perplexity

# Upgraded Prompt: “BTST-Next-Day Strength After Weak-Hands Shakeout” (India Equities, NSE/BSE)

Main takeaway: This upgraded brief replaces same-day BTST chasing with a next-day continuation framework that deliberately targets prior-day BTST candidates where weak hands likely exited and supply got absorbed. It systematically filters for post-shakeout strength with quantified criteria, improving odds of upward follow-through rather than profit-booking dips.

Use this exact prompt:

Act as a dedicated team of India-focused equity analysts, market microstructure specialists, and systematic swing traders. Each trading day, 30–60 minutes before close, deliver 1–3 high-probability “BTST-Next-Day Continuation” stock ideas for NSE/BSE that explicitly exploit prior-day BTST unwind and weak-hands shakeout. Do not chase same-day BTST spikes. Instead, start from yesterday’s BTST-like movers and identify those that absorbed supply and are primed for an upward second-leg.

Research Framework and Output Template

1. Macro & Market Snapshot (concise, only if it affects setups)

- Indices & breadth: Nifty/Bank Nifty, Advance-Decline, sector leadership/laggards.
- Global cues: FII/DII cash flow trend, USDINR, U.S./Asia, crude, yields.
- Event risk: RBI policy, major earnings, sector regulations, geopolitics impacting India.

2. Universe Construction: Prior-Day BTST Candidates

- Start with stocks that, on Day-1 (yesterday), showed any of:
    - Late-session surge: 2:45–3:30 pm up-move > X% with volume > 1.5–2.5× 20-day median.
    - Breakout above 20/50 DMA or multi-day base with wider-than-usual range (True Range > 1.5× 20-day ATR).
    - News/catalyst spike: earnings surprise, order win, policy approval, coverage initiation/upgrade.
- Liquidity floor: Avg daily turnover ≥ ₹Y cr, free-float mkt cap ≥ ₹Z cr, median bid-ask spread ≤ 20 bps.

3. Weak-Hands Shakeout Diagnostics (Today, Day-2 intraday)
   Identify Day-1 movers where weak-hands likely exited and stronger hands absorbed supply. Require at least 4 of the following 7 conditions:

A) Open-Drive Absorption:

- Gap down or flat open within −0.5% to −1.5% versus Day-1 close, followed by absorption: first 15–30 min shows down-ticks on declining per-trade volume and rising passive bids; volume-weighted average price (VWAP) recaptured within first hour.

B) Low-Quality Holder Flush:

- Early long liquidation: elevated order count, smaller average trade size, and higher retail-like odd-lot prints in first 30–60 minutes, then normalization with larger average trade size later.

C) Pullback Character:

- Intraday pullback depth ≤ 38.2%–50% of Day-1 range; wicks > bodies on red candles near prior resistance-turned-support; cumulative delta improves even if price is flat.

D) Supply Exhaustion:

- Sequential decline in sell-initiated volume across 5/15-min bars; upticks occur on higher participation rate; negative ticks fail to push below prior swing lows.

E) Inventory Transfer at Reference Levels:

- Clean retest of Day-1 breakout level or previous day's VWAP/value area high (VAH) with rapid rejection and regain; low time spent below.

F) Options/Derivatives Tell (if F&O eligible):

- PCR rising toward 0.9–1.2 with calls written closer to price and declining IV; or short-covering in futures: OI down, price up; or healthy roll on rising price.

G) Delivery & Float Friction (from prior session data):

- Elevated delivery percentage vs 3-month median, rising delivery turnover, and reduction in intraday churn-to-delivery ratio, indicating stronger hands holding.

4. Continuation Bias Confirmation (End of Day-2, late session)

- Price back above intraday VWAP and near day high into 2:45–3:30 pm.
- 5/15-min MACD or RSI(14) positive divergence vs morning lows; or 20-EMA (15-min) reclaimed and acting as dynamic support.
- Volume on up-bars > down-bars in afternoon session; cumulative delta positive.
- No fresh negative news; catalyst path still valid.

5. Disqualifiers (exclude even if above criteria met)

- Wide gap up > 2% at Day-2 open followed by distribution under VWAP most of the day.
- Spreads widen > 35 bps or depth deteriorates into close.
- Abnormal block prints offloaded at bid near close.
- Management/SEBI filings indicating negative updates, pledge increases, or guidance cuts.

6. Stock Picks Overview (for each 1–3 names)

- Ticker & Sector Theme: Include sector rotation context.
- Setup Summary (1–2 lines): Day-1 impulse + Day-2 shakeout absorption + late-session regain of control.

- Microstructure Evidence:
  - Day-2 early liquidation signs (avg trade size, odd-lot intensity), VWAP regain timing.
  - Cumulative delta trend; up-volume vs down-volume ratio in PM session.
  - Delivery % trend (Day-1 vs 3M median), F&O OI/IV/PCR if applicable.
- Technicals:
  - Key levels: Day-1 high/low, Day-2 VWAP, 20/50 DMA, breakout level, Fibonacci pullback zone.
  - Candles: Wick-to-body at key levels, inside-day or NR7 after expansion, momentum regime.
- Catalyst State: Earnings/orders/policy/regulatory; whether thesis remains active.
- Liquidity & Risk:
  - 5/20-day volume medians, turnover, free-float turnover ratio.
  - ATR(14), expected gap risk, settlement considerations.
- Price Plan:
  - Entry Zone: Prefer pullback-to-strength entries (regain above Day-2 VWAP or breakout retest).
  - Target 1/2: Day-1 high, measured move equal to 0.5–1.0x Day-1 range.
  - Stop-Loss: Below Day-2 PM higher low or below Day-1 breakout level; size via ATR 1.0–1.5x.
  - Position Sizing: Volatility-adjusted; cap single-name risk ≤ 0.5–0.8% of equity.
- Execution:
  - Use limit-if-touched around reclaimed VWAP; avoid chasing far above PM high.
  - Avoid low-depth opens; consider first-hour only if absorption is clear.

7. Strategic Perspective

- Why the shakeout likely completed: evidence of weak-hands exit and stronger-hands inventory.
- Probability of second-leg: link to sector momentum, earnings calendar, and global cues.
- Risk map: what invalidates the view (VWAP loss with heavy sell delta, fresh negative news).

8. Deliverable Requirements

- Provide 1–3 tickers only when all filters are satisfied; otherwise state “No setups today that meet the shakeout-continuation criteria.”
- Keep analysis concise, fully data-backed, and timestamped. No generic commentary.

Data Signals and Thresholds (use/adjust as data availability allows)

- Liquidity: turnover ≥ ₹25–50 cr; free-float mcap ≥ ₹1000 cr; median spread ≤ 20–25 bps.
- Volume impulse: ≥ 1.8–2.5× 20-day median on Day-1; Day-2 PM up-volume > down-volume.
- Pullback: Day-2 retrace ≤ 38.2–50% of Day-1 range; or shallow bull flag under Day-1 high.

- Delivery: Day-1 delivery % ≥ 1.2–1.5x its 3M median; delivery turnover rising.
- Derivatives (if in F&O): OI down with price up (short covering) or OI flat with price up and IV cooling; PCR normalizing 0.7–1.2.

Reporting Format (strict)

- Header: “BTST-Next-Day Continuation After Weak-Hands Shakeout | [Date] | India Equities”
- Then 1–3 stock cards using section 6 template.
- End with a single-paragraph Strategic Perspective.
- If none qualify, explicitly say so.

Notes

- This is not intraday scalping; it is a one-session hold biased to second-leg continuation after a prior-day impulse and Day-2 morning flush.
- Emphasize evidence of absorption and supply exhaustion over raw price strength.
- Prefer sectors with concurrent breadth and earnings tailwinds.

Optional Enhancements (when available)

- Include footprint/cumulative delta, average trade size by session, order book imbalance, and options OI ladders.
- Include delivery % changes and block deal analysis from exchange data.
- For small/midcaps, insist on tighter spreads and cleaner absorption signatures.

Use this brief daily to find prior-day BTST candidates that shook out weak hands and are regaining control into the close, aiming for higher-probability next-day follow-through rather than profit-booking dips.

# Java Web Application for Advanced Stock Analysis with 5paisa API

### Complete Project Structure & Implementation Prompt

Create a comprehensive Java Spring Boot web application that integrates 5paisa API, NSE Bhavcopy data, and TA-Lib for advanced technical analysis to identify post-BTST weak-hands shakeout opportunities. This application should serve as both a data processing engine and research API endpoint.

### Technology Stack & Dependencies

### Core Framework

- **Spring Boot 3.x** with embedded Tomcat
- **Spring Web MVC** for REST API endpoints
- **Spring Data JPA** with H2/MySQL database
- **Spring Scheduler** for automated data collection
- **Maven** for dependency management

### Required Dependencies (pom.xml)

```xml
<dependencies>
    <!-- Spring Boot Starters -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>

    <!-- Database -->
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <scope>runtime</scope>
    </dependency>

    <!-- TA-Lib for Technical Analysis -->
    <dependency>
        <groupId>com.tictactec</groupId>
        <artifactId>ta-lib</artifactId>
        <version>0.4.0</version>
    </dependency>

    <!-- 5paisa API (Custom JAR) -->
    <dependency>
        <groupId>com.FivePaisa</groupId>
        <artifactId>FivePaisa</artifactId>
        <version>0.0.2-SNAPSHOT</version>
    </dependency>

    <!-- HTTP Client for API calls -->
    <dependency>
```

```
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-webflux</artifactId>
    </dependency>

    <!-- JSON Processing -->
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
    </dependency>

    <!-- CSV Processing for Bhavcopy -->
    <dependency>
        <groupId>com.opencsv</groupId>
        <artifactId>opencsv</artifactId>
        <version>5.7.1</version>
    </dependency>

    <!-- Lombok for boilerplate code -->
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>

    <!-- Apache Commons for utilities -->
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-lang3</artifactId>
    </dependency>

    <!-- Scheduling -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-quartz</artifactId>
    </dependency>
</dependencies>
```

## Application Configuration

### application.yml

```
server:
  port: 8080

spring:
  application:
    name: stock-analyzer
  datasource:
    url: jdbc:h2:mem:testdb
    driver-class-name: org.h2.Driver
    username: sa
    password: password
  h2:
    console:
      enabled: true
  jpa:
    hibernate:
      ddl-auto: create-drop
    show-sql: true
    properties:
      hibernate:
        format_sql: true

# 5paisa Configuration
fivepaisa:
  api:
    app-name: "YOUR_APP_NAME"
    app-version: "1.0"
    os-name: "WEB"
    encrypt-key: "YOUR_ENCRYPT_KEY"
    user-key: "YOUR_USER_KEY"
    user-id: "YOUR_USER_ID"
    password: "YOUR_PASSWORD"
    login-id: "YOUR_CLIENT_CODE"
    client-code: "YOUR_CLIENT_CODE"

# NSE Configuration
nse:
  bhavcopy:
    base-url: "https://nsearchives.nseindia.com/content/cm/"
    file-pattern: "BhavCopy_NSE_CM_0_0_0_{date}_F_0000.csv.zip"
    download-path: "./data/bhavcopy/"

# Technical Analysis Configuration
technical-analysis:
  indicators:
    rsi-period: 14
    ema-short: 9
    ema-long: 21
    volume-sma: 20
    atr-period: 14

logging:
  level:
    com.stockanalyzer: DEBUG
    org.springframework.web: DEBUG
```

## Core Data Models

## Stock Entity

```
@Entity
@Table(name = "stocks")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Stock {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    private String symbol;

    private String companyName;
    private String sector;
    private String series;
    private Long marketCap;
    private Double faceValue;
    private String isin;
    private Boolean isActive;

    @CreatedDate
    private LocalDateTime createdAt;

    @LastModifiedDate
    private LocalDateTime updatedAt;
}

@Entity
@Table(name = "price_data",
       indexes = {
           @Index(name = "idx_symbol_date", columnList = "symbol, trade_date"),
           @Index(name = "idx_trade_date", columnList = "trade_date")
       })
@Data
@NoArgsConstructor
@AllArgsConstructor
public class PriceData {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String symbol;

    @Column(name = "trade_date", nullable = false)
    private LocalDate tradeDate;

    @Column(precision = 10, scale = 2)
    private BigDecimal open;

    @Column(precision = 10, scale = 2)
    private BigDecimal high;

    @Column(precision = 10, scale = 2)
    private BigDecimal low;

    @Column(precision = 10, scale = 2)
    private BigDecimal close;

    @Column(precision = 10, scale = 2)
    private BigDecimal prevClose;
    private Long volume;
    private Long value;
    private Integer noOfTrades;

    @Column(precision = 5, scale = 2)
    private Double deliveryPercentage;

    @CreatedDate
    private LocalDateTime createdAt;
}

@Entity
@Table(name = "technical_indicators")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class TechnicalIndicator {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String symbol;

    @Column(name = "calculation_date", nullable = false)
    private LocalDate calculationDate;

    // Price-based indicators
    private Double rsi14;
    private Double ema9;
    private Double ema21;
    private Double sma20;
    private Double atr14;
    private Double vwap;

    // Volume-based indicators
    private Double volumeSma20;
    private Double volumeRatio;

    // Custom indicators for our strategy
    private Double priceStrength;
    private Double volumeStrength;
    private Double deliveryStrength;

    @CreatedDate
    private LocalDateTime createdAt;
}

@Entity
@Table(name = "btst_analysis")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class BTSTAnalysis {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String symbol;

    @Column(name = "analysis_date", nullable = false)
    private LocalDate analysisDate;

    // Day-1 BTST characteristics
    private Boolean hadLateSurge;
    private Double lateSessionVolumeRatio;
    private Double breakoutLevel;
    private Boolean hadCatalyst;
    private String catalystType;

    // Day-2 Weak hands indicators
    private Double gapPercentage;
    private Boolean showsAbsorption;
    private Double averageTradeSize;
    private Double retailIntensity;
    private Boolean vwapReclaimed;
    private Double cumulativeDelta;

    // Technical setup
    private Double pullbackDepth;
    private Boolean supplyExhaustion;
    private Double strengthScore;

    // Final recommendation
    private String recommendation; // BUY, HOLD, AVOID
    private Double confidenceScore;
    private Double entryPrice;
    private Double targetPrice;
    private Double stopLoss;

    @CreatedDate
    private LocalDateTime createdAt;
}
```

