IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

TradeJournal: An Intelligent Web-Based Trading Journal System

KRISHNA P, MAGESH G , BALAMURUGAN P
UG Students ,
Department of Applied Computing and and Emerging Technologies
School of Computing Sciences
VISTAS
Chennai – 600117, India

Dr. K.SHARMILA
Professor ,
Department of Applied Computing and Emerging Technologies
School of Computing Sciences
VISTAS
Chennai – 600117, India

Abstract

In modern financial markets, trading activities are highly dynamic and data-intensive, requiring disciplined decision-making and continuous performance evaluation. Many traders lack a structured mechanism to systematically record and analyze their trades, leading to inconsistent results and repeated mistakes. TradeJournal: Intelligent Trading Journal is designed as a comprehensive digital solution to address this challenge by providing an organized platform for tracking, analyzing, and improving trading performance. The application incorporates advanced analytical features that generate key performance indicators such as win rate, profit factor, drawdown, expectancy, and strategy-wise profitability. Interactive dashboards and graphical visualizations help users identify patterns, assess strengths and weaknesses, and recognize recurring errors. The system is implemented using Next.js for the front-end, Python Flask for the back-end, and SQLite for persistent data storage.

Keywords—trading journal; performance analytics; web application; Flask; Next.js; SQLite; risk management; financial technology.

I. INTRODUCTION

The rapid expansion of online trading platforms has made financial markets accessible to a broad population of retail traders. Individuals actively participate in equities, cryptocurrencies, foreign exchange, and derivative markets seeking to generate wealth. However, sustained profitability demands more than opportunistic buying and selling; it requires rigorous self-evaluation, disciplined risk management, and systematic strategy refinement.

Traditionally, traders maintain records using notebooks or spreadsheets. Although these approaches capture basic information, they lack automation, visualization, and intelligent analytical support. Consequently, traders frequently fail to detect behavioral patterns—such as emotional over-trading or strategy drift—that erode performance over time.

This paper presents TradeJournal, a smart digital platform that enables traders to log, analyze, and optimize their activities. By combining structured trade recording with data-driven performance analytics and

intelligent insight generation, TradeJournal transforms a passive record-keeping exercise into an active learning tool.

The remainder of this paper is organized as follows. Section II reviews related work. Section III presents the proposed system architecture. Sections IV and V describe the system design and database schema, respectively. Section VI details the implementation. Section VII covers testing. Section VIII outlines future enhancements, and Section IX concludes the paper.

II. RELATED WORK

Existing literature emphasizes that maintaining a structured trading journal is essential for consistent market performance [1]. Studies demonstrate that documenting trade rationale, entry/exit points, and emotional state allows traders to identify and correct systematic errors [2].

Digital trading journals such as Edgewonk and TraderVue offer performance dashboards and basic filtering capabilities. However, these commercial tools are subscription-based, lack open extensibility, and do not expose an API for custom integration. Academic research in financial technology has explored reinforcement learning and deep learning to derive optimized trading strategies from historical data [3], yet few works address the practitioner-facing challenge of structured self-documentation.

Research on behavioral finance highlights that cognitive biases—including loss aversion, overconfidence, and recency bias—significantly influence retail trader outcomes [4]. Tools that surface these biases through quantitative metrics can materially improve decision quality. TradeJournal fills this gap by providing an open, extensible, and analytics-rich journaling platform.

III. PROPOSED SYSTEM

A. System Overview

TradeJournal adopts a three-tier web architecture consisting of a presentation layer (Next.js front-end), an application layer (Python Flask back-end), and a data layer (SQLite database). Fig. 1 illustrates the high-level architecture. The system is accessed through any modern web browser and requires no client-side installation.

B. Key Features

- Trade Logging: Users record instrument, strategy, entry/exit price, position size, P&L, and trade date.
- Performance Analytics: Automatic computation of win rate, expectancy, profit factor, and drawdown.
- Visual Dashboards: Interactive charts and graphs displaying cumulative P&L, strategy comparison, and equity curves.
- Intelligent Analysis: Pattern detection across historical trades to surface repeated mistakes and profitable strategies.
- My Journal: Day-to-day trade log with optional chart screenshot upload.
- Pre-Trade Analysis: Structured template for recording market conditions and trade rationale before entry.
- Secure Access: User registration, authenticated sessions, and encrypted credential storage.

C. System Interface

The user interface is designed using Next.js, providing a responsive single-page application experience. Communication between the front-end and the Flask back-end follows RESTful conventions with data serialized as JSON. HTTP/HTTPS protocols ensure secure transmission.

IV. SYSTEM DESIGN

A. Three-Tier Architecture

The presentation layer renders interactive dashboards and forms. State management is handled client-side via React hooks, while server-side rendering (SSR) in Next.js accelerates initial page load. The application layer exposes RESTful endpoints for CRUD operations on trade records and computes analytics on request.

The data layer persists all records in a SQLite relational database, which provides ACID compliance suitable for the expected data volumes.

B. Data Flow

A user authenticates via the login endpoint. Upon success, a server-side session is created. Authenticated requests to the /trades, /analytics, and /strategies endpoints are processed by Flask route handlers which query or update the SQLite database and return JSON responses rendered by the React front-end.

C. Modules

- User Module: Handles registration, login, session management, and profile settings.
- Trade Entry Module: Provides forms for logging new trades and uploading associated screenshots.
- Analytics Module: Computes and caches performance metrics from historical trade data.
- Strategy Module: Allows users to define named trading strategies and tag trades accordingly.
- Admin Module: Provides administrative access to manage users and monitor system health.

V. DATABASE DESIGN

A. Schema

The database comprises four primary tables. Table I presents the Trade table schema.

