Progetto di Ingegneria del Software

Francesco Corrini [1067709], Enrico Perani [1066174]

A.A. 2021/2022

Sommario

Questa relazione rappresenta la relazione finale del progetto per l'esame di Ingegneria del Software, presentato da Francesco Corrini ed Enrico Perani.
Nel primo capitolo viene presentato il project plan, nel quale vengono riassunti i punti fondamentali del progetto, dal modello organizzativo del team agli standard e le procedure utilizzate. Viene anche fatta un'introduzione dei due relatori. Nel secondo capitolo sono descritti in modo dettagliato tutti i requisiti che il nostro software deve rispettare. Nel terzo capitolo sono analizzati gli aspetti di management del nostro progetto: il ciclo di vita del software, la gestione delle configurazioni e l'organizzazione del team. Nel quarto capitolo viene descritta l'architettura del nostro software ed i design pattern utilizzati per risolvere alcuni problemi. Nel quinto capitolo entriamo nel cuore del progetto: la modellazione. Vengono presentati i cinque diagrammi con cui abbiamo modellato il nostro software accompagnati da una breve descrizione. Nel sesto capitolo vengono analizzati gli aspetti principali della fase di implementazione, di test e di manutenzione del codice Infine, nel settimo capitolo, verrano presentati i risultati di questo progetto insieme alle considerazioni finali.

Indice

Elenco delle figure

Capitolo 1

Project Plan

1.1 Introduzione al progetto

Nel panorama dei mercati finanziari si sta inserendo sempre più la figura dell'analista quantitativo, ovvero colui che utilizza analisi statistiche per studiare i mercati finanziari. L'obbiettivo dei traders è quello di creare strategie algoritmiche, che possano essere testate sui dati passati tramite un software, così da poterne valutare le prestazioni. L'obbiettivo del nostro software è quello di fornire ad un analista uno strumento con il quale possa fare le seguenti operazioni:

- ottenere le serie storiche di diversi titoli da più piattaforme (Binance, Yahoo Finance)
- scrivere ed implementare diverse strategie come algoritmi
- testare le strategie sui dati

E' importante, visti questi tre punti, che il software abbia un'architettura modulare la quale permetta di modificare, in base alle necessità, una delle tre componenti del software; ad esempio la possibilità di implementare una nuova strategia senza modificare la raccolta dati o la fase di test. Il software viene curato e prodotto da Francesco Corrini ed Enrico Perani.

1.2 Process model

Il modello utilizzato per la progettazione del software è quello dei prototipi: divideremo lo sviluppo in time slot da una settimana l'uno, in ogni slot si svilupperà un prototipo.

1.3 Organizzazione del progetto

Ogni venerdì verrà fatto un briefing nel quale si valuteranno gli sviluppi della settimana passata, si discuteranno/approveranno eventuali proposte di modifica, si scriverà del codice insieme (programmazione a coppie) e si programmeranno le attività per la settimana successiva, le quali saranno divise equamente fra i due membri.

1.4 Standard e procedure

Il progetto sarà scritto in linguaggio java con la convenzione lower Camel case mentre l'ambiente di sviluppo utilizzato sarà Eclipse. Per scrivere la documentazione in linguaggio naturale verranno usati file di testo; per produrre la documentazione sotto forma di diagrammi useremo il linguaggio UML con il software StarUML. Per gestire le versioni del software nonché lavorare contemporaneamente al progetto e per condividere la documentazione verrà utilizzato GitHub.

1.5 People management

Ad ogni briefing il "management" (entrambi perché il progetto è agile) produrrà un documento di sintesi del briefing nel quale saranno riportati:

- Obiettivi raggiunti e mancati;
- Stato di avanzamento generale del progetto;
- Nuove richieste di implementazione;
- Organizzazione del lavoro nel team.

1.6 Rischi

Il rischio principale è costituito dal fatto che il software prodotto non sia user friendly: per poter utilizzare questo software bisogna essere in grado di implementare le strategie prodotte dagli analisti in codice java per poi aggiungerle nel software. Risulta necessario che l'utente sia un programmatore, o che abbia alle spalle un team di sviluppo (e quindi sostenga un costo) per utilizzarlo. Questo fatto potrebbe scoraggiare molti utenti, che preferirebbero software più commerciali.

1.7 Staff

Francesco Corrini, 21 anni, Studente di Ingegneria Informatica presso l'Università degli studi di Bergamo, esperienza di tesi con l'azienda Huawei nello sviluppo di un'applicazione smartwatch per segnalare terremoti.
Enrico Perani, 31 anni, Studente di Ingegneria Informatica presso l'Università di Bergamo, ha lavorato presso Banca BCC, SAME Deutz-Fahr ed aeroporto "Il Caravaggio".

1.8 Tecniche e metodi

Per la gestione delle versioni del software, per comunicare nel gruppo e per condividere la documentazione è usato github, soprattutto utilizzando la sua versione desktop insieme al plugin di eclipse per fare le operazioni di push direttamente dall'IDE.
Per la parte di test verrà utilizzato Junit in eclipse che permette di scrivere casi di test che vengono in automatico applicati al software (quando lo strumento viene lanciato) e da in output una misura di copertura.

1.9 Qualità del software

Per assicurare una buona qualità del software dopo lo sviluppo di ogni prototipo vengono implementati dei nuovi casi di test e si modifica il codice finché questi non vengono superati.

1.10 Work packages

Francesco Corrini si occuperà prevalentemente del design, implementazione e della coordinazione del gruppo; Enrico Perani dei requisiti del progetto, della documentazione e della gestione delle risorse.

1.11 Risorse

Si hanno a disposizione dalle 40 alle 60 ore lavorative per ogni membro del gruppo, il quale lavorerà prevalentemente dalla postazione presso la sua abitazione (smart working).

1.12 Schedule

Il termine dello sviluppo è previsto per venerdì 7 gennaio.

1.13 Cambiamenti ed aggiornamenti del software

Il software reagirà solidamente ai cambiamenti necessari per lo sviluppo visto che il modello utilizzato è quello dei prototipi. La gestione delle versioni, come scritto in questo documento, sarà effettuata con GitHub.

1.14 Consegna

La consegna al "cliente" (prof. Angelo Gargantini) sarà effettuata tramite GitHub, a cui avrà accesso e grazie al quale potrà verificare in tempo reale lo sviluppo e le versioni del software.

Capitolo 2

Requisiti

Introduzione ai requisiti

I requisiti del software sono stati divisi in 4 macrocategorie, le quali al loro interno sono specificate in punti numerati per garantire la completa tracciabilità di ogni singolo requisito.

