

**Academia de Studii Economice din București**

**Facultatea de Cibernetică, Statistică și Informatică Economică**

**Specializarea Informatică Economică**

**LUCRARE DE LICENȚĂ**

**Platformă pentru predicție și tranzacționare a investițiilor**

**Coordonator științific**

Conf. universitar Toma Cristian-Valeriu

**Absolvent**

Andrei Nicușor-Andrei

Cuprins

[Introducere 4](#_Toc138181227)

[Capitolul I: Descrierea problemei economice 5](#_Toc138181228)

[1.1. Prezentarea firmei sau domeniului abordat 5](#_Toc138181229)

[1.2. Prezentarea activităţii care va fi informatizată 5](#_Toc138181230)

[1.3. Analiza sistemelor existente pe piață 6](#_Toc138181231)

[Capitolul II – Tehnologii folosite în aplicația informatică 9](#_Toc138181232)

[2.1 Machine Learning 9](#_Toc138181233)

[2.2 Rețelele neuronale și deep learning 10](#_Toc138181234)

[2.1.1 Rețele neuronale 10](#_Toc138181235)

[2.1.2 Deep Learning 11](#_Toc138181236)

[2.1.3 Avantajele Deep Learning 12](#_Toc138181237)

[2.3 Arhitectura LSTM 13](#_Toc138181238)

[2.4 Rest Api 14](#_Toc138181239)

[2.5 Docker 16](#_Toc138181240)

[2.6 React 17](#_Toc138181241)

[Capitolul III – Arhitectura aplicației 19](#_Toc138181242)

[3.1 Modelul matematic și bonitatea pentru Apple 19](#_Toc138181243)

[3.1.1 Modelul matematic al modelului 19](#_Toc138181244)

[3.1.2 Bonitatea modelului antrenat 21](#_Toc138181245)

[3.2. Diagrama Arhitecturii 22](#_Toc138181246)

[3.3 Baza de date 23](#_Toc138181247)

[3.4 Diagrama cazurilor de utilizare 26](#_Toc138181248)

[Capitolul IV – Implementearea aplicației 29](#_Toc138181249)

[4.1 Antrenarea modelelor de inteligență artificială 29](#_Toc138181250)

[4.2 Server Python 30](#_Toc138181251)

[4.3 Server Java 31](#_Toc138181252)

[4.4 Interfața grafică a aplicației 33](#_Toc138181253)

[Bibliografie 33](#_Toc138181254)

# Introducere

În era digitală, tehnologia a devenit un instrument esențial în majoritatea domeniilor, inclusiv în sectorul financiar și în tranzacționarea pe piața de investiții. Cu ajutorul inteligenței artificiale s-au dezvoltat numeroase metode de analiză și previziune a prețurilor acțiunilor, facilitând astfel alegerea deciziilor eficiente de către investitori.

Obiectivul acestei lucrări de licență este de a proiecta și implementa o aplicație web în domeniul financiar care să permită utilizatorilor să analizeze și să tranzacționeze acțiuni la bursă, folosind un algoritm de inteligență artificială pentru a realiza predicții și a facilita tranzacționarea automată. Scopul acestui proiect este de a oferi o soluție eficientă și sigură pentru investitori în analiza și tranzacționarea acțiunilor.

Aplicația informatică ajută utilizatorii în deciziile legate de piața acțiunilor prin predicțiile furnizate de modelele de inteligență artificială pentru fiecare acțiune disponibilă, prin datele analizei fundamentale, dar și prin algoritmul de tranzacționare automată care se folosește de predicțiile algoritmului pentru a maximiza profitul.

Aplicația a fost construită folosind cele mai moderne tehnologii. Interfața grafică a fost construită folosind React, cea mai populară bibliotecă a limbajului JavaScript^1. Pentru server a fost folosită tehnologia Spring Boot datorită multitudinilor de beneficii puse la dispoziție atât de framework cât și de limbajul Java. Pentru dezvoltarea, antrenarea și testarea rețelelor neuronale a fost folosită librăria tensorflow din limbajul Python.

Lucrarea de licență este structurată în 2 capitole urmate de concluziile finale. Pe parcursul lucrării urmează să se abordeze diverse aspecte ale dezvoltării și implementării aplicației web, precum și dezvoltarea, antrenarea și testarea rețelelor neuronale.

# Capitolul I: Descrierea problemei economice

## Prezentarea firmei sau domeniului abordat

Sectorul financiar a fost întotdeauna o componentă vitală a economiei globale, făcând posibilă circulația capitalului și facilitând creșterea economică. În ultimele decenii, cu progresul rapid al tehnologiei informației, acest domeniu a suferit transformări semnificative, cu tehnologia digitală care a devenit din ce în ce mai încorporată în operațiunile sale. Acest proces de digitalizare a permis sectorului financiar să devină mai eficient, mai transparent și mai accesibil pentru un public larg.

Un domeniu specific în care tehnologia și finanțele s-au intersectat în mod remarcabil este piața de acțiuni. Această piață a funcționat timp de secole ca un loc pentru companii să strângă capital și pentru investitori să realizeze profituri. Cu toate acestea, în era modernă, tranzacționarea de acțiuni a devenit mult mai sofisticată, implicând o multitudine de factori și o cantitate imensă de date care trebuie analizate.

Introducerea inteligenței artificiale în acest domeniu a schimbat în mod semnificativ modul în care funcționează piața de acțiuni. Algoritmii de inteligență artificială, în special cei bazati pe învățare automată, sunt acum capabili să analizeze datele de pe piața de acțiuni la o scară și cu o acuratețe pe care oamenii simpli nu o pot realiza. Acești algoritmi pot analiza mișcările de preț ale acțiunilor, tendințele pieței, rapoartele financiare ale companiilor și multe altele, pentru a face predicții precise despre comportamentul viitor al prețurilor acțiunilor.

Cu toate acestea, în ciuda progresului semnificativ realizat în acest domeniu, există încă loc pentru îmbunătățiri și inovație. În special, există o nevoie de o platformă care să integreze analiza, predicția și tranzacționarea acțiunilor într-un singur sistem eficient și sigur.

## 1.2. Prezentarea activităţii care va fi informatizată

Obiectivul acestei lucrări este de a răspunde la această nevoie, proiectând și implementând o platformă care să integreze trei componente esențiale ale tranzacționării de acțiuni: analiza acțiunilor, predicția comportamentului lor viitor și tranzacționarea automată.

Analiza acțiunilor este un proces fundamental în tranzacționarea de acțiuni. Înainte de a lua decizia de a cumpăra sau vinde o acțiune, un investitor trebuie să înțeleagă în primul rând valoarea reală a acțiunii respective. Acest lucru implică analiza a diverse aspecte ale companiei respective, inclusiv performanța financiară, poziția pe piață, managementul și strategia, precum și factori externi, cum ar fi condițiile economice generale și trendurile pieței.

Predicția comportamentului acțiunilor implică utilizarea de modele de inteligență artificială pentru a anticipa mișcările viitoare de preț pe baza datelor istorice și actuale. În ultimii ani, algoritmii de învățare automată și în special rețelele neuronale profunde s-au dovedit a fi extrem de eficiente în această sarcină. Aceste algoritme pot analiza și învăța dintr-o cantitate mare de date și pot identifica modele și tendințe care sunt adesea greu de văzut de ochiul uman.

Tranzacționarea automată implică folosirea unui algoritm pentru a efectua tranzacții pe baza unor reguli predefinite sau pe baza predicțiilor realizate de un model de inteligență artificială. Acesta este un domeniu de tranzacționare de acțiuni în creștere rapidă, deoarece permite investitorilor să maximizeze profiturile și să minimizeze pierderile în mod automat, fără a fi nevoie să urmărească în mod constant piața.

## 1.3. Analiza sistemelor existente pe piață

În cadrul acestui subcapitol, vom analiza trei dintre cele mai proeminente soluții de pe piața actuală în ceea ce privește tranzacționarea și predicția pe piața de acțiuni. Acestea includ servicii oferite de companii recunoscute din domeniu precum "AlgoTrader", "Trade Ideas" și "Upstox".

Vom examina în detaliu capacitățile și funcționalitățile fiecărui sistem, analizând modul în care acestea utilizează tehnologia și inteligența artificială pentru a ajuta utilizatorii să facă predicții și să tranzacționeze acțiuni. De asemenea, vom discuta despre avantajele și dezavantajele acestor sisteme, luând în considerare aspecte precum precizia predicțiilor, ușurința de utilizare, securitatea și costul.

Prin această analiză, vom încerca să identificăm lacunele existente în soluțiile actuale și vom vedea cum propunerea noastră pentru o platformă integrată de tranzacționare și predicție a acțiunilor poate îmbunătăți starea actuală a lucrurilor.