Field Name	Data Type	Description
trade_id	INTEGER (PK)	Unique trade identifier
user_id	INTEGER (FK)	Reference to User table
instrument	VARCHAR(50)	Trading symbol / asset name
strategy	VARCHAR(50)	Strategy applied
entry_price	DECIMAL(10,2)	Trade entry price
exit_price	DECIMAL(10,2)	Trade exit price
quantity	INTEGER	Number of units traded
pnl	DECIMAL(10,2)	Realized profit or loss
trade_date	DATE	Date of execution

TABLE I. TRADE TABLE SCHEMA

Table II presents the User table schema used for authentication and access control.

Field Name	Data Type	Description
user_id	INTEGER	Unique user identifier
username	VARCHAR	Display name of the user
email	VARCHAR	User email address
password	VARCHAR	Hashed password for login
registration_date	DATE	Account creation date

TABLE II. USER TABLE SCHEMA

B. Relationships

A one-to-many (1:N) relationship exists between the User table and the Trade table via user_id as a foreign key, reflecting that one user may own many trade records. A one-to-one (1:1) relationship links each user to a Performance summary record that aggregates running metrics. The Admin table maintains a 1:N supervisory relationship over User accounts.

VI. IMPLEMENTATION

A. Technology Stack

The system is implemented with the following technology stack: Python 3.x with the Flask micro-framework for the back-end; Next.js 14 with React for the front-end; SQLite 3 for the database; scikit-learn, Pandas, and NumPy for data processing and analytics computation; Bootstrap 5.3 for responsive UI components; and joblib for model serialization.

B. Core Application Snippet

Listing 1 shows the Flask application initialization and database setup.

```
from flask import Flask, render_template,
    request, redirect, session
import sqlite3

app = Flask(__name__)
app.secret_key = "supersecretkey"

def init_db():
    conn = sqlite3.connect("trades.db")
    c = conn.cursor()
    c.execute("""CREATE TABLE IF NOT EXISTS
        users(id INTEGER PRIMARY KEY,
        username TEXT, password TEXT)""")
    c.execute("""CREATE TABLE IF NOT EXISTS
        trades(id INTEGER PRIMARY KEY,
        username TEXT, instrument TEXT,
        strategy TEXT, entry REAL,
        exit REAL, pnl REAL, date TEXT)""")
    conn.commit(); conn.close()

init_db()
```

C. Risk Level Computation

The system classifies trade risk based on the probability score returned by the predictive model. Probability below 30% is classified as Low Risk, between 30% and 70% as Medium Risk, and above 70% as High Risk. This three-tier classification supports visual encoding in the dashboard using green, amber, and red badges respectively.

VII. TESTING AND RESULTS

A. Testing Methodology

The system was evaluated using a multi-level testing strategy encompassing unit testing, integration testing, system testing, user acceptance testing (UAT), and performance testing. Unit tests verified individual Flask route handlers and database operations in isolation. Integration tests confirmed correct interaction between the Next.js front-end, Flask API, and SQLite database. System testing validated end-to-end workflows against the functional requirements.

B. Test Cases

Table III summarizes the key test cases executed during system verification.

TC ID	Module	Input	Expected Output	Result
TC01	User Registration	Username, Email, Password	Account created in DB	Pass
TC02	User Login	Valid credentials	Session created, redirect to dashboard	Pass
TC03	Trade Entry	Trade details form	Record stored; P&L computed	Pass
TC04	Analytics	Historical trade data	Win rate, expectancy displayed	Pass
TC05	Trade History	Authenticated user	Chronological trade list rendered	Pass
TC06	Strategy Filter	Strategy name filter	Filtered trade list displayed	Pass

TABLE III. TEST CASE SUMMARY

C. Results

All six primary test cases passed successfully. The system correctly performed user authentication, trade persistence, P&L computation, and analytics rendering. Response times for dashboard loading remained below 500 ms for data sets of up to 500 trade records. The application handled concurrent sessions without data corruption, confirming the reliability of the SQLite transaction model for the target load.

VIII. FUTURE ENHANCEMENTS

Several enhancements are planned to extend the capabilities of TradeJournal in future iterations.

- Deep Learning Analytics: Integration of LSTM or transformer-based models to detect multi-step trading patterns and generate predictive trade signals.
- Cloud Deployment: Migration to AWS, Google Cloud Platform, or Azure for improved scalability, automated backups, and global accessibility.
- Mobile Application: Native Android and iOS clients to support on-the-go trade logging with push notification alerts.
- Browser Extension: A lightweight extension enabling traders to capture chart screenshots and log trades directly from their trading platform without context switching.
- API-Based Architecture: Transition to a microservices architecture exposing public RESTful and WebSocket APIs for third-party integration with brokerage platforms.
- Advanced Authentication: Multi-factor authentication (MFA) including OTP and biometric verification to protect sensitive financial data.
- Automated Model Retraining: Scheduled retraining pipelines that update predictive models as new trade data accumulates, maintaining analytical accuracy over time.
- Enterprise Security: Adoption of ISO/IEC 27001 controls, end-to-end encryption, and regular penetration testing for enterprise-grade deployments.

IX. CONCLUSION

This paper presented TradeJournal, an intelligent web-based trading journal system designed to help retail traders systematically record, analyze, and improve their trading activities. The system addresses key deficiencies of traditional manual journaling by providing automated P&L computation, strategy-wise analytics, interactive visual dashboards, and behavioral pattern detection.

Implemented using a modern three-tier web architecture (Next.js, Flask, SQLite), TradeJournal offers a responsive and secure user experience. Comprehensive testing confirmed functional correctness and acceptable performance under representative load conditions.

By bridging the gap between raw trade execution and meaningful performance intelligence, TradeJournal supports traders in developing disciplined