2.1 Scaricare i dati

1. L'utente deve poter scegliere la piattaforma che vuole utilizzare fra le disponibili (Binance, YhaooFinance)
2. Un dato è composto da cinque valori:
   - Prezzo di chiusura (close)
   - Prezzo di apertura (open)
   - Prezzo massimo
   - Prezzo minimo
   - Data e ora
3. L'utente deve poter scaricare i dati da Binance:
   (a) l'utente deve inserire data di inizio e fine dei dati da scaricare nel formato DD-MM-YYYY
   (b) l'utente deve inserire il timeframe (intervallo tra un dato e l'altro) fra 1m, 3m, 5m, 15m, 30m, 1h, 2h, 4h, 6h, 8h, 12h, 1d, 3d, 1w, 1M
   (c) l'utente deve inserire il simbolo (strumento finanziario che vuole studiare). I simboli ottenibili sono quelli presenti sulla piattaforma binance.com
   (d) il software deve connettersi alle API di binance.com e scaricare i dati relativi ai parametri inseriti
4. L'utente deve potere scaricare i dati da yhaooFinance:
   (a) l'utente deve inserire data di inizio e fine dei dati da scaricare nel formato DD-MM-YYYY
   (b) l'utente deve inserire il timeframe scegliendo fra i disponibili (1d, 1w, 1M)
   (c) l'utente deve inserire il simbolo (strumento finanziario che vuole studiare). I simboli ottenibili sono quelli presenti sulla piattaforma yhaooFinance
   (d) il software deve connettersi alle API di yhaooFinance e scaricare i dati relativi ai parametri inseriti

2.2 Applicare delle strategie ai dati

1. implementare una strategia sui dati significa definire delle regole (algoritmo) con il quale si decide quando acquistare e vendere uno strumento finanziario del quale abbiamo scaricato i dati; il software deve dare in output il risultato della strategia ovvero quando (data e ora) effettua le operazioni

2. l'utente deve scegliere la strategia da applicare sui dati fra le disponibili (media mobile, ROC)

3. l'utente deve poter applicare la strategia delle medie mobili:

   (a) La media mobile di periodo p di una serie di valori s[N] è essa stessa una serie il cui 0-esimo elemento si calcola come
   $$\frac{\sum_{i=0}^{p-1} s[i]}{p}$$
   L'1-esimo elemento si ottiene aggiungendo 1 ai due estremi della serie, il 2-esimo aggiungendo 2, e così si calcolano tutti gli elementi finché l'estremo superiore non diventa maggiore di N-1. Ne si evince che, se la dimensione della serie di partenza è N, la media mobile avrà dimensione N - p + 1. Non può quindi essere calcolata una media mobile con p > N

   (b) Per la strategia è necessario calcolare due medie mobili; l'utente deve inserire i due periodi, uno maggiore dell'altro (non possono essere uguali)

   (c) La strategia prevede che il software acquisti lo strumento finanziario scelto se il valore della media mobile con periodo minore (detta veloce) supera quello della media con periodo maggiore (detta lenta)
   $$\textit{if veloce[i] > lenta[i] and veloce[i-1] < lenta[i-1] then acquista}$$

   (d) La strategia prevede che il software venda lo strumento finanziario acquistato se il valore della media mobile con periodo minore (detta veloce) diventa inferiore a quello della media con periodo maggiore (detta lenta)
   $$\textit{if veloce[i] < lenta[i] and veloce[i-1] > lenta[i-1] then vendi}$$

4. L'utente deve poter applicare la strategia del ROC:

   (a) L'indicatore ROC di periodo p di una serie di valori s[N] è a sua volta una serie il cui 0-esimo elemento si calcola come
   $$100\frac{s[p-1]-s[0]}{s[0]}$$
   L'1-esimo elemento si calcola aggiungendo 1 alla posizione di ogni termine della formula, il 2-esimo aggiungendo 2 e così fino a che il primo termine diventa s[N-1]. Ne si evince che la dimensione della serie ottenuta sarà N - p + 1, e che quindi non si potrà avere p > N. Alla serie così calcolata viene applicata una media mobile (requisito 3a) per smussare il suo andamento. Definendo quindi q il periodo della media mobile, la dimensione dell'indicatore ROC sarà N - p - q + 2. Non è possibile avere q > N - p + 1, ovvero p + q > N + 1

   (b) L'utente deve inserire il periodo del ROC e della media

   (c) La strategia prevede che il software acquisti lo strumento finanziario se il ROC diventa positivo

if roc[i] > 0 and roc[i-1] < 0 then acquista

(d) La strategia prevede che il software venda lo strumento finanziario acquistato se il ROC diventa negativo

if roc[i] < 0 and roc[i-1] > 0 then vendi

2.3 Testare la strategia scelta in precedente calcolando alcuni parametri

1. Preso l'output della strategia il software deve calcolare:
   (a) La lista delle operazioni eseguite specificando:
      - Data di inizio dell'operazione
      - Data di fine dell'operazione
      - Profitto/perdita in percentuale dell'operazione
   (b) Profitto/perdita cumulate di tutte le operazioni
   (c) Numero di operazioni totale
   (d) Numero di operazioni in profitto e percentuale
   (e) Massimo drowdown (perdita massima maturata rispetto al picco di profitto)
   (f) Calcolo dell'equity-curve, ovvero la funzione di ritorno del capitale che associa ad ogni operazione il profitto/perdita cumulata fino a quell'istante
2. Il software deve mostrare i risultati ottenuti all'utente
3. Il software deve eseguire il plot dell'equity curve

2.4 Salvataggio dei risultati su file esterno

1. Salvataggio della lista delle operazioni su un file
2. Caricamento della lista di operazioni da un file e stampa dei risultati

Capitolo 3

Management, organizzazione ed altri aspetti di sviluppo

3.1 Ciclo di vita del software

Come speficicato nel project plan abbiamo scelto di utilizzare la tecnica del prototyping. Questa scelta è dovuta principalmente al fatto che produrre un progetto con il metodo waterfall sarebbe stato molto oneroso per via delle nostre competenze: abbiamo imparato l'utilizzo dei diagrammi durante lo sviluppo stesso del progetto ed è stato fondamentale avere un feedback immediato su tutte le varie fasi di sviluppo (requirements engineering, design, implementazione e test). Siamo quindi partiti, come da project plan, sviluppando un prototipo per ogni time slot della durata di una settimana.
Durante il primo prototipo, grazie alla scelta fatta, abbiamo potuto comprendere le problematiche principali che si sono presentate nell'implementazione ed abbiamo agito sui requisiti e sul design in modo rapido e senza difficoltà; viceversa in un modello waterfall sarebbe stato molto più complesso intervenire. Quindi siamo partiti con lo sviluppo del secondo prototipo arrivando ad avere un software già utilizzabile per eseguire delle analisi.
Il terzo prototipo sarebbe dovuto essere implementato nella settimana del 27 dicembre. Alcuni problemi del team (principalmente la consegna di un altro software) hanno interrotto momentaneamente lo sviluppo che è ripreso il tre di gennaio. In quel giorno abbiamo optato per accorpare le ultime feature da implementare in un unico prototipo finale che costituirà l'elaborato finale. Lo sviluppo è terminato il giorno tredici gennaio (una settimana dopo la scadenza prevista nel project plan). Il giorno seguente è stata eseguita un'operazione di controllo ed una fase finale di test per garantire il corretto funzionamento del software prima della consegna.

