

SOUTHEAST EUROPEAN UNIVERSITY

FACULTY OF CONTEMPORARY SCIENCE AND TECHNOLOGY

STUDY PROGRAM: “COMPUTER ENGINEERING”

SEMINAR PAPER FROM THE SUBJECT: SOFTWARE DESIGN AND ARCHITECTURE

Theme: “Real-Time price drop alert platform”

SUBJECT LECTURE: STUDENT:

Asst. Prof. Dr. Nuhi Besimi Irfan Uruchi

TETOV, MAY 2025

# Introduction

## Problem statement

Stock prices can move several percent in a matter of seconds where large trading firms handle this speed with premium data feeds and automated orders, but most individual investors rely mostly on web dashboards or delayed e-mail alerts supplied by their broker. And because these tools refresh slowly and cannot check user-defined conditions in real time, many traders learn about a favorable price only after the market has already reversed.

## Example

A student investor likes to buy Samsung stocks if the price falls below 38$. But during a two hour laboratory session the share dips briefly to 34$ but then recovers back to around 40$ and closes the day higher, when the student finally reloads the trading app the opportunity has passed where not because the analysis was incorrect, but because the alert system failed to react fast enough.

## Why the issue deserves attention

It has a financial impact where missing a 1% intraday swing even a few times a semester can add up to a meaningful sum over the several years of study and early employment, another reason is cognitive load where reliable, real-time alerts let users focus on classes or work instead of repeatedly checking price screens. And also, another benefit is a better market data, where aggregated and anonymous alert thresholds reveal where many small investors consider a share “fair value” information that can improve liquidity planning for brokers.

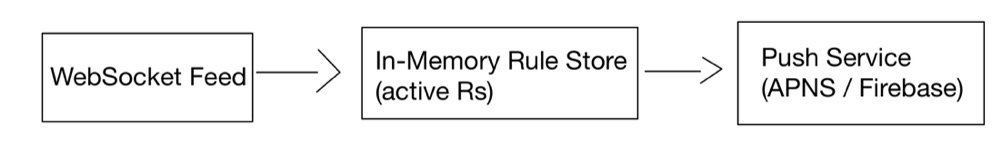
For pulling this off the system must keep an eye on live market tricks, run thousands of “is it at my price yet?” checks every few seconds and tap users on the shoulder in under a few seconds. Where that calls for a speedy data stream, a database that will be scalable and a notification path that’s quick but still easy on the cloud bill and of course locked down on security.

# Related work

Real-time price-alert services are not new where some several public systems I found related to this project:

## 2.1 Robinhood Alert Pipeline

- The **Robinhood app**[(Robinhood)](https://robinhood.com/us/en/support/articles/price-alerts/?utm_source=chatgpt.com) hat steams live market tics(via WebSocket’s) into a in-memory evaluation where each user’s limits are kept in memory , and when a tick matches a even fires immediately through Apple Push or Firebase. The main trade-off is that of the RAM where they keep the “active” rules in memory, this is a fast system but they pay extra for RAM to keep the hot rules handy.



## 2.2 LeetCode “Design a Stock Alert System

* **LeetCode** **“design a stock-alert system” thread** [(LeetCode)](https://leetcode.com/discuss/post/5745135/design-stock-price-notification-system-b-trj4/) **which is a community blueprint that proposes a pub-sub backbone where first market producers write ticks to Kafka, then a rule engine reads them after loads user rules from Redis and filters machines. And in the end the matches go to a notifier service. They highlight two pain points one for sharding millions of user-symbol rules and deduplication to avoid alert storms**

A white rectangular sign with black text

AI-generated content may be incorrect.

## 2.3 Bhavin’s Micro-Trading Blog Architecture

* In this Medium post for **Hobby micro-trading blog** [(Micro-trading blog)](https://medium.com/%40datajedi/trading-system-design-using-microservices-256cda0dc60a) , Bhavin splits the workload into three tiny services: Quote Service, Strategy Service and Execution Service where each is stateless and behind a messaging queue. Although focused at automated trades rather than human alerts, the piece shows how **stateless quote processors** can scale horizontally behind a message queue.

A white rectangular sign with black text

AI-generated content may be incorrect.

## 2.4 One-Box Python Poll-and-Email Flow

* One-box Python tracker on GitHub [(One-box python tracker)](https://github.com/mustang519/STOCK-MARKET-PRICE-TRACK-AND-ALERT-SYSTEM?utm_source=chatgpt.com) it polls Yahoo Finance once a minute and sends e-mails when thresholds are crossed , because it’s a single-machine design it cannot handle thousands of users.

A black and white image of a computer

AI-generated content may be incorrect.

# Solution

## High-level view

The platform follows a stream-based with a micro-service design:

Where first it ingests continuous market feed, then on the right the system delivers a push, e-mail or SMS to the user. Everything in between is broken into small, single purpose service so that each part can scale or fail independently. Diagram for demonstration:

A diagram of a software flowchart

AI-generated content may be incorrect.

## End-to-end flow

When a user taps “Alert me if Samsung ≤ 38$ in the mobile app, the request travels through the API Gateway, where is authenticated and written to the Rules database. A copy goes straight to the Redis so system can act on it within seconds. Meanwhile a lightweight listener is sipping the broker’s WebSocket feed where every fresh tick is normalized and dropped onto a price-topic stream. Stateless rule-engine workers pick up each tick, glance at the matching rules in Redis and if the condition hits, then places an entry on the alert queue. The notification service pulls that entry, checks it hasn’t fired for the same user in the last few minutes and then pushes through Firebase, e-mail or SMS. Logs and metric flow into Grafana and any failed notification fall into a dead-letter queue for safe retry. From first market tick to the user’s phone vibration the path is typically withing a few seconds.

A diagram of a diagram

AI-generated content may be incorrect.

From the first market tick to user’s vibration, the path is withing a few seconds, a diagram for high level architecture:

