o vizează, restul se întâmplă automat, așa cum am dat exemplu adăugarea companiei „DIGIRO” în **Figura 5.2.1**, folosind logica ce consuma API-urile externe.

5.3 Vizualizarea structurată a entităților

Din studiul de caz am tras concluzia că o aplicație financiară complexă ar trebui să poată gestiona în același mod date ale companiilor listate pe piețe financiare diferite. În această direcție am început prin a afișa numele companiei, ticker-ul, prețul curent și diferența dintre acesta și ultimul preț înregistrat, ca procent, în partea de sus a paginii. Totodată, am observat

că afișarea unui chart cu prețurile istorice este importantă așa că am poziționat acest prim element pe prima pagină a unei companii, așa cum putem observa în **Figura 5.3.1**.

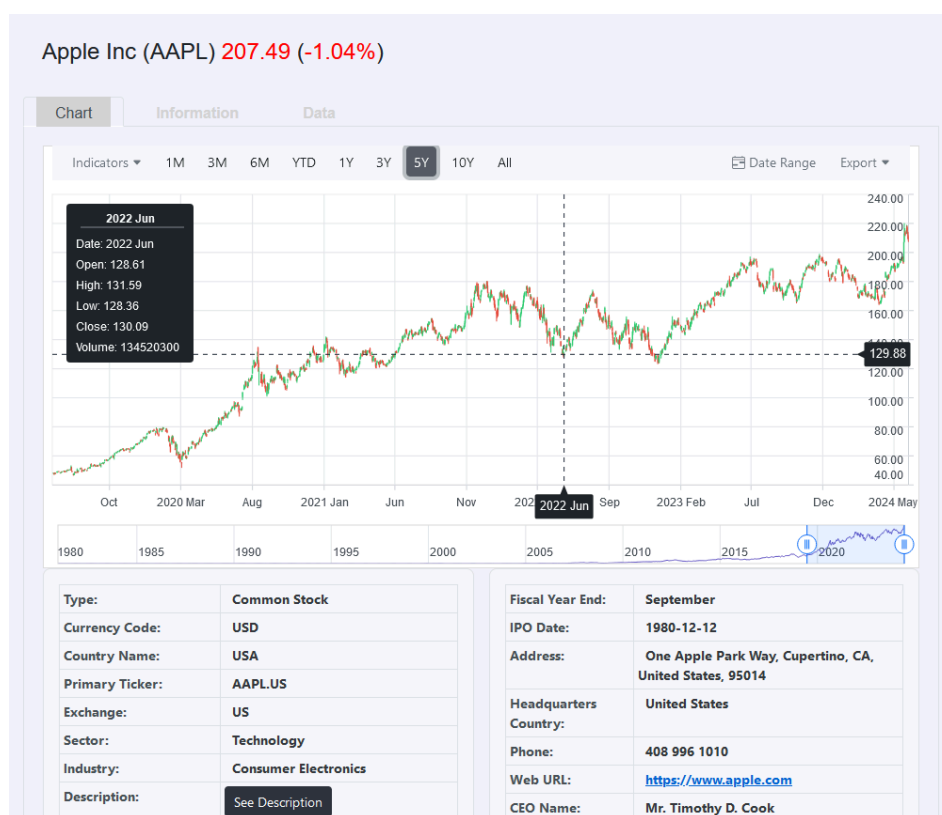


Figura 5.3.1: Prima pagină a unei companii.

Tot în cadrul acestei pagini am creat un tabel cu cele mai importante detalii ce țin de profilul unei companii. Spre exemplu, câmpul „Headquarters Country” conține numele țării unde compania are înregistrată sediul social. După cum am discutat în capitolele anterioare, este o informație la care utilizatorul ar trebui să aibă acces clar și rapid, în vederea plății impozitului pe dividende.

În următoarea secțiune din pagina unei companii, numită „Information” utilizatorul poate vedea informații cum ar fi evaluări, statistici despre acțiuni, indicatori tehnici, informații despre dividende și altele. În **Figura 5.3.2** prezint secțiunea „Splits & Dividends” a acestei pagini, ce separă într-un mod clar valoarea dividendului și procentul acestuia în raport cu prețul acțiunii. Acest lucru este disponibil pentru companiile listate atât la bursa americană cât și pentru cele catalogate drept străine, ca de exemplu cele românești.

Apple Inc (AAPL) 207.49 (-1.04%)

Chart Information Data

Highlights Valuation Shares Stats Technicals Splits & Dividends Analyst Ratings

Forward Annual Dividend Rate:	1.00
Forward Annual Dividend Yield:	0.48%
Payout Ratio:	0.15
Dividend Date:	2024-05-16
Ex Dividend Date:	2024-05-10
Last Split Factor:	4:1
Last Split Date:	2020-08-31

Figura 5.3.2: Secțiunea Information: Splits & Dividends.

Ultima secțiune a acestei pagini este numită „Data”. Aceasta conține date financiare ce ar putea fi folosite de către investitori pentru o analiză detaliată. În **Figura 5.3.3** am ales ca exemplu fila în care sunt prezentați acționarii companiei, mai exact instituțiile. Aceste valori pot fi ordonate după nevoile utilizatorului.