3.2 Configuration management

Per il configuration management abbiamo utilizzato github. I vantaggi principali di questa scelta sono:

- la possibilità di poter lavorare contemporaneamente al progetto utilizzando delle brenches per modificare il codice
- condividere la documentazione sia tra di noi che con il prof. Gargantini (utilizzeremo github anche per la consegna)
- avere una history di tutte le versioni passate del software con la possibilità di tornare ad una di queste in caso di una modifica instabile

- utilizzare la funzione "issues" per comunicare, proporre modifiche e mettere in evidenza le problematiche del software
- utilizzare il plugin integrato in eclipse per effettuare il push (quasi) automaticamente su github grazie all'app desktop

Su github sono quindi presenti tutte le versioni del software e della documentazione (requisiti e diagrammi UML). Nella maggior parte dei casi, in prima battuta caricavamo le modifiche al software in una brench, per poi avviare un processo di "change request": si avvisava l'altro membro della modifica e si valutava insieme se procedere o meno con una pull request. Per imporre un formalismo, chi proponeva la modifica creava la pull request in github, mentre l'altro la accettava, in modo da garantire che entrambi fossimo d'accordo sulla modifica.
Solo in un'occasione, in data 27/12/2021, siamo dovuti intervenire caricando una copia di backup del progetto nella repository e scavalcando il processo di change request, questo perché github desktop (a causa di un bug) non consentiva il push delle modifiche al progetto.

3.3 Organizzazione del team

Lavorando in due ed avendo lavorato insieme a due progetti contemporaneamente abbiamo deciso di organizzarci come un team agile. Nessuno dei due componenti aveva un ruolo gerarchicamente superiore all'altro: ad ogni prototipo abbiamo diviso i compiti da eseguire, ma ci siamo sempre aiutati l'un l'altro nel momento in cui uno dei due si è trovato in difficoltà con le sue task. Una volta la settimana organizzavamo una sessione di pair programming nella quale, scrivendo codice insieme, abbiamo trovato nuove soluzioni ai problemi che riscontravamo. Ovviamente la nostra soluzione è possibile solo per team molto piccoli: nel momento in cui un'organizzazione si espande, è necessario stabilire delle regole (gerarchiche) per la gestione del personale, altrimenti il rischio è di perdere molto in efficienza.

Capitolo 4

Architettura del software e design pattern

4.1 Architettura del software

La caratteristica principale del nostro software, così importante da essere specificata nel primo punto del project plan, è quella di poter aggiungere nel tempo nuove strategie, nuove piattaforme o magari nuovi modi di testare i dati. Dal punto di vista statico serve quindi un software con almeno tre componenti: una che scarica i dati, una che effettua il calcolo della strategia ed una che calcola i risultati.
La view "components and connectors" è stata formalizzata in UML con un diagramma delle componenti e si presenta così:

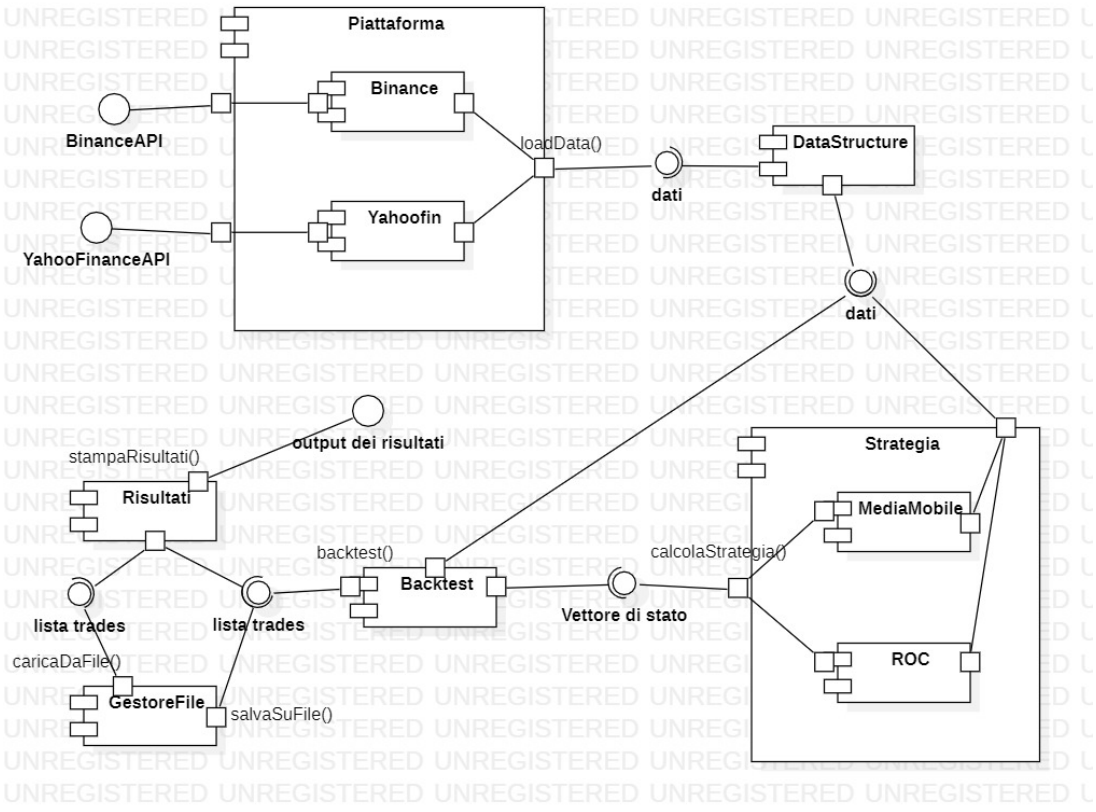

Figura 4.1: Component diagram sulla vista "components and connectors" dell'architettura

Il componente "Piattaforma" è diviso in una sottocomponente per ogni piattaforma implementata nel software. Nel codice ogni piattaforma sarà rappresentata da una classe. Ogni sottocomponente è collegata ad un'interfaccia (solitamente delle API) dalla quale è in grado di ottenere i dati richiesti. Tutti i sottocomponenti danno in output un vettore (classe Vector di java) di "Dato" (una classe volta a soddisfare il requisito 1.2) e questo viene salvato in memoria dal software. Aggiungere una nuova piattaforma è relativamente semplice: bisogna creare una nuova sottocomponente che si interfacci con le API della nuova piattaforma e far si che l'output sia un vettore di Dato.
Il componente "Strategia" deve utilizzare i dati scaricati dal componente precedente applicandovi un algoritmo il quale stabilisce se il software deve o meno aprire un'operazione di trading (requisito 2.1). Anche qui il componente è suddiviso in sottocomponenti ognuna delle quali rappresenta una strategia (ogni strategia sarà implementata come una classe). E' importante che ogni strategia dia lo stesso output il quale deve rappresentare il momento (data e ora) di quando l'operazione avviene. Abbiamo deciso di realizzare questo utilizzando un "vettore di stato", implementato come un vettore di interi della stessa dimensione del vettore dei dati. Il metodo calcolaStrategia assegna a questo vettore il valore 0 in corrispondenza di quei dati in cui il software NON ha acquistato il prodotto finanziario (attenzione: non i momenti in cui non effettua l'acquisto, ma quelli su cui non possiede l'asset). Assegna invece 1 quando possiede l'asset. Questo permette di capire quando viene fatto un'acquisto o una vendita: nel momento in cui avviene una transizione da 0 ad 1 è stato effettuato l'acquisto, mentre nel momento in cui si effettua la transizione da 1 a 0 viene venduto (realizzazione dei requisiti 2.3c, 2.3d, 2.4c, 2.4d). Abbiamo quindi associato ad ogni Dato un'informazione che ci dice se in quel momento possediamo o meno lo strumento finanziario e questa informazione è univoca per ogni strategia. Ogni qual volta si vorrà aggiungere una strategia sarà sufficiente rispettare queste caratteristiche.
La complessità vista nel calcolo delle strategie è necessaria affinchè si abbia un output univoco che possa essere preso in input dal componente che effettua il backtest della strategia. Questo componente sarà anch'esso mappato in una classe java. Il componente presi in input i dati ed il vettore di stato deve produrre la lista di operazioni (un'operazione sarà anch'essa una classe java con le caratteristiche descritte nel requisito 3.1a) che la strategia ha calcolato. Una soluzione poteva essere quella di implementare direttamente il backtest all'interno delle strategie, tuttavia si sarebbe dovuto riscrivere l'algoritmo di backtest in ogni strategia, invece separando le due fasi, all'aggiunta di una strategia dobbiamo solo definire le condizioni di apertura/chiusura di una posizione. L'output di questo componente sarà sempre un un vettore (classe Vector di java) di Trade (la classe che rappresenta il requisito 3.1a).
Il componente risultati (anch'esso una classe java) avrà come input il vettore calcolato nel vettore precedente e restituirà un'interfaccia verso i risultati calcolati dal software, descritti nel requisito 3.1 (realizzando quindi i requisiti 3.2 e 3.3). Il componente può prendere in input la lista di operazioni anche dal componente "GestoreFile" (un'altra classe java) tramite il metodo "caricaDaFile". Questo componente ha anche una funzione "salvaSuFile", che prende l'output del componente di backtest e lo salva su un file.