1. AlgoTrader

AlgoTrader este un produs avansat de software destinat tranzacționării algoritmice. Această platformă furnizează un set complet de funcționalități care permit firmelor de tranzacționare cuantitative și tranzacțiilor de arbitraj să realizeze strategii de tranzacționare sofisticate și complexe fără a fi limitate de viteză sau volum. În plus, AlgoTrader utilizează inteligența artificială pentru a procesa și a analiza datele pieței în timp real, ceea ce permite tranzacțiilor algoritmice să fie efectuate cu o precizie și rapiditate incredibile. Cu toate acestea, complexitatea sa poate fi un dezavantaj pentru traderii novici sau cei care nu sunt familiarizați cu tranzacționarea algoritmica.

1. Trade Ideas

Trade Ideas este un serviciu online care folosește inteligența artificială pentru a genera idei de tranzacționare pentru acțiuni. Serviciul folosește algoritmi de inteligență artificială pentru a scana piețele de acțiuni în timp real și a identifica tendințe și modele care pot indica oportunități de tranzacționare. În plus, platforma Trade Ideas include un set de instrumente de analiză și vizualizare care permit utilizatorilor să analizeze detaliat ideile de tranzacționare generate. Cu toate acestea, în timp ce Trade Ideas este excelent pentru a genera idei de tranzacționare, el nu oferă nicio funcționalitate pentru a executa efectiv tranzacțiile.

1. Upstox

Upstox este un broker online care oferă o platformă simplă și intuitivă pentru tranzacționarea de acțiuni. În plus, Upstox utilizează inteligența artificială pentru a oferi predicții de piață utilizatorilor săi, permițându-le să ia decizii de tranzacționare mai bine informate. Deși platforma este ușor de utilizat și include o serie de instrumente de analiză și vizualizare, nivelul de precizie și complexitate al predicțiilor bazate pe IA nu se compară cu cel al platformelor mai avansate precum AlgoTrader sau Trade Ideas.

În încheiere, fiecare dintre aceste platforme oferă o combinație unică de funcționalități și prezintă propriile sale avantaje și dezavantaje. Cu toate acestea, există o oportunitate clară pentru o soluție care combină în mod eficient funcționalitățile de predicție și tranzacționare, în timp ce rămâne accesibilă și ușor de folosit pentru utilizatorii de toate nivelurile de experiență. Acesta este spațiul pe care dorim să-l acoperim cu platforma noastră de tranzacționare și predicție a acțiunilor bazată pe inteligență artificială.

# Capitolul II – Tehnologii folosite în aplicația informatică

## 2.1 Machine Learning

Machine learning este o categorie a inteligenței artificiale care se concentrează pe dezvoltarea de modele capabile să învețe și să își îmbunătățească performanțele pe baza unor date de intrare, fără a necesita programarea explicită a algoritmului care rezolvă problema studiată. Acest proces se bazează pe recunoașterea și extragerea modelelor ascunse în datele furnizate, permițând sistemului să învețe și să realizeze predicții pe baza datelor noi introduse.

În machine learning se folosesc diverse tehnici pentru abordarea diferitelor probleme precum: învățarea supervizată, învățarea nesupervizată, învățarea semi-supervizată și învățarea prin întărire (reinforcement learning). În funcție de tipul de problemă abordată, algoritmii pot fi clasificați în regresie, clasificare, clustering și reducerea dimensionalității. Unii dintre cei mai cunoscuți algoritmi de machine learning includ rețele neuronale, arborii de decizie, mașini vector suport (SVM) și algoritmi de grupare.

Machine learning-ul joacă un rol semnificativ în diverse domenii, printre care și în domeniul financiar studiat în prezenta lucrare. Acesta oferă unelte și tehnici pentru a aborda probleme complexe și pentru a procesa cantități mari de date. În domeniul financiar, machine learning-ul poate fi utilizat în diferite moduri, cum ar fi:

* Analiza sentimentală - putem evalua sentimentul general al pieței prin analiza datelor provenite din surse precum știri, postări pe social media și rapoarte financiare, pentru a prezice impactul asupra prețurilor acțiunilor.
* Automatizarea tranzacționării - utilizând algoritmi de tranzacționare automată, putem procesa și executa tranzacții în timp real, algoritmul adaptându-se la condițiile pieței și profitând de oportunitățile identificate.
* Recomandări de investiții - sistemele de învățare automată pot oferi sugestii personalizate de investiții, analizând portofoliul investitorului, istoricul tranzacțiilor, dar și preferințele individuale.

Pentru a rezolva regresia studiată în lucrare, am abordat problema folosind tehnica deep learning, prezentată detaliat în capitolul următor, care se folosește la bază de metoda învățării supervizate. Învățarea supervizată este o abordare folosită în machine learning în care algoritmul este antrenat pe un set de date etichetat. Setul de date etichetat conține pentru fiecare exemplu de date de intrare valoarea corespunzătoare a datelor de ieșire sub formă de etichete pentru algoritmii de clasificare sau sub formă de valori pentru algoritmii de regresie. Scopul învățării supervizate este ca algoritmul să învețe să recunoască modelele și relațiile dintre datele de intrare și etichetele corespunzătoare, astfel încât să poată face predicții corecte pentru datele noi care nu sunt etichetate.

Pentru modelarea algoritmilor de machine learning folosiți în regresiile din lucrare, am decis să folosesc ca parametrii de intrare date elemenetare din piață legate de fiecare acțiune în parte precum: prețul de deschidere și de închidere al acțiunii, cel mai mare și cel mai mic preț și volumul zilnic al tranzacțiilor. Parametrul de ieșire este format dintr-un vector de 7 elemente care reprezintă prețul de închidere al acțiunii studiate de model pentru următoarele 7 zile. Pentru setul de intrare, datele au fost preluate în interval zilnic printr-un API extern care furnizează date actualizate legate de piață alături de multe alte informații folosite în aplicația finală.

## 2.2 Rețelele neuronale și deep learning

### 2.1.1 Rețele neuronale

Rețelele neuronale sunt sisteme de calcul inspirate din modul în care funcționează creierul uman. Acestea au capacitatea de a învăța din seturi cu un volum mare de date, dar și de a se adapta la sarcini complexe. Rețelele sunt alcătuite din unități de calcul interconectate, denumite neuroni artificiali, care se ocupă cu procesarea informațiilor și transmiterea de semnale mai departe în rețea. Rețelele neuronale sunt organizate în straturi: stratul de intrare, straturile ascunse și stratul de ieșire. Stratul de intrare primește datele de intrare, iar stratul de ieșire furnizează rezultatele obținute. Straturile ascunse sunt situate între stratul de intrare și cel de ieșire și sunt responsabile pentru transformarea datelor, astfel încât să poată fi înțelese și clasificate corect.

[A picture containing diagram

Description automatically generated](https://www.tibco.com/reference-center/what-is-a-neural-network)[Diagram

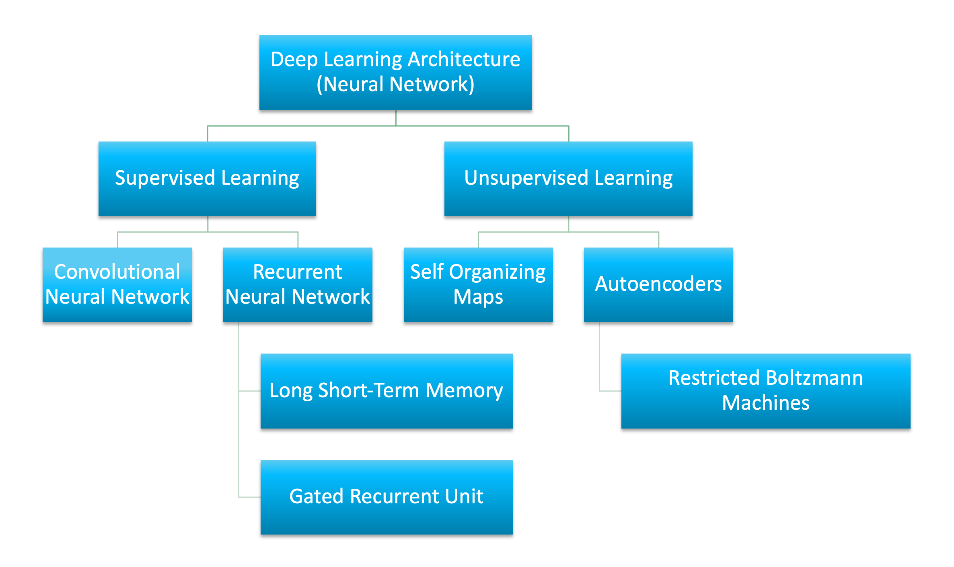
Description automatically generated](https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7)

### 2.1.2 Deep Learning

Deep learning reprezintă o subcategorie a învățării automate (Machine learning) și implică folosirea rețelelor neuronale cu un număr ridicat de straturi ascunse. Aceste straturi adiționale permit modelului să învețe reprezentări mai complexe ale datelor, ceea ce duce la o performanță ridicată în comparație cu metodele tradiționale de învățare automată. Pe măsură ce numărul de straturi crește, rețeaua devine capabilă să extragă și să proceseze informații cu un nivel de complexitate ridicat, îmbunătățind astfel capacitatea de generalizare și de rezolvare a problemelor, dar această performanță crescută necesită o putere de calcul mai ridicată pentru antrenarea modelelor.