Apple Inc (AAPL) 207.49 (-1.04%)

Chart Information Data

Outstanding Shares Earnings Insider Transactions Holders ESG Scores Balance Sheet

Institutions Funds

Name	Total Shares (%) ↓	Total Assets (%)	Current Shares	Change	Change (%)
Vanguard Group Inc	8.60	4.48	1,318,859,614.00	795,265.00	0.06
BlackRock Inc	6.79	4.15	1,040,523,406.00	-1,868,402.00	-0.18
Berkshire Hathaway Inc	5.15	40.81	789,368,450.00	-116,191,550.00	-12.83
State Street Corporation	3.84	4.51	588,590,779.00	2,538,722.00	0.43
Geode Capital Management...	2.01	4.87	307,720,852.00	6,898,229.00	2.29
FMR Inc	1.88	3.33	288,979,418.00	-10,891,934.00	-3.63
Morgan Stanley - Brokerage ...	1.45	3.05	221,959,262.00	3,998,035.00	1.83
T. Rowe Price Associates, Inc.	1.35	4.35	206,971,786.00	-3,855,311.00	-1.83
NORGES BANK	1.15	5.93	176,141,203.00	8,766,925.00	5.24
JPMorgan Chase & Co	1.02	2.26	156,058,717.00	18,548,178.00	13.49
Northern Trust Corp	1.01	4.57	154,171,342.00	-7,943,858.00	-4.90
Legal & General Group PLC	1.00	6.17	153,042,666.00	14,581,590.00	10.53

« < 1 2 > » 1 of 2 pages (20 items)

Figura 5.3.3: Acționarii unei companii: instituții.

Pe lângă acestea, pagina curentă oferă și date precum câștigurile, tranzacții din interior, scorurile ESG și altele, dar nu toate acestea sunt disponibile pentru companiile românești.

Referitor la piețele financiare, aplicația oferă informații despre acestea, cum ar fi statusul curent, orele și zilele săptămânii în care piața este deschisă pentru tranzacționare, exact cum putem vedea în **Figura 5.3.4**.

US Exchange	
Information	Holidays
Status	
Status:	Closed
Time Until Next Open:	01:02:20:45
Code:	US
Name:	USA Stocks
Operating MIC:	XNAS, XNYS
Country:	USA
Currency Code:	USD
Time Zone:	America/New_York
Trading Hours	
Open:	9:30 AM
Close:	4:00 PM
Open (UTC):	1:30 PM
Close (UTC):	8:00 PM
Working Days:	Mon, Tue, Wed, Thu, Fri

Figura 5.3.4: Informații despre piața americană.

De asemenea, tot pe aceeași pagină avem și o secțiune unde putem vedea datele calendaristice în care tranzacționarea este închisă conform sărbătorilor naționale din țara respectivă, cu rolul de a oferi investitorilor o cale rapidă de a-și planifica următoarele investiții, așa cum observăm în **Figura 5.3.5**.

US Exchange		
Information	Holidays	
Name	Date	Type
Independence Day	2024-07-04	official
Dr. Martin Luther King Jr's Birthday	2024-01-15	official
Christmas Day	2023-12-25	official
Juneteenth Holiday	2024-06-19	official
Memorial Day	2024-05-27	official
Good Friday	2024-03-29	official
Washington's Birthday	2024-02-19	official
Thanksgiving Day	2024-11-28	official
Labour Day	2024-09-02	official
New Year's Day	2024-01-01	official

Figura 5.3.5: Zilele libere pentru piața americană.

5.4 Portofolii și liste de urmarire

Pe lângă o structurare eficientă, aplicația Investing Wizard dispune și de o gestionare organizată a portofoliilor și a listelor de urmărire. După cum putem vedea în **Figura 5.4.1**, utilizatorul are acces la o pagină specială unde poate adăuga, șterge și edita portofoliile iar în același timp vedea suma estimată totală a dividendelor pe care ar trebui să le plătească toate companiile din portofoliile acestuia. Valoarea respectivă reprezintă totalul brut, afișat în moneda preferată de utilizator.

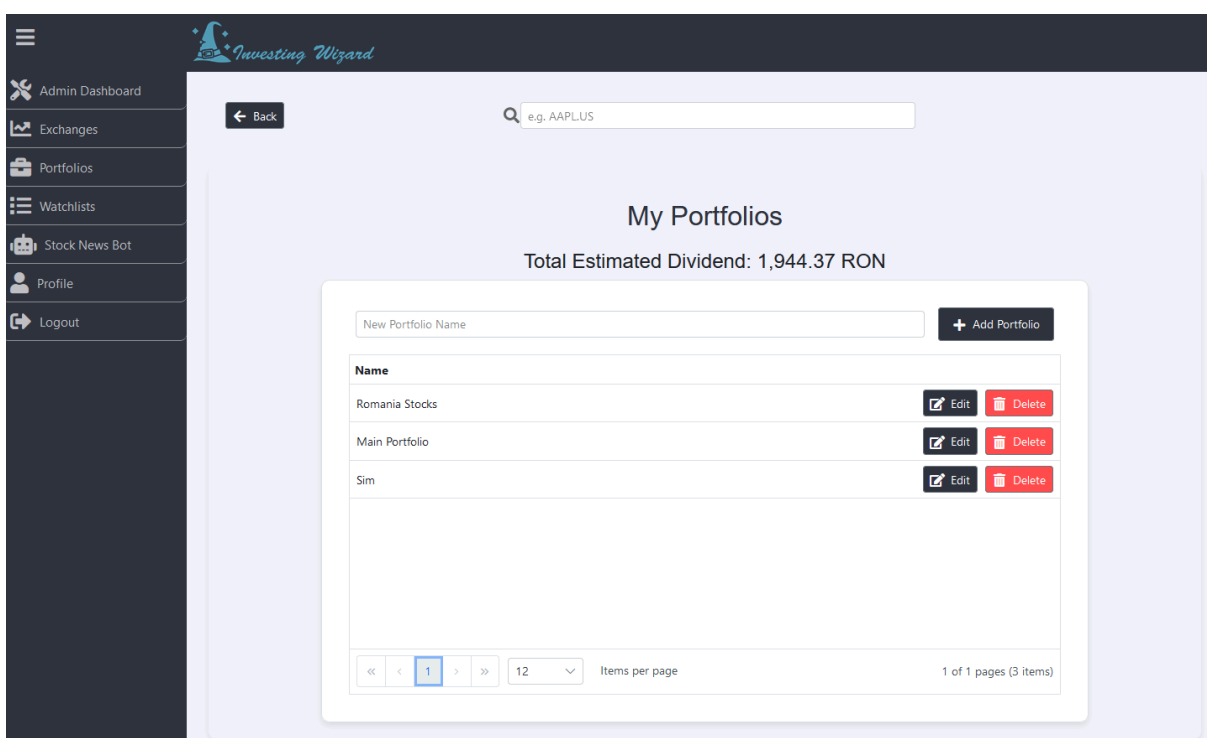


Figura 5.4.1: Pagina de portofolii.

Studiul de piață a arătat clar un avantaj în folosirea tranzacțiilor, așa că am implementat două moduri de prezentare a acestora. În primul rând, pagina expune dividendul total estimat, de data asta doar cel din cadrul portofoliului. Apoi avem înregistrări care conțin numele companiei, numărul total de unități deținute, suma investită total și cea curentă, ambele în moneda în care a fost făcută investiția, prețul curent, profitul total și greutatea companiei în raport cu portofoliul. Toate acestea sunt încapsulate într-un tabel ce suportă paginarea, pentru a face căutarea mai simplă. Aceste înregistrări reprezintă de fapt un total rezultat în urma

tranzacțiilor făcute de utilizator. În **Figura 5.4.2** sunt afișate patru elemente, după modelul celor din studiul de piață.

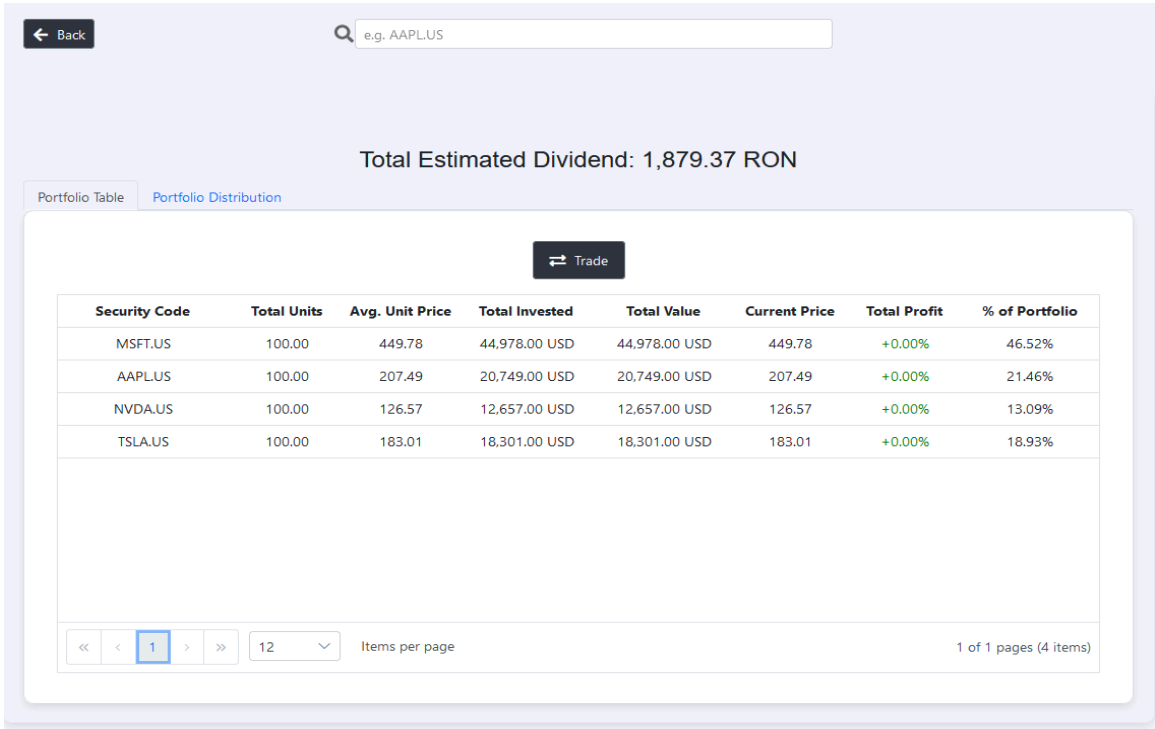


Figura 5.4.2: Pagina unui portofoliu.

Încă o modalitate de afișare a acestei pagini este sub forma distribuției portofoliului, așa cum vedem în **Figura 5.4.3**, un exemplu afișat în format mobile.

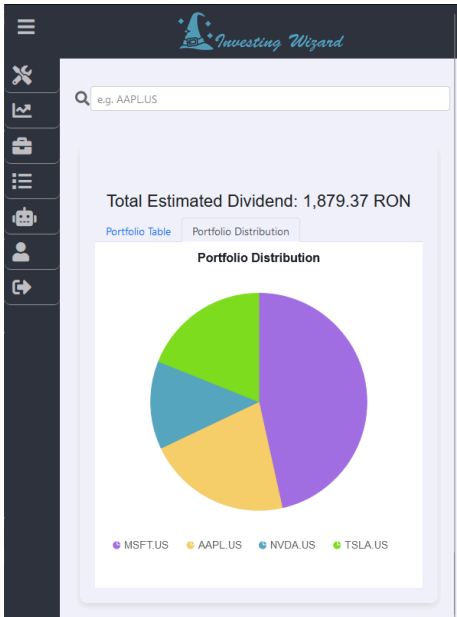


Figura 5.4.3: Pagina de portofolii.