4.2 Design Pattern

Per la gestione della scelta della piattaforma da parte dell'utente sono stati utilizzati due design pattern: il Singleton pattern ed il Factory pattern.
La prima cosa che abbiamo fatto è creare l'interfaccia "PlatformFactory" che contiene il metodo "createPlatform", al quale viene passato un parametro intero "scelta" che rappresenta la piattaforma decisa dall'utente. La classe "PlatformCreator" implementa quest'interfaccia specificando createPlatform: nel caso in cui scelta sia 1, il metodo ritorna un oggetto Binance, nel caso in cui sia 2 ritorna un oggetto Yahoofin, altrimenti lancia una "PlatformException" specificando che

la scelta dell'utente non corrisponde con nessuna piattaforma.
Dato che nel nostro software è necessario un solo oggetto PlatformCreator, il quale deve essere chiamato per restituire una nuova piattaforma, risulta necessario implementare un meccanismo per impedire che vengano istanziati più oggetti che non servono. Il Singleton pattern ci è utile per questo scopo: la classe PlatformCreator ha il costruttore impostato a private, in modo tale che non possa essere istanziata se non da se stessa. Viene quindi aggiunto un attributo statico "istanza" che viene definito dalla funzione pubblica e statica "getPlatformCreator": questo metodo, nel momento in cui istanza è null, crea il nuovo oggetto, altrimenti restituisce l'oggetto precedentemente creato. Questo impedisce la creazione di più istanze.
Utilizzando questi due pattern viene quindi semplicissimo gestire la decisione dell'utente nel main: una volta inserita la variabile scelta, si lancia getPlatformCreator per ottenere l'oggetto Factory, il quale lancerà createPlatform per ottenere l'oggetto Platform con cui poi potremo scaricare i dati.

Capitolo 5

Progettazione e modeling

Introduzione ai diagrammi

Per la progettazione del software abbiamo prodotto 5 diagrammi: use case diagram, class diagram, state machine diagram, sequence diagram ed activity diagram. Di seguito mostriamo i diagrammi allegando una breve descrizione degli stessi.

5.1 Use case diagram

Il diagramma mostra la possibilità dell'utente (attore) di accedere ai vari use cases, in particolare, può accedere a "download dati", che lo porta (con include) anche a "gestione strategia" e "stampa risultati", mentre può decidere in via opzionale (con extend) di fare "salva operazioni". L'altro use case a cui ha accesso è "carica operazioni" al quale segue nuovamente "stampa risultati". L'attore platform (che generalizza binance e yahooFinance) non accede a nessuno use case (per via della direzionalità). E' l'utente che vi accede mediante "download dati".

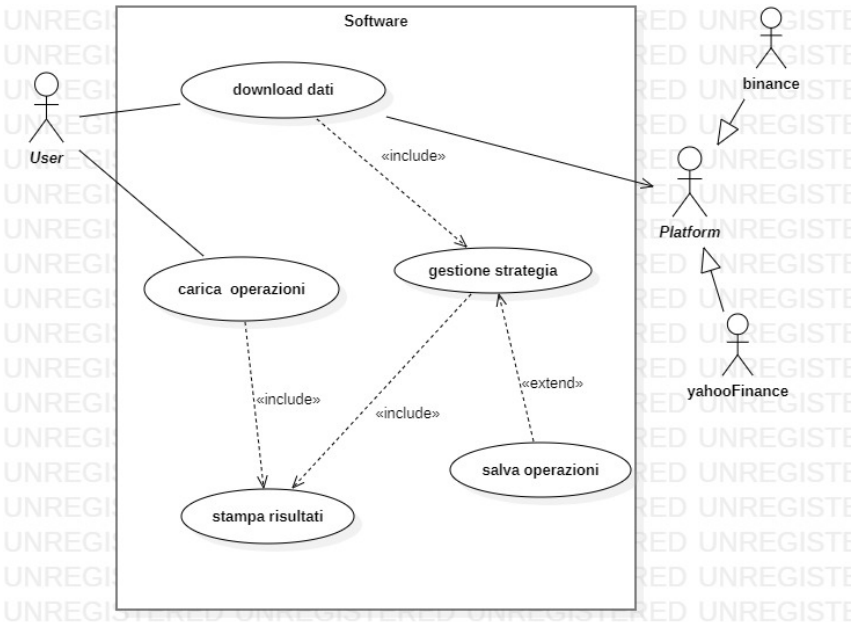

Figura 5.1: Use case diagram

5.2 Class diagram

Il class diagram rappresenta le classi principali del nostro software. Sono state escluse da questo diagramma le classi che rappresentano eccezioni (Binance Exception, YahoofinException, StrategiaException). Per rappresentare la relazione XOR, non trovando la funzione in starUML, abbiamo usato un testo.