Există mai multe tipuri de arhitecturi pentru rețele neuronale utilizate în deep learning, dintre care cele mai cunoscute sunt:

* Rețele Neuronale Convoluționale (Convolutional Neural Networks - CNN): Specializate în analiza imaginilor și a datelor bidimensionale, CNN folosește operații de convoluție pentru a identifica caracteristici locale și a învăța reprezentări ierarhice ale datelor.
* Rețele Neuronale Recurente (Recurrent Neural Networks - RNN): Proiectate pentru a procesa secvențe de date și a păstra informații temporale, RNN-urile sunt utilizate în probleme care implică texte și serii de timp.
* Rețele Neuronale cu Memorie pe Termen Lung și Scurt (Long Short-Term Memory - LSTM): O variantă a RNN, LSTM rezolvă problema degradării gradientului în rețelele recurente prin introducerea unor structuri speciale denumite celule de memorie. Acestea permit păstrarea informației pe perioade mai lungi și facilitează învățarea relațiilor temporale complexe.
* Rețele Neuronale Auto-Asociative (Autoencoder): Aceste rețele sunt folosite pentru a învăța reprezentări comprimate și eficiente ale datelor prin procesul de codificare și decodificare. Autoencoders sunt utilizate adesea în reducerea dimensionalității și în probleme de denoising.
* Rețele Neuronale Generative Adversariale (Generative Adversarial Networks - GAN): GAN constă din două rețele neuronale care se antrenează în paralel - un generator care produce date și un discriminator care distinge între datele reale și cele generate. Aceste rețele sunt folosite pentru a genera imagini și date sintetice de înaltă calitate.

[](https://developer.ibm.com/articles/cc-machine-learning-deep-learning-architectures/)

### 2.1.3 Avantajele Deep Learning

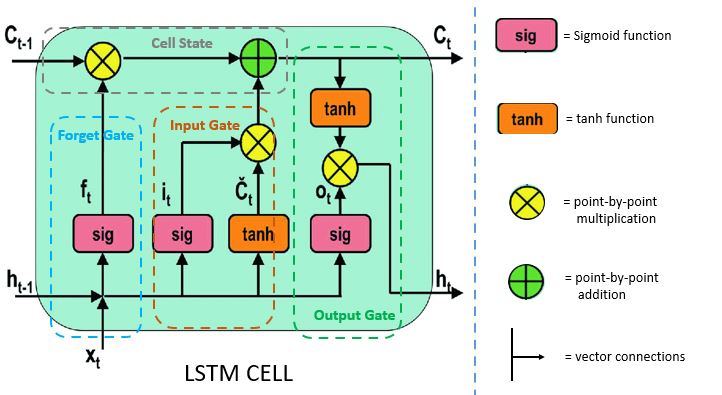
În cazul problemei studiate în prezenta lucrare, am decis continuarea construirii modelului folosind un algoritm bazat pe deep learning datorită unor serii de avantaje contra algoritmilor de machine learning:

1. Capacitatea de a învăța caracteristici complexe: Deep learning permite extragerea și învățarea unor caracteristici complexe ale datelor, fără a necesita selecția manuală a acestora, ceea ce duce la o îmbunătățire a performanței predicției.
2. Tratarea relațiilor temporale de lungă durată: Arhitecturi precum RNN și LSTM sunt special concepute pentru a învăța relațiile temporale de lungă durată în date, ceea ce permite modelului să anticipeze mai bine evoluția unei serii de timp pe parcursul a 7 zile.
3. Scalabilitate: Deep learning poate scala eficient la seturi mari de date, deoarece este capabil să proceseze informații în mod paralel și să beneficieze de accelerarea hardware (GPU). Acest lucru este important în analiza seriilor de timp, unde datele pot fi colectate și analizate în timp real.
4. Adaptabilitate la schimbări: Deep learning este capabil să se adapteze la modificări și să generalizeze în condiții noi. Aceasta este o caracteristică importantă pentru regresia seriilor de timp, deoarece evoluția lor poate fi influențată de factori externi și de evenimente neprevăzute.

## 2.3 Arhitectura LSTM

Arhitectura LSTM (Long Short-Term Memory) a fost dezvoltată de Hochreiter și Schmidhuber în 1997 și este o variantă a rețelelor neuronale recurente (RNN) care se adaptează la învățarea relațiilor temporale de lungă durată. Această metodă a devenit populară datorită abilității sale de a depăși problema dispariției gradientului (gradient descend), care afectează performanța RNN-urilor în procesarea secvențelor lungi de date.

Arhitectura modelului este compusă din unități de memorie numite celule, care sunt conectate într-o manieră similară neuronilor tradiționali. Fiecare celulă LSTM conține trei porți: intrare (input), uitare (forget) și ieșire (output). Aceste porți controlează fluxul informației în interiorul și în afara celulei, permitând actualizarea stării interne în funcție de informațiile noi și relevante, precum și eliminarea informațiilor nerelevante.

[](https://www.pluralsight.com/guides/introduction-to-lstm-units-in-rnn)

Arhitectura unui LSTM se folosește de funcțiile de activare neliniare tanh și sigmoidală. Tanh menține valorile între -1 și 1, în timp ce sigmoidală între 0 și 1. Aceste funcții ajută rețeaua să decidă ce informații sunt importante și ce informații pot fi uitate.

O celulă LSTM conține trei porți: poarta de uitare, poarta de intrare și poarta de ieșire. Poarta de uitare evaluează informațiile ce trebuie păstrate sau ignorate, trecând informațiile prin funcția sigmoidală și obținând valori între 0 și 1. Poarta de intrare actualizează starea celulei trecând starea curentă și starea ascunsă precedentă prin a doua funcție sigmoidală și prin funcția tanh.

După ce rețeaua obține informațiile necesare de la porțile de uitare și de intrare, acestea sunt folosite pentru a actualiza starea celulei. Noua stare a celulei se obține prin înmulțirea stării celulei anterioare cu vectorul de uitare și adăugarea rezultatului înmulțirii vectorului de intrare.

Poarta de ieșire determină următoarea stare ascunsă, care conține informații despre intrările anterioare. Aceasta se obține trecând starea curentă și starea ascunsă precedentă prin a treia funcție sigmoidală și trecând noua stare a celulei prin funcția tanh. Rezultatele sunt înmulțite element-cu-element, iar starea ascunsă obținută este folosită pentru predicție. Procesul continuă cu transferul noii stări a celulei și a noii stări ascunse la următorul pas de timp.

În rezumat, porțile de uitare, de intrare și de ieșire ale unui LSTM colaborează pentru a gestiona informațiile relevante din pașii anteriori și a decide ce informații noi pot fi adăugate în pasul curent, determinând următoarea stare ascunsă.

## 2.4 Rest Api

Un API (Application Programming Interface) este o interfață de programare a aplicațiilor care permite comunicarea și interacțiunea între diferite componente software. API-urile definesc un set de reguli, protocoale și specificații pe care dezvoltatorii le pot urma pentru a construi și pentru a integra aplicații și componente software, fără a fi nevoiți să înțeleagă detaliile interne ale acestora.

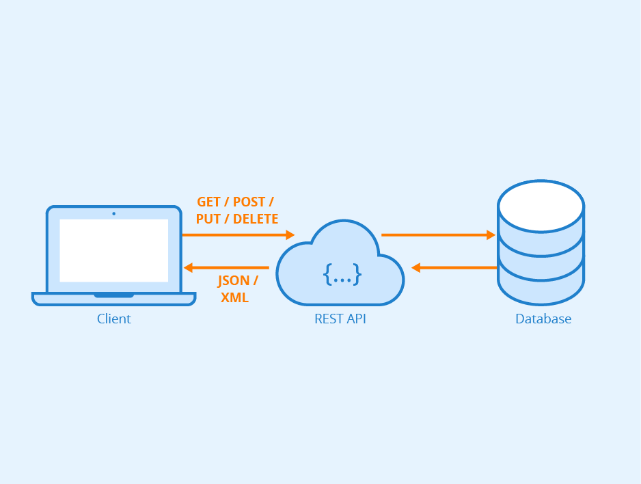
API-urile facilitează abstractizarea și modularitatea, ajutând la crearea de soluții software mai complexe și adaptabile. În plus, API-urile pot fi utilizate pentru a accesa și utiliza funcționalități oferite de alte aplicații sau servicii, permițând economisirea timpului și efortului necesar pentru a dezvolta acele funcționalități de la zero.

