LAPORAN TUGAS BESAR

PEMROGRAMAN BEORIENTASI OBJEK

DISUSUN OLEH:

L0124054 - FARREL AZHAR VAHREZI

L0124042 - BAGAS ADITAMA SURYO. N.

L0124098 - FADHOLI RIZQI NOVANDRA. A.

PROGRAM STUDI S-1 INFORMATIKA

FAKULTAS TEKNOLOGI INFORMASI DAN SAINS DATA

UNIVERSITAS SEBELAS MARET

SURAKARTA

2025

BAB I

PENDAHULUAN

1.1. Latar Belakang Masalah

Perkembangan teknologi informasi di era revolusi industri 4.0 telah membawa dampak signifikan pada sektor ekonomi, khususnya di bidang keuangan (*Financial Technology* atau *Fintech*). Salah satu tren yang mencolok adalah pergeseran pola pikir masyarakat dari sekadar menabung (saving) menjadi berinvestasi (investing). Minat masyarakat terhadap instrumen investasi seperti saham, *forex*, hingga aset kripto terus menunjukkan peningkatan yang tajam, terutama di kalangan generasi muda yang semakin sadar akan pentingnya kebebasan finansial.

Namun, tingginya minat investasi ini tidak selalu berbanding lurus dengan tingkat literasi keuangan yang memadai. Banyak calon investor merasa terintimidasi oleh kompleksitas pasar modal, grafik analisis teknikal yang rumit, serta terminologi asing yang sulit dipahami. Selain itu, volatilitas pasar yang tinggi sering kali menjadi penghalang bagi pemula yang takut mengalami kerugian besar akibat kurangnya pengetahuan dan strategi perdagangan (trading) yang matang. Fenomena Fear of Missing Out (FOMO) sering kali menjebak investor pemula untuk mengambil keputusan impulsif tanpa dasar analisis yang kuat.

Masalah lain yang sering ditemui adalah antarmuka (user interface) pada aplikasi trading konvensional yang sering kali didesain untuk para profesional. Hal ini menciptakan hambatan masuk (barrier to entry) bagi masyarakat awam yang ingin memulai. Mereka membutuhkan sebuah ekosistem yang tidak hanya berfungsi sebagai alat transaksi, tetapi juga sebagai mentor digital yang mampu membimbing mereka memahami dinamika pasar.

Berangkat dari permasalahan tersebut, pengembangan aplikasi Neostock dirancang sebagai solusi komprehensif. Aplikasi ini bertujuan untuk menjembatani kesenjangan antara keinginan berinvestasi dan kemampuan teknis penggunanya. Fokus utama dari pengembangan aplikasi ini adalah penyederhanaan proses investasi melalui antarmuka yang intuitif, sehingga siapa pun dapat memulai investasi dengan mudah.

Lebih dari sekadar alat transaksi, aplikasi ini juga menekankan pada aspek edukasi mendalam. Melalui fitur-fitur pembelajaran yang terintegrasi—seperti simulasi pasar, materi analisis fundamental dan teknikal, serta manajemen risiko—aplikasi ini diharapkan dapat membantu pengguna untuk tidak hanya menjadi investor yang pasif, tetapi juga trader yang cerdas dan mandiri. Dengan demikian, aplikasi ini hadir untuk mendemokratisasi akses investasi, membuatnya lebih mudah, aman, dan edukatif bagi semua lapisan masyarakat.

1.2. Rumusan Masalah

Berdasarkan latar belakang yang telah diuraikan di atas, maka rumusan masalah dalam pembuatan aplikasi ini adalah sebagai berikut,

1. Bagaimana merancang UI dan pengalaman pengguna UX aplikasi *trading* yang intuitif agar mudah dipahami oleh investor pemula?
2. Bagaimana mengimplementasikan fitur edukasi yang efektif di dalam aplikasi untuk membantu pengguna mempelajari analisis teknikal, fundamental, dan manajemen risiko secara lebih mendalam?
3. Bagaimana membangun sistem aplikasi yang mampu mengintegrasikan fungsi transaksi investasi dengan fitur pembelajaran (simulasi) secara *real-time* dan akurat?