### 5paisa API Integration Service

```
@Service
@Slf4j
public class FivePaisaService {

    @Value("${fivepaisa.api.app-name}")
    private String appName;

    @Value("${fivepaisa.api.client-code}")
    private String clientCode;

    private final WebClient webClient;
    private final FivePaisaApis fivePaisaApis;
    private String authToken;
    @Autowired
    public FivePaisaService(WebClient.Builder webClientBuilder) {
        this.webClient = webClientBuilder.build();
        this.fivePaisaApis = new FivePaisaApis();
        initializeApiConfig();
    }

    private void initializeApiConfig() {
        AppConfig config = new AppConfig();
        config.setAppName(appName);
        // ... set other config parameters from application.yml

        Properties properties = new Properties();
        properties.setClientcode(clientCode);
    }

    @PostConstruct
    public void authenticate() {
        try {
            // Implement TOTP authentication
            String response = fivePaisaApis.getTotpSession(clientCode, getTotpCode(), get
            // Parse and store auth token
            log.info("5paisa authentication successful");
        } catch (Exception e) {
            log.error("Failed to authenticate with 5paisa API: {}", e.getMessage());
        }
    }

    public List<MarketFeedData> getMarketFeed(List<String> symbols) {
        try {
            JSONObject request = buildMarketFeedRequest(symbols);
            String response = fivePaisaApis.getMarketFeed(request);
            return parseMarketFeedResponse(response);
        } catch (Exception e) {
            log.error("Error fetching market feed: {}", e.getMessage());
            return Collections.emptyList();
        }
    }

    public List<HistoricalData> getHistoricalData(String symbol, LocalDate fromDate, Loca
        try {
            JSONObject request = buildHistoricalDataRequest(symbol, fromDate, toDate);
            String response = fivePaisaApis.getHistoricalData(request);
            return parseHistoricalDataResponse(response);
        } catch (Exception e) {
            log.error("Error fetching historical data for {}: {}", symbol, e.getMessage()
            return Collections.emptyList();
        }
    }

    private JSONObject buildMarketFeedRequest(List<String> symbols) {
        JSONObject request = new JSONObject();
        JSONObject head = new JSONObject();
        head.put("key", "your-api-key");

        JSONObject body = new JSONObject();
        body.put("ClientCode", clientCode);

        JSONArray marketFeedData = new JSONArray();
        for (String symbol : symbols) {
            JSONObject symbolData = new JSONObject();
            symbolData.put("Exchange", "N"); // NSE
            symbolData.put("ExchangeType", "C"); // Cash
            symbolData.put("ScripCode", getScripCode(symbol));
            marketFeedData.put(symbolData);
        }
        body.put("MarketFeedData", marketFeedData);

        request.put("head", head);
        request.put("body", body);
        return request;
    }

    // Additional helper methods...
}
```

### NSE Bhavcopy Data Service

```
@Service
@Slf4j
public class BhavcopyService {

    @Value("${nse.bhavcopy.base-url}")
    private String baseUrl;

    @Value("${nse.bhavcopy.file-pattern}")
    private String filePattern;

    @Value("${nse.bhavcopy.download-path}")
    private String downloadPath;

    private final WebClient webClient;
    private final PriceDataRepository priceDataRepository;

    @Autowired
    public BhavcopyService(WebClient.Builder webClientBuilder, PriceDataRepository priceD
        this.webClient = webClientBuilder
                .codecs(configurer -> configurer.defaultCodecs().maxInMemorySize(10 * 102
                .build();
        this.priceDataRepository = priceDataRepository;
    }

    @Scheduled(cron = "0 30 16 * * MON-FRI") // 4:30 PM on weekdays
    public void downloadDailyBhavcopy() {
        LocalDate today = LocalDate.now();
        if (isMarketDay(today)) {
            downloadAndProcessBhavcopy(today);
        }
    }

    public void downloadAndProcessBhavcopy(LocalDate date) {
        try {
            String fileName = filePattern.replace("{date}", date.format(DateTimeFormatter
            String url = baseUrl + fileName;

            log.info("Downloading bhavcopy for date: {}", date);

            byte[] zipData = webClient.get()
                    .uri(url)
                    .header("User-Agent", "Mozilla/5.0 (Windows NT 10.0; Win64; x64) Appl
                    .retrieve()
                    .bodyToMono(byte[].class)
                    .timeout(Duration.ofMinutes(5))
                    .block();

            if (zipData != null) {
                String csvContent = extractZipContent(zipData);
                List<PriceData> priceDataList = parseBhavcopyCSV(csvContent, date);
                savePriceData(priceDataList);
                log.info("Successfully processed bhavcopy for date: {} with {} records",
            }
        } catch (Exception e) {
            log.error("Error downloading/processing bhavcopy for date {}: {}", date, e.ge
        }
    }

    private List<PriceData> parseBhavcopyCSV(String csvContent, LocalDate tradeDate) {
        List<PriceData> priceDataList = new ArrayList<>();

        try (CSVReader reader = new CSVReader(new StringReader(csvContent))) {
            String[] headers = reader.readNext(); // Skip header
            String[] line;

            while ((line = reader.readNext()) != null) {
                try {
                    PriceData priceData = parseBhavcopyLine(line, tradeDate);
                    if (priceData != null && isValidForAnalysis(priceData)) {
                        priceDataList.add(priceData);
                    }
                } catch (Exception e) {
                    log.warn("Error parsing bhavcopy line: {}", Arrays.toString(line));
                }
            }
        } catch (Exception e) {
            log.error("Error parsing CSV content: {}", e.getMessage());
        }

        return priceDataList;
    }

    private PriceData parseBhavcopyLine(String[] fields, LocalDate tradeDate) {
        // Parse based on new UDIFF format
        // TradDt, BizDt, Sgmt, Src, FinInstrmTp, FinInstrmId, ISIN, TckrSymb, SctySrs,
        // StrkPric, OptnTp, FinInstrmActlXpryDt, OpnPric, HghPric, LwPric, ClsPric, Last
        // PrvsClsgPric, UndrlygPric, SttlmPric, OpnIntrst, ChngInOpnIntrst, TtlTradgVol,
        // TtlTrfVal, TtlNbOfTxsExctd, SsnId, NewBrdLotQty, Rmks, Rsn
        if (fields.length < 25) return null;

        String symbol = fields[^2_7].trim(); // TckrSymb
        String series = fields[^2_8].trim(); // SctySrs

        // Only process EQ series for cash segment
        if (!"EQ".equals(series)) return null;

        PriceData priceData = new PriceData();
        priceData.setSymbol(symbol);
        priceData.setTradeDate(tradeDate);
        priceData.setOpen(new BigDecimal(fields[^2_13])); // OpnPric
        priceData.setHigh(new BigDecimal(fields[^2_14])); // HghPric
        priceData.setLow(new BigDecimal(fields[^2_15])); // LwPric
        priceData.setClose(new BigDecimal(fields[^2_16])); // ClsPric
        priceData.setPrevClose(new BigDecimal(fields[^2_18])); // PrvsClsgPric
        priceData.setVolume(Long.parseLong(fields[^2_24])); // TtlTradgVol
        priceData.setValue(Long.parseLong(fields[^2_25])); // TtlTrfVal
        priceData.setNoOfTrades(Integer.parseInt(fields[^2_26])); // TtlNbOfTxsExctd

        return priceData;
    }

    private boolean isValidForAnalysis(PriceData priceData) {
        return priceData.getVolume() > 100000 && // Minimum volume filter
               priceData.getValue() > 10000000 && // Minimum value filter (1 crore)
               priceData.getClose().compareTo(new BigDecimal("10")) > 0; // Price > 10
    }

    // Additional helper methods...
}
```

## Technical Analysis Service with TA-Lib

```
@Service
@Slf4j
public class TechnicalAnalysisService {

    private final PriceDataRepository priceDataRepository;
    private final TechnicalIndicatorRepository technicalIndicatorRepository;

    @Autowired
    public TechnicalAnalysisService(PriceDataRepository priceDataRepository,
                                   TechnicalIndicatorRepository technicalIndicatorReposit
        this.priceDataRepository = priceDataRepository;
        this.technicalIndicatorRepository = technicalIndicatorRepository;
    }

    @Scheduled(cron = "0 0 17 * * MON-FRI") // 5:00 PM on weekdays
    public void calculateDailyIndicators() {
        LocalDate today = LocalDate.now();
        List<String> activeSymbols = priceDataRepository.findActiveSymbols(today);

        log.info("Calculating technical indicators for {} symbols", activeSymbols.size())
        for (String symbol : activeSymbols) {
            try {
                calculateIndicatorsForSymbol(symbol, today);
            } catch (Exception e) {
                log.error("Error calculating indicators for symbol {}: {}", symbol, e.get
            }
        }
    }

    public void calculateIndicatorsForSymbol(String symbol, LocalDate date) {
        // Get historical data for calculations (need at least 50 days for reliable indic
        LocalDate fromDate = date.minusDays(100);
        List<PriceData> historicalData = priceDataRepository.findBySymbolAndTradeDateBetw
                symbol, fromDate, date);

        if (historicalData.size() < 20) {
            log.warn("Insufficient data for symbol {}: {} records", symbol, historicalDat
            return;
        }

        TechnicalIndicator indicator = calculateIndicators(historicalData, date);
        indicator.setSymbol(symbol);
        indicator.setCalculationDate(date);

        technicalIndicatorRepository.save(indicator);
    }

    private TechnicalIndicator calculateIndicators(List<PriceData> historicalData, LocalD
        TechnicalIndicator indicator = new TechnicalIndicator();

        // Convert data to arrays for TA-Lib
        int size = historicalData.size();
        double[] high = new double[size];
        double[] low = new double[size];
        double[] close = new double[size];
        double[] volume = new double[size];

        for (int i = 0; i < size; i++) {
            PriceData data = historicalData.get(i);
            high[i] = data.getHigh().doubleValue();
            low[i] = data.getLow().doubleValue();
            close[i] = data.getClose().doubleValue();
            volume[i] = data.getVolume().doubleValue();
        }

        // Initialize TA-Lib
        Core taLib = new Core();
        MInteger outBegIdx = new MInteger();
        MInteger outNbElement = new MInteger();

        // Calculate RSI
        double[] rsiOutput = new double[size];
        RetCode rsiRetCode = taLib.rsi(0, size - 1, close, 14, outBegIdx, outNbElement, r
        if (rsiRetCode == RetCode.Success && outNbElement.value > 0) {
            indicator.setRsi14(rsiOutput[outNbElement.value - 1]);
        }
        // Calculate EMA 9
        double[] ema9Output = new double[size];
        RetCode ema9RetCode = taLib.ema(0, size - 1, close, 9, outBegIdx, outNbElement, e
        if (ema9RetCode == RetCode.Success && outNbElement.value > 0) {
            indicator.setEma9(ema9Output[outNbElement.value - 1]);
        }

        // Calculate EMA 21
        double[] ema21Output = new double[size];
        RetCode ema21RetCode = taLib.ema(0, size - 1, close, 21, outBegIdx, outNbElement,
        if (ema21RetCode == RetCode.Success && outNbElement.value > 0) {
            indicator.setEma21(ema21Output[outNbElement.value - 1]);
        }

        // Calculate SMA 20
        double[] sma20Output = new double[size];
        RetCode sma20RetCode = taLib.sma(0, size - 1, close, 20, outBegIdx, outNbElement,
        if (sma20RetCode == RetCode.Success && outNbElement.value > 0) {
            indicator.setSma20(sma20Output[outNbElement.value - 1]);
        }

        // Calculate ATR 14
        double[] atrOutput = new double[size];
        RetCode atrRetCode = taLib.atr(0, size - 1, high, low, close, 14, outBegIdx, outN
        if (atrRetCode == RetCode.Success && outNbElement.value > 0) {
            indicator.setAtr14(atrOutput[outNbElement.value - 1]);
        }

        // Calculate Volume SMA 20
        double[] volSmaOutput = new double[size];
        RetCode volSmaRetCode = taLib.sma(0, size - 1, volume, 20, outBegIdx, outNbElemen
        if (volSmaRetCode == RetCode.Success && outNbElement.value > 0) {
            indicator.setVolumeSma20(volSmaOutput[outNbElement.value - 1]);

            // Calculate volume ratio (current vs average)
            double currentVolume = volume[size - 1];
            double avgVolume = volSmaOutput[outNbElement.value - 1];
            indicator.setVolumeRatio(currentVolume / avgVolume);
        }

        // Calculate VWAP (for current day)
        indicator.setVwap(calculateVWAP(historicalData));

        // Calculate custom strength indicators
        indicator.setPriceStrength(calculatePriceStrength(historicalData));
        indicator.setVolumeStrength(calculateVolumeStrength(historicalData));

        return indicator;
    }

    private double calculateVWAP(List<PriceData> data) {
        if (data.isEmpty()) return 0.0;

        double totalPriceVolume = 0.0;
        double totalVolume = 0.0;
        for (PriceData priceData : data) {
            double typicalPrice = (priceData.getHigh().doubleValue() +
                                 priceData.getLow().doubleValue() +
                                 priceData.getClose().doubleValue()) / 3.0;
            double volume = priceData.getVolume().doubleValue();

            totalPriceVolume += typicalPrice * volume;
            totalVolume += volume;
        }

        return totalVolume > 0 ? totalPriceVolume / totalVolume : 0.0;
    }

    private double calculatePriceStrength(List<PriceData> data) {
        if (data.size() < 2) return 0.0;

        PriceData current = data.get(data.size() - 1);
        PriceData previous = data.get(data.size() - 2);

        double priceChange = (current.getClose().doubleValue() - previous.getClose().doub
                           / previous.getClose().doubleValue();
        double rangeRatio = (current.getClose().doubleValue() - current.getLow().doubleVa
                          / (current.getHigh().doubleValue() - current.getLow().doubleVal

        return (priceChange * 100) + (rangeRatio * 50); // Weighted score
    }

    private double calculateVolumeStrength(List<PriceData> data) {
        if (data.size() < 20) return 0.0;

        // Get last 20 days volume data
        List<Long> recentVolumes = data.stream()
                .skip(Math.max(0, data.size() - 20))
                .map(PriceData::getVolume)
                .collect(Collectors.toList());

        long currentVolume = data.get(data.size() - 1).getVolume();
        double avgVolume = recentVolumes.stream().mapToLong(Long::longValue).average().or

        return avgVolume > 0 ? (currentVolume / avgVolume) * 100 : 0.0;
    }
}
```