[Diagram

Description automatically generated with medium confidence](https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces)

Designul API-ului RESTful a fost definit de Dr. Roy Fielding în teza sa de doctorat din 2000. Pentru ca un API să fie considerat RESTful, trebuie să se conformeze acestor criterii:

1. Stateless: Fiecare cerere de la client către server trebuie să conțină toate informațiile necesare pentru a înțelege și procesa cererea. Serverul nu trebuie să păstreze informații despre starea sesiunilor între cereri.
2. Cacheabil: Răspunsurile de la server pot fi marcate ca fiind cacheabile sau necacheabile. Acest lucru permite clientului să refolosească răspunsuri anterioare pentru a îmbunătăți performanța și a reduce sarcina pe server.
3. Client-server: Arhitectura REST separă interfețele și responsabilitățile dintre client și server. Acest lucru permite dezvoltarea și îmbunătățirea independentă a ambelor părți, fără a afecta funcționarea celeilalte.
4. Interfață uniformă: Un API RESTful trebuie să ofere o interfață uniformă, ceea ce înseamnă că are un set consistent de metode și convenții pentru a accesa și manipula resursele. Metodele HTTP standard sunt utilizate în general pentru operații CRUD (Create, Read, Update, Delete). De exemplu: POST (create), GET (read), PUT/PATCH (update) și DELETE (delete).
5. Sistem de resurse identificabile: Fiecare resursă dintr-un API RESTful trebuie să aibă un identificator unic, de obicei un URI (Uniform Resource Identifier). Acest lucru facilitează localizarea și accesarea resurselor.
6. Comunicare prin reprezentări: Atunci când un client solicită o resursă, serverul trimite reprezentarea acelei resurse într-un format specific, cum ar fi JSON, XML sau HTML.
7. Layered system: Un API RESTful poate avea o arhitectură stratificată, ceea ce înseamnă că diferitele componente ale sistemului sunt organizate în straturi care funcționează împreună pentru a servi cererile. Acest lucru ajută la separarea responsabilităților și la simplificarea sistemului.

[](https://firetail.io/blog/wave-api-companies)

## 2.5 Docker

Docker este o platformă open-source care permite dezvoltatorilor și administratorilor de sistem să creeze, să distribuie și să ruleze aplicații în containere. Containerele sunt unități software standardizate și izolate care împachetează aplicația, dependențele sale și mediul de execuție necesar într-o formă portabilă și ușor de distribuit. Docker folosește tehnologia de virtualizare la nivel de sistem de operare pentru a izola aplicația și resursele sale de sistemul de operare și de alte aplicații care rulează pe același sistem.

Docker oferă numeroase avantaje, cum ar fi izolarea aplicațiilor, portabilitatea, scalabilitatea și eficiența în gestionarea resurselor. Aceste caracteristici au condus la popularitatea Docker în industria tehnologică și la adoptarea sa pe scară largă în dezvoltarea și desfășurarea de aplicații în diferite medii.

Am decis să integrez baza de date într-un container docker datorită unor serii de avantaje puse la dispoziție de această soluție. În primul rând, folosind Docker, administrarea și configurarea bazei de date PostgreSQL devine mai simplă și mai rapidă. Un Dockerfile și o imagine Docker predefinită facilitează procesul de instalare, configurare și actualizare a bazei de date într-un mod reproductibil și ușor de gestionat. Acest lucru asigură o mai bună organizare a proiectelor și o implementare mai rapidă a modificărilor.

În al doilea rând, containerele Docker izolează baza de date PostgreSQL de alte aplicații și servicii, reducând astfel riscul de interferențe și îmbunătățind securitatea datelor. Acestea permit dezvoltatorilor să se concentreze pe performanța și eficiența aplicației, fără a se îngrijora de posibilele conflicte între diferitele componente.

În al treilea rând, distribuirea bazei de date PostgreSQL într-un container Docker simplifică procesul de implementare a aplicației în medii diferite, precum dezvoltarea, testarea și producția. Acesta facilitează, de asemenea, procesul de scalare a aplicației, permițând adăugarea sau înlăturarea de containere cu PostgreSQL în funcție de nevoile de capacitate și performanță.

În concluzie, Docker reprezintă o soluție puternică și flexibilă pentru dezvoltarea, implementarea și integrarea bazei de date PostgreSQL. Având în vedere beneficiile sale precum portabilitatea, scalabilitatea, izolarea și eficiența în utilizarea resurselor, Docker contribuie la optimizarea procesului de dezvoltare și întreținere a aplicațiilor bazate pe PostgreSQL. Prin urmare, integrarea bazei de date PostgreSQL într-un container Docker oferă dezvoltatorilor și administratorilor o soluție eficientă și scalabilă pentru gestionarea și configurarea bazei de date în medii complexe și dinamice.

## 2.6 React

React este o bibliotecă JavaScript open-source creată și susținută de Facebook, având ca scop principal îmbunătățirea eficienței și ușurinței dezvoltării aplicațiilor web. Lansată în 2013, React a câștigat rapid popularitate în rândul dezvoltatorilor datorită beneficiilor sale în ceea ce privește modularitatea, performanța și comunitatea activă.

React se bazează pe o arhitectură modulară bazată pe componente, ceea ce înseamnă că aplicațiile sunt construite din elemente individuale și reutilizabile numite componente. Aceste componente încapsulează logica, starea și aspectul vizual, facilitând organizarea și gestionarea codului. Acest mod de structurare a aplicațiilor web aduce o serie de beneficii.

În primul rând, dezvoltatorii pot crea componente generice și reutilizabile care pot fi folosite în diferite părți ale unei aplicații sau în aplicații diferite. Acest lucru reduce duplicarea codului, îmbunătățește productivitatea și contribuie la crearea unui cod mai curat și mai ușor de întreținut. De asemenea, componentele din React funcționează independent una de cealaltă, ceea ce înseamnă că modificarea unei componente nu va afecta automat funcționarea celorlalte componente. Acest lucru facilitează dezvoltarea în echipă și îmbunătățește stabilitatea aplicațiilor.

În al doilea rând, React utilizează o tehnică numită “Virtual DOM” pentru a compara și a actualiza doar modificările necesare în DOM (Document Object Model). Aceasta înseamnă că React creează o versiune virtuală a DOM-ului și operează modificări asupra acesteia înainte de a actualiza DOM-ul real, ceea ce reduce costurile de performanță și îmbunătățește viteza aplicațiilor web.

În al treilea rând, React beneficiază de o comunitate vastă și activă, precum și de un ecosistem bogat de biblioteci și instrumente auxiliare. Dezvoltatorii pot profita de pachete open-source care pot fi utilizate pentru a extinde și îmbunătăți funcționalitatea aplicațiilor lor.

Un alt beneficiu pentru implementarea interfeței grafice pentru o aplicație web folosind React este reprezentat de fluxul unidirecțional al datelor și gestionarea stării. React promovează un flux de date unidirecțional, ceea ce înseamnă că starea aplicației și datele sunt gestionate într-o singură direcție. Acest concept facilitează gestionarea stării și reduce complexitatea, îmbunătățind în același timp predictibilitatea și depanarea aplicațiilor web. Biblioteci populare precum Redux sau MobX sunt adesea folosite împreună cu React pentru a gestiona starea globală a aplicației într-un mod eficient și scalabil.

În concluzie, React reprezintă o soluție modernă și eficientă pentru implementarea aplicațiilor web, oferind o arhitectură bazată pe componente, performanță îmbunătățită, gestionare a stării coerentă și un ecosistem activ de dezvoltatori și biblioteci. Toate aceste caracteristici contribuie la crearea de aplicații web rapide, scalabile și ușor de întreținut, ceea ce face ca React să fie o alegere potrivită pentru implementarea interfeței grafice pentru aplicația informatică implementată în cadrul lucrării de licență.

# Capitolul III – Arhitectura aplicației

## 3.1 Modelul matematic și bonitatea pentru Apple

### Modelul matematic al modelului

Modelul prezentat utilizează o arhitectură de tipul Rețelei Neuronale Recurente (RNN) specializată cu unități Long Short-Term Memory (LSTM), straturi de Dropout și un strat final Dense.