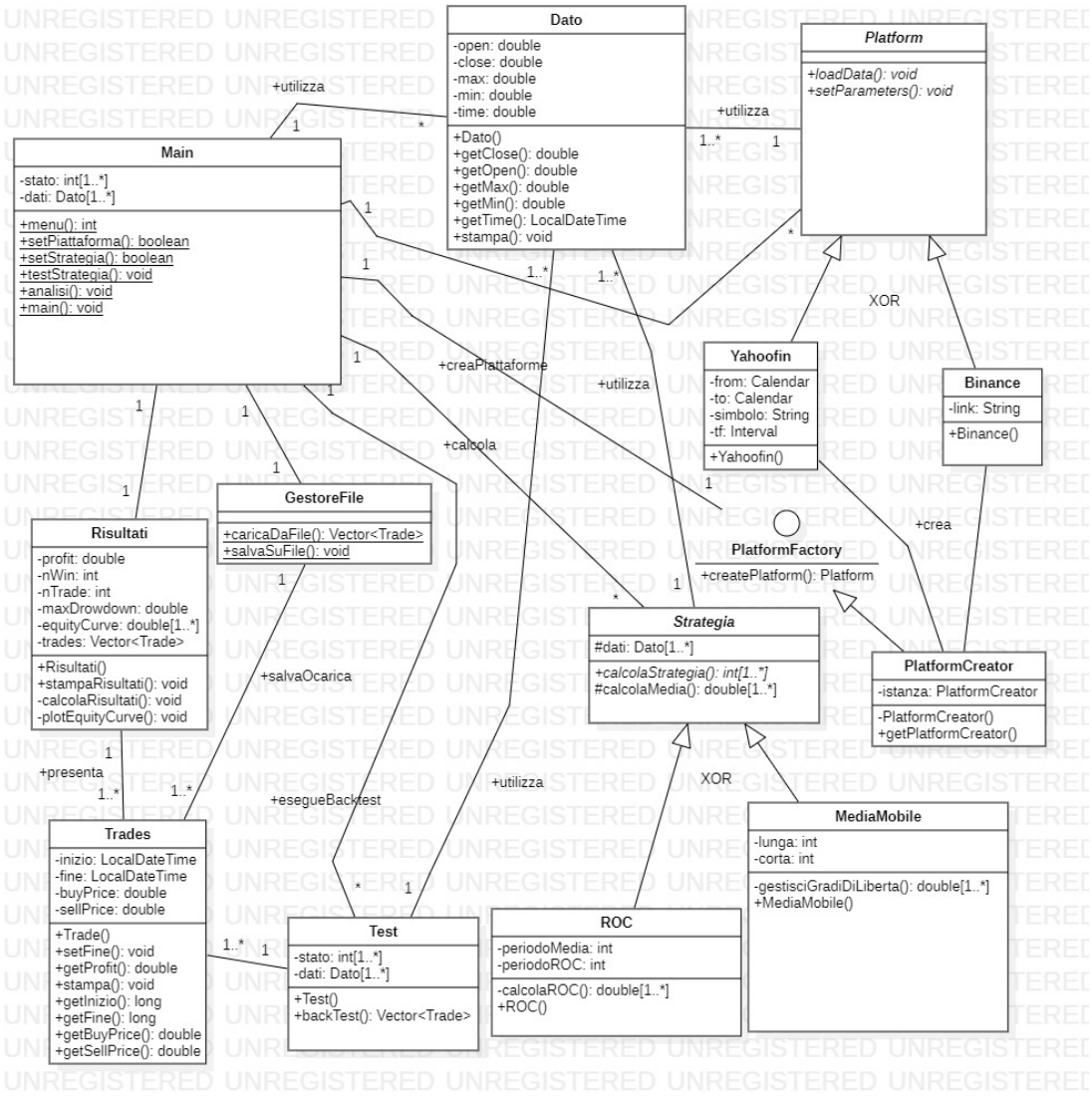

Figura 5.2: Class diagrams

5.3 State machine diagram

Lo state machine diagram evidenzia gli stati in cui si trova il nostro software. Non aggiungiamo particolari commenti poiché il diagramma è abbastanza chiaro, tuttavia specifichiamo che la submachine CaricaDati si trova nella seconda immagine di questo paragrafo.

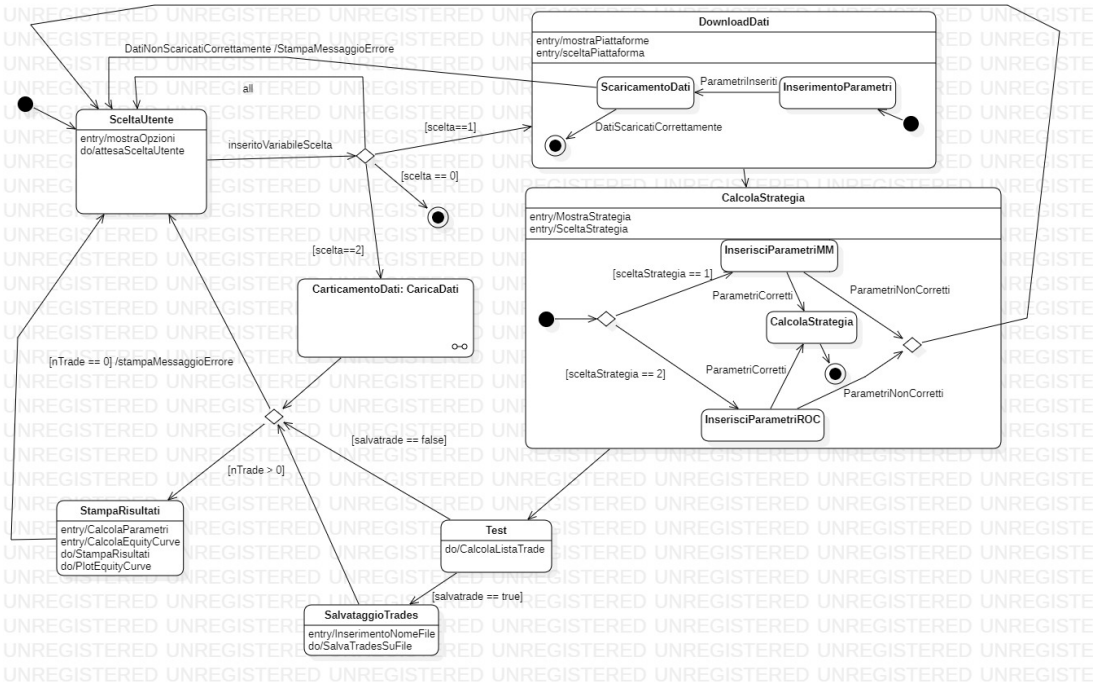

Figura 5.3: State machine diagram

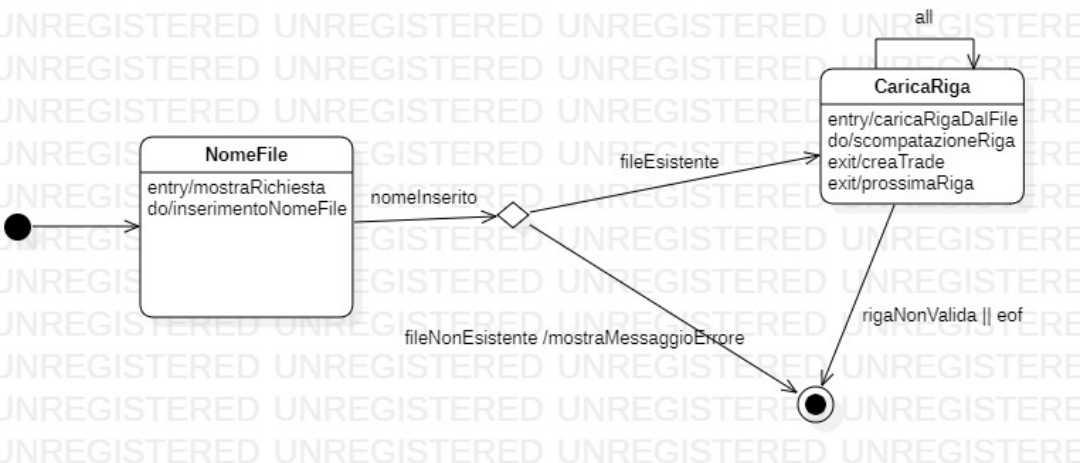

Figura 5.4: Submachine CaricaDati

5.4 Sequence diagram

Il sequence diagram è stato diviso in cinque parti per motivi di impaginazione. Tuttavia si può ricomporre leggendo le immagini in verticale (dove termina un'immagine inizia la successiva). Inoltre si può trovare il diagramma originale nella cartella Documentazione/UML della repository github.