1.3. Tujuan Pengembangan

Tujuan dari pembuatan aplikasi Neostock adalah,

1. Mempermudah akses investasi, merancang, dan membangun aplikasi *trading* dengan navigasi yang sederhana untuk meminimalkan hambatan teknis bagi masyarakat awam yang ingin mulai berinvestasi.
2. Meningkatkan literasi keuangan dan menyediakan sarana edukasi yang terstruktur di dalam aplikasi, sehingga pengguna dapat belajar memahami dinamika pasar dan strategi *trading* yang benar sambil melakukan praktik.
3. Membantu pengambilan keputusan dengan menciptakan alat bantu analisis yang mudah dibaca agar pengguna dapat mengambil keputusan investasi yang didasarkan pada data dan pengetahuan, bukan sekadar spekulasi.

1.4. Solusi yang Ditawarkan

Untuk mengatasi permasalahan rendahnya literasi keuangan dan kompleksitas platform investasi yang ada saat ini, aplikasi kami menawarkan solusi melalui pendekatan teknologi yang berfokus pada pengalaman pengguna (User Experience) dan edukasi berkelanjutan. Berikut adalah rincian solusi yang dikembangkan:

1.4.1. Desain Antarmuka Adaptif (*Adaptive User Interface*)

Aplikasi ini dirancang dengan konsep User-Centric yang memprioritaskan kemudahan navigasi. Solusi ini diimplementasikan melalui fitur "Dual Mode Interface":

- *Guest Mode* (Mode Tamu): Tampilan yang disederhanakan, menghilangkan grafik teknikal yang rumit, dan fokus pada informasi fundamental dasar serta tombol transaksi yang jelas. Bahasa yang digunakan menghindari jargon teknis yang membingungkan.
- *Pro Mode* (Mode Lanjutan): Pengguna dapat beralih ke mode ini ketika sudah mahir, yang menampilkan *charting tools* lengkap, indikator teknikal, dan data *order book* mendalam.

1.4.2. Integrasi Fitur *Paper Trading* (Simulasi Real-Time)

Untuk menjawab ketakutan pemula terhadap risiko kerugian, aplikasi menyediakan fitur simulasi perdagangan menggunakan dana virtual (dummy money).

- Data pergerakan harga diambil secara *real-time* dari pasar sesungguhnya.
- Pengguna dapat mempraktikkan strategi *trading* tanpa risiko finansial sedikitpun sebelum terjun menggunakan uang asli.
- Riwayat transaksi simulasi akan dievaluasi oleh sistem untuk memberikan *feedback* performa pengguna.

BAB II

SOURCE CODE DAN ANALISIS

2.1. Source Code

Link Source Code: *https://github.com/jempolbagas/Neostock.git*

2.2. Penjelasan Source Code:

Dalam pengembangan aplikasi Neostock, kami menerapkan empat pilar utama Pemrograman Berorientasi Objek (PBO) secara ketat. Penerapan ini bukan sekadar pemenuhan syarat tugas, melainkan strategi fundamental untuk memastikan kode program bersifat modular, aman, mudah dipelihara (maintainable), dan dapat dikembangkan (scalable) di masa depan. Berikut adalah analisis mendalam mengenai penerapan masing-masing pilar:

2.2.1. *Encapsulation* (Enkapsulasi)

Enkapsulasi adalah mekanisme membungkus data (variabel) dan kode yang memanipulasinya (metode) menjadi satu unit tunggal. Dalam konteks aplikasi keuangan seperti Neostock, enkapsulasi memegang peranan vital untuk menjaga integritas data finansial.