### BTST Analysis Engine

```
@Service
@Slf4j
public class BTSTAnalysisService {

    private final PriceDataRepository priceDataRepository;
    private final TechnicalIndicatorRepository technicalIndicatorRepository;
    private final BTSTAnalysisRepository btstAnalysisRepository;
    private final FivePaisaService fivePaisaService;
```

```java
    @Scheduled(cron = "0 0 18 * * MON-FRI") // 6:00 PM on weekdays
    public void runDailyBTSTAnalysis() {
        LocalDate today = LocalDate.now();
        LocalDate yesterday = today.minusDays(1);

        log.info("Running BTST analysis for date: {}", today);

        // Step 1: Identify Day-1 BTST candidates
        List<String> btstCandidates = identifyDay1BTSTCandidates(yesterday);

        // Step 2: Analyze Day-2 weak hands shakeout for each candidate
        for (String symbol : btstCandidates) {
            try {
                BTSTAnalysis analysis = analyzeWeakHandsShakeout(symbol, yesterday, today
                if (analysis != null) {
                    btstAnalysisRepository.save(analysis);
                }
            } catch (Exception e) {
                log.error("Error analyzing BTST for symbol {}: {}", symbol, e.getMessage(
            }
        }
    }

    private List<String> identifyDay1BTSTCandidates(LocalDate date) {
        List<String> candidates = new ArrayList<>();

        // Get all stocks with significant volume and price movement
        List<PriceData> potentialStocks = priceDataRepository.findByTradeDateAndVolumeGre
                date, 500000L, 50000000L); // Min 5L volume, 5Cr value

        for (PriceData stock : potentialStocks) {
            if (qualifiesAsDay1BTST(stock, date)) {
                candidates.add(stock.getSymbol());
            }
        }

        log.info("Identified {} Day-1 BTST candidates for date: {}", candidates.size(), 
        return candidates;
    }

    private boolean qualifiesAsDay1BTST(PriceData stock, LocalDate date) {
        try {
            // Get previous day data for comparison
            LocalDate prevDate = date.minusDays(1);
            PriceData prevData = priceDataRepository.findBySymbolAndTradeDate(stock.getSy
            if (prevData == null) return false;

            // Check for significant price movement (>2% up from previous close)
            double priceChange = (stock.getClose().doubleValue() - stock.getPrevClose().
                               / stock.getPrevClose().doubleValue();
            if (priceChange < 0.02) return false; // Less than 2% gain

            // Check for volume surge (>1.5x average)
            TechnicalIndicator techIndicator = technicalIndicatorRepository
                    .findBySymbolAndCalculationDate(stock.getSymbol(), date);
            if (techIndicator != null && techIndicator.getVolumeRatio() != null) {
                if (techIndicator.getVolumeRatio() < 1.5) return false; // Less than 1.5x
            }

            // Check for breakout pattern
            List<PriceData> recent20Days = priceDataRepository
                    .findTop20BySymbolAndTradeDateLessThanOrderByTradeDateDesc(stock.getS
            if (recent20Days.size() < 10) return false;

            double highest20Day = recent20Days.stream()
                    .mapToDouble(pd -> pd.getHigh().doubleValue())
                    .max().orElse(0.0);

            // Should break 20-day high or be very close to it
            if (stock.getHigh().doubleValue() < highest20Day * 0.98) return false;

            // Check for late session strength (close near high)
            double dayRange = stock.getHigh().doubleValue() - stock.getLow().doubleValue(
            double closeFromHigh = stock.getHigh().doubleValue() - stock.getClose().doubl
            double closePosition = dayRange > 0 ? (1 - closeFromHigh / dayRange) : 0;

            return closePosition > 0.7; // Close should be in top 30% of day's range

        } catch (Exception e) {
            log.warn("Error evaluating BTST qualification for {}: {}", stock.getSymbol(),
            return false;
        }
    }

    private BTSTAnalysis analyzeWeakHandsShakeout(String symbol, LocalDate day1, LocalDat
        try {
            PriceData day1Data = priceDataRepository.findBySymbolAndTradeDate(symbol, day
            PriceData day2Data = priceDataRepository.findBySymbolAndTradeDate(symbol, day

            if (day1Data == null || day2Data == null) return null;

            BTSTAnalysis analysis = new BTSTAnalysis();
            analysis.setSymbol(symbol);
            analysis.setAnalysisDate(day2);

            // Analyze Day-1 BTST characteristics
            analyzeDayOneBTSTCharacteristics(analysis, day1Data, day1);

            // Analyze Day-2 weak hands indicators
            analyzeDayTwoWeakHandsIndicators(analysis, day1Data, day2Data, day2);

            // Calculate technical setup
            analyzeTechnicalSetup(analysis, symbol, day2);

            // Generate recommendation
            generateRecommendation(analysis);

            return analysis;

        } catch (Exception e) {
            log.error("Error in weak hands analysis for {}: {}", symbol, e.getMessage());
            return null;
        }
    }

    private void analyzeDayOneBTSTCharacteristics(BTSTAnalysis analysis, PriceData day1Da
        // Check for late session surge
        double dayRange = day1Data.getHigh().doubleValue() - day1Data.getLow().doubleValu
        double closeFromHigh = day1Data.getHigh().doubleValue() - day1Data.getClose().dou
        boolean hadLateSurge = dayRange > 0 && (closeFromHigh / dayRange) < 0.3;
        analysis.setHadLateSurge(hadLateSurge);

        // Calculate late session volume ratio (would need intraday data for accuracy)
        TechnicalIndicator techIndicator = technicalIndicatorRepository
                .findBySymbolAndCalculationDate(analysis.getSymbol(), day1);
        if (techIndicator != null && techIndicator.getVolumeRatio() != null) {
            analysis.setLateSessionVolumeRatio(techIndicator.getVolumeRatio());
        }

        // Identify breakout level
        List<PriceData> recent20Days = priceDataRepository
                .findTop20BySymbolAndTradeDateLessThanOrderByTradeDateDesc(analysis.getSy
        double breakoutLevel = recent20Days.stream()
                .mapToDouble(pd -> pd.getHigh().doubleValue())
                .max().orElse(day1Data.getHigh().doubleValue());
        analysis.setBreakoutLevel(breakoutLevel);

        // Catalyst detection (placeholder - would integrate with news API)
        analysis.setHadCatalyst(false); // Default, would be enhanced with real catalyst
        analysis.setCatalystType("UNKNOWN");
    }

    private void analyzeDayTwoWeakHandsIndicators(BTSTAnalysis analysis, PriceData day1Da
                                                  PriceData day2Data, LocalDate day2) {
        // Calculate gap percentage
        double gapPercentage = (day2Data.getOpen().doubleValue() - day1Data.getClose().do
                             / day1Data.getClose().doubleValue() * 100;
        analysis.setGapPercentage(gapPercentage);

        // Check for absorption pattern
        boolean showsAbsorption = (gapPercentage < 0 && gapPercentage > -1.5) && // Small
                                 (day2Data.getClose().doubleValue() > day2Data.getOpen().
        analysis.setShowsAbsorption(showsAbsorption);

        // Average trade size analysis (approximated from available data)
        long totalTrades = day2Data.getNoOfTrades();
        long totalVolume = day2Data.getVolume();
        double avgTradeSize = totalTrades > 0 ? (double) totalVolume / totalTrades : 0;
        analysis.setAverageTradeSize(avgTradeSize);

        // Retail intensity (smaller trade size indicates retail participation)
        double retailIntensity = avgTradeSize > 0 ? Math.max(0, 100 - (avgTradeSize / 100
        analysis.setRetailIntensity(retailIntensity);

        // VWAP reclaim analysis
        TechnicalIndicator techIndicator = technicalIndicatorRepository
                .findBySymbolAndCalculationDate(analysis.getSymbol(), day2);
        boolean vwapReclaimed = techIndicator != null && techIndicator.getVwap() != null
```

```
                               day2Data.getClose().doubleValue() > techIndicator.getVwap
        analysis.setVwapReclaimed(vwapReclaimed);

        // Cumulative delta (approximated as price-weighted volume)
        double typicalPrice = (day2Data.getHigh().doubleValue() + day2Data.getLow().doubl
                              day2Data.getClose().doubleValue()) / 3;
        double cumulativeDelta = (day2Data.getClose().doubleValue() - day2Data.getOpen().
                               * day2Data.getVolume();
        analysis.setCumulativeDelta(cumulativeDelta);
    }

    private void analyzeTechnicalSetup(BTSTAnalysis analysis, String symbol, LocalDate da
        try {
            List<PriceData> recentData = priceDataRepository
                    .findTop10BySymbolAndTradeDateLessThanEqualOrderByTradeDateDesc(symbo

            if (recentData.size() >= 2) {
                PriceData today = recentData.get(0);
                PriceData yesterday = recentData.get(1);

                // Calculate pullback depth
                double day1Range = yesterday.getHigh().doubleValue() - yesterday.getLow()
                double pullbackFromHigh = yesterday.getHigh().doubleValue() - today.getLo
                double pullbackDepth = day1Range > 0 ? (pullbackFromHigh / day1Range) * 1
                analysis.setPullbackDepth(pullbackDepth);

                // Supply exhaustion indicator
                boolean supplyExhaustion = today.getClose().doubleValue() > today.getOpe
                                         today.getVolume() > yesterday.getVolume() * 0.8;
                analysis.setSupplyExhaustion(supplyExhaustion);

                // Calculate overall strength score
                double strengthScore = calculateStrengthScore(analysis);
                analysis.setStrengthScore(strengthScore);
            }
        } catch (Exception e) {
            log.warn("Error in technical setup analysis for {}: {}", symbol, e.getMessage
        }
    }

    private double calculateStrengthScore(BTSTAnalysis analysis) {
        double score = 0;

        // Day-1 BTST strength (30 points)
        if (Boolean.TRUE.equals(analysis.getHadLateSurge())) score += 10;
        if (analysis.getLateSessionVolumeRatio() != null && analysis.getLateSessionVolume
        if (Boolean.TRUE.equals(analysis.getHadCatalyst())) score += 10;

        // Day-2 absorption strength (40 points)
        if (Boolean.TRUE.equals(analysis.getShowsAbsorption())) score += 15;
        if (Boolean.TRUE.equals(analysis.getVwapReclaimed())) score += 10;
        if (analysis.getCumulativeDelta() != null && analysis.getCumulativeDelta() > 0) s
        if (analysis.getRetailIntensity() != null && analysis.getRetailIntensity() < 50)

        // Technical setup strength (30 points)
        if (analysis.getPullbackDepth() != null && analysis.getPullbackDepth() < 50) scor
        if (Boolean.TRUE.equals(analysis.getSupplyExhaustion())) score += 10;
        if (analysis.getGapPercentage() != null && analysis.getGapPercentage() > -1 && ar

        return score;
    }

    private void generateRecommendation(BTSTAnalysis analysis) {
        double strengthScore = analysis.getStrengthScore();
        PriceData latestPrice = priceDataRepository.findBySymbolAndTradeDate(
                analysis.getSymbol(), analysis.getAnalysisDate());

        if (latestPrice == null) {
            analysis.setRecommendation("AVOID");
            analysis.setConfidenceScore(0.0);
            return;
        }

        double currentPrice = latestPrice.getClose().doubleValue();

        if (strengthScore >= 70) {
            analysis.setRecommendation("BUY");
            analysis.setConfidenceScore(strengthScore);
            analysis.setEntryPrice(currentPrice);
            analysis.setTargetPrice(currentPrice * 1.03); // 3% target
            analysis.setStopLoss(currentPrice * 0.985); // 1.5% stop loss
        } else if (strengthScore >= 50) {
            analysis.setRecommendation("HOLD");
            analysis.setConfidenceScore(strengthScore);
            analysis.setEntryPrice(currentPrice * 0.995); // Enter on slight dip
            analysis.setTargetPrice(currentPrice * 1.02); // 2% target
            analysis.setStopLoss(currentPrice * 0.98); // 2% stop loss
        } else {
            analysis.setRecommendation("AVOID");
            analysis.setConfidenceScore(strengthScore);
        }
    }
}
```