În **Figura 5.4.3** se poate observa un buton „Trade”. Acesta are rolul de a trimite utilizatorul pe pagina de tranzacții a acelui portofoliu, la care am făcut referire anterior, când am menționat studiul de piață.

Security Code

Units

6/23/2024

☐ Broker is Resident

+ Add Transaction

Investment	Date	Units	Unit Price	Invested	Current Amount	Current Price	Profit %	Resident Broker
TSLA.US	23/06/2024	100.00	183.01	18,301.00 USD	18,301.00 USD	183.01 (+0.79%)	+0.00%	False
MSFT.US	23/06/2024	100.00	449.78	44,978.00 USD	44,978.00 USD	449.78 (+0.92%)	+0.00%	False
NVDA.US	23/06/2024	100.00	126.57	12,657.00 USD	12,657.00 USD	126.57 (-3.22%)	+0.00%	False
AAPL.US	23/06/2024	100.00	207.49	20,749.00 USD	20,749.00 USD	207.49 (-1.04%)	+0.00%	False

<<

<

1

>

>>

12

Items per page

1 of 1 pages (4 items)

Activate Optimizer

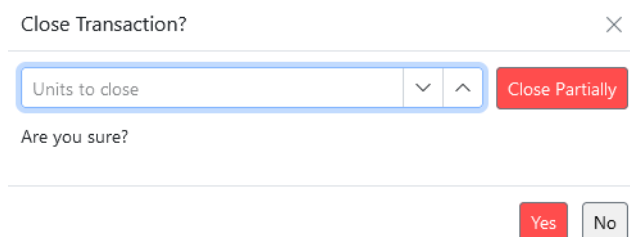
Information

Suggestions

Figura 5.4.4: Pagină de tranzacții.

O pagină de tranzacții este prezentată în **Figura 5.4.4**. O tranzacție conține în principiu aceleași câmpuri ca o înregistrare de pe pagina de portofoliu, dar se adaugă și data la care a fost făcută aceasta și o proprietate prin care se ține cont de țara unde este rezident brokerul. Acest lucru ajută algoritmul de optimizare să poată face referire la normele legale impuse, discutate în prealabil. Diferența majoră dintre aplicația Investing Wizard și cele din studiul de piață constă în pașii de adăugare ai unei tranzacții. În general în cadrul aplicațiilor financiare, un utilizator trebuie să ofere data tranzacției împreună cu prețul și numărul de unități, dar acest lucru poate duce la neconcordanțe. Deoarece aplicația are în baza de date istoricul prețurilor activelor și schimbul valutar dintre moneda de cumpărare și moneda preferată de utilizator, trebuie doar să furnizăm codul companiei, numărul de unități și data tranzacției iar în aplicație se va deschide o tranzacție corespunzătoare prețului de închidere al acțiunilor. Pe lângă asta, se va reține valoarea totală în moneda preferată de utilizator din acel moment, pentru a putea determina în viitor profitul sau pierderea netă cu o precizie ridicată.

Pentru a păstra un mod de lucru cât mai apropiat de parcursul real, o tranzacție nu poate fi modificată, ci doar închisă parțial. În practică, dacă vreau să elimin o tranzacție o închid, iar dacă vreau să modific numărul de unități pot ori să mai deschid o tranzacție, pentru a-l crește, ori să închid parțial tranzacția curentă, pentru a-l scădea. Așadar, funcționalitatea implementată în aplicație apare sub forma unui mesaj de confirmare, atunci când utilizatorul apasă pe butonul de închidere a unei tranzacții. După cum putem vedea în **Figura 5.4.5**, acesta poate decide dacă vrea să închidă complet tranzacția sau doar parțial.



Close Transaction? ×

Units to close ▼ ▲ Close Partially

Are you sure?

Yes No

Figura 5.4.5: Confirmarea închiderii unei tranzacții.

5.4.1 Optimizarea tranzacțiilor

Dacă utilizatorul alege opțiunea de a activa optimizatorul, atunci de fiecare dată când închidem o tranzacție vom primi diverse informații și sugestii. În **Figura 5.4.6** avem o pagină de tranzacții diversificată.

Security Code

Units

6/23/2024

☐ Broker is Resident

+ Add Transaction

Investment	Date	Units	Unit Price	Invested	Current Amount	Current Price	Profit %	Resident Broker	
TSLA.US	15/11/2023	10,000.00	242.84	2,428,400.00 USD	1,830,100.00 USD	183.01 (+0.79%)	-24.64%	False	<div>✕</div>
TSLA.US	15/11/2023	15,000.00	242.84	3,642,600.00 USD	2,745,150.00 USD	183.01 (+0.79%)	-24.64%	True	<div>✕</div>
TSLA.US	15/11/2023	100.00	242.84	24,284.00 USD	18,301.00 USD	183.01 (+0.79%)	-24.64%	False	<div>✕</div>
AAPL.US	15/06/2023	5,000.00	185.03	925,137.00 USD	1,037,450.00 USD	207.49 (-1.04%)	+12.14%	False	<div>✕</div>
BVB.RO	04/06/2024	1,000.00	64.00	64,000.00 RON	64,400.00 RON	64.40 (+0.63%)	+0.63%	True	<div>✕</div>
WIN.RO	18/06/2020	1,000.00	11.00	11,000.00 RON	15,620.00 RON	15.62 (+1.30%)	+42.00%	True	<div>✕</div>

«

<

1

>

»

12

Items per page

1 of 1 pages (6 items)

Activate Optimizer

Information

Suggestions

Figura 5.4.6: Pagină de tranzacții diversificată.