Kami menerapkan konsep ini pada seluruh kelas di dalam paket Model. Seluruh atribut sensitif, seperti saldo, password, dan portfolio, dideklarasikan dengan access modifier private. Hal ini mencegah akses langsung atau modifikasi sembarangan dari luar kelas yang dapat menyebabkan inkonsistensi data. Akses terhadap data tersebut dikontrol secara ketat melalui metode public (getter dan setter) yang telah dilengkapi dengan logika validasi.

Dengan pendekatan ini, objek Akun menjamin bahwa saldonya tidak akan pernah bernilai negatif akibat kesalahan input eksternal, menjaga validitas keadaan objek (object state) sepanjang siklus hidup aplikasi.

2.2.2. *Inheritance* (Pewarisan)

Pewarisan digunakan untuk menciptakan hierarki kelas yang logis, memungkinkan penggunaan kembali kode (code reusability), dan menurunkan sifat atau perilaku dari superclass ke subclass. Dalam proyek ini, implementasi pewarisan terlihat dominan pada struktur GUI dan manajemen kesalahan (Error Handling).

Analisis Implementasi:

- Integrasi Framework (JavaFX):

  Class utama StockTradingApp mewarisi (extends) class Application dari pustaka JavaFX. Dengan mewarisi kelas ini, aplikasi kami secara otomatis

mendapatkan kemampuan manajemen siklus hidup aplikasi GUI (seperti metode init(), start(), dan stop()) tanpa perlu menulis kode dasar dari nol.

- Granularitas Penanganan Error:

  Kami mengembangkan sistem custom exception yang mewarisi class Exception. Hal ini dilakukan agar aplikasi dapat membedakan jenis error secara spesifik dan memberikan respon yang tepat kepada pengguna, alih-alih hanya menangkap error generik.

2.2.3. *Polymorphism* (Polimorfisme)

Polimorfisme memungkinkan sebuah objek memiliki banyak bentuk. Dalam aplikasi ini, kami menggunakan Overriding untuk mengubah perilaku method bawaan Java.

Contoh Implementasi:

- Method toString():

  Pada class Transaksi.java, kami melakukan override method toString() untuk menghasilkan format cetak riwayat transaksi yang rapi dan mudah dibaca oleh manusia.

- JavaFX Cell Factory:

  Pada PortfolioView.java, kami melakukan override method updateItem pada TableCell untuk mengubah tampilan angka (warna hijau untuk profit, merah untuk rugi) secara dinamis.

2.2.4. *Abstraction (Abstraksi)*

Abstraksi diterapkan untuk menyembunyikan kerumitan logika internal. Pengguna kelas MarketService atau TradingService tidak perlu tahu bagaimana thread bekerja atau bagaimana data disimpan ke JSON, mereka cukup memanggil method yang tersedia.

2.3. Arsitektur Aplikasi dan Penjelasan Kode

Aplikasi ini dibangun menggunakan pola arsitektur yang memisahkan antara Tampilan (GUI), Logika Bisnis (Service), dan Data (Model/Repository).

2.3.1. Model Layer (Data Representation)

Paket ini berisi representasi objek nyata:

- Akun: Menyimpan data pengguna, saldo, portfolio (HashMap), dan riwayat transaksi.
- Saham: Menyimpan data pasar seperti harga sekarang, harga pembukaan, dan histori harga untuk grafik.

- Portfolio: Menyimpan data kepemilikan saham, termasuk perhitungan Average Down (rata-rata harga beli) saat pengguna membeli saham yang sama berulang kali.

2.3.2. Service Layer (Business Logic)

Ini adalah otak dari aplikasi yang menangani aturan bisnis:

- MarketService.java (Simulasi Pasar Real-Time) Kelas ini menggunakan konsep Multithreading. Sebuah daemon thread berjalan di latar belakang untuk memperbarui harga saham secara acak setiap 10 detik, mensimulasikan fluktuasi pasar nyata.