Nella prima parte, il software mostra all'utente le due opzioni di scelta: eseguire un'analisi o caricare una precedente analisi da file.

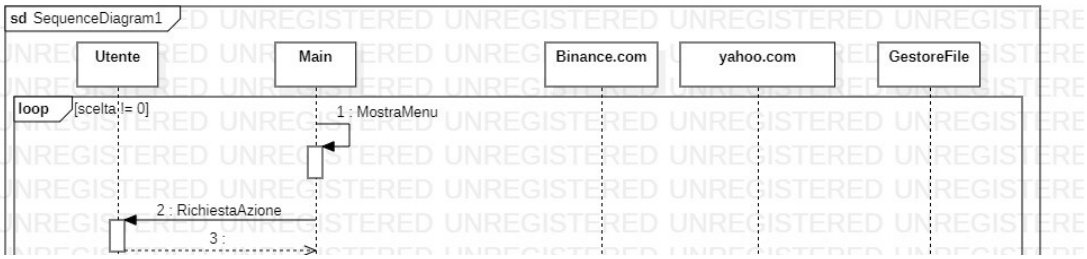

Figura 5.5: Sequence diagram, scelta iniziale dell'utente

Nella seconda parte, chiede all'utente di inserire la sua scelta. Tutto il programma è eseguito in loop: una volta terminata l'operazione scelta, il software tornerà a questo punto e chiederà nuovamente all'utente di fare una scelta.
Dopo ciò, il fragment alt divide le decisioni dell'utente: se scelta==1, viene chiesto all'utente di scegliere la piattaforma ed inserire i parametri per ottenere i dati che vuole, dopodiché il software creerà un'istanza piattaforma ed scaricherà le informazioni richieste grazie alle API del servizio (Binance o Yahoo).

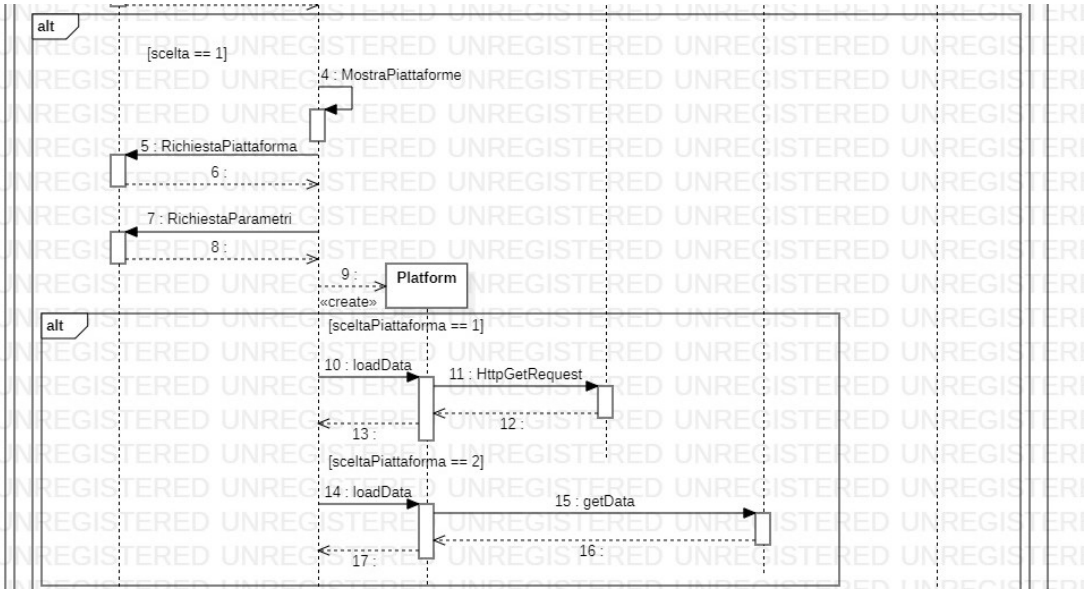

Figura 5.6: Sequence diagram, scelta della piattaforma

Nella terza parte, viene chiesto all'utente quale strategia vuole applicare ai dati: se sceglie media mobile, verrà chiesto di inserire i parametri per questa strategia, verrà creato un oggetto MediaMobile con il quale si calcoleranno le operazioni della strategia. Discorso analogo vale nel caso l'utente scelga la strategia ROC.

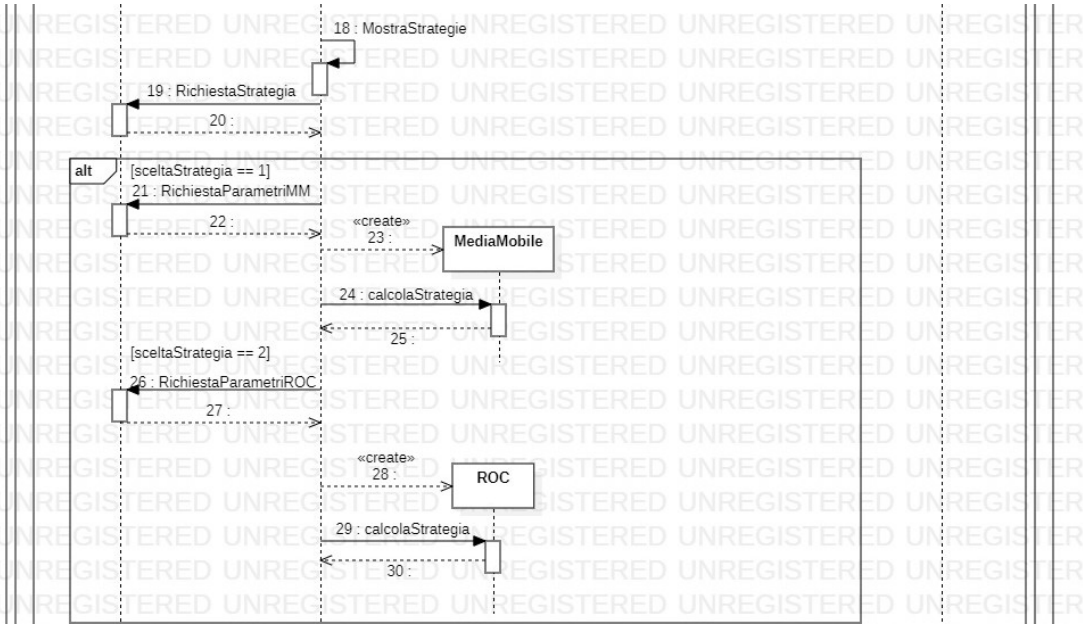

Figura 5.7: Sequence diagram, scelta della strategia

Nella quarta parte, il Main crea un'istanza della classe Test e lancia il backtest della strategia. Quando termina, mostra i risultati ed esegue il plot dell'equity curve. Viene poi chiesto all'utente se vuole salvare i dati in un file e con il fragment opt. In caso di risposta affermativa, viene chiesto all'utente il nome del file su cui salvare la lista dei trades.