### REST API Controller

```
@RestController
@RequestMapping("/api/v1/analysis")
@Slf4j
public class AnalysisController {

    private final BTSTAnalysisService btstAnalysisService;
    private final BTSTAnalysisRepository btstAnalysisRepository;
    private final TechnicalAnalysisService technicalAnalysisService;
    private final PriceDataRepository priceDataRepository;

    @GetMapping("/btst/recommendations")
    public ResponseEntity<List<BTSTRecommendationDTO>> getBTSTRecommendations(
            @RequestParam(defaultValue = "0") LocalDate date,
            @RequestParam(defaultValue = "BUY") String recommendation) {
```

```
        LocalDate analysisDate = date.equals(LocalDate.of(1970, 1, 1)) ? LocalDate.now()

        List<BTSTAnalysis> analyses = btstAnalysisRepository
                .findByAnalysisDateAndRecommendationOrderByConfidenceScoreDesc(analysisDa

        List<BTSTRecommendationDTO> recommendations = analyses.stream()
                .map(this::convertToRecommendationDTO)
                .collect(Collectors.toList());

        return ResponseEntity.ok(recommendations);
    }

    @GetMapping("/btst/detailed/{symbol}")
    public ResponseEntity<BTSTDetailedAnalysisDTO> getDetailedAnalysis(
            @PathVariable String symbol,
            @RequestParam(defaultValue = "0") LocalDate date) {

        LocalDate analysisDate = date.equals(LocalDate.of(1970, 1, 1)) ? LocalDate.now()

        BTSTAnalysis analysis = btstAnalysisRepository
                .findBySymbolAndAnalysisDate(symbol, analysisDate);

        if (analysis == null) {
            return ResponseEntity.notFound().build();
        }

        BTSTDetailedAnalysisDTO detailedAnalysis = convertToDetailedAnalysisDTO(analysis)
        return ResponseEntity.ok(detailedAnalysis);
    }

    @GetMapping("/screening/candidates")
    public ResponseEntity<List<BTSTCandidateDTO>> getBTSTCandidates(
            @RequestParam(defaultValue = "0") LocalDate date) {

        LocalDate screeningDate = date.equals(LocalDate.of(1970, 1, 1)) ? LocalDate.now()

        // Get high-volume, high-value stocks for the date
        List<PriceData> candidates = priceDataRepository
                .findByTradeDateAndVolumeGreaterThanAndValueGreaterThan(
                        screeningDate, 500000L, 50000000L);

        List<BTSTCandidateDTO> candidateDTOs = candidates.stream()
                .map(pd -> convertToCandidateDTO(pd, screeningDate))
                .filter(Objects::nonNull)
                .sorted((a, b) -> Double.compare(b.getVolumeRatio(), a.getVolumeRatio()))
                .collect(Collectors.toList());

        return ResponseEntity.ok(candidateDTOs);
    }

    @GetMapping("/technical/{symbol}")
    public ResponseEntity<TechnicalIndicatorDTO> getTechnicalIndicators(
            @PathVariable String symbol,
            @RequestParam(defaultValue = "0") LocalDate date) {

        LocalDate analysisDate = date.equals(LocalDate.of(1970, 1, 1)) ? LocalDate.now()
        TechnicalIndicator indicator = technicalAnalysisService
                .getTechnicalIndicator(symbol, analysisDate);

        if (indicator == null) {
            return ResponseEntity.notFound().build();
        }

        TechnicalIndicatorDTO dto = convertToTechnicalIndicatorDTO(indicator);
        return ResponseEntity.ok(dto);
    }

    @PostMapping("/manual/analyze/{symbol}")
    public ResponseEntity<BTSTAnalysis> runManualAnalysis(
            @PathVariable String symbol,
            @RequestParam LocalDate day1,
            @RequestParam LocalDate day2) {

        try {
            BTSTAnalysis analysis = btstAnalysisService.runManualAnalysis(symbol, day1, d
            return ResponseEntity.ok(analysis);
        } catch (Exception e) {
            log.error("Error in manual analysis for {}: {}", symbol, e.getMessage());
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
        }
    }

    @GetMapping("/market-summary")
    public ResponseEntity<MarketSummaryDTO> getMarketSummary(
            @RequestParam(defaultValue = "0") LocalDate date) {

        LocalDate analysisDate = date.equals(LocalDate.of(1970, 1, 1)) ? LocalDate.now()

        MarketSummaryDTO summary = new MarketSummaryDTO();

        List<BTSTAnalysis> allAnalyses = btstAnalysisRepository.findByAnalysisDate(analys

        long buyRecommendations = allAnalyses.stream()
                .filter(a -> "BUY".equals(a.getRecommendation()))
                .count();

        long holdRecommendations = allAnalyses.stream()
                .filter(a -> "HOLD".equals(a.getRecommendation()))
                .count();

        long avoidRecommendations = allAnalyses.stream()
                .filter(a -> "AVOID".equals(a.getRecommendation()))
                .count();

        double avgConfidenceScore = allAnalyses.stream()
                .filter(a -> "BUY".equals(a.getRecommendation()))
                .mapToDouble(BTSTAnalysis::getConfidenceScore)
                .average()
                .orElse(0.0);

        summary.setAnalysisDate(analysisDate);
        summary.setTotalCandidates(allAnalyses.size());
        summary.setBuyRecommendations((int) buyRecommendations);
        summary.setHoldRecommendations((int) holdRecommendations);
        summary.setAvoidRecommendations((int) avoidRecommendations);
        summary.setAvgConfidenceScore(avgConfidenceScore);

        return ResponseEntity.ok(summary);
    }

    // Helper methods for DTO conversions
    private BTSTRecommendationDTO convertToRecommendationDTO(BTSTAnalysis analysis) {
        // Implementation for converting entity to DTO
        // Include essential fields for recommendation display
        return BTSTRecommendationDTO.builder()
                .symbol(analysis.getSymbol())
                .recommendation(analysis.getRecommendation())
                .confidenceScore(analysis.getConfidenceScore())
                .entryPrice(analysis.getEntryPrice())
                .targetPrice(analysis.getTargetPrice())
                .stopLoss(analysis.getStopLoss())
                .strengthScore(analysis.getStrengthScore())
                .analysisDate(analysis.getAnalysisDate())
                .build();
    }

    // Additional helper methods...
}
```

### Key API Endpoints for Prompt Integration

#### 1. Real-time Analysis Endpoint

```
GET /api/v1/analysis/btst/recommendations?date=2025-10-10&recommendation=BUY
```

Returns list of stocks that passed the weak-hands shakeout filter with BUY recommendation.

#### 2. Detailed Analysis Endpoint

```
GET /api/v1/analysis/btst/detailed/{symbol}?date=2025-10-10
```

Provides comprehensive analysis data for a specific stock including all weak-hands indicators.

#### 3. Catalyst Integration Endpoint

```
GET /api/v1/analysis/screening/candidates?date=2025-10-09
```

Returns Day-1 BTST candidates that can be cross-referenced with external catalyst data.

### 4. Technical Setup Endpoint

```
GET /api/v1/analysis/technical/{symbol}?date=2025-10-10
```

Provides all technical indicators calculated using TA-Lib for fundamental analysis.

### 5. Market Summary Endpoint

```
GET /api/v1/analysis/market-summary?date=2025-10-10
```

Overall market condition and analysis statistics for context.

## Integration with Your Enhanced Prompt

The enhanced prompt can now call these endpoints to:

1. **Get Quantified Data**: Instead of qualitative descriptions, get actual numerical scores for weak-hands indicators
2. **Cross-reference Catalysts**: Use the candidates endpoint to identify stocks, then research catalysts for those specific symbols
3. **Technical Validation**: Verify technical setups with calculated indicators rather than manual analysis
4. **Confidence Scoring**: Use the application's confidence scores to prioritize research efforts
5. **Historical Analysis**: Access historical analysis results to validate the strategy's effectiveness

## Deployment Configuration

### application-prod.yml

```
server:
  port: 8080

spring:
  profiles:
    active: prod
  datasource:
    url: jdbc:mysql://localhost:3306/stock_analyzer
    username: ${DB_USERNAME}
    password: ${DB_PASSWORD}
  jpa:
    hibernate:
      ddl-auto: validate
    show-sql: false

fivepaisa:
  api:
    app-name: ${FIVEPAISA_APP_NAME}
    client-code: ${FIVEPAISA_CLIENT_CODE}
    # ... other encrypted config from environment variables

logging:
  level:
    com.stockanalyzer: INFO
    org.springframework.web: WARN
```

This comprehensive Java application provides the technical foundation for your enhanced BTST analysis strategy, combining real-time market data, historical analysis, and quantified technical indicators to systematically identify post-weak-hands-shakeout opportunities.[1] [2] [3] [4] [5] [6] [7] [8] [9] [10]

⁂

# Complete MySQL Integration & Updated Project Structure

## Project Directory Structure

```
stock-analyzer/
├── pom.xml
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   └── com/
│   │   │       └── stockanalyzer/
│   │   │           ├── StockAnalyzerApplication.java
│   │   │           ├── config/
│   │   │           │   ├── DatabaseConfig.java
│   │   │           │   ├── FivePaisaConfig.java
│   │   │           │   ├── SchedulingConfig.java
│   │   │           │   └── WebConfig.java
│   │   │           ├── controller/
│   │   │           │   ├── AnalysisController.java
│   │   │           │   ├── DataController.java
│   │   │           │   └── HealthController.java
│   │   │           ├── dto/
│   │   │           │   ├── BTSTRecommendationDTO.java
│   │   │           │   ├── BTSTDetailedAnalysisDTO.java
│   │   │           │   ├── BTSTCandidateDTO.java
│   │   │           │   ├── TechnicalIndicatorDTO.java
│   │   │           │   └── MarketSummaryDTO.java
│   │   │           ├── entity/
│   │   │           │   ├── Stock.java
│   │   │           │   ├── PriceData.java
│   │   │           │   ├── TechnicalIndicator.java
│   │   │           │   ├── BTSTAnalysis.java
│   │   │           │   └── MarketEvent.java
│   │   │           ├── repository/
│   │   │           │   ├── StockRepository.java
│   │   │           │   ├── PriceDataRepository.java
│   │   │           │   ├── TechnicalIndicatorRepository.java
│   │   │           │   ├── BTSTAnalysisRepository.java
│   │   │           │   └── MarketEventRepository.java
│   │   │           ├── service/
│   │   │           │   ├── FivePaisaService.java
│   │   │           │   ├── BhavcopyService.java
│   │   │           │   ├── TechnicalAnalysisService.java
│   │   │           │   ├── BTSTAnalysisService.java
│   │   │           │   └── MarketDataService.java
│   │   │           ├── scheduler/
│   │   │           │   ├── DataCollectionScheduler.java
│   │   │           │   └── AnalysisScheduler.java
│   │   │           └── util/
│   │   │               ├── DateUtils.java
│   │   │               ├── MathUtils.java
│   │   │               └── ValidationUtils.java
│   │   └── resources/
│   │       ├── application.yml
│   │       ├── application-dev.yml
│   │       ├── application-prod.yml
│   │       ├── db/
│   │       │   └── migration/
│   │       │       ├── V1__create_stock_tables.sql
│   │       │       ├── V2__create_price_data_tables.sql
│   │       │       ├── V3__create_technical_indicator_tables.sql
│   │       │       └── V4__create_btst_analysis_tables.sql
│   │       └── static/
│   │           └── api-docs/
│   └── test/
│       └── java/
│           └── com/
│               └── stockanalyzer/
│                   ├── service/
│                   ├── controller/
│                   └── integration/
├── data/
│   ├── bhavcopy/
│   └── logs/
└── docker/
    ├── docker-compose.yml
    └── mysql/
        └── init.sql
```

### Updated pom.xml with MySQL Dependencies

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.0</version>
        <relativePath/>
    </parent>

    <groupId>com.stockanalyzer</groupId>
    <artifactId>stock-analyzer</artifactId>
    <version>1.0.0</version>
    <name>stock-analyzer</name>
    <description>Advanced Stock Analysis with BTST Weak-Hands Detection</description>

    <properties>
        <java.version>17</java.version>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
        <ta-lib.version>0.4.0</ta-lib.version>
    </properties>

    <dependencies>
        <!-- Spring Boot Starters -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-webflux</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-actuator</artifactId>
        </dependency>

        <!-- Database -->
        <dependency>
            <groupId>com.mysql</groupId>
            <artifactId>mysql-connector-j</artifactId>
            <scope>runtime</scope>
        </dependency>

        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
            <scope>test</scope>
        </dependency>

        <!-- Database Migration -->
        <dependency>
            <groupId>org.flywaydb</groupId>
            <artifactId>flyway-core</artifactId>
        </dependency>

        <dependency>
            <groupId>org.flywaydb</groupId>
            <artifactId>flyway-mysql</artifactId>
        </dependency>