- TradingService.java (Transactional Logic) Menangani pembelian dan penjualan. Fitur kuncinya adalah Transaction Rollback. Jika data berhasil diubah di memori tetapi gagal disimpan ke file JSON, sistem akan membatalkan perubahan saldo dan portfolio untuk mencegah korupsi data.

2.3.3. Data Persistence (Penyimpanan Data)

- DataManager.java: Menggunakan library GSON untuk melakukan serialisasi objek Java ke format JSON (neostock.json). Ini memungkinkan data pengguna tetap tersimpan meskipun aplikasi ditutup.

2.3.4. GUI Layer (JavaFX)

Kami menggunakan JavaFX tanpa FXML (Pure Java Code) untuk fleksibilitas maksimal dalam *styling*.

- Styling (CSS): Tampilan futuristik "Quantum" diatur melalui styles.css dengan tema warna gelap (dark mode), gradasi neon (cyan/green), dan elemen glassmorphism.

- Dynamic Views:
  - MarketView: Menampilkan grafik saham (LineChart).
  - TradeView: Formulir jual beli dengan validasi input.
  - PortfolioView: Tabel kepemilikan aset dengan perhitungan Profit/Loss real-time.

BAB III

FITUR UTAMA APLIKASI DAN PENJELASAN SOURCE CODE

3.1. Fitur Autentikasi Pengguna dan Keamanan Data

3.1.1. Deskripsi Fitur

Fitur autentikasi merupakan gerbang utama keamanan aplikasi yang memastikan bahwa hanya pengguna terdaftar yang dapat mengakses fitur perdagangan. Sistem ini mencakup proses registrasi akun baru, validasi kredensial saat login, serta manajemen sesi pengguna aktif. Selain itu, fitur ini menjamin persistensi data, di mana saldo dan portfolio pengguna tersimpan aman secara lokal dan dapat dimuat kembali saat aplikasi dibuka.

3.1.2. Penjelasan Source Code dan Implementasi

Logika autentikasi terpusat pada kelas AuthService.java dan UserRepository.java. Kami menerapkan pendekatan validasi berlapis (layered validation) untuk menjaga integritas data sejak awal input.

A. Validasi Input Ketat (AuthService.java):

Pada metode createAccount, aplikasi tidak sekadar menerima data mentah. Setiap input diverifikasi menggunakan kondisional if yang ketat. Jika input tidak memenuhi syarat, sistem akan melempar custom exception yang spesifik, bukan generic exception, untuk memberikan umpan balik yang jelas.

- Username & Password: Panjang karakter divalidasi (username min. 4, password min. 6).
- Saldo Awal: Aplikasi mewajibkan deposit minimal (misal: Rp 100.000) menggunakan perbandingan BigDecimal.
- Duplikasi Data:
  Sistem memanggil userRepository.existsByUsername(username) untuk mencegah pembuatan akun ganda dengan username yang sama.

B. Manajemen Data Persisten (DataManager.java & UserRepository.java):

Untuk penyimpanan data, kami menggunakan format JSON yang dikelola oleh library GSON.

- Kelas DataManager bertindak sebagai Data Access Object (DAO) tingkat rendah yang menangani operasi I/O file (FileReader/FileWriter).
- Kami menggunakan teknik serialisasi objek, di mana objek Java Akun (beserta Portfolio dan RiwayatTransaksi di dalamnya) dikonversi menjadi string JSON secara otomatis.

- Saat aplikasi dimulai (startup), UserRepository memuat seluruh data JSON ke dalam ConcurrentHashMap di memori. Penggunaan ConcurrentHashMap dipilih untuk menjamin keamanan akses data (thread-safety) jika ada operasi baca/tulis yang terjadi bersamaan dari thread yang berbeda.