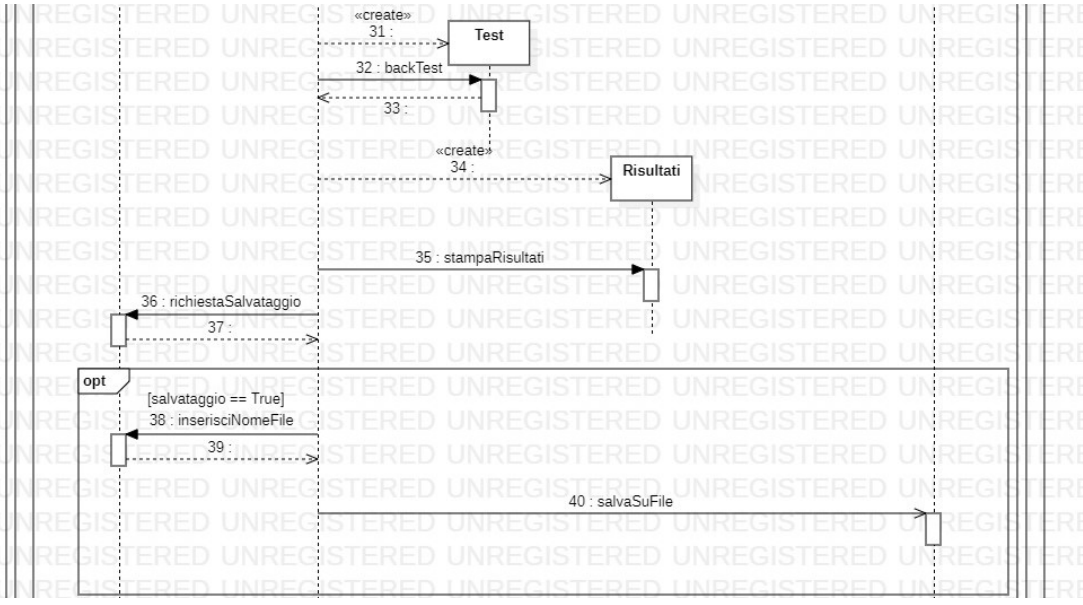

Figura 5.8: Sequence diagram, test della strategia e salvataggio su file

Nella quinta parte del diagramma viene mostrato cosa accade se l'utente (nella scelta iniziale) sceglie di caricare i dati da file. Viene quindi chiesto all'utente il nome del file, e se questo esiste si avvia il caricamento. Quando questo è finito vengono stampati i risultati caricati ed eseguito il plot dell'equity curve.

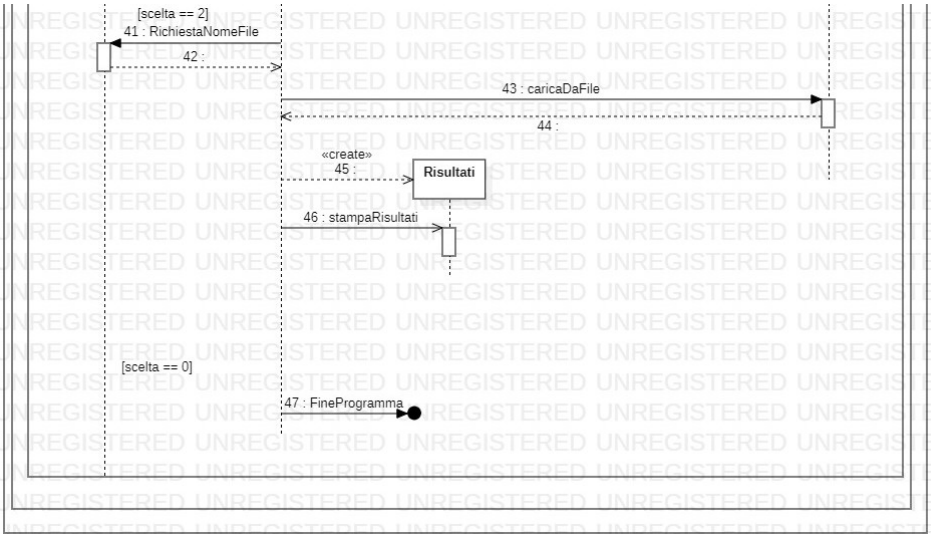

Figura 5.9: Sequence diagram, caricamento da file

5.5 Activity diagram

L'activity diagram che abbiamo prodotto non specifica il flusso dell'intero software ma solo la parte di download e analisi dei dati (viene quindi trascurata quella di caricamento e salvataggio su file).