Putem observa tranzacții cu acțiuni americane, acțiuni românești, realizate atât la brokeri rezidenți în România cât și la brokeri străini. Acesta este un scenariu de utilizare practic, în care un utilizator își poate monitoriza într-un singur loc investițiile făcute pe mai multe platforme. Am presupus că utilizatorul vrea să vândă în profit acțiunile la compania Apple, atunci optimizatorul va oferi informații despre închiderea acestei tranzacții, ca în **Figura 5.4.7**.

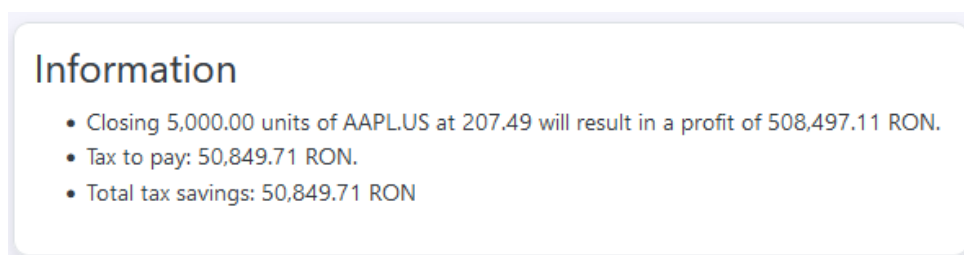


Figura 5.4.7: Informații despre închiderea unei tranzacții.

Putem remarca faptul că algoritmul calculează profitul net în lei, indiferent de moneda în care a fost făcută tranzacția, folosind schimbul valutar datei tranzacției împreună cu schimbul valutar curent, astfel investitorul român nu va trebui să se preocupe de acest aspect.

În **Figura 5.4.8** observăm și sugestiile pe care aplicația ni le oferă cu scopul de a plăti cât mai puține taxe. Algoritmul ajunge la această sugestie deși există trei tranzacții în pierdere ce pot servi drept candidați. Acest lucru se întâmplă deoarece a doua tranzacție este făcută la un broker rezident român, ceea ce nu permite deducerea de pierderi iar a treia tranzacție este semnificativ mai mică decât prima și nu acoperă nici pe departe profitul încasat, ceea ce înseamnă că oricum ar trebui făcute două tranzacții. Algoritmul este conceput în manieră greedy, să execute cât mai puține tranzacții, cu rolul de a minimiza costurile adiționale și taxele plătite brokerilor.

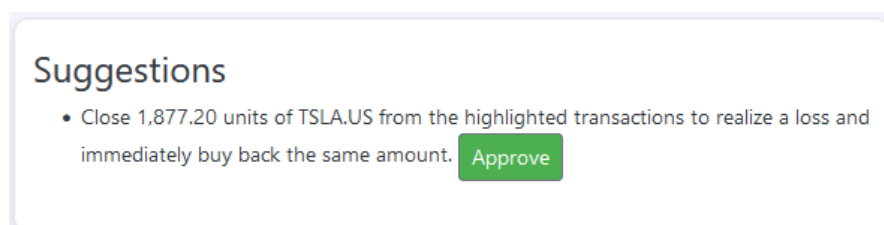


Figura 5.4.8: Sugestii oferite de algoritmul de optimizare.

Utilizatorul are posibilitatea de a aproba sugestii, în acest caz aplicația va executa mișcările corespunzătoare. În **Figura 5.4.9** se poate vedea cum arată pagina completă exact după închiderea unei tranzacții și primirea informațiilor și sugestiilor de la algoritmul de optimizare. Pe ecran vor fi marcate cu galben tranzacțiile care sunt sugerate de optimizator pentru a fi închise, astfel încât utilizatorul să nu fie indus în eroare în cazul în care există două sau mai multe tranzacții identice sau cu mici diferențe.

Investment	Date	Units	Unit Price	Invested	Current Amount	Current Price	Profit %	Resident Broker	
WINE.RO	18/06/2020	1,000.00	11.00	11,000.00 RON	15,620.00 RON	15.62 (+1.30%)	+42.00%	True	×
BVB.RO	04/06/2024	1,000.00	64.00	64,000.00 RON	64,400.00 RON	64.40 (+0.63%)	+0.63%	True	×
TSLA.US	15/11/2023	100.00	242.84	24,284.00 USD	18,301.00 USD	183.01 (+0.79%)	-24.64%	False	×
TSLA.US	15/11/2023	15,000.00	242.84	3,642,600.00 USD	2,745,150.00 USD	183.01 (+0.79%)	-24.64%	True	×
TSLA.US	15/11/2023	10,000.00	242.84	2,428,400.00 USD	1,830,100.00 USD	183.01 (+0.79%)	-24.64%	False	×

« < 1 > » 12 Items per page 1 of 1 pages (5 items)

Activate Optimizer ☒

Information

- Closing 5,000.00 units of AAPL.US at 207.49 will result in a profit of 508,497.11 RON.
- Tax to pay: 50,849.71 RON.
- Total tax savings: 50,849.71 RON

Suggestions

- Close 1,877.20 units of TSLA.US from the highlighted transactions to realize a loss and immediately buy back the same amount. Approve

Figura 5.4.9: Scenariu practic ce folosește optimizatorul.

Dacă de exemplu vreau să închid o tranzacție de genul celei care vizează WINE.RO, companie românească, tranzacție realizată cu ajutorul unui broker rezident sau cu sediul în România, atunci vom fi informați despre taxa pe care ar trebui să o plătim, de asemenea în moneda preferată de noi, în funcție de diverși factori. Un exemplu de închidere a unei tranzacții mai vechi de un an este afișat în **Figura 5.4.10**.

Information

- Closing 1,000.00 units of WINE.RO at 15.62 will result in a profit of 4,620.00 RON. You should be taxed at source with 1%, which means 46.20 RON.