3.2. Simulasi Pasar Saham Real-Time (Live Market Engine)

3.2.1. Deskripsi Fitur

Salah satu keunggulan utama Neostock adalah kemampuannya mensimulasikan dinamika pasar saham secara real-time. Harga saham tidak statis, melainkan bergerak naik atau turun secara otomatis setiap interval waktu tertentu. Fitur ini dirancang untuk melatih mental pengguna dalam menghadapi volatilitas pasar, memberikan pengalaman psikologis yang mirip dengan trading sesungguhnya tanpa risiko finansial.

3.2.2. Penjelasan Source Code dan Implementasi

Mesin simulasi ini dibangun di atas kelas MarketService.java dan PasarSaham.java dengan memanfaatkan konsep Multithreading dan pola desain Observer.

A. Mekanisme Volatilitas Harga (PasarSaham.java):

Setiap objek Saham memiliki metode updateHarga() yang dipicu secara berkala. Di dalamnya, terdapat algoritma pengacak (java.util.Random) yang menentukan persentase perubahan harga antara -5% hingga +5% pada setiap tick.

- Logika ini memastikan pergerakan harga tidak terprediksi namun tetap dalam batas wajar.
- Harga baru dihitung dengan rumus: Harga Baru = Harga Lama + (Harga Lama * Persentase Acak).
- Kami menambahkan batasan (guard clause) agar harga tidak pernah jatuh di bawah nilai minimum (misal: Rp 50), menjaga realisme pasar.

B. Concurrency dengan Daemon Thread (MarketService.java):

Agar pembaruan harga tidak membekukan antarmuka pengguna (UI Freeze), proses ini dijalankan di Thread terpisah (Background Thread).

- Kami membuat sebuah Thread baru yang berjalan dalam infinite loop (while(true)).
- Di dalam loop, thread "tidur" (Thread.sleep(10000)) selama 10 detik, lalu bangun untuk memperbarui seluruh harga saham.
- Thread ini diset sebagai Daemon Thread, yang artinya ia akan otomatis mati ketika aplikasi utama ditutup, mencegah kebocoran memori (memory leak).

C. Sinkronisasi UI (StockTradingApp.java & MarketView.java):

Tantangan utama dalam multithreading adalah memperbarui UI dari thread non-UI, yang dilarang dalam JavaFX. Kami mengatasi ini dengan membungkus kode pembaruan tampilan di dalam blok Platform.runLater(() -> { ... }). Ini menginstruksikan JavaFX Application Thread untuk melakukan refresh grafik dan tabel harga segera setelah antrian tugasnya kosong, menghasilkan animasi pergerakan harga yang mulus.

3.3. Mesin Transaksi Perdagangan (Trading Engine)

3.3.1. Deskripsi Fitur

Inti dari aplikasi ini adalah kemampuan untuk melakukan transaksi beli (Buy) dan jual (Sell). Fitur ini bukan sekadar pengurangan atau penambahan angka, melainkan melibatkan perhitungan matematis presisi untuk biaya transaksi, rata-rata harga beli (average price), serta validasi saldo dan kepemilikan aset yang ketat.

3.3.2. Penjelasan Source Code dan Implementasi

Logika transaksi ditangani oleh TradingService.java yang bekerja sama dengan model Akun dan Portfolio.

A. Presisi Keuangan dengan BigDecimal:

Kesalahan umum dalam aplikasi pemula adalah menggunakan tipe data double atau float untuk uang, yang dapat menyebabkan masalah presisi desimal (contoh: 0.1 + 0.2 hasilnya 0.30000000004).

- Dalam TradingService, seluruh variabel moneter (harga saham, saldo, total bayar) menggunakan kelas java.math.BigDecimal.
- Operasi matematika dilakukan menggunakan metode objek (.add(), .subtract(), .multiply()) dengan pengaturan skala desimal (RoundingMode.HALF_UP) untuk pembulatan yang akurat sesuai standar akuntansi.