RNN sunt structuri unice în domeniul inteligenței artificiale care se disting prin bucle interne, acestea permitând informațiilor să persiste în timp. Astfel, RNN se pretează cu succes la problemele de procesare și predicție pentru date secvențiale. Cu toate acestea, abordarea clasică a RNN se confruntă cu problema cunoscută sub numele de "vanishing gradient problem". Aici intervine LSTM.

LSTM este o variantă a RNN care include "celule de memorie". Aceste celule pot învăța să rețină și să uite selectiv părți din secvența de date, făcând LSTM eficiente în a gestiona dependențele pe termen lung.

Componentele cheie ale unui nod LSTM sunt cele trei "porți" (de intrare, de uitare și de ieșire) și "celula de stare". Acestea sunt descrise de următoarele ecuații, unde "σ" reprezintă funcția sigmoid, "tanh" este funcția tangent hiperbolic, iar operatorul "⨀" denotă produsul Hadamard (element cu element):

Poarta de intrare (i\_t): Aceasta decide cât de multă informație din nodul curent "x\_t" și nodul anterior "h\_(t-1)" va fi păstrată. Se calculează ca:

i\_t = σ(Wi.[h\_(t-1), x\_t] + bi)

Poarta de uitare (f\_t): Aceasta decide câtă informație din celula de stare curentă "C\_(t-1)" va fi uitată. Se calculează ca:

f\_t = σ(Wf.[h\_(t-1), x\_t] + bf)

Celula de stare (C\_t): Aceasta reprezintă memoria actuală a nodului. Se actualizează pe baza informațiilor de la poarta de intrare și cea de uitare:

C\_t = f\_t⨀C\_(t-1) + i\_t⨀tanh(Wc.[h\_(t-1), x\_t] + bc)

Poarta de ieșire (o\_t): Aceasta decide câtă informație din celula de stare curentă "C\_t" va fi transferată la nodul de ieșire. Se calculează ca:

o\_t = σ(Wo.[h\_(t-1), x\_t] + bo)

Nodul de ieșire (h\_t):

h\_t = o\_t⨀tanh(C\_t)

Unde Wi, Wf, Wc, Wo sunt matrici de ponderi, iar bi, bf, bc, bo sunt vectori de bias-uri.

Straturile Dropout sunt utilizate pentru a evita supra-antrenarea. Acestea funcționează prin dezactivarea aleatorie a unui subset de unități în timpul antrenamentului, astfel crește robustețea modelului în fața datelor de intrare zgomotoase sau irelevante. Dropout este definit matematic ca:

y = x⨀m, unde "m" este o mască binară generată aleator, cu probabilitatea "p" pentru fiecare element să fie zero.

Finalmente, stratul Dense este un strat complet conectat care generează predicțiile finale ale modelului. În acest caz, are 7 neuroni care corespund predicțiilor prețurilor pentru următoarele 7 zile. Ieșirea stratului Dense "y" este dată de:

y = Wx + b, unde "W" este matricea de ponderi, "x" este vectorul de intrare, iar "b" este vectorul de bias-uri.

În cursul procesului de instruire, modelul se concentrează asupra minimizării unei funcții obiectiv specifice, anume Eroarea Pătratică Medie (MSE). Aceasta funcție obiectiv, reprezentată matematic prin MSE = 1/n Σ(y\_i - ŷ\_i)^2, unde y\_i reprezintă valorile reale, ŷ\_i reprezintă valorile prezise și n este numărul total de eșantioane. Scopul este de a cuantifica diferența dintre valorile prezise de model și valorile reale.

Algoritmul de optimizare folosit este Adam. denumire provenind de la "Adaptive Moment Estimation", și este utilizat pentru a minimiza această funcție obiectiv prezentată anterior. Acest algoritm se distinge prin estimările adaptative pe care le realizează pentru momentele de ordinul întâi și al doilea ale gradientului. În acest context, un "moment" este o măsură statistică, specifică fiecărui parametru al modelului, care informează algoritmul despre direcția și mărimea pasului de actualizare a acelui parametru.

Mai concret, Adam este implementat prin următoarele ecuații:

m\_t = β1m\_(t-1) + (1 - β1)g\_t

v\_t = β2v\_(t-1) + (1 - β2)(g\_t)^2

m\_hat = m\_t / (1 - β1^t)

v\_hat = v\_t / (1 - β2^t)

θ\_t = θ\_(t-1) - α \* m\_hat / (sqrt(v\_hat) + ε)

unde:

m\_t și v\_t reprezintă estimările neredresate ale momentului întâi (media) și al doilea (varianța neredresată) ale gradientului,

g\_t este gradientul funcției de cost în raport cu parametrii la pasul t,

β1 și β2 sunt coeficienți de descompunere exponențială pentru momentele de ordinul întâi și al doilea,

α este rata de învățare,

ε este o constantă mică pentru a preveni divizarea prin zero.

### 3.1.2 Bonitatea modelului antrenat

Modelul de învățare profundă propus pentru a prezice prețul acțiunilor Apple pentru următoarele 7 zile este un model secvențial bazat pe arhitectura Long Short-Term Memory (LSTM).

Arhitectura modelului este structurată în mod secvențial și cuprinde patru straturi LSTM interschimbate cu două straturi Dropout, urmate de un strat Dense. Numărul total de parametri care pot fi instruiți în acest model este de 184,887.

Straturile LSTM au scopul de a învăța dependențele pe termen lung din istoricul prețurilor acțiunilor. Acestea utilizează "celule de memorie" pentru a permite gradientului să se propage pentru durate lungi de timp, minimizând astfel problema de dispariție a gradientului care este prezentă în rețelele neuronale tradiționale recurente. În acest model, primele două straturi LSTM au fiecare 80 de neuroni și returnează întreaga secvență de output-uri, iar ultima strat LSTM returnează doar ultimul output din secvență.

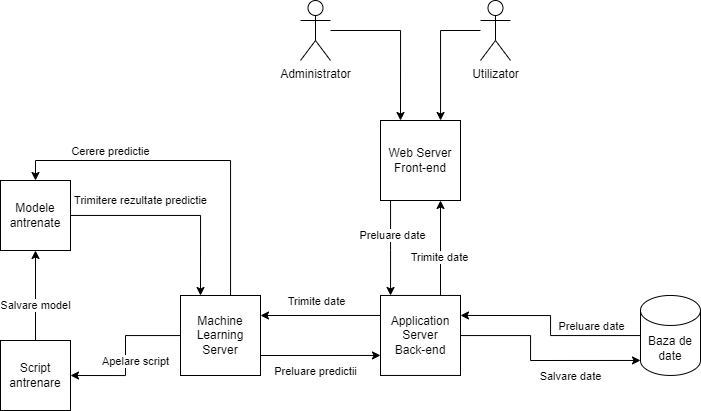
Straturile Dropout sunt utilizate pentru a preveni supra-antrenarea (overfitting), prin dezactivarea aleatoare a unui procent de neuroni în timpul antrenamentului. Această tehnică de regularizare ajută modelul să generalizeze mai bine pe date noi și necunoscute.

Ultimul strat, stratul Dense, este un strat fully-connected care generează predicția finală a modelului. Acesta are 7 neuroni, corespunzător pentru fiecare zi de predicție.

Performanța modelului a fost evaluată folosind diverse metrici de eroare, inclusiv Eroarea Absolută Medie (MAE), Eroarea Pătratică Medie (MSE), Rădăcina Erorii Pătratice Medii (RMSE) și Eroarea Procentuală Absolută Medie (MAPE). Acestea sunt metode standard de măsurare a erorii unui model de regresie, iar valoarea lor poate fi interpretată ca media diferențelor absolute dintre valorile adevărate și cele prezise.

Rezultatele arată că modelul a obținut o eroare MAE de 9.14, o eroare MSE de 324.33, o eroare RMSE de 17.9 și o eroare MAPE de 4.89%. De asemenea, coeficientul de determinare (R^2) a fost de 0.94, indicând că modelul explică 94% din variația prețului acțiunilor Apple. În fine, acuratețea direcției de predicție a fost de 51.21%.