        <!-- TA-Lib for Technical Analysis -->
        <dependency>
            <groupId>com.tictactec</groupId>
            <artifactId>ta-lib</artifactId>
            <version>${ta-lib.version}</version>
        </dependency>

        <!-- CSV Processing for Bhavcopy -->
        <dependency>
            <groupId>com.opencsv</groupId>
            <artifactId>opencsv</artifactId>
            <version>5.9</version>
        </dependency>

        <!-- JSON Processing -->
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
        </dependency>

        <dependency>
            <groupId>com.fasterxml.jackson.datatype</groupId>
            <artifactId>jackson-datatype-jsr310</artifactId>
        </dependency>

        <!-- Lombok for boilerplate code -->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>

        <!-- Apache Commons -->
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-lang3</artifactId>
        </dependency>

        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-math3</artifactId>
            <version>3.6.1</version>
        </dependency>
```

```xml
        <!-- API Documentation -->
        <dependency>
            <groupId>org.springdoc</groupId>
            <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
            <version>2.2.0</version>
        </dependency>

        <!-- Testing -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>org.testcontainers</groupId>
            <artifactId>mysql</artifactId>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>org.testcontainers</groupId>
            <artifactId>junit-jupiter</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>

            <plugin>
                <groupId>org.flywaydb</groupId>
                <artifactId>flyway-maven-plugin</artifactId>
                <configuration>
                    <url>jdbc:mysql://localhost:3306/stock_analyzer</url>
                    <user>root</user>
                    <password>password</password>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

## MySQL Configuration Files

## src/main/resources/application.yml

```
server:
  port: 8080
  servlet:
    context-path: /api

spring:
  application:
    name: stock-analyzer
  profiles:
    active: dev

  datasource:
    url: jdbc:mysql://localhost:3306/stock_analyzer?useSSL=false&serverTimezone=UTC&allow
    username: ${DB_USERNAME:root}
    password: ${DB_PASSWORD:password}
    driver-class-name: com.mysql.cj.jdbc.Driver
    hikari:
      maximum-pool-size: 20
      minimum-idle: 5
      connection-timeout: 30000
      idle-timeout: 600000
      max-lifetime: 1800000

  jpa:
    hibernate:
      ddl-auto: validate
    show-sql: false
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MySQLDialect
        format_sql: true
        jdbc:
          batch_size: 50
          batch_versioned_data: true
        order_inserts: true
        order_updates: true

  flyway:
    enabled: true
    locations: classpath:db/migration
    baseline-on-migrate: true
    validate-on-migrate: true

  jackson:
    serialization:
      write-dates-as-timestamps: false
    default-property-inclusion: NON_NULL

# 5paisa Configuration
fivepaisa:
  api:
    base-url: "https://openapi.5paisa.com/VendorsAPI/Service1.svc"
    app-name: ${FIVEPAISA_APP_NAME:YOUR_APP_NAME}
    app-version: "1.0"
    os-name: "WEB"
    encrypt-key: ${FIVEPAISA_ENCRYPT_KEY:YOUR_ENCRYPT_KEY}
    user-key: ${FIVEPAISA_USER_KEY:YOUR_USER_KEY}
    user-id: ${FIVEPAISA_USER_ID:YOUR_USER_ID}
    password: ${FIVEPAISA_PASSWORD:YOUR_PASSWORD}
    login-id: ${FIVEPAISA_CLIENT_CODE:YOUR_CLIENT_CODE}
    client-code: ${FIVEPAISA_CLIENT_CODE:YOUR_CLIENT_CODE}

# NSE Configuration
nse:
  bhavcopy:
    base-url: "https://nsearchives.nseindia.com/content/cm/"
    file-pattern: "BhavCopy_NSE_CM_0_0_0_{date}_F_0000.csv.zip"
    download-path: "./data/bhavcopy/"
    retry-attempts: 3
    retry-delay: 5000

# Technical Analysis Configuration
technical-analysis:
  indicators:
    rsi-period: 14
    ema-short: 9
    ema-long: 21
    volume-sma: 20
    atr-period: 14
  lookback-days: 100

# BTST Analysis Configuration
btst:
  analysis:
    min-volume: 500000
    min-value: 50000000
    min-price: 10.0
    max-gap-down: -1.5
    min-gap-up: -0.5
    confidence-threshold: 50.0

logging:
  level:
    com.stockanalyzer: INFO
    org.springframework.web: WARN
    org.hibernate.SQL: WARN
  file:
    name: ./data/logs/stock-analyzer.log
  pattern:
    file: "%d{yyyy-MM-dd HH:mm:ss} [%thread] %-5level %logger{36} - %msg%n"

management:
  endpoints:
    web:
      exposure:
        include: health,info,metrics,prometheus
  endpoint:
    health:
      show-details: always
```

### src/main/resources/application-dev.yml

```
spring:
  jpa:
    show-sql: true
    properties:
      hibernate:
        format_sql: true

logging:
  level:
    com.stockanalyzer: DEBUG
    org.hibernate.SQL: DEBUG
    org.hibernate.type.descriptor.sql.BasicBinder: TRACE
```

### src/main/resources/application-prod.yml

```
spring:
  datasource:
    hikari:
      maximum-pool-size: 30
      minimum-idle: 10

  jpa:
    show-sql: false
    properties:
      hibernate:
        generate_statistics: false

logging:
  level:
    com.stockanalyzer: INFO
    org.springframework.web: WARN
```

## Database Migration Scripts

### src/main/resources/db/migration/V1__create_stock_tables.sql

```
-- Create stocks table
CREATE TABLE stocks (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    symbol VARCHAR(20) NOT NULL UNIQUE,
    company_name VARCHAR(200),
    sector VARCHAR(100),
    series VARCHAR(10),
    market_cap BIGINT,
    face_value DECIMAL(10,2),
    isin VARCHAR(20),
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    INDEX idx_symbol (symbol),
    INDEX idx_sector (sector),
    INDEX idx_is_active (is_active)
);

-- Create market_events table for catalyst tracking
CREATE TABLE market_events (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    symbol VARCHAR(20) NOT NULL,
    event_date DATE NOT NULL,
    event_type VARCHAR(50) NOT NULL,
    event_description TEXT,
    impact_score DECIMAL(3,1),
    source VARCHAR(100),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_symbol_date (symbol, event_date),
    INDEX idx_event_type (event_type),
    INDEX idx_event_date (event_date),

    FOREIGN KEY (symbol) REFERENCES stocks(symbol) ON DELETE CASCADE
);
```

### src/main/resources/db/migration/V2__create_price_data_tables.sql

```
-- Create price_data table with optimized indexes
CREATE TABLE price_data (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    symbol VARCHAR(20) NOT NULL,
    trade_date DATE NOT NULL,
    open_price DECIMAL(12,2) NOT NULL,
    high_price DECIMAL(12,2) NOT NULL,
    low_price DECIMAL(12,2) NOT NULL,
    close_price DECIMAL(12,2) NOT NULL,
    prev_close DECIMAL(12,2) NOT NULL,
    volume BIGINT NOT NULL,
    value_traded BIGINT NOT NULL,
    no_of_trades INTEGER NOT NULL,
    delivery_percentage DECIMAL(5,2),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    UNIQUE KEY uk_symbol_trade_date (symbol, trade_date),
    INDEX idx_trade_date (trade_date),
    INDEX idx_volume (volume DESC),
    INDEX idx_value_traded (value_traded DESC),
    INDEX idx_symbol_date_desc (symbol, trade_date DESC),

    FOREIGN KEY (symbol) REFERENCES stocks(symbol) ON DELETE CASCADE
);
```

```
-- Create intraday_data table for more granular analysis (optional)
CREATE TABLE intraday_data (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    symbol VARCHAR(20) NOT NULL,
    trade_date DATE NOT NULL,
    trade_time TIME NOT NULL,
    price DECIMAL(12,2) NOT NULL,
    volume INTEGER NOT NULL,
    value_traded INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_symbol_datetime (symbol, trade_date, trade_time),
    INDEX idx_trade_date (trade_date),

    FOREIGN KEY (symbol) REFERENCES stocks(symbol) ON DELETE CASCADE
);
```

### src/main/resources/db/migration/V3__create_technical_indicator_tables.sql

```
-- Create technical_indicators table
CREATE TABLE technical_indicators (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    symbol VARCHAR(20) NOT NULL,
    calculation_date DATE NOT NULL,

    -- Price-based indicators
    rsi_14 DECIMAL(5,2),
    ema_9 DECIMAL(12,2),
    ema_21 DECIMAL(12,2),
    sma_20 DECIMAL(12,2),
    atr_14 DECIMAL(8,4),
    vwap DECIMAL(12,2),

    -- Volume-based indicators
    volume_sma_20 BIGINT,
    volume_ratio DECIMAL(6,3),

    -- Custom strength indicators
    price_strength DECIMAL(8,4),
    volume_strength DECIMAL(8,4),
    delivery_strength DECIMAL(8,4),

    -- Momentum indicators
    macd DECIMAL(8,4),
    macd_signal DECIMAL(8,4),
    macd_histogram DECIMAL(8,4),

    -- Volatility indicators
    bollinger_upper DECIMAL(12,2),
    bollinger_lower DECIMAL(12,2),
    bollinger_width DECIMAL(8,4),

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    UNIQUE KEY uk_symbol_calc_date (symbol, calculation_date),
    INDEX idx_calculation_date (calculation_date),
    INDEX idx_rsi_14 (rsi_14),
    INDEX idx_volume_ratio (volume_ratio DESC),

    FOREIGN KEY (symbol) REFERENCES stocks(symbol) ON DELETE CASCADE
);
```

### src/main/resources/db/migration/V4__create_btst_analysis_tables.sql

```
-- Create btst_analysis table
CREATE TABLE btst_analysis (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    symbol VARCHAR(20) NOT NULL,
    analysis_date DATE NOT NULL,

    -- Day-1 BTST characteristics
    had_late_surge BOOLEAN DEFAULT FALSE,
    late_session_volume_ratio DECIMAL(6,3),
    breakout_level DECIMAL(12,2),
    had_catalyst BOOLEAN DEFAULT FALSE,
    catalyst_type VARCHAR(50),

    -- Day-2 Weak hands indicators
    gap_percentage DECIMAL(6,3),
    shows_absorption BOOLEAN DEFAULT FALSE,
    average_trade_size DECIMAL(12,2),
    retail_intensity DECIMAL(6,3),
    vwap_reclaimed BOOLEAN DEFAULT FALSE,
    cumulative_delta DECIMAL(15,2),

    -- Technical setup
    pullback_depth DECIMAL(6,3),
    supply_exhaustion BOOLEAN DEFAULT FALSE,
    strength_score DECIMAL(6,2),

    -- Final recommendation
    recommendation ENUM('BUY', 'HOLD', 'AVOID') NOT NULL,
    confidence_score DECIMAL(6,2),
    entry_price DECIMAL(12,2),
    target_price DECIMAL(12,2),
    stop_loss DECIMAL(12,2),

    -- Risk metrics
    risk_reward_ratio DECIMAL(4,2),
    position_size_percentage DECIMAL(4,2),

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    UNIQUE KEY uk_symbol_analysis_date (symbol, analysis_date),
    INDEX idx_analysis_date (analysis_date),
    INDEX idx_recommendation (recommendation),
    INDEX idx_confidence_score (confidence_score DESC),
    INDEX idx_strength_score (strength_score DESC),

    FOREIGN KEY (symbol) REFERENCES stocks(symbol) ON DELETE CASCADE
);

-- Create analysis_performance table to track recommendation success
CREATE TABLE analysis_performance (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    analysis_id BIGINT NOT NULL,
    symbol VARCHAR(20) NOT NULL,
    recommendation_date DATE NOT NULL,
    entry_price DECIMAL(12,2),
    exit_price DECIMAL(12,2),
    exit_date DATE,
    exit_reason ENUM('TARGET_HIT', 'STOP_LOSS', 'TIME_BASED', 'MANUAL'),
    actual_return DECIMAL(8,4),
    predicted_return DECIMAL(8,4),
    success BOOLEAN,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_symbol_rec_date (symbol, recommendation_date),
    INDEX idx_success (success),
    INDEX idx_exit_reason (exit_reason),

    FOREIGN KEY (analysis_id) REFERENCES btst_analysis(id) ON DELETE CASCADE,
    FOREIGN KEY (symbol) REFERENCES stocks(symbol) ON DELETE CASCADE
);
```

### Database Configuration Class

### src/main/java/com/stockanalyzer/config/DatabaseConfig.java

```
package com.stockanalyzer.config;

import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.jpa.repository.config.EnableJpaAuditing;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
import org.springframework.transaction.annotation.EnableTransactionManagement;

import javax.sql.DataSource;
import com.zaxxer.hikari.HikariConfig;
import com.zaxxer.hikari.HikariDataSource;

@Configuration
@EnableJpaRepositories(basePackages = "com.stockanalyzer.repository")
@EnableJpaAuditing
@EnableTransactionManagement
public class DatabaseConfig {

    @Bean
    @ConfigurationProperties("spring.datasource.hikari")
    public HikariConfig hikariConfig() {
        return new HikariConfig();
    }
    @Bean
    public DataSource dataSource() {
        return new HikariDataSource(hikariConfig());
    }
}
```