B. Logika Transaksi Atomik dan Rollback:

Kami menerapkan prinsip Atomicity dalam transaksi. Artinya, sebuah transaksi harus berhasil sepenuhnya atau gagal sepenuhnya; tidak boleh setengah-setengah.

- Proses Beli:
  1. Cek saldo (if saldo < totalBiaya).
  2. Kurangi saldo di memori.
  3. Update portfolio (tambah lot atau buat entry baru).
  4. Simpan ke File JSON (auth.saveData()).

- Mekanisme Rollback: Jika langkah ke-4 (penyimpanan file) gagal—misalnya karena disk full atau error I/O—blok catch akan memicu prosedur Rollback. Sistem secara otomatis mengembalikan saldo yang sudah terpotong dan menghapus saham yang baru ditambahkan di memori. Hal ini menjamin bahwa data di aplikasi tidak akan pernah tidak sinkron dengan data di penyimpanan.

C. Manajemen Portfolio (Portfolio.java):

Saat pengguna membeli saham yang sudah dimiliki, sistem menerapkan logika Average Down.

- Rumus: Harga Rata-Rata Baru = ((Harga Lama * Lot Lama) + (Harga Baru * Lot Baru)) / Total Lot.
- Kode ini memastikan bahwa nilai investasi pengguna tercatat secara adil berdasarkan harga beli rata-rata, yang krusial untuk perhitungan profit/loss yang akurat.

3.4. Manajemen Portfolio dan Pelaporan (Reporting)

3.4.1. Deskripsi Fitur

Fitur ini berfungsi sebagai dasbor kinerja investasi pengguna. Aplikasi menyajikan tabel portfolio yang dinamis, menampilkan aset yang dimiliki, nilai pasar saat ini, serta keuntungan atau kerugian yang belum direalisasikan (Unrealized P/L). Selain itu, fitur ekspor laporan memungkinkan pengguna mengunduh riwayat aktivitas mereka ke dalam file eksternal untuk keperluan arsip atau analisis lebih lanjut.

3.4.2. Penjelasan Source Code dan Implementasi

Fitur ini menggabungkan logika tampilan di PortfolioView.java dan logika I/O di LaporanManager.java.

A. Kalkulasi Profit/Loss Real-Time:

Pada PortfolioView, tabel tidak hanya menampilkan data statis. Setiap kali harga pasar berubah (dipicu oleh MarketService), tabel menghitung ulang kolom "Nilai Sekarang" dan "Profit/Loss".

- Logika: Profit = (Harga Pasar Sekarang - Harga Beli Rata-Rata) * Jumlah Lembar.
- Kami menggunakan Custom Cell Factory pada JavaFX TableView. Ini memungkinkan kode untuk memeriksa nilai profit secara dinamis: jika positif, teks diwarnai hijau; jika negatif, teks diwarnai merah. Logika visual ini membantu pengguna memahami kondisi portofolionya dalam sekali pandang.

B. Generator Laporan Teks (LaporanManager.java):

Kelas ini bertanggung jawab untuk mencetak data kompleks menjadi format yang mudah dibaca manusia (human-readable).

- Menggunakan PrintWriter dan FileWriter untuk membuat file .txt.
- Kami memanfaatkan String.format() secara ekstensif untuk merapikan tata letak kolom (menggunakan padding spasi) sehingga laporan terlihat seperti tabel yang rapi meskipun hanya berupa teks biasa.
- Laporan mencakup tiga segmen utama: Header Informasi Akun, Tabel Ringkasan Portfolio (dengan total valuasi aset), dan Daftar Riwayat Transaksi Kronologis.

BAB IV

CHALLENGE DAN OPPORTUNITY

4.1. Challenge (Tantangan) dari Program StockTradingApp.java:

1. Kompleksitas Manajemen State

   Aplikasi memiliki banyak state yang harus dijaga sinkronisasi (akun aktif, dashboard, market data, portfolio, dll.). Ada kemungkinan risiko memory leak karena listener dan view yang tidak selalu dibersihkan dengan benar saat berpindah layar.