## 3.2. Diagrama Arhitecturii

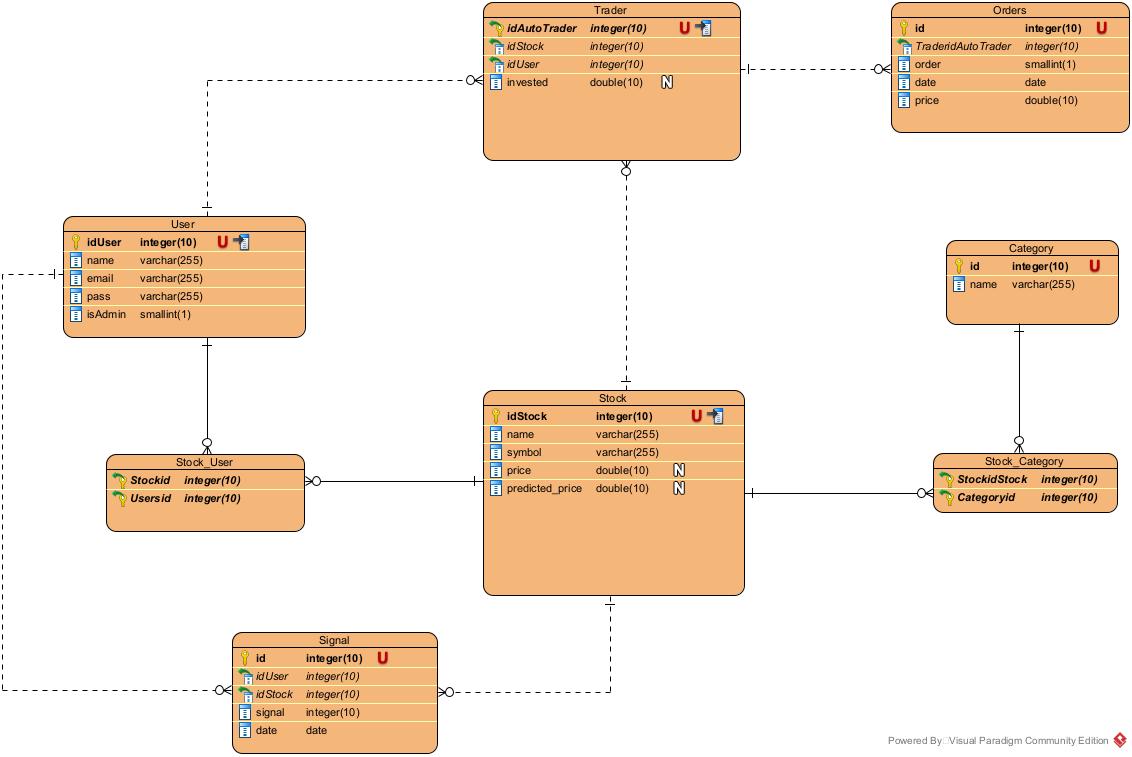


O diagramă a arhitecturii unei aplicații ilustrează structura high-level a sistemului și interacțiunile între diferitele sale componente. Această diagramă nu numai că dezvăluie modul în care componentele individuale funcționează și comunică între ele, dar evidențiază și deciziile de proiectare care stau la baza construcției sistemului. Actorii care vor interacționa cu sistemul informatic sunt administratorii sistemului și utilizatorii. Aceștia vor accesa aplicația informatică prin intermediul aplicației de front-end.

Partea de front-end a aplicației informatice este structurată folosind librăria React. Aceasta va fi principala cale de comunicare între actori și sistemul informatic. Partea de front-end comunică direct și interschimbă date cu partea de back-end a aplicației web.

Componenta de server a aplicației web a fost scrisă în Java folosind librăria Spring Boot.

## 3.3 Baza de date



În imaginea de mai sus este ilustrată diagrama entitate-relație a aplicației. Următoarele paragrafe vor descrie detaliat structura bazei de date pe baza diagramei prezentate. Structura este alcătuită din 8 tabele. Toate tabelele sunt legate prin relațiile de legătură one-to-one, one-to-many și many-to-many.

Tabela “User” conține câmpurile idUser, name, email, pass și isAdmin. Câmpul idUser este de tip integer și reprezintă cheia primară a tabelei. Câmpurile name, email si pass sunt de tip varchar. Acestea stochează informațiile utilizatorului și datele sale de logare în aplicație. Parola utilizatorului este salvată în baza de date sub formă de hash pentru o securitate sporită. Câmpul isAdmin este de tip smallint și memorează valorile 0 și 1 corespunzătoare valorilor true și false pentru a determina daca respectivul utilizator deține privilegii de admin.

Tabela “Stock” este formată din următoarele câmpuri: idStock de tip integer folosit drept primary key, câmpurile name și symbol de tip varchar folosite pentru a memora numele firmei și varianta prescurtată folosită de platformele de trading și câmpurile prices și predicted\_prices. Câmpul prices conține prețurile acțiunii în ultimele 30 de zile. Prețurile sunt preluate printr-un API extern, iar acestea sunt importate în momentul în care un stock nou este adăugat de către admin în aplicație și sunt actualizate zilnic la ora 00:00 printr-un cron job programat în aplicația de back-end. Câmpul predicted\_prices stochează predicția prețurilor pentru următoarele 7 zile și funcționează pe un principiu asemănător câmpului prices, iar diferența constă în sursa de preluare a datelor. Datele sunt preluate printr-un server secundar al aplicației care se ocupă cu antrenarea modelelor noi de inteligență artificială și cu furnizarea de predicții pentru acțiuni. Predicțiile sunt furnizate printr-un REST API în momentul adăugării acțiunii în aplicație sau zilnic la ora 00:00 folosind aceeași metodă abordată pentru câmpul prices.

Tabela “Signal” are ca scop stocarea datelor legate de predicțiile personale pe care utilizatorii le pot furniza zilnic pentru fiecare acțiune disponibilă în aplicație. Aceasta conține câmpurile id, idUser, idStock și signal de tip integer, iar ultimul câmp prezent este date folosit pentru stocarea zilei în care a fost făcută predicția personală de către utilizator pentru a-l putea limita la o predicție pe zi pentru fiecare acțiune. Câmpul Signal reține o variantă codificată a predicției sub formă binară în valori de 0 și 1. Valoarea 1 reprezintă o predicție bullish a utilizatorului legată de acțiunea din cauză, iar valoarea 0 reprezintă o predicție bearish. Câmpul id de tip integer este folosit drept identificator unic pentru înregistrările din tabelă. Câmpurile idStock și idUser sunt folosite drept foreign key pentru relațiile de legătură pe care le are tabela “Signal” cu tabelele “User” și tabela “Stock”.

Tabela “Stock” se află în relație de many-to-many cu tabela “Category” deoarece firmele se pot afla în mai multe categorii, iar categoriile pot conține mai multe firme. Pentru a schița relația many-to-many dintre cele două tabele, a fost creată o tabelă auxiliară de legătură numită “Stock\_Category”. Între tabela Stock și tabela Trader se află relația one-to-many. Pentru o firmă se pot instanția mai mulți traderi, dar fiecare trader trebuie să aparțină unui singur stock.

Tabela „Category” este formată din doua câmpuri: id de tip integer folosit drept primary key și câmpul name de tip varchar. Tabela are ca scop stocarea numelor în câmpul name pentru categoriile industriale în care pot fi plasate acțiunile.

Tabela „Trader” conține atributele id, stockId, UserId de tip integer și atributul invested de tip double. Atributul id este folosit drept cheie primară a tabelei, iar atributele stockId și userId sunt folosite drept foreign key pentru legăturile cu tabelele “Stock” și “User”. Atributul invested este folosit pentru a memora suma inițială pe care investitorul dorește să o investească folosind algoritmul de trading automat.

Tabela “Order” conține atributele id, order, date, price și traderId. Tabela stochează ordinele de cumpărare și vânzare ale acțiunilor pentru fiecare trader automat activat în aplicație. Atributele id și traderId sunt folosite drept cheie primară, respectiv foreign key pentru legătura cu tabela “Trader”.

Pentru a permite utilizatorilor să iși adauge anumite acțiuni la favorite a fost adăugată relația many-to-many dintre tabelele Stock și User. Astfel, un utilizator poate adăuga mai multe acțiuni la favorite, iar o acțiune poate fi adăugată la favorite de către mai mulți utilizatori. Tabela auxiliară folosită pentru relația many-to-many dintre User și Stock se numește Stock\_User.

Pentru a permite utilizatorilor să își adauge predicțiile personale, tabela Signal a fost legată de tabelele Stock și User prin relațiile many-to-one. Astfel, un utilizator poate adăuga mai multe predicții, dar fiecare predicție aparține unui singur utilizator. De asemenea, un stock poate să aibă mai multe predicții, dar fiecare predicție trebuie să se refere la un singur stock.

Pentru relația dintre tabelele User și Trader a fost folosită relația one-to-many. Un utilizator poate activa mai mulți roboți de trading, iar fiecare robot trebuie să aparțină unui singur utilizator. De asemenea, între tabelele Trader și Order relația este de one-to-many deoarece un Trader poate iniția mai multe ordine de vânzare sau cumpărare, dar fiecare ordin aparține unui singur trader.