Figura 5.4.10: Informații despre închiderea altui tip de tranzacții.

5.5 Accesul rapid la știri importante

După cum am prezentat în capitolul trecut, această funcționalitate îmbină două API-uri de inteligență artificială cu scopul de a afișa un rezultat cât mai uman. În **Figura 5.5.1** putem observa chat-ul în sine, în partea stângă, unde asistentul va interpreta știrile pe care le găsește iar în partea dreaptă link-uri către toate site-urile de unde au fost preluate, în scopul de a putea continua o analiză calitativă amănunțită.

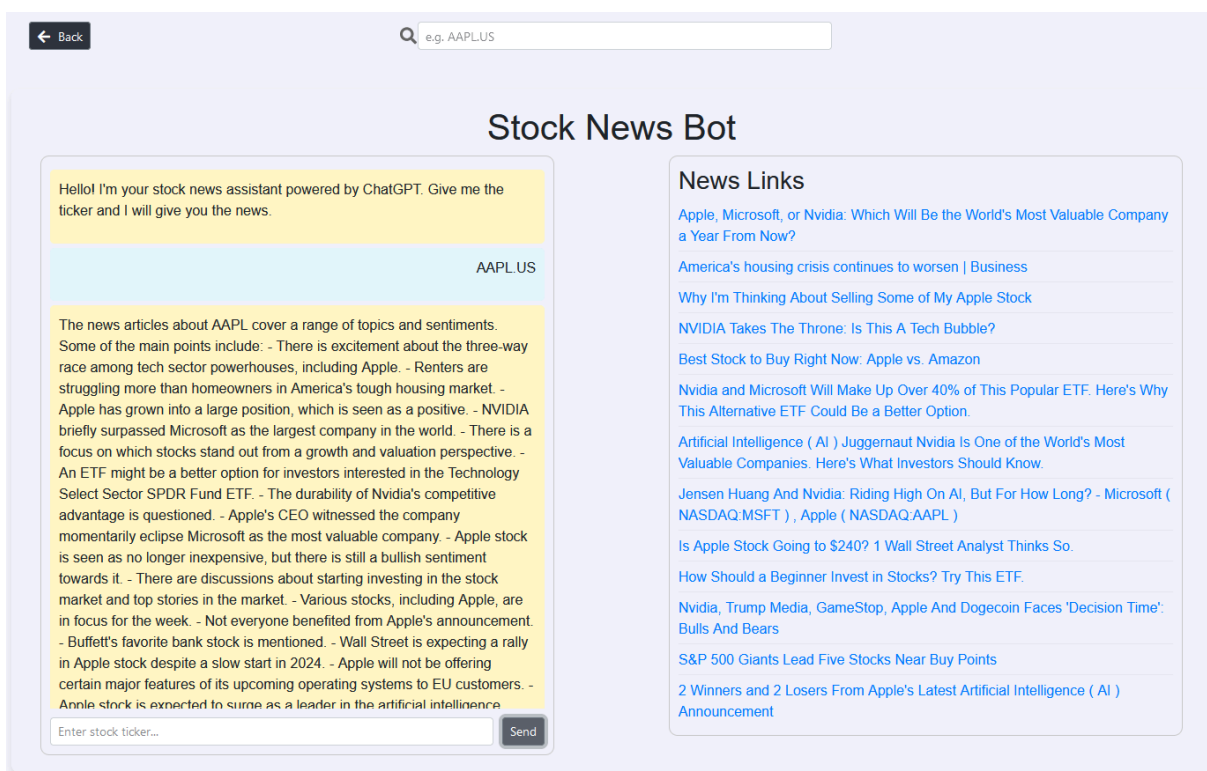


Figura 5.5.1: Asistent virtual pentru știri.

5.6 Concluzii

În concluzie, aplicația Investing Wizard reușește să reducă la o structură asemănătoare companiile românești și străine, oferind o soluție complexă de a analiză cantitativă, poate gestiona diverse tranzacții făcute cu ajutorul unor brokeri diferiți și conferă investitorilor metode inteligente de a gestiona taxele și dividendele. Pe lângă aceste lucruri, utilizatorii dispun și de analiză calitativă, prin accesul în timp competitiv la cele mai recente știri și sentimente ale pieței și alți indicatori.

Concluzii

În cadrul studiului de piață am luat în considerare unele dintre cele mai bune aplicații financiare la ora actuală, reușind să evidențiez punctele tari și slabe pentru fiecare dintre acestea. În acest parcurs am dedus că există niște caracteristici prezente în aplicațiile respective care le fac să iasă în evidență. Combinând aceste funcționalități am reușit să aduc piața financiară din România într-un punct destul de apropiat de piețele mai dezvoltate, păstrând o structură standard pentru toate entitățile. Am reușit această performanță utilizând API-uri externe pentru prețuri în timp real și istoric, care folosesc 30 de surse diferite și API-uri externe pentru date fundamentale care folosesc peste 15 furnizori pentru a livra date de calitate. Aplicația simulează scenarii cât mai apropiate de realitate, fiind gândită să se conformeze cu legislația în vigoare, astfel încât experiența investitorului care folosește aplicația să fie un bonus, nu o necesitate.

