**HW2:**

Il presente documento descrive l'estensione del progetto realizzato per HW1, concentrandosi sulle nuove funzionalità implementate per soddisfare i requisiti di HW2. Tramite l’utilizzo del pattern cqrs abbiamo migliorato la gestione delle read e delle write e tutto ciò è stato fatto in un file cqrs.py che viene richiamato nei file sui quali si scrive e si legge dal db, quindi il dataCollector, il server e infine l’AlertSystem. In questo secondo homework sono stati inseriti due ulteriori componenti oltre a quelli gia presenti nell’HW1 ed essi sono i seguenti:

-Alert system

-alert Notifier system

Per quanto riguarda il server è stato leggermente modificato per un controllo della cache migliore, infatti abbiamo aggiunto una funzione che all’inizio dell’esecuzione di esso sincronizza la tabella utenti con la cache. Inoltre, nella tabella utenti sono stati inseriti due campi che sono high\_value e low\_value ed in funzione di essi tramite kafka avverrà o meno l’invio di messaggi all’utente interessato ad un determinato ticker dicendo che il valore di tale ticker ha superato la soglia superiore oppure è sceso sotto la soglia inferiore, se ovviamente l’utente ha deciso di specificare uno o entrambe le soglie. Le soglie possono essere inserite sia durante la registrazione sia se l’utente vuole aggiornare esse. Chiaramente high value deve essere maggiore di low value. Il sistema è stato arricchito con un meccanismo di notifica basato su Kafka, una gestione migliorata del database con il pattern CQRS, e nuove funzionalità utente per gestire soglie sui valori monitorati. Dentro questo file cqrs.py creiamo la connessione al db e definiamo le classi inerente ai command e ai “servizi” di lettura e di scrittura e questo l’abbiamo fatto per le due tabelle presenti nel db cioè utenti e dati\_finanziari. Kafka è stato “integrato” per gestire la comunicazione tra i componenti che sono il **DataCollector, AlertSystem, AlertNotifierSystem,** i quali comunicano in questo modo:

* **DataCollector:** il datacollector per prima cosa preleva gli utenti dal db, poi recupera i dati da yahoofinance, inserisci i seguenti dati nella tabella dati\_finanziari e dopo di ciòfunge da producer Kafka, scrivendo messaggi sul topic to-alert-system al termine di ogni aggiornamento. Il messaggio che produce contiene email, ticker e valore. \*
* **AlertSystem:** invecelui funge da consumer e producer infattilegge i messaggi da to-alert-system, verifica le soglie prendendo le soglie dalla tabella utenti nel db e in caso di superamento, invia un messaggio al topic to-notifier dove tale messaggio contiene email,ticker e condizione. \*
* **AlertNotifierSystem:** esso funge da consumer, infatti riceve i messaggi dal topic to-notifier e invia email all’utente dicendo che il valore del ticker di suo interesse ha superato la soglia da lui definita.

(\*ovviamente letture e scritture del db avvengono tramite cqrs)

Visto l’inserimento di questi componenti ovviamente serve l’aggiunta di nuovi container che sono l’Alert System e l’alertNotifierSystem, ma non sono gli unici container inseriti visto che serve ovviamente il broker, infatti è stato inserito il container kafka, quindi un altro componente aggiuntivo, che funge da broker e di conseguenza, essendoci kafka, abbiamo inserito un container Zookeeper.

Di seguito è riportato un diagramma architetturale che mostra quello che succede con le modifiche richieste dall’homework:

DATACOLLECTOR

(PRODUCER SUL TOPIC TO-ALERT-SYSTEM)

SERVER SMTP

ALERT-NOTIFIER-SYSTEM

(CONSUMER SU TO-NOTIFIER)

ALERTSYSTEM

(CONSUMER SU TO-ALERT-SYSTEM E PRODUCER SU TO-NOTIFIER)

KAKFA BROKER

**JOSE’ LUIS AQUILA**

**VINCENZO COSTANZA**