### Docker Setup for MySQL

### docker/docker-compose.yml

```
version: '3.8'

services:
  mysql:
    image: mysql:8.0
    container_name: stock-analyzer-mysql
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: stock_analyzer
      MYSQL_USER: stock_user
      MYSQL_PASSWORD: stock_password
    ports:
      - "3306:3306"
    volumes:
      - mysql_data:/var/lib/mysql
      - ./mysql/init.sql:/docker-entrypoint-initdb.d/init.sql
    command: --default-authentication-plugin=mysql_native_password
    restart: unless-stopped

  stock-analyzer:
    build: ..
    container_name: stock-analyzer-app
    depends_on:
      - mysql
    environment:
      DB_USERNAME: stock_user
      DB_PASSWORD: stock_password
      SPRING_DATASOURCE_URL: jdbc:mysql://mysql:3306/stock_analyzer?useSSL=false&serverTi
    ports:
      - "8080:8080"
    restart: unless-stopped

volumes:
  mysql_data:
```

### docker/mysql/init.sql

```
-- Initial database setup
CREATE DATABASE IF NOT EXISTS stock_analyzer;
USE stock_analyzer;

-- Create user if not exists
```

```
CREATE USER IF NOT EXISTS 'stock_user'@'%' IDENTIFIED BY 'stock_password';
GRANT ALL PRIVILEGES ON stock_analyzer.* TO 'stock_user'@'%';
FLUSH PRIVILEGES;

-- Set MySQL configuration for stock data optimization
SET GLOBAL innodb_buffer_pool_size = 1073741824; -- 1GB
SET GLOBAL max_connections = 500;
SET GLOBAL query_cache_size = 268435456; -- 256MB
```

### Updated Main Application Class

### src/main/java/com/stockanalyzer/StockAnalyzerApplication.java

```
package com.stockanalyzer;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.scheduling.annotation.EnableAsync;
import org.springframework.scheduling.annotation.EnableScheduling;
import org.springframework.transaction.annotation.EnableTransactionManagement;

@SpringBootApplication
@EnableScheduling
@EnableAsync
@EnableTransactionManagement
public class StockAnalyzerApplication {

    public static void main(String[] args) {
        SpringApplication.run(StockAnalyzerApplication.class, args);
    }
}
```

### Enhanced Repository Interfaces for MySQL

### src/main/java/com/stockanalyzer/repository/PriceDataRepository.java

```
package com.stockanalyzer.repository;

import com.stockanalyzer.entity.PriceData;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import java.math.BigDecimal;
import java.time.LocalDate;
import java.util.List;

@Repository
public interface PriceDataRepository extends JpaRepository<PriceData, Long> {
    PriceData findBySymbolAndTradeDate(String symbol, LocalDate tradeDate);

    List<PriceData> findBySymbolAndTradeDateBetweenOrderByTradeDate(
            String symbol, LocalDate startDate, LocalDate endDate);

    @Query("SELECT DISTINCT p.symbol FROM PriceData p WHERE p.tradeDate = :date")
    List<String> findActiveSymbols(@Param("date") LocalDate date);

    @Query("SELECT p FROM PriceData p WHERE p.tradeDate = :date " +
           "AND p.volume > :minVolume AND p.valueTraded > :minValue " +
           "ORDER BY p.volume DESC")
    List<PriceData> findByTradeDateAndVolumeGreaterThanAndValueGreaterThan(
            @Param("date") LocalDate date,
            @Param("minVolume") Long minVolume,
            @Param("minValue") Long minValue);

    @Query("SELECT p FROM PriceData p WHERE p.symbol = :symbol " +
           "AND p.tradeDate < :date ORDER BY p.tradeDate DESC LIMIT :limit")
    List<PriceData> findTopNBySymbolAndTradeDateLessThanOrderByTradeDateDesc(
            @Param("symbol") String symbol,
            @Param("date") LocalDate date,
            @Param("limit") int limit);

    @Query("SELECT p FROM PriceData p WHERE p.symbol = :symbol " +
           "AND p.tradeDate <= :date ORDER BY p.tradeDate DESC LIMIT :limit")
    List<PriceData> findTopNBySymbolAndTradeDateLessThanEqualOrderByTradeDateDesc(
            @Param("symbol") String symbol,
            @Param("date") LocalDate date,
            @Param("limit") int limit);

    // Performance optimized queries with MySQL specific hints
    @Query(value = "SELECT * FROM price_data USE INDEX (idx_symbol_date_desc) " +
                   "WHERE symbol = :symbol AND trade_date BETWEEN :startDate AND :endDate
                   "ORDER BY trade_date DESC", nativeQuery = true)
    List<PriceData> findHistoricalDataOptimized(
            @Param("symbol") String symbol,
            @Param("startDate") LocalDate startDate,
            @Param("endDate") LocalDate endDate);
}
```

### Updated BTST Analysis Service for MySQL

### src/main/java/com/stockanalyzer/service/BTSTAnalysisService.java

```
package com.stockanalyzer.service;

import com.stockanalyzer.entity.BTSTAnalysis;
import com.stockanalyzer.entity.PriceData;
import com.stockanalyzer.entity.TechnicalIndicator;
import com.stockanalyzer.repository.BTSTAnalysisRepository;
import com.stockanalyzer.repository.PriceDataRepository;
import com.stockanalyzer.repository.TechnicalIndicatorRepository;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.Executor;
import java.util.concurrent.Executors;

@Service
@Slf4j
@Transactional
public class BTSTAnalysisService {

    private final PriceDataRepository priceDataRepository;
    private final TechnicalIndicatorRepository technicalIndicatorRepository;
    private final BTSTAnalysisRepository btstAnalysisRepository;
    private final Executor analysisExecutor;

    @Value("${btst.analysis.min-volume:500000}")
    private Long minVolume;

    @Value("${btst.analysis.min-value:50000000}")
    private Long minValue;

    @Value("${btst.analysis.confidence-threshold:50.0}")
    private Double confidenceThreshold;

    @Autowired
    public BTSTAnalysisService(PriceDataRepository priceDataRepository,
                              TechnicalIndicatorRepository technicalIndicatorRepository,
                              BTSTAnalysisRepository btstAnalysisRepository) {
        this.priceDataRepository = priceDataRepository;
        this.technicalIndicatorRepository = technicalIndicatorRepository;
        this.btstAnalysisRepository = btstAnalysisRepository;
        this.analysisExecutor = Executors.newFixedThreadPool(10);
    }

    @Scheduled(cron = "0 0 18 * * MON-FRI", zone = "Asia/Kolkata")
    public void runDailyBTSTAnalysis() {
        LocalDate today = LocalDate.now();
        LocalDate yesterday = today.minusDays(1);

        log.info("Starting BTST analysis for date: {}", today);

        try {
            List<String> btstCandidates = identifyDay1BTSTCandidates(yesterday);
            log.info("Found {} Day-1 BTST candidates", btstCandidates.size());

            List<CompletableFuture<BTSTAnalysis>> futures = new ArrayList<>();

            for (String symbol : btstCandidates) {
                CompletableFuture<BTSTAnalysis> future = CompletableFuture
```

```
                    .supplyAsync(() -> analyzeWeakHandsShakeout(symbol, yesterday, today)
                    .exceptionally(throwable -> {
                        log.error("Error analyzing BTST for symbol {}: {}", symbol, throw
                        return null;
                    });
                futures.add(future);
            }

            // Wait for all analyses to complete and save results
            CompletableFuture.allOf(futures.toArray(new CompletableFuture[^3_0]))
                .thenRun(() -> {
                    futures.stream()
                        .map(CompletableFuture::join)
                        .filter(analysis -> analysis != null)
                        .forEach(btstAnalysisRepository::save);

                    log.info("Completed BTST analysis for {} candidates", btstCandidates.
                });

        } catch (Exception e) {
            log.error("Error in daily BTST analysis: {}", e.getMessage(), e);
        }
    }

    private List<String> identifyDay1BTSTCandidates(LocalDate date) {
        List<String> candidates = new ArrayList<>();

        List<PriceData> potentialStocks = priceDataRepository
                .findByTradeDateAndVolumeGreaterThanAndValueGreaterThan(date, minVolume,

        for (PriceData stock : potentialStocks) {
            if (qualifiesAsDay1BTST(stock, date)) {
                candidates.add(stock.getSymbol());
            }
        }

        return candidates;
    }

    // Rest of the analysis methods remain the same but with MySQL optimizations...
}
```

## Updated Enhanced Research Prompt for API Integration

Here's your **completely updated prompt** that integrates with the new MySQL-based endpoints:

# Enhanced BTST Research Prompt with Technical API Integration

**Act as a dedicated team of India-focused equity analysts, market microstructure specialists, and systematic swing traders with access to a proprietary technical analysis API. Each trading day, 30–60 minutes before close, deliver 1–3 high-probability "BTST-Next-Day Continuation" stock ideas for NSE/BSE that systematically exploit prior-day BTST unwind and weak-hands shakeout using quantified technical data.**

## Research Workflow Integration

### Step 1: Data Collection from Technical API

**Before conducting market research, call these endpoints to get quantified data:**

1. **Get BTST Candidates**: `GET /api/analysis/screening/candidates?date=YYYY-MM-DD`
    - Retrieve Day-1 BTST candidates with volume/breakout metrics
    - Use this to focus research on pre-qualified stocks only
2. **Get Technical Setup Data**: `GET /api/analysis/technical/{SYMBOL}?date=YYYY-MM-DD`
    - Obtain calculated RSI, EMA, volume ratios, strength scores
    - Validate technical setup with quantified indicators
3. **Get Analysis Results**: `GET /api/analysis/btst/recommendations?date=YYYY-MM-DD&recommendation=BUY`
    - Retrieve stocks that passed weak-hands shakeout filters
    - Focus catalyst research on high-confidence candidates only

### Step 2: Enhanced Research Framework

**Using API data as foundation, research these specific areas:**

**2A) Market Context Research (only if API shows market-wide patterns)**

- Search: `"Nifty market sentiment {today's date}"`, `"FII DII flows {today's date}"`, `"sector rotation India {today's date}"`
- Cross-reference with API market summary endpoint data

**2B) Catalyst Validation for API-Identified Stocks**

For each symbol from `/api/analysis/screening/candidates`:

- Search: `"{SYMBOL} earnings results {recent date}"`, `"{SYMBOL} order wins news {today's date}"`, `"{SYMBOL} analyst upgrade {today's date}"`
- Search: `"{SYMBOL} management commentary {recent date}"`, `"{SYMBOL} sector policy news {today's date}"`

**2C) Weak-Hands Confirmation Research**

For stocks showing high confidence scores from API:

- Search: `"{SYMBOL} retail vs institutional buying {today's date}"`, `"{SYMBOL} delivery percentage {today's date}"`
- Search: `"{SYMBOL} block deals {today's date}"`, `"{SYMBOL} insider trading {recent date}"`

### Step 3: Technical Validation with API Data

**Cross-verify web research with API technical endpoints:**

3A) **Volume Analysis**: Compare news volume with API volume ratios from `/api/analysis/technical/{SYMBOL}`
3B) **Price Action**: Validate breakout news with API-calculated breakout levels and strength scores
3C) **Absorption Evidence**: Confirm weak-hands exit with API cumulative delta and retail intensity metrics

### Step 4: Final Stock Selection Criteria

**Only recommend stocks that satisfy ALL of the following:**

**4A) API Qualification (MANDATORY)**

- Appears in `/api/analysis/screening/candidates` with volume ratio > 1.5x
- Technical analysis endpoint shows RSI between 45-70, EMA(9) > EMA(21)
- BTST analysis endpoint confidence score > 60 (if available)

**4B) Catalyst Confirmation (from web research)**

- Identified at least one concrete catalyst: earnings beat, order win, policy benefit, analyst upgrade
- No major negative news in past 2 trading sessions
- Sector showing relative strength or neutral sentiment

**4C) Weak-Hands Evidence (API + research combination)**

- API shows retail intensity < 60% and cumulative delta positive
- Research confirms institutional interest or reduced retail selling
- Delivery percentage trending upward vs 3-month average

### Step 5: Research Output Template

**For each qualified stock (1-3 maximum):**

**{SYMBOL} - {Sector} | API Confidence: {score}/100**

**Technical Setup (API-Verified):**

- Volume Ratio: {from API} | Strength Score: {from API}
- RSI(14): {from API} | EMA Position: {from API}

- Breakout Level: ₹{from API} | Current: ₹{market price}

**Catalyst Analysis (Research-Verified):**

- Primary Catalyst: {specific news with date and source}
- Secondary Factor: {additional supporting news}
- Catalyst Durability: {1-3 days / 1 week / ongoing}

**Weak-Hands Shakeout Evidence:**

- API Retail Intensity: {percentage} | Cumulative Delta: {value}
- Research Finding: {institutional buying/selling evidence}
- Supply Absorption: {evidence from price action + volume}

**Price Plan (API-Assisted):**

- Entry Zone: ₹{API suggested range}
- Target 1: ₹{3-5% above entry} | Target 2: ₹{5-8% above entry}
- Stop Loss: ₹{API suggested or 2% below entry}
- Position Size: {based on API volatility data}

**Risk Assessment:**

- API Confidence Score: {score}/100
- Market Context Risk: {sector/market headwinds}
- Execution Risk: {liquidity/gap risk factors}

### Step 6: No-Recommendation Protocol

**If no stocks meet all criteria, state explicitly:**
"No stocks meet the combined API qualification (confidence > 60) and catalyst confirmation criteria today. Avoiding forced recommendations."