Arhitectura aplicației noastre ASP.NET Core, un monolit gândit pe baza arhitecturilor folosite pe termen lung în aplicații la scară largă, este un punct de plecare excelent deoarece costurile sunt mici, complexitatea este destul de simplă dar performanța este înaltă. Îmbinând principii din DDD și Clean Architecture cu pachete din cadrul ecosistemului precum MediatR, AutoMapper și alte servicii personalizate, am reușit să creez componente independente, stabile, organizate și ușor de întreținut în puține linii de cod. Viteza noilor tehnologii a fost la același nivel cu performanța bazelor de date PostgreSQL configurate și expuse cu ajutorul EFCore. Aplicația se folosește de mecanismul de caching pe care îl are la dispoziție pentru a salva în memorie prețurile în timp real, pentru o performanță ridicată și o experiență mai bună a utilizatorului. Tot în această direcție, am reușit să implementez un background job cu ajutorul bibliotecii Quartz, rolul acestuia fiind de a stoca ultimele prețuri din cache ca prețuri istorice, la sfârșit de zi. Aceste lucruri au un impact asupra costurilor totale, deoarece sunt gândite special pentru a reduce semnificativ numărul de apeluri către API și de asemenea implică o administrare locală a prețurilor istorice, astfel încât în viitorul apropiat să nu mai fie nevoie de o sursă externă.

Alegerea de a construi aplicația ca un Single Page Application în Blazor, folosind componente Syncfusion pe partea de prezentare a contribuit major atât la crearea unei interfețe moderne ce poate afișa în mod pragmatic datele de care investitorul are nevoie cât și la experiența utilizatorului, fiind o aplicație interactivă.

Deoarece aplicația prezintă o funcționalitate pe bază de inteligență artificială avansată, folosind API-urile de la Alpha Vantage și OpenAI, pentru a crea o componentă asistată de ChatGPT, putem spune că se încadrează în tendințele curente și arată promițător pentru viitor.

Investing Wizard reușește să cuprindă suficiente funcționalități încât să fie considerată o aplicație complexă, de sine stătătoare, care să poată să fie folosită exclusiv de un investitor român pentru a reuși să facă o analiză completă a unei companii.

Directii viitoare

Un prim lucru de luat în considerare în vederea dezvoltării este după părerea mea expunerea aplicației către public într-o variantă demo. Putem numi acest lucru partea a doua a studiului de piață, după care putem modela aplicația, ajungând astfel la o primă versiune completă în vederea utilizatorilor.

Pe lângă asta, colaborarea cu experți atât din domeniul financiar cât și din cel tehnic ar putea aduce valoare în crearea de noi funcționalități ușor de integrat în aplicație. Spre exemplu framework-ul ML.NET⁵⁰, ce se așteaptă să fie îmbunătățit în versiunile următoare, poate fi o opțiune bună în dezvoltarea unui model.

Acesta nu trebuie neapărat să fie foarte complex, dar poate fi de impact, având la dispoziție seturile de date provenite de la API-urile externe și posibilitatea de a simula portofoliile. Spre exemplu, dacă putem crea un model de ML care să fie antrenat pe date de acum 20 de ani și testat pe date de acum 10 ani, putem stabili într-o oarecare măsură dacă se pot găsi tipare în ciclurile pieței, folosind diverși indicatori, cum ar fi de exemplu ratele dobânzilor istorice și curente. Deși nu putem demonstra că acțiunile trecute influențează viitorul, nu se știe cum va evolua inteligența artificială și care poate fi plafonul în această direcție.

Pe plan funcțional, cea mai bună metodă de dezvoltare cred că o reprezintă parteneriate cu diverși brokeri, astfel încât utilizatorul să poată importa detaliile despre tranzacții din aplicația brokerului, direct în aplicația Investing Wizard. Acest lucru ar putea automatiza și adăugarea de tranzacții noi, cu o vedere mai precisă a tipului de broker cu ajutorul căruia se realizează. Dacă

⁵⁰ <https://learn.microsoft.com/en-us/dotnet/core/whats-new/dotnet-9/overview>

aplicația reușește să combine cu succes cele mai bune tehnologii existente în vederea adunării câtor mai multe informații și prelucrarea inteligentă a acestora, singurul lucru rămas este scalabilitatea. Cred că scalabilitatea în legătură cu aplicația Investing Wizard poate fi realizată în mai multe direcții. Cum entitățile din baza de date sunt aproape decuplate, deoarece sunt preluate cu ajutorul API-urilor externe, se poate migra într-un mod rezonabil la o arhitectură bazată pe microservicii, implementând un sistem de mesagerie.

Tot în următoarele versiuni de .NET se va introduce o metodă de caching hibridă⁵¹, ce cu siguranță ar putea îmbunătăți formatul actual, care folosește doar memoria serverului web, pentru rapiditate. Combinarea a diverse metode de caching distribuit, ca de exemplu Redis⁵², cu noile tehnologii .NET va putea susține consistent un volum mult mai mare decât cel posibil în varianta dată. De altfel, folosirea în paralel a mai multor tipuri de baze de date sau fragmentarea modelului curent, poate contribui la scalarea într-o anumită direcție a aplicației.

Într-un final, cea mai ambițioasă privire de ansamblu ar fi oferirea mai multor tipuri de soluții, personalizate. De exemplu aplicația Investing Wizard s-ar putea folosi așa cum este de către utilizatori, sau ar putea fi configurată o soluție care să folosească doar runtime-ul personalizat iar datele financiare să fie administrate de către client.

Consider că aplicația poate aduce valoare și în forma în care este, dar poate fi și semnificativ îmbunătățită, iar tehnologiile viitoare ar putea face aceste extinderi ambițioase să fie posibile, într-un mod sau altul.

⁵¹ <https://learn.microsoft.com/en-us/aspnet/core/performance/caching/hybrid?view=aspnetcore-9.0>

⁵² <https://redis.io/learn/develop/dotnet>

Bibliografie

- [1] Evans, Eric. 2004. *Domain-driven design: tackling complexity in the heart of software*.
- [2] Gamma, E., R. Helm, R. Johnson, and J. Vlissides. *Elements of reusable object-oriented software. Design Patterns*.
- [3] Martin, Robert C. 2000. *Design principles and design patterns*.
- [4] Martin, Robert C. 2017. *Clean architecture*.
- [5] Dahan, Udi. 2009. "Clarified CQRS." <http://www.udidahan.com/2009/12/09/clarified-cqrs>.