2. Thread Safety

   MarketService berjalan di thread terpisah untuk update harga saham, namun UI di-update di thread JavaFX (Platform.runLater). Berpotensi race condition jika ada akses konkuren ke data akun atau portfolio.

3. Error Handling yang Tidak Konsisten

   Beberapa bagian menggunakan try-catch dengan Exception umum, sementara lainnya tidak menangani error sama sekali. Validasi input di form registrasi/login masih sederhana dan rentan terhadap input yang tidak valid.

4. Keterikatan dengan JavaFX

   UI sepenuhnya dibangun dengan JavaFX, sehingga sulit untuk mengganti framework UI di masa depan tanpa menulis ulang kode. CSS styling yang terikat langsung di kode Java mengurangi fleksibilitas dan maintainability.

5. Skalabilitas Data

   DataManager dan UserRepository menggunakan penyimpanan lokal (file), yang tidak cocok untuk banyak pengguna atau data besar. Tidak ada mekanisme caching untuk market data, sehingga setiap update mungkin memperlambat UI.

6. Keamanan

   Password disimpan tanpa enkripsi (hanya di-hash). Tidak ada autentikasi dua faktor atau session management yang kuat.

7. Maintainability Kode UI

   Kode UI sangat panjang dan padat dalam satu kelas (StockTradingApp), sulit untuk di-test dan dimodularisasi. Banyak duplikasi kode untuk pembuatan form dan tombol.

4.2. Opportunity (Peluang) untuk Pengembangan

1. Restrukturisasi Arsitektur (Layered/Modular) yaitu memisahkan business logic, data layer, dan presentation layer menggunakan pola desain MVP atau MVVM untuk meningkatkan keterbacaan dan perawatan kode.

2. Manajemen Dependensi (Dependency Injection) dengan mengadopsi framework Spring atau Dagger untuk pengelolaan dependensi yang lebih terstruktur dan mempermudah proses unit testing.

3. Optimalisasi Aliran Data (Real-time Streaming) dengan mengganti mekanisme polling dengan WebSocket untuk data pasar real-time serta integrasi langsung dengan penyedia data finansial eksternal.

4. Transisi ke Microservices, yaitu memecah fungsionalitas kritis menjadi layanan mikro independen untuk fleksibilitas pengembangan dan skalabilitas sistem.

5. Modernisasi Penyimpanan Data, migrasi ke solusi database RDBMS atau NoSQL yang didukung oleh ORM (Hibernate/JPA) untuk integritas dan kecepatan transaksi data.

6. Penguatan Keamanan (Security Hardening) dengan menerapkan standar enkripsi modern, otentikasi berbasis token (JWT), dan pencatatan jejak audit (logging) untuk setiap aktivitas trading.

7. Pembaruan Antarmuka Pengguna (UI/UX). Mengembangkan antarmuka yang responsif dan lintas platform, baik melalui JavaFX maupun teknologi web modern/desktop wrapper (Electron).

8. Otomatisasi Pengujian (QA Automation) dengan membangun cakupan tes menyeluruh meliputi Unit Test, Integration Test, dan UI Test (TestFX) untuk menjamin stabilitas rilis.

9. Deployment Berbasis Cloud. Menggunakan Docker container dan infrastruktur cloud dengan fitur auto-scaling untuk menangani lonjakan trafik pengguna.
10. Ekspansi Fitur Trading dengan menyediakan fitur analisis teknikal canggih, charting tools (TradingView), dan dukungan untuk algorithmic trading.
11. Sistem Monitoring Terintegrasi. Mengimplementasikan dashboard analitik dan monitoring stack (ELK) untuk pengawasan kesehatan sistem secara real-time.
12. Ekosistem Multi-Platform. Mengembangkan aplikasi mobile (Android/iOS) yang terintegrasi penuh dengan backend utama untuk aksesibilitas pengguna.