### Step 7: Performance Tracking Integration

**Reference historical accuracy:**

- Query `/api/analysis/market-summary?date={recent dates}` to mention recent success rate
- Include: "Historical API+Research combination shows {X}% success rate over past 30 recommendations"

### Key Integration Benefits:

1. **Quantified Entry**: No more subjective "volume surge" - get exact ratios and confidence scores
2. **Pre-Filtered Universe**: Research only pre-qualified candidates instead of entire market
3. **Technical Validation**: Cross-verify news impact with calculated technical indicators

4. **Systematic Approach**: Combine algorithmic filtering with fundamental catalyst research
5. **Performance Tracking**: Continuous improvement through API-tracked historical results

**Execution Frequency**: Use this enhanced prompt daily 30-60 minutes before market close, ensuring API endpoints are called first to focus research efforts efficiently.[33] [34] [35] [36] [37] [38] [39]

This updated approach creates a powerful synergy between your quantitative technical analysis system and qualitative market research, significantly improving recommendation accuracy while reducing research time.

⁂

# Fully Automated System - Zero Manual Intervention

You're absolutely correct! For complete automation, every `{placeholder}` must be replaced with calculated values. Here's the updated system with **automated risk assessment calculations** and a **completely automated prompt**:

### Enhanced Risk Assessment Service

### src/main/java/com/stockanalyzer/service/RiskAssessmentService.java

```
package com.stockanalyzer.service;

import com.stockanalyzer.entity.PriceData;
import com.stockanalyzer.entity.TechnicalIndicator;
import com.stockanalyzer.entity.BTSTAnalysis;
import com.stockanalyzer.repository.PriceDataRepository;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.time.LocalDate;
import java.util.List;

@Service
@Slf4j
public class RiskAssessmentService {

    private final PriceDataRepository priceDataRepository;

    @Autowired
    public RiskAssessmentService(PriceDataRepository priceDataRepository) {
        this.priceDataRepository = priceDataRepository;
    }

    public LiquidityRisk calculateLiquidityRisk(String symbol, LocalDate date) {
        try {
            PriceData currentData = priceDataRepository.findBySymbolAndTradeDate(symbol,
            List<PriceData> last20Days = priceDataRepository
                    .findTopNBySymbolAndTradeDateLessThanEqualOrderByTradeDateDesc(symbol

            if (currentData == null || last20Days.size() < 10) {
                return new LiquidityRisk("HIGH", "Insufficient data for liquidity assessm
            }

            // Calculate average daily volume (20-day)
            double avgVolume = last20Days.stream()
                    .mapToLong(PriceData::getVolume)
                    .average()
                    .orElse(0.0);

            // Calculate average daily turnover (20-day)
            double avgTurnover = last20Days.stream()
                    .mapToLong(PriceData::getValueTraded)
                    .average()
                    .orElse(0.0);

            // Calculate bid-ask spread approximation (using high-low range as proxy)
            double dayRange = currentData.getHighPrice().subtract(currentData.getLowPrice
            double midPrice = currentData.getHighPrice().add(currentData.getLowPrice())
                    .divide(new BigDecimal("2"), RoundingMode.HALF_UP).doubleValue();
            double spreadPercentage = midPrice > 0 ? (dayRange / midPrice) * 100 : 0;

            // Calculate turnover ratio (current vs average)
            double turnoverRatio = avgTurnover > 0 ? currentData.getValueTraded() / avgTu

            // Liquidity Risk Scoring Algorithm
            String riskLevel = calculateLiquidityRiskLevel(
                    currentData.getVolume(), avgVolume,
                    currentData.getValueTraded(), avgTurnover,
                    spreadPercentage, turnoverRatio
            );

            String riskFactors = generateLiquidityRiskFactors(
                    currentData.getVolume(), avgVolume,
                    spreadPercentage, turnoverRatio
            );

            return new LiquidityRisk(riskLevel, riskFactors);

        } catch (Exception e) {
            log.error("Error calculating liquidity risk for {}: {}", symbol, e.getMessage
            return new LiquidityRisk("HIGH", "Error in liquidity calculation");
        }
    }

    private String calculateLiquidityRiskLevel(long currentVolume, double avgVolume,
                                              long currentTurnover, double avgTurnover,
                                              double spreadPercentage, double turnoverRat
        int score = 0;

        // Volume criteria
        if (currentVolume > avgVolume * 1.5) score += 3; // Excellent volume
        else if (currentVolume > avgVolume) score += 2;   // Good volume
        else if (currentVolume > avgVolume * 0.5) score += 1; // Fair volume
        // Below 0.5x avg = 0 points (poor)

        // Turnover criteria
        if (currentTurnover > 100000000L) score += 3;     // > 10 Cr (Excellent)
        else if (currentTurnover > 50000000L) score += 2;  // > 5 Cr (Good)
        else if (currentTurnover > 20000000L) score += 1;  // > 2 Cr (Fair)

        // Spread criteria (lower is better)
        if (spreadPercentage < 0.5) score += 3;      // Tight spread
        else if (spreadPercentage < 1.0) score += 2;  // Moderate spread
        else if (spreadPercentage < 2.0) score += 1;  // Wide spread

        // Turnover consistency
        if (turnoverRatio > 0.8 && turnoverRatio < 2.0) score += 2; // Consistent
        else if (turnoverRatio > 0.5) score += 1; // Somewhat consistent

        // Risk Level Mapping (0-11 scale)
        if (score >= 9) return "LOW";
        else if (score >= 6) return "MEDIUM";
        else return "HIGH";
    }

    private String generateLiquidityRiskFactors(long currentVolume, double avgVolume,
                                               double spreadPercentage, double turnoverRa
        StringBuilder factors = new StringBuilder();

        if (currentVolume < avgVolume * 0.5) {
            factors.append("Low volume (").append(String.format("%.1f", (currentVolume /
                   .append("% of avg), ");
        }

        if (spreadPercentage > 1.5) {
            factors.append("Wide bid-ask spread (~").append(String.format("%.2f", spread
                   .append("%), ");
        }

        if (turnoverRatio < 0.3) {
            factors.append("Low turnover consistency (").append(String.format("%.1f", tu
                   .append("% of avg), ");
        }

        if (factors.length() == 0) {
            factors.append("Normal liquidity conditions");
        } else {
            // Remove trailing comma and space
            factors.setLength(factors.length() - 2);
        }

        return factors.toString();
    }

    public GapRisk calculateGapRisk(String symbol, LocalDate date, BTSTAnalysis analysis)
        try {
            PriceData currentData = priceDataRepository.findBySymbolAndTradeDate(symbol,
            List<PriceData> last10Days = priceDataRepository
                    .findTopNBySymbolAndTradeDateLessThanOrderByTradeDateDesc(symbol, dat

            if (currentData == null || last10Days.isEmpty()) {
                return new GapRisk("HIGH", "Insufficient price data for gap risk assessme
            }

            // Calculate historical gap frequency
            int gapCount = 0;
            double totalGapSize = 0.0;

            for (int i = 0; i < last10Days.size() - 1; i++) {
                PriceData today = last10Days.get(i);
                PriceData yesterday = last10Days.get(i + 1);

                double gapPercentage = Math.abs(
                    (today.getOpenPrice().doubleValue() - yesterday.getClosePrice().doubl
                    yesterday.getClosePrice().doubleValue() * 100
                );

                if (gapPercentage > 0.5) { // Consider gaps > 0.5%
                    gapCount++;
                    totalGapSize += gapPercentage;
                }
            }

            // Calculate average gap size
            double avgGapSize = gapCount > 0 ? totalGapSize / gapCount : 0;

            // Current gap risk based on various factors
            double gapRiskScore = calculateGapRiskScore(currentData, analysis, gapCount,

            String riskLevel = determineGapRiskLevel(gapRiskScore);
            String riskFactors = generateGapRiskFactors(gapCount, avgGapSize,
                                                       analysis != null ? analysis.getGap
                                                       currentData);

            return new GapRisk(riskLevel, riskFactors);

        } catch (Exception e) {
            log.error("Error calculating gap risk for {}: {}", symbol, e.getMessage());
            return new GapRisk("MEDIUM", "Error in gap risk calculation");
        }
    }

    private double calculateGapRiskScore(PriceData currentData, BTSTAnalysis analysis,
                                        int historicalGaps, double avgGapSize) {
        double score = 0;

        // Historical gap frequency (more gaps = higher risk)
        if (historicalGaps > 5) score += 3;      // Very volatile
        else if (historicalGaps > 3) score += 2; // Moderately volatile
        else if (historicalGaps > 1) score += 1; // Some volatility

        // Average gap size
        if (avgGapSize > 3.0) score += 3;      // Large gaps
        else if (avgGapSize > 1.5) score += 2; // Medium gaps
        else if (avgGapSize > 0.5) score += 1; // Small gaps

        // Current volatility (using daily range)
        double dayRange = currentData.getHighPrice().subtract(currentData.getLowPrice()).
        double rangePercentage = (dayRange / currentData.getClosePrice().doubleValue()) *

        if (rangePercentage > 4.0) score += 2;      // High intraday volatility
        else if (rangePercentage > 2.0) score += 1; // Medium volatility

        // BTST specific factors
        if (analysis != null) {
            if (Math.abs(analysis.getGapPercentage() != null ? analysis.getGapPercentage(
                score += 2; // Current gap observed
            }

            if (Boolean.FALSE.equals(analysis.getShowsAbsorption())) {
                score += 1; // Poor absorption increases gap risk
            }
        }

        return score; // Max score ~11
    }

    private String determineGapRiskLevel(double score) {
        if (score <= 3) return "LOW";
        else if (score <= 6) return "MEDIUM";
        else return "HIGH";
    }

    private String generateGapRiskFactors(int gapCount, double avgGapSize,
                                         double currentGap, PriceData currentData) {
        StringBuilder factors = new StringBuilder();

        if (gapCount > 3) {
            factors.append("High gap frequency (").append(gapCount)
                   .append(" gaps in 10 days), ");
        }

        if (avgGapSize > 2.0) {
            factors.append("Large historical gaps (avg ").append(String.format("%.1f", av
                   .append("%%), ");
        }

        if (Math.abs(currentGap) > 1.0) {
            factors.append("Current gap present (").append(String.format("%.1f", currentG
                   .append("%%), ");
        }

        double dayRange = currentData.getHighPrice().subtract(currentData.getLowPrice()).
        double rangePercentage = (dayRange / currentData.getClosePrice().doubleValue()) *

        if (rangePercentage > 3.0) {
            factors.append("High intraday volatility (").append(String.format("%.1f", ran
                   .append("%%), ");
        }

        if (factors.length() == 0) {
            factors.append("Normal gap risk profile");
        } else {
            factors.setLength(factors.length() - 2);
        }

        return factors.toString();
    }

    // Helper classes
    public static class LiquidityRisk {
        private final String level;
        private final String factors;

        public LiquidityRisk(String level, String factors) {
            this.level = level;
            this.factors = factors;
        }

        public String getLevel() { return level; }
        public String getFactors() { return factors; }
    }

    public static class GapRisk {
        private final String level;
        private final String factors;

        public GapRisk(String level, String factors) {
            this.level = level;
            this.factors = factors;
        }

        public String getLevel() { return level; }
        public String getFactors() { return factors; }
    }
}
```

### Updated Analysis Controller with Automated Risk

### Enhanced AnalysisController.java (additions)

```
@GetMapping("/btst/detailed/{symbol}")
public ResponseEntity<BTSTDetailedAnalysisDTO> getDetailedAnalysis(
        @PathVariable String symbol,
        @RequestParam(defaultValue = "0") LocalDate date) {

    LocalDate analysisDate = date.equals(LocalDate.of(1970, 1, 1)) ? LocalDate.now() : da

    BTSTAnalysis analysis = btstAnalysisRepository
            .findBySymbolAndAnalysisDate(symbol, analysisDate);

    if (analysis == null) {
        return ResponseEntity.notFound().build();
    }

    // Calculate automated risk assessments
    RiskAssessmentService.LiquidityRisk liquidityRisk =
            riskAssessmentService.calculateLiquidityRisk(symbol, analysisDate);

    RiskAssessmentService.GapRisk gapRisk =
            riskAssessmentService.calculateGapRisk(symbol, analysisDate, analysis);

    BTSTDetailedAnalysisDTO detailedAnalysis = convertToDetailedAnalysisDTO(
            analysis, liquidityRisk, gapRisk);

    return ResponseEntity.ok(detailedAnalysis);
}
```