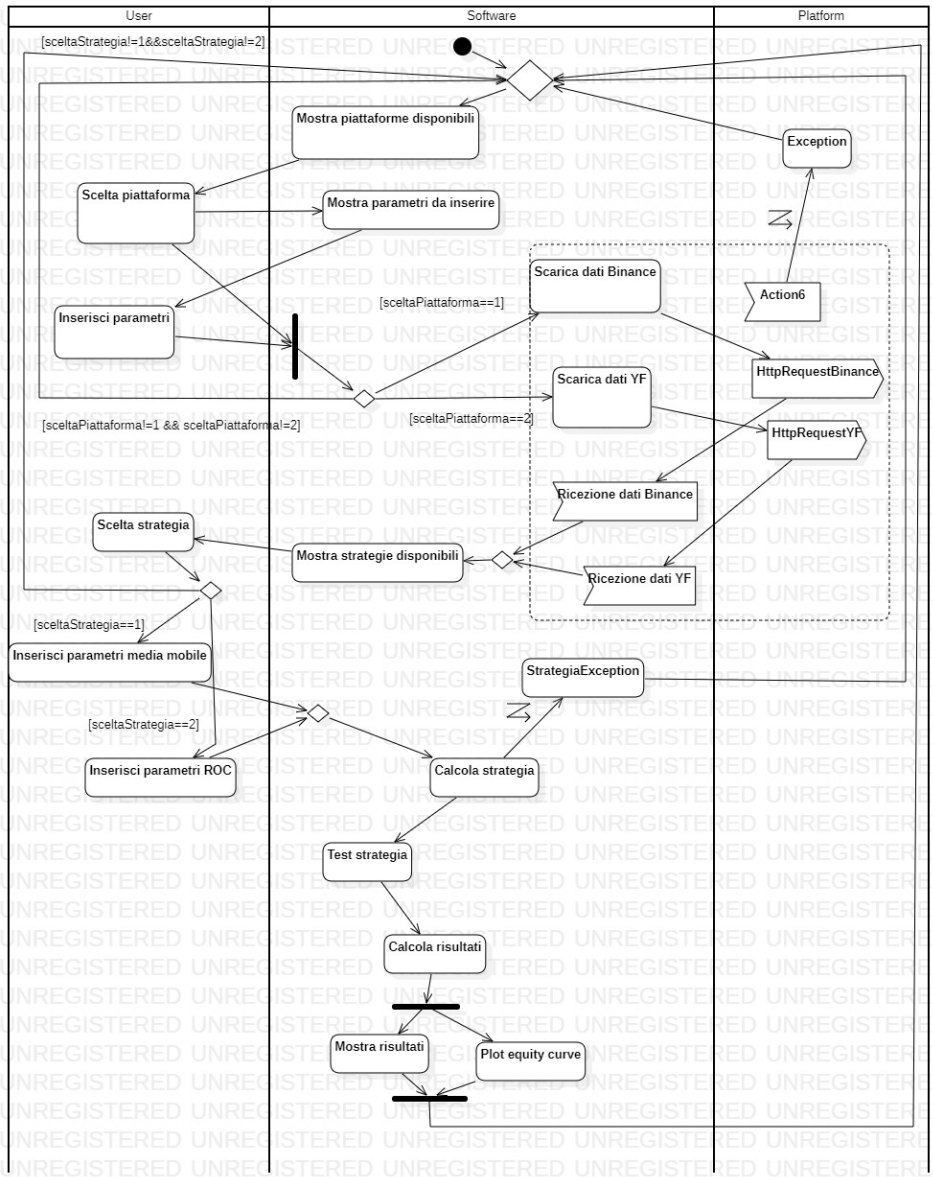

Figura 5.10: Activity diagram

Capitolo 6

Implementazione, test e manutenzione

6.1 Implementazione

Le fasi di implementazione sono avvenute durante lo sviluppo di ogni prototipo dopo aver definito il design nei diagrammi UML e prima delle fasi di test del software. Il software è strutturato in package che cercano di rappresentare ognuno dei macro-componenti dell'architettura (capitolo 4) mentre ogni classe (o insieme di classi) rappresentano un sottocomponente.
Le interfacce delle classi piattaforma sono differenti: in Binance abbiamo implementato direttamente un'HttpRequest ad un link costruito con le informazioni inserite dall'utente che chiama le API del servizio Binance.com, mentre per YahooFinance ci siamo avvalsi di una libreria esterna. Un'altra libreria importata è org.json poiché i dati scaricati da binance seguono lo standard JSON. Abbiamo poi aggiunto la libreria prog.io per gestire input/output dalla console e input/output da file. Ultima libreria importata, ma non meno importante, è jMathPlot, per gestire il plot dell'equity curve. Le librerie in formato .jar sono state inserite in un package in modo che rimanessero in github e non fosse necessario scaricarle/importarle ogni volta. Alcune librerie sono state aggiunte come dipendenza in maven senza necessità di avere nessun file (esempio Yahoofinance).
Abbiamo cercato di implementare il software in modo che fosse il più robusto possibile cercando di gestire il più possibile le eccezzioni che venivano generate. Sono state create due classi, "PlatformException" e "StrategiaException", che estendono entrambe "RuntimeException" di java: queste due eccezioni vengono lanciate ogni qual volta ci sia un problema negli oggetti piattaforma o strategia cercando di restituire all'utente anche la motivazione che ha causato l'eccezione.
Nel complesso crediamo di aver scritto un buon software, da un lato robusto ma anche semplice e facile da comprendere, anche grazie allo studio preliminare fatto sull'architettura e sui design pattern implementati (vedi capitolo 4).

6.2 Test

Il focus dei test è stato rivolto a quelle classi con metodi e operazione di calcolo matematico, i test (scritti durante lo sviluppo del codice) vogliono assicurare il coretto funzionamento dei metodi principalmente appartenenti alle strategie.
Sin da subito è stato chiara l'impossibilità di una copertura totale del codice (non si possono testare i metodi che stampano a video, le http request, il main, ecc.); abbiamo comunque raggiunto una buona copertura specialmente nelle classi strategia (MediaMobile, ROC). Si è prestata particolare attenzione a coprire tutte le decisioni presenti nei metodi testati garantendo che il software funzioni correttamente qualsiasi sia la condizione che si verifica.

Per i casi di test è stato utilizzato jUnit nella sua quarta versione; di seguito i metodi che abbiamo testato:

- Trade: sono stati testati i metodi getBuyPrice, getSellPrice e getProfit;
- Dato: sono stati testati tutti i metodi (di tipo get) ad eccezione di stampa. Il test verifica che i metodi diano in ritorno i valori utilizzati per costruire l'oggetto;
- GestoreFile: è stato testato il metodo con il quale avviene il caricamento di un file ed è stato esaminato se i valori salvati nel vettore tarde (scelti a campione) fossero quelli attesi;
- Medi Mobile: sono stati implementati due diversi casi di test: nel primo si verifica la corretta esecuzione dei metodi per il calcolo della strategia e controlla che il vettore risultante (stato) sia corretto. Il secondo caso vuole invece vericare se, in presenza di determinati eventi (media corta maggiore o uguale alla lunga o somma delle due media superiore alla dimensione del vettore di dati), si verificano le eccezioni previste.
- ROC: per la classe ROC è bastato testare il solo metodo calcolaStrategia per verificare che tutto il codice operi correttamente; il verificarsi delle eccezioni per questa classe, si è limitato al singolo caso in cui la somma tra i valori periodoMedia e periodoRoc fosse maggiore della dimensione del vettore dei dati.

Copertura del codice: 38.3%.

6.3 Manutenzione e refactoring

Durante il progetto (in particolare alla fine di ogni prototipo) sono state fatte delle operazioni di manutenzione per rendere più leggibile e comprensibile il software, in particolare adattando i nomi delle variabili a degli standard. E' stata inoltre fatta un'operazione di refactor (utilizzando la funzione di eclipse) al package "backtest" (in precedenza "test") e la classe "Backtest" (in precedenza "Test") per non creare ambiguità fra le funzioni di backtest delle strategie e quelle di testing del software.
Su richiesta del prof. con una issues abbiamo spostato i casi di test in una source folder parallela (di nome test) con le classi di test divise negli stessi package presenti nella directory del codice. In questo il software ha una sua cartella indipendente.

Capitolo 7

Conclusioni

Nelle conclusioni vogliamo dare un feedbeck su quello che è stato per noi lo sviluppo del progetto e sul risultato che abbiamo ottenuto.
Per la prima volta abbiamo utilizzato un approccio ingegneristico allo sviluppo software, seguendo in primis lo schema "requirements, design, implementation, testing, manutenzione". Lo sviluppo a prototipi ci ha anche permesso di sperimentare e osservare l'evolversi del software da una sua prima versione fino al software "completo" (in realtà, si potrebbe continuare a sviluppare per mesi, aggiungendo nuove piattaforme, strategie e funzionalità).
Riteniamo che aver studiato questo progetto nei minimi dettagli, partendo da una buona scrittura dei requisiti e sviluppando l'architettura prima dell'implementazione ci abbia portati ad un prodotto (software) robusto ed efficace. Il nostro auspicio è che sia apprezzato anche dallo "stakeholder".