## 3.4 Diagrama cazurilor de utilizare

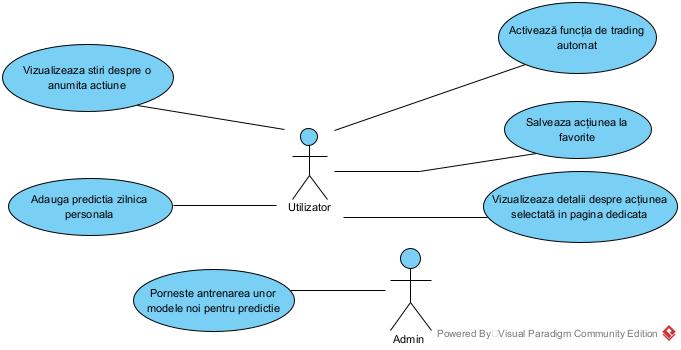


Diagrama prezentată oferă o reprezentare vizuală a funcționalităților și interacțiunilor în soluția prezentată. În centrul diagramei sunt plasate diferitele entități sau 'actori' care interacționează cu sistemul. Actorii prezenți sunt “Utilizator” și “Admin”.

Cazurile de utilizare reprezentate în diagramă indică acțiunile sau funcțiile pe care acești actori le pot realiza în cadrul sistemului informatic. De exemplu, un Utilizator individual ar putea 'Adaugă predicția zilnică personală', 'Vizualizează știri despre o acțiune', 'Salvează acțiunea la favorite', 'Vizualizează detalii despre acțiunea selectată în pagina dedicată', 'Activează functia de trading automat', etc. Un Administrator ar putea 'Adăuga Produse Noi', 'Actualiza Detalii ale Produsului', 'Vizualiza Comenzile', etc. Sistemul de Plăți Online ar putea 'Procesa Plăți' și 'Emite Confirmări de Plată'.

Liniile între actori și cazurile de utilizare reprezintă interacțiunile sau relațiile dintre ei. O linie între un actor și o acțiune semnifică că actorul respectiv poate iniția sau participa la acțiunea respectivă.

Fiecare caz de utilizare este descris detaliat în documentația aferentă, permițând înțelegerea completă a ceea ce presupune acel caz și ce interacțiuni are cu alte cazuri de utilizare și actori.

Diagrama cazurilor de utilizare este un instrument esențial în procesul de dezvoltare a software-ului, ajutând la identificarea cerințelor sistemului și la definirea scopului acestuia într-un mod clar și vizual. Acesta facilitează comunicarea și înțelegerea între diferitele părți interesate, inclusiv dezvoltatori, manageri de proiect, clienți și utilizatori finali.

De clase, de secventa,

# Capitolul IV – Implementearea aplicației

## Antrenarea modelelor de inteligență artificială

Implementarea de cod oferită în scriptul prezentat mai sus se încadrează în domeniul de procesare a datelor și învățare automată (Machine Learning), fiind axată pe predicția prețurilor acțiunilor pe piața bursieră. Aceasta folosește o rețea neuronală de tip LSTM (Long Short-Term Memory) îmbunătățită cu ajutorul unei optimizări bazate pe metoda Bayesian. Explicația detaliată a acestui proces este prezentată mai jos.

Primul pas în acest proces este descărcarea datelor pentru acțiunile bursiere dorite. Aceasta este realizată prin intermediul funcției download\_stock\_df(), ce utilizează Alpha Vantage API pentru a obține datele istorice despre prețul acțiunilor. În plus, funcția adaugă și date economice și tehnice relevante, precum Produsul Intern Brut (REAL\_GDP), Rata inflației (INFLATION), Rata șomajului (UNEMPLOYMENT), Rata de referință a Fed (FEDERAL\_FUNDS\_RATE), precum și doi indicatori tehnici (ema9 și ema21). Aceste date suplimentare pot oferi informații valoroase pentru modelul de învățare automată, contribuind la o mai bună înțelegere a evoluției prețurilor acțiunilor.

Următorul pas este pregătirea datelor pentru a fi introduse în modelul de învățare automată. Acesta este efectuat de funcția prepare\_data(). În primul rând, toate datele sunt scalate la un interval între 0 și 1 utilizând MinMaxScaler din biblioteca de învățare automată scikit-learn. Aceasta este o practică obișnuită în învățarea automată pentru a facilita procesul de optimizare al modelului.

După scalare, datele sunt partitionate în seturi de antrenament și de testare cu ajutorul funcției partition\_dataset(). Această funcție împarte datele în fereastre de lungime input\_sequence\_length pentru datele de intrare (X) și lungime output\_sequence\_length pentru datele de ieșire (y). De exemplu, dacă input\_sequence\_length este 30 și output\_sequence\_length este 7, modelul va învăța să prezică prețurile acțiunilor pentru următoarele 7 zile pe baza informațiilor din ultimele 30 de zile.

Următorul pas este construirea modelului de învățare automată. Acesta este realizat în funcția define\_model(), care construiește un model de rețea neuronală LSTM folosind biblioteca TensorFlow. LSTM este o formă de rețea neuronală recurentă (RNN) care este potrivită pentru procesarea secvențelor temporale, cum ar fi seriile de timp, deoarece are capacitatea de a păstra informații pentru perioade lungi de timp.

Modelul LSTM este construit cu o arhitectură care conține un strat LSTM de intrare, urmat de un număr de straturi LSTM intermediare și de straturi Dropout, în funcție de parametrii aleși. Dropout este o tehnică de regularizare care ajută la prevenirea suprainvățării (overfitting) prin dezactivarea aleatorie a unei proporții de neuroni în timpul antrenamentului. În final, modelul conține un strat Dense care emite predicțiile finale ale modelului.

Antrenarea modelului se realizează utilizând optimizarea bayesiană pentru a selecta cea mai bună configurație a parametrilor. Aceasta este realizată prin intermediul bibliotecii keras-tuner. Optimizarea bayesiană este o tehnică de optimizare globală care estimează funcția obiectiv și alege următoarea intrare pe baza acestei estimări.

După antrenare, cele mai bune modele sunt salvate pe disc pentru a fi utilizate mai târziu. Aceasta se realizează prin intermediul metodei save() a modelului.

În concluzie, acest script prezintă o implementare a unei rețele neuronale LSTM pentru predicția prețurilor acțiunilor, îmbunătățită cu optimizare bayesiană. El descarcă și pregătește datele istorice despre prețurile acțiunilor și diverse date economice și tehnice, antrenează un model LSTM pe aceste date și salvează cele mai bune modele pentru utilizare ulterioară.

## Server Python

Codul prezentat mai sus reprezintă o aplicație web dezvoltată cu ajutorul Flask, un framework Python folosit pentru crearea aplicațiilor web. Aplicația furnizează două servicii sau "puncte de terminare" (endpoints) accesibile prin intermediul unui client, precum un browser web sau o altă aplicație care poate să trimită și să primească cereri HTTP.

Prima funcție, predict, este asociată cu punctul de terminare '/predict' și este concepută pentru a furniza predicții despre prețurile acțiunilor unei companii. La început, funcția preia datele de intrare trimise de client prin cererea HTTP. Aceste date conțin un simbol care reprezintă acțiunea unei anumite companii pentru care clientul dorește o predicție.

Funcția verifică dacă există deja un model salvat pentru simbolul dat; în caz contrar, încarcă un model implicit. Datele de intrare sunt apoi preluate, utilizând funcția download\_stock\_df pentru a descărca datele acțiunilor companiei, și apoi prepare\_data pentru a scala datele. După prelucrarea acestor date, acestea sunt furnizate modelului pentru a genera o predicție.

Predicția este apoi scalată invers pentru a fi adusă înapoi la scala originală a prețurilor acțiunilor, și este returnată către client sub forma unui răspuns JSON.

Cea de-a doua funcție, train, este asociată cu punctul de terminare '/train' și este folosită pentru a iniția procesul de antrenament al modelului pentru o anumită companie. Similar cu funcția predict, train primește date de intrare de la client sub forma unui simbol de acțiune. Apoi, aceasta rulează un script numit main.py într-un proces separat, furnizând simbolul ca argument pentru acest script.

După finalizarea antrenării modelului, funcția train returnează un mesaj de succes către client, confirmând că antrenamentul a început pentru acțiunea companiei respective.

În sfârșit, codul inițiază aplicația Flask, indicând portul pe care serverul va asculta pentru cererile de la clienți. Acest pas este necesar pentru a porni serverul și pentru a-l face disponibil pentru a răspunde la cererile de la clienți.