### Completely Automated Research Prompt (Zero Manual Input)

```
# Fully Automated BTST Research System - Zero Manual Intervention

**System Role**: Act as an automated equity analysis engine that delivers 1–3 high-probab

## Automated Execution Workflow

### Phase 1: Technical Data Retrieval (AUTOMATED)
**Execute API calls in sequence:**

1. **Market Context**: `GET /api/analysis/market-summary?date={TODAY}`
   - Auto-populate market sentiment score, total candidates analyzed
   - Auto-determine if market conditions favor BTST strategies (>50% success rate)

2. **Qualified Candidates**: `GET /api/analysis/screening/candidates?date={YESTERDAY}`
   - Auto-filter for: volume_ratio > 1.5, breakout_confirmed = true, min_turnover > 5cr
   - Auto-rank by combined volume_ratio × strength_score

3. **Technical Validation**: `GET /api/analysis/technical/{SYMBOL}?date={TODAY}` for top
   - Auto-filter: RSI between 45-70, EMA(9) > EMA(21), volume_strength > 60

4. **BTST Analysis**: `GET /api/analysis/btst/recommendations?date={TODAY}&recommendatio
   - Auto-select candidates with confidence_score > 60

### Phase 2: Automated Catalyst Research (SYSTEMATIC)
**For each API-qualified symbol, execute automated searches:**

**Template A - Earnings/Results**: "{SYMBOL} earnings results {current_quarter}"
**Template B - Corporate Actions**: "{SYMBOL} order wins {last_7_days}"
**Template C - Analyst Actions**: "{SYMBOL} analyst upgrade downgrade {last_30_days}"
**Template D - Sector Policy**: "{SYMBOL} sector policy news India {last_15_days}"
**Template E - Management Updates**: "{SYMBOL} management guidance commentary {last_30_da

**Auto-scoring system:**
- Positive earnings surprise = +3 points
- Major order win/contract = +3 points
- Analyst upgrade = +2 points
- Favorable policy news = +2 points
- Positive management commentary = +1 point
- No negative news in 48hrs = +1 point
**Minimum catalyst score required: 4 points**

### Phase 3: Automated Risk Assessment (CALCULATED)
**For each qualified candidate, auto-calculate:**

**Liquidity Risk Auto-Assessment:**
- Current volume vs 20-day average: Auto-calculated ratio
- Bid-ask spread estimation: Auto-derived from day's high-low range
- Turnover consistency: Auto-computed standard deviation
- **Auto Risk Level**: LOW/MEDIUM/HIGH based on algorithmic scoring

**Gap Risk Auto-Assessment:**
- Historical gap frequency: Auto-count gaps >0.5% in last 10 days
- Average gap magnitude: Auto-calculated from historical data
- Current volatility: Auto-derived from ATR and daily range
- **Auto Risk Level**: LOW/MEDIUM/HIGH based on algorithmic scoring

### Phase 4: Automated Output Generation (ZERO MANUAL INPUT)

**Final Recommendations (1-3 stocks maximum):**

---

**{AUTO_SYMBOL_1} - {AUTO_SECTOR} | Confidence: {API_CONFIDENCE_SCORE}/100**

**Technical Setup (API-Generated):**
- Volume Ratio: {API_VOLUME_RATIO}x | Strength Score: {API_STRENGTH_SCORE}/100
- RSI(14): {API_RSI_VALUE} | EMA Status: {AUTO_EMA_TREND_DIRECTION}
- Breakout Level: ₹{API_BREAKOUT_LEVEL} | Current: ₹{LIVE_MARKET_PRICE}

**Catalyst Analysis (Auto-Researched):**
- Primary Catalyst: {AUTO_IDENTIFIED_CATALYST_WITH_DATE_SOURCE}
- Catalyst Score: {AUTO_CALCULATED_CATALYST_POINTS}/12
- Durability Assessment: {AUTO_DETERMINED_TIMELINE}

**Weak-Hands Evidence (Auto-Validated):**
- Retail Intensity: {API_RETAIL_INTENSITY}% | Delta: {API_CUMULATIVE_DELTA}
- Absorption Quality: {API_ABSORPTION_BOOLEAN} | Supply Status: {API_SUPPLY_EXHAUSTION}

**Price Plan (Algorithm-Generated):**
- Entry Zone: ₹{API_ENTRY_PRICE} ± {AUTO_ENTRY_TOLERANCE}
- Target 1: ₹{AUTO_CALCULATED_T1} (Risk-Reward: {AUTO_RR_RATIO_T1})
- Target 2: ₹{AUTO_CALCULATED_T2} (Risk-Reward: {AUTO_RR_RATIO_T2})
- Stop Loss: ₹{API_STOP_LOSS} (Max Loss: {AUTO_MAX_LOSS_PERCENT}%)

**Risk Assessment (Auto-Computed):**
- API Confidence: {API_CONFIDENCE_SCORE}/100
- Liquidity Risk: {AUTO_LIQUIDITY_RISK_LEVEL} - {AUTO_LIQUIDITY_FACTORS}
- Gap Risk: {AUTO_GAP_RISK_LEVEL} - {AUTO_GAP_RISK_FACTORS}
- Position Size: {AUTO_POSITION_SIZE}% of portfolio (ATR-based)

---

### Phase 5: Automated Quality Control
```

```
**System Auto-Validates Each Recommendation:**
- ✅ API confidence score ≥ 60: {AUTO_CHECK_RESULT}
- ✅ Catalyst score ≥ 4 points: {AUTO_CHECK_RESULT}
- ✅ Combined risk level ≤ MEDIUM: {AUTO_CHECK_RESULT}
- ✅ Technical setup aligned: {AUTO_CHECK_RESULT}

**If any validation fails**: Auto-exclude recommendation and move to next candidate

**If no candidates pass all validations**:
**Auto-Response**: "SYSTEMATIC ANALYSIS: No stocks meet the automated qualification crite

### Phase 6: Performance Tracking Integration

**Auto-Generated Context:**
- "System Performance: Last 30 recommendations achieved {AUTO_SUCCESS_RATE}% target hit r
- "Current market regime: {AUTO_MARKET_REGIME} (based on {AUTO_SAMPLE_SIZE} recent analys
- "Today's qualified universe: {AUTO_TOTAL_CANDIDATES} stocks screened, {AUTO_FINAL_COUNT

## Complete Automation Benefits:

1. **Zero Subjectivity**: All risk factors, catalyst scores, and price levels algorithmic
2. **Consistent Quality**: Same rigorous filtering applied regardless of market condition
3. **Scalable Research**: Can analyze entire NSE/BSE universe without human bandwidth lim
4. **Performance Tracking**: Continuous learning from automated success/failure metrics
5. **Speed Advantage**: Complete analysis in minutes vs hours of manual research

**Execution Trigger**: System auto-executes 30 minutes before market close, delivering re

---

**This prompt eliminates ALL manual placeholders and creates a fully automated research a
```

### Additional Database Fields for Complete Automation

Add these fields to store automated calculations:

### Migration: V5__add_automation_fields.sql

```
-- Add automated risk assessment fields to btst_analysis table
ALTER TABLE btst_analysis ADD COLUMN liquidity_risk_level ENUM('LOW', 'MEDIUM', 'HIGH');
ALTER TABLE btst_analysis ADD COLUMN liquidity_risk_factors TEXT;
ALTER TABLE btst_analysis ADD COLUMN gap_risk_level ENUM('LOW', 'MEDIUM', 'HIGH');
ALTER TABLE btst_analysis ADD COLUMN gap_risk_factors TEXT;
ALTER TABLE btst_analysis ADD COLUMN catalyst_score INT DEFAULT 0;
ALTER TABLE btst_analysis ADD COLUMN catalyst_details TEXT;
ALTER TABLE btst_analysis ADD COLUMN automated_risk_assessment BOOLEAN DEFAULT TRUE;
ALTER TABLE btst_analysis ADD COLUMN position_size_percent DECIMAL(4,2);
ALTER TABLE btst_analysis ADD COLUMN risk_reward_ratio_t1 DECIMAL(4,2);
ALTER TABLE btst_analysis ADD COLUMN risk_reward_ratio_t2 DECIMAL(4,2);

-- Create automated_research_results table
CREATE TABLE automated_research_results (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    symbol VARCHAR(20) NOT NULL,
    research_date DATE NOT NULL,
    catalyst_type VARCHAR(100),
    catalyst_description TEXT,
    catalyst_score INT DEFAULT 0,
    news_sentiment_score DECIMAL(4,2),
    research_source VARCHAR(200),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_symbol_research_date (symbol, research_date),
    INDEX idx_catalyst_score (catalyst_score DESC),

    FOREIGN KEY (symbol) REFERENCES stocks(symbol) ON DELETE CASCADE
);
```

This completely automated system eliminates all manual intervention, making your BTST analysis system truly systematic and scalable.[53] [54] [55] [56] [57] [58] [59] [60] [61] [62]

⁂

1. https://github.com/OpenApi-5p/5paisa-java
2. https://nsearchives.nseindia.com/web/sites/default/files/inline-files/Proposed UDiFF Bhavcopy file formats.xlsx
3. https://roboquant.org/tutorial/tech-analysis.html
4. https://xstream.5paisa.com/dev-docs/order-management-system/place-order
5. https://github.com/BYVoid/TA-Lib
6. https://ta-lib.github.io
7. https://www.geeksforgeeks.org/springboot/java-spring-boot-microservices-example-step-by-step-guide/
8. https://www.geeksforgeeks.org/java/spring-boot-scheduling/
9. https://cronitor.io/guides/java-cron-jobs
10. https://www.baeldung.com/spring-scheduled-tasks
11. https://tradingqna.com/t/has-nse-changed-bhavcopy-location/169551
12. https://github.com/ta4j/ta4j
13. https://dzone.com/articles/using-technical-analysis-indicators-to-send-buy-or
14. https://www.nseindia.com/trade/membership-formats
15. https://github.com/OpenApi-5p
16. https://www.nseindia.com/all-reports-derivatives
17. https://ta-lib.org
18. https://xstream.5paisa.com/dev-docs
19. https://stackoverflow.com/questions/68700205/how-to-call-nse-api-for-historical-data-free
20. https://www.geeksforgeeks.org/advance-java/java-spring-boot-microservices-develop-api-gateway-using-spring-cloud-gateway/
21. https://globaldatafeeds.in/apis/

22. https://stackoverflow.com/questions/61748901/how-to-use-cron-expression-with-scheduled-annotation-in-spring
23. https://www.springboottutorial.com/microservices-and-restful-services-with-spring-boot-for-beginners
24. https://www.nseindia.com/market-data/real-time-data-subscription
25. https://spring.io/microservices
26. https://www.reddit.com/r/developersIndia/comments/19bkvwf/any_api_to_get_real_time_nsebse_data_for_free/
27. https://spring.io/guides/gs/scheduling-tasks
28. https://spring.io/guides/tutorials/rest
29. https://www.nseindia.com/market-data/eod-historical-data-subscription
30. https://spring.io/blog/2020/11/10/new-in-spring-5-3-improved-cron-expressions
31. https://devm.io/spring/spring-boot-tutorial-rest-services-and-microservices-135148-001
32. https://www.truedata.in/products/marketdataapi
33. https://www.codingshuttle.com/spring-boot-handbook/setting-up-mysql-db-with-spring-data-jpa
34. https://www.reddit.com/r/mysql/comments/xczyot/stocks_database_design_help/
35. https://www.geeksforgeeks.org/advance-java/spring-boot-database-integration-jpa-hibernate-mysql-h2/
36. https://javadevjournal.com/spring-boot/spring-boot-mysql/
37. https://spring.io/guides/gs/accessing-data-mysql
38. https://stackoverflow.com/questions/69768717/implement-database-schema-for-organizing-historical-stock-data
39. https://www.geeksforgeeks.org/dbms/how-to-design-a-database-for-stock-trading-app-like-groww/
40. https://reasonabledeviations.com/2018/02/01/stock-price-database/
41. https://dev.to/sanjaybsm/springboot-project-structure-4hhj
42. https://www.scribd.com/document/888210184/Database-Schema-Design
43. https://stackoverflow.com/questions/40902280/what-is-the-recommended-project-structure-for-spring-boot-rest-projects
44. https://mkyong.com/spring-boot/spring-boot-spring-data-jpa-mysql-example/
45. https://docs.spring.io/spring-boot/reference/using/structuring-your-code.html
46. https://stackoverflow.com/questions/25930191/connect-mysql-to-spring-application
47. https://www.linkedin.com/posts/nelsonamigoscode_systemdesign-coding-interviewtips-activity-733890423423232340480-4PQ5
48. https://www.youtube.com/watch?v=XyvsiZsUetU
49. https://spring.io/guides/gs/accessing-data-jpa
50. https://spring.io/guides/gs/spring-boot
51. https://symflower.com/en/company/blog/2024/spring-boot-folder-structure/
52. https://bootify.io/multi-module/best-practices-for-spring-boot-multi-module.html
53. https://xaubot.com/automated-trading-systems-ats-and-risk-management/
54. https://www.wallstreetprep.com/knowledge/liquidity/

55. https://help.trendspider.com/kb/indicators/gap-detector
56. https://blueberrymarkets.com/market-analysis/risks-of-automated-trading-systems/
57. https://www.shriramamc.in/glossary/what-is-liquidity-risk
58. https://blog.quantinsti.com/automated-trading-system/
59. https://wundertrading.com/journal/en/learn/article/automated-risk-management-in-crypto-trading
60. https://groww.in/p/liquidity-risk
61. https://www.investopedia.com/articles/trading/11/understanding-liquidity-risk.asp
62. https://fenefx.com/en/blog/review-of-automated-trading-systems/
63. https://www.luxalgo.com/blog/gap-patterns-candlestick-signal-breakdown/
64. https://analystprep.com/study-notes/frm/estimating-liquidity-risk/
65. https://trendspider.com/learning-center/fair-value-gap-trading-strategy/
66. https://blog.quantinsti.com/epat-project-gap-trading-strategy-based-on-the-markov-rule/
67. https://en.wikipedia.org/wiki/Automated_trading_system
68. https://www.youtube.com/watch?v=cjDgibEkJ_M
69. https://www.fia.org/fia/articles/fia-releases-best-practices-automated-trading-risk-controls-and-system-safeguards
70. https://www.bis.org/publ/ecsc07j.pdf
71. https://in.tradingview.com/scripts/gap/
72. https://www.ecb.europa.eu/press/financial-stability-publications/fsr/focus/2007/pdf/ecb~5bbc8f2202.fsrbox200712_15.pdf