Clean Architecture: <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>

Domain-Driven Design: <https://martinfowler.com/bliki/DomainDrivenDesign.html>

ASP.NET Core: <https://learn.microsoft.com/en-us/aspnet/core/>

[introduction-to-aspnet-core?view=aspnetcore-8.0](https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-8.0)

.NET: <https://learn.microsoft.com/en-us/dotnet/core/introduction>

NuGet: <https://learn.microsoft.com/en-us/nuget/what-is-nuget>

ASP.NET Core Identity: <https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-8.0&tabs=visual-studio>

Identity in Single Page Applications: <https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity-api-authorization?view=aspnetcore-8.0>

Blazor: <https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-8.0>

Blazor hosting: <https://learn.microsoft.com/en-us/aspnet/core/blazor/hosting-models?view=aspnetcore-8.0>

Funcționalități noi în ASP.NET Core 8: <https://learn.microsoft.com/en-us/aspnet/core/release-notes/aspnetcore-8.0?view=aspnetcore-8.0>

Entity Framework Core: <https://learn.microsoft.com/en-us/ef/core/>

HttpClient: <https://learn.microsoft.com/en-us/dotnet/fundamentals/networking/http/httpclient>

Caching: <https://learn.microsoft.com/en-us/aspnet/core/performance/caching/memory?view=aspnetcore-8.0>
<https://learn.microsoft.com/en-us/aspnet/core/performance/caching/hybrid?view=aspnetcore-9.0>

Dependency Injection: <https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection>

ASP.NET Core API: <https://learn.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-3.1>

.NET Transactions: <https://www.bytehide.com/blog/dotnet-transactions>

.Net 9: <https://learn.microsoft.com/en-us/dotnet/core/whats-new/dotnet-9/overview>

Database Seeding: <https://learn.microsoft.com/en-us/ef/core/modeling/data-seeding>

Antiforgery in ASP.NET Core: <https://learn.microsoft.com/en-us/aspnet/core/security/anti-request-forgery?view=aspnetcore-8.0>

Tax-Loss Harvesting: https://www.gsam.com/content/dam/gsam/pdfs/us/en/articles/2021/FAQ_Tax_Loss_Harvesting_Strategies.pdf?sa=n&rd=n

Wash-Sale Rule: https://apps.irs.gov/app/vita/content/10s/10_04_011.jsp?level=advanced

Patterns:

Mediator Pattern: <https://www.dofactory.com/net/mediator-design-pattern>

Repository Pattern și Unit of Work Pattern:

<https://learn.microsoft.com/en-us/dotnet/api/system.data.entity.dbcontext?view=entity-framework-6.2.0>

<https://learn.microsoft.com/en-us/dotnet/api/system.data.entity.dbset-1?view=entity-framework-6.2.0>

https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design?trk=article-ssr-frontend-pulse_little-text-block

Factory Method Pattern: https://www.tutorialspoint.com/design_pattern/factory_pattern.htm

Result Pattern: <https://www.milanjovanovic.tech/blog/functional-error-handling-in-dotnet-with-the-result-pattern>

Aplicații:

TrackYourDividends: <https://trackyourdividends.com/>

DividendWatch: <https://dividend.watch>

Yahoo Finance: <https://finance.yahoo.com/>

Bursa de Valori București: <https://bvb.ro/>

Legislație:

<https://legislatie.just.ro/Public/DetaliiDocument/255605>

https://static.anaf.ro/static/10/Anaf/legislatie/Cod_fiscal_norme_2023.htm

Alte tehnologii utilizate:

PostgreSQL: <https://www.postgresql.org/about/>

AutoMapper: <https://docs.automapper.org/en/stable/Getting-started.html#what-is-automapper>

MediatR: <https://www.jimmybogard.com/sharing-context-in-mediatr-pipelines/>

Serilog: <https://serilog.net/>

Quartz.NET: <https://www.quartz-scheduler.net/>

Twilio SendGrid: <https://sendgrid.com/en-us/solutions/email-api>

EODHD: <https://eodhd.com/financial-apis/quick-start-with-our-financial-data-apis>

Exchanges API: <https://eodhd.com/financial-apis/exchanges-api-trading-hours-and-stock-market-holidays>

Fundamental Data API: <https://eodhd.com/financial-apis/stock-etfs-fundamental-data-feeds>

End Of Day API: <https://eodhd.com/financial-apis/api-for-historical-data-and-volumes>

Live Stock Prices API: <https://eodhd.com/financial-apis/live-realtime-stocks-api>

OpenAI API: <https://platform.openai.com/docs/quickstart>

Alpha Vantage API: <https://www.alphavantage.co/documentation/>

Syncfusion: <https://www.syncfusion.com/company/about-us>

TimeZoneConverter: <https://www.nuget.org/packages/timezoneconverter/>

QRCoder: <https://www.nuget.org/packages/QRCoder/>

Bootstrap: <https://getbootstrap.com/docs/5.0/getting-started/introduction/>

Referințe suplimentare:

Railway Oriented Programming: <https://fsharpforfunandprofit.com/rop/>

CQRS: <https://martinfowler.com/bliki/CQRS.html>

OWASP Top 10: <https://owasp.org/www-project-top-ten/>

Atac DDoS: <https://www.microsoft.com/ro-ro/security/business/security-101/what-is-a-ddos-attack>

Redis: <https://redis.io/learn/develop/dotnet>