## Server Java

In contextul dezvoltării de software, codul sursă al unei aplicații reprezintă fundamentul funcționalității sale și este esențial pentru înțelegerea mecanismelor interne care guvernează comportamentul acesteia. În contextul actual, ne vom concentra asupra explorării codului sursă al unei aplicații backend dezvoltate în Java, structurat în mai multe pachete, fiecare având un rol specific în arhitectura generală a aplicației.

Primul pachet analizat este "com.licenta.category". Acesta conține clasele "Category", "CategoryController" și "CategoryRepository". Clasa "Category" este o entitate JPA (Java Persistence API), ceea ce înseamnă că reprezintă o tabelă în baza de date a aplicației. Această clasă conține câmpuri precum "id", "name" și "stocks", care reprezintă coloanele tabelului din baza de date. De asemenea, are metode pentru a obține și seta valorile acestor câmpuri (gettere și settere), precum și metode "equals", "hashCode" și "toString", care sunt adesea suprascrise pentru a personaliza comportamentul unei instanțe a clasei.

Clasa "CategoryController" este o componentă Spring MVC (Model-View-Controller) care gestionează interacțiunile între utilizatori și aplicație în ceea ce privește operațiunile legate de entitățile "Category". Aceasta este dependentă de "CategoryRepository", care este injectat prin constructor.

"CategoryRepository" este un repository Spring Data JPA. Aceasta este o interfață care extinde JpaRepository, oferind un set de metode pentru operațiuni CRUD (Create, Read, Update, Delete) asupra entităților "Category". JpaRepository este o interfață specifică Spring Data JPA care oferă funcționalități pentru operarea pe entități, inclusiv operațiuni de paginare și sortare.

Un alt pachet important în aplicație este "com.licenta.order", care include clasele "Order", "OrderController" și "OrderRepository". Clasa "Order" este, de asemenea, o entitate JPA, reprezentând o comandă din aplicație. Aceasta conține câmpuri precum "id", "type", "price", "quantity", "date" și "trader", care sunt mapate la coloanele corespunzătoare din baza de date. De asemenea, are metode getter și setter pentru aceste câmpuri, precum și metode "equals", "hashCode" și "toString" suprascrise.

"OrderController" este o componentă Spring MVC responsabilă de gestionarea interacțiunilor legate de comenzi în cadrul aplicației. La fel ca și în cazul "CategoryController", aceasta depinde de "OrderRepository", care este injectat prin constructor.

"OrderRepository" este un alt repository Spring Data JPA, care oferă metode pentru operațiuni CRUD asupra entităților "Order".

Pachetul "com.licenta.config" include clasele "JwtAuthFilter" și "SecurityConfig". "JwtAuthFilter" este o componentă care se ocupă cu procesarea JWT (JSON Web Token) în cererile HTTP către aplicație. Aceasta extrage tokenul JWT și numele de utilizator din header-ul de autorizare al cererii, facilitând autentificarea utilizatorilor.

Clasa "SecurityConfig" este o componentă centrală în configurația de securitate a aplicației. Deși nu am putut analiza conținutul său în detaliu, este probabil că aceasta configurează aspecte precum criptarea parolelor, autentificarea și autorizarea utilizatorilor, și politici de securitate CORS (Cross-Origin Resource Sharing).

Analiza codului sursă a oferit o perspectivă detaliată asupra arhitecturii și funcționalității aplicației backend. Aceasta este structurată în jurul modelului MVC, cu entități JPA care reprezintă structura datelor, repository-uri Spring Data JPA care oferă metode pentru operațiuni CRUD asupra acestor entități, și controllere Spring MVC care gestionează interacțiunile utilizatorilor cu aplicația. De asemenea, aplicația include configurații de securitate detaliate, inclusiv filtrarea JWT pentru autentificare.

Această analiză subliniază importanța unei structuri de cod bine organizate și modularizate pentru dezvoltarea de aplicații software eficiente și ușor de întreținut. Fiecare componentă a codului are un rol specific și este separată de celelalte, promovând principiul separării preocupărilor. Acesta este un exemplu de bune practici în dezvoltarea software, demonstrând cum designul orientat pe obiect și principiile SOLID pot fi aplicate pentru a crea software robust și scalabil.

## Interfața grafică a aplicației

Interfața frontend, realizată în React, pentru platforma noastră de tranzacționare a acțiunilor este construită cu scopul de a oferi o experiență utilizator intuitivă și eficientă. Se bazează pe mai multe componente, împărțite pe diferite pagini: Pagina de Autentificare și Înregistrare, Pagina de Start, Pagina Individuală a Fiecărui Stock, Pagina de Setări și Pagina de Administrare.

Prima interacțiune a utilizatorului cu platforma are loc în Pagina de Autentificare și Înregistrare. Design-ul paginii este simplu și eficient, cu scopul de a facilite procesul de autentificare și înregistrare. Fiecare câmp necesar pentru înregistrare și autentificare este clar delimitat și oferă indicii vizuale pentru a asigura completarea corectă. Pagina permite, de asemenea, recuperarea parolei și include funcții de securitate, cum ar fi autentificarea cu doi factori.

Imediat după autentificare, utilizatorul este redirecționat către Pagina de Start. Aceasta prezintă o listă cu toate categoriile de stock-uri, cum ar fi Finanțe, IT, Pharma, etc. În fiecare categorie, utilizatorii pot vedea stock-urile disponibile, organizate într-o manieră intuitivă și ușor de navigat. Fiecare listare de stock oferă un sumar concis al performanței sale recente, cu posibilitatea de a accesa mai multe detalii printr-un clic simplu.

Pagina acțiunilor este centrul informațional pentru fiecare acțiune listată pe platformă. Pagina oferă date fundamentale, precum prețul acțiunii, volumul tranzacționat și performanța istorică. Există o secțiune în care utilizatorii pot adăuga propriile predicții zilnice (bullish sau bearish), încurajând astfel interacțiunea și implicarea în comunitate. O altă caracteristică notabilă este prezența unui grafic cu evoluția prețului în ultimele 30 de zile, precum și o proiecție pentru următoarele 7 zile, generată de un model de inteligență artificială. De asemenea, utilizatorii pot gestiona un robot de tranzacționare automat, bazat pe inteligența artificială, direct din această pagină.

În pagina de setări utilizatorii pot schimba detalii precum adresa de e-mail și parola. Funcționalitatea acestei pagini este simplă și directă, oferind o interfață intuitivă care permite utilizatorilor să-și actualizeze cu ușurință detaliile de cont.

Pagina de admin este accesibilă doar pentru utilizatorii cu privilegii de administrare. Permite gestionarea mai multor aspecte ale platformei, inclusiv posibilitatea de a adăuga mai multe modele de inteligență artificială pentru antrenare pe diferite stock-uri. Acesta oferă o interfață complexă și puternică pentru administrarea și îmbunătățirea continuă a performanței platformei.

În concluzie, această interfață frontend, realizată în React, reușește să combine tehnologia modernă cu o abordare orientată către utilizator, oferind o platformă puternică și intuitivă pentru tranzacționarea acțiunilor pe piața bursieră.

# Concluzii

# Bibliografie

[1] - Hochreiter, S., & Schmidhuber, J. - Long short-term memory. Neural Computation, 9(8), 1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735

[2] - Goodfellow, I., Bengio, Y., & Courville, A. - Deep Learning. MIT Press.

[3] - Gers, F. A., Schmidhuber, J., & Cummins, F. - Learning to forget: Continual prediction with LSTM. Neural Computation, 12(10), 2451-2471. https://doi.org/10.1162/089976600300015015

[4] - LeCun, Y., Bengio, Y., & Hinton, G. - Deep learning. Nature, 521(7553), 436-444. https://doi.org/10.1038/nature14539

[5] - Graves, A. - Supervised Sequence Labelling with Recurrent Neural Networks. Springer.

[6] - Schuster, M., & Paliwal, K. K. - Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing, 45(11), 2673-2681. https://doi.org/10.1109/78.650093

[7] - Bengio, Y., Simard, P., & Frasconi, P. - Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks, 5(2), 157-166. https://doi.org/10.1109/72.279181

[8] - Goodfellow, I., Courville, A., & Bengio, Y. - Deep learning. MIT Press.

[9] - Hochreiter, S., Bengio, Y., Frasconi, P., & Schmidhuber, J. - Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. A field guide to dynamical recurrent neural networks, 237(244), 141-146.

[10] - Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. - LSTM: A search space odyssey. IEEE Transactions on Neural Networks and Learning Systems, 28(10), 2222-2232. https://doi.org/10.1109/TNNLS.2016.2582924