NSE Stock Performance Tracker

Complete Code Documentation & Architecture Guide

Table of Contents

[1. Project Overview 4](#_Toc211872725)

[Key Capabilities: 4](#_Toc211872726)

[2. Architecture & Design 4](#_Toc211872727)

[3. File Structure 5](#_Toc211872728)

[4. Module-by-Module Explanation 6](#_Toc211872729)

[4.1 config.py 6](#_Toc211872730)

[Key Components: 6](#_Toc211872731)

[CSS Features: 6](#_Toc211872732)

[4.2 app.py 7](#_Toc211872733)

[Key Functions: 7](#_Toc211872734)

[Application Flow: 7](#_Toc211872735)

[Fetching Logic: 7](#_Toc211872736)

[4.3 data\_fetchers.py 8](#_Toc211872737)

[Key Functions: 8](#_Toc211872738)

[Stock Performance Logic: 8](#_Toc211872739)

[NSE CSV Strategy: 8](#_Toc211872740)

[4.4 ui\_components.py 9](#_Toc211872741)

[Key Functions: 9](#_Toc211872742)

[Live Ticker: 9](#_Toc211872743)

[4.5 utils.py 10](#_Toc211872744)

[Key Functions: 10](#_Toc211872745)

[Color Logic: 10](#_Toc211872746)

[4.6 cache\_manager.py 10](#_Toc211872747)

[Features: 10](#_Toc211872748)

[Key Functions: 10](#_Toc211872749)

[4.7 file\_manager.py 11](#_Toc211872750)

[Key Functions: 11](#_Toc211872751)

[5. Data Flow & Logic 11](#_Toc211872752)

[6. Key Features 12](#_Toc211872753)

[7. Performance Optimizations 13](#_Toc211872754)

[Caching: 13](#_Toc211872755)

[Parallel Fetching: 13](#_Toc211872756)

[Streamlit Caching: 13](#_Toc211872757)

[UI Optimizations: 13](#_Toc211872758)

[8. Usage Guide 14](#_Toc211872759)

[Installation: 14](#_Toc211872760)

[Running: 14](#_Toc211872761)

[Using the App: 14](#_Toc211872762)

[Uploading Custom Lists: 14](#_Toc211872763)

[9. Dependencies 15](#_Toc211872764)

# 1. Project Overview

The NSE Stock Performance Tracker is a Streamlit web application for tracking Indian stock market performance with real-time data, caching, and advanced visualizations.

## Key Capabilities:

* Track Nifty 50, Nifty Next 50, Nifty Total Market
* Real-time market indices
* 1-week to 3-month performance metrics
* Live rolling ticker
* Custom stock lists
* Persistent caching (6-hour expiry)
* Parallel data fetching
* Commodity prices

# 2. Architecture & Design

Modular architecture with clear separation of concerns:

**config.py:** Constants, fallback data, CSS styling

**app.py:** Main orchestration and business logic

**data\_fetchers.py:** External API calls and data retrieval

**ui\_components.py:** Streamlit UI rendering

**utils.py:** Helper functions for formatting

**cache\_manager.py:** Persistent caching with Pickle

**file\_manager.py:** File I/O for custom lists

# 3. File Structure

windsurf-project/  
├── app.py # Main entry point  
├── config.py # Configuration  
├── data\_fetchers.py # Data fetching  
├── ui\_components.py # UI rendering  
├── utils.py # Utilities  
├── cache\_manager.py # Cache management  
├── file\_manager.py # File operations  
├── requirements.txt # Dependencies  
├── cache/ # Cache storage  
└── saved\_stock\_lists/ # Custom lists

# 4. Module-by-Module Explanation

## 4.1 config.py

Centralizes configuration, constants, and styling.

### Key Components:

**SAVED\_LISTS\_DIR:** Directory for user stock lists

**ITEMS\_PER\_PAGE:** Pagination size (10)

**FALLBACK\_NIFTY\_50:** Hardcoded Nifty 50 stocks

**FALLBACK\_NIFTY\_NEXT\_50:** Hardcoded Nifty Next 50

**INDICES\_ROW1:** Major indices (Nifty, Sensex, etc.)

**INDICES\_ROW2:** Sectoral indices (IT, Pharma, etc.)

**COMMODITIES:** Commodity tickers (Oil, Gold, BTC)

**CUSTOM\_CSS:** Dark theme styling (440+ lines)

**METRIC\_CSS:** Metric component styling

### CSS Features:

* Dark theme (#1e1e1e)
* Responsive fonts (13-14px)
* Custom sidebar
* Rolling ticker animation
* Color-coded percentages
* Hover effects

## 4.2 app.py

Main application orchestrating the workflow.

### Key Functions:

**main():** Entry point, initializes app and manages workflow

**render\_stock\_selection\_sidebar():** Sidebar for category selection

**handle\_file\_upload():** Manages file upload and saved lists

**fetch\_stocks\_data():** Intelligent data fetching with mode selection

### Application Flow:

1. 1. Apply CSS
2. 2. Initialize session state
3. 3. Render header
4. 4. Render ticker
5. 5. Display sidebar
6. 6. Fetch stock list
7. 7. Apply sorting
8. 8. Fetch data
9. 9. Create DataFrame
10. 10. Paginate
11. 11. Display table
12. 12. Show performers

### Fetching Logic:

Smart mode selection based on dataset size:

• >100 stocks: Bulk mode (3 workers, aggressive caching)

• 50-100 stocks: Parallel mode (3 workers)

• <50 stocks: Sequential mode

## 4.3 data\_fetchers.py

Handles all external API interactions.

### Key Functions:

**fetch\_nse\_csv\_list():** Fetches from NSE CSV archives (24hr cache)

**get\_index\_performance():** Index performance (5min cache)

**get\_stock\_performance():** Core stock data fetching

**get\_commodities\_prices():** Oil, Gold, Silver, BTC, USD/INR

**fetch\_stocks\_bulk():** Bulk fetching for 100+ stocks

**get\_stock\_list():** Main function with fallback

### Stock Performance Logic:

1. Retry logic: 3 attempts with exponential backoff (3s, 9s, 27s)

2. Fetch 4 months of historical data

3. Get semi-live current price from info dict

4. Calculate historical prices:

* 1 Week: 5 trading days back
* 1 Month: 30 calendar days back
* 2 Months: 60 days back
* 3 Months: 90 days back

5. Calculate percentage changes

6. Format and return result

### NSE CSV Strategy:

To avoid API 421 errors:

1. Set browser-like headers

2. Warm-up: Visit homepage to set cookies

3. Fetch CSV from archives

4. Parse and validate (min 40 stocks)

## 4.4 ui\_components.py

Streamlit UI rendering functions.

### Key Functions:

**render\_header():** Title, time, commodity prices

**render\_market\_indices():** Major and sectoral indices

**render\_live\_ticker():** Animated rolling ticker

**render\_gainer\_loser\_banner():** Top gainer/loser

**render\_top\_bottom\_performers():** Top 3 and bottom 3

**render\_averages():** 1-year index performance

**render\_pagination\_controls():** Pagination UI

### Live Ticker:

1. Fetch data (60s cache)

2. Sort alphabetically

3. Duplicate for infinite scroll

4. CSS animation (120s duration)

5. Hover to pause

## 4.5 utils.py

Helper functions for formatting and processing.

### Key Functions:

**color\_percentage():** Color-coded HTML for percentages

**get\_current\_times():** IST and EDT times

**format\_time\_display():** Format header display

**create\_html\_table():** HTML table with colors

**get\_ticker\_data():** Live ticker data (60s cache)

### Color Logic:

• Positive: Green (#00ff00)

• Negative: Red (#ff4444)

• Zero: White (#ffffff)

## 4.6 cache\_manager.py

Persistent caching with Pickle.

### Features:

• Single-file Pickle cache (25x faster than JSON)

• 6-hour expiry

• Bulk operations

• Automatic validation

### Key Functions:

**save\_to\_cache():** Save single stock

**load\_from\_cache():** Load single stock

**save\_bulk\_cache():** Save multiple stocks

**load\_bulk\_cache():** Load multiple, return cached and missing

**clear\_cache():** Remove cache file

**get\_cache\_stats():** Total, valid, expired counts

## 4.7 file\_manager.py

Manages custom stock lists.

### Key Functions:

**ensure\_saved\_lists\_dir():** Create directory

**save\_list\_to\_csv():** Save list

**load\_list\_from\_csv():** Load list

**delete\_list\_csv():** Delete list

**load\_all\_saved\_lists():** Load all on startup

# 5. Data Flow & Logic

Complete workflow:

1. User opens app → main() initializes

2. Load session state and saved lists

3. Apply CSS styling

4. Render UI (header, ticker, indices)

5. User selects category

6. Fetch stock list (NSE CSV or fallback)

7. User clicks fetch

8. Check cache, load cached, fetch missing

9. Calculate performance metrics

10. Create DataFrame, sort, paginate

11. Display HTML table with colors

12. Show top/bottom performers

# 6. Key Features

* Multi-category support (Nifty 50, Next 50, Total Market)
* Real-time data with ~15min delay
* 1-week to 3-month performance metrics
* Market indices (major and sectoral)
* Live animated ticker (50 stocks)
* Commodity prices (Oil, Gold, Silver, BTC, USD/INR)
* Custom list upload (.txt/.csv)
* Persistent 6-hour cache
* Automatic parallel fetching
* Pagination (10 per page)
* Dark theme with responsive fonts
* Top/bottom performers
* Flexible sorting
* Cache management UI

# 7. Performance Optimizations

## Caching:

• Pickle format (25x faster than JSON)

• Single-file atomic operations

• 6-hour expiry

• Bulk load/save

## Parallel Fetching:

• 3 workers for 50-100 stocks

• 3 workers for 100+ stocks (bulk mode)

• Exponential backoff for rate limits

• Timeout handling (30s)

## Streamlit Caching:

• @st.cache\_data for stock lists (24hr)

• @st.cache\_data for indices (5min)

• @st.cache\_data for ticker (60s)

## UI Optimizations:

• CSS animations (GPU-accelerated)

• Pagination (10 items)

• Lazy loading

• Responsive font sizing

# 8. Usage Guide

## Installation:

1. Clone repository

2. Create virtual environment: python3 -m venv venv

3. Activate: source venv/bin/activate

4. Install: pip install -r requirements.txt

## Running:

streamlit run app.py

## Using the App:

1. Select category (Nifty 50, Next 50, Total Market, Custom, Upload)

2. Choose sorting (3 Months %, 2 Months %, etc.)

3. Enable parallel fetching for speed

4. View results with pagination

5. Check top/bottom performers

6. Manage cache as needed

## Uploading Custom Lists:

1. Prepare .txt or .csv file (one symbol per line)

2. Add .NS for NSE or .BO for BSE

3. Upload via sidebar

4. Select exchange (Auto-detect, NSE, BSE)

5. Name and save list

6. Load anytime from saved lists

# 9. Dependencies

* streamlit - Web framework
* pandas - Data manipulation
* yfinance - Stock data
* requests - HTTP requests
* pytz - Timezone handling
* beautifulsoup4 - HTML parsing
* lxml - XML processing
* plotly - Visualizations
* curl-cffi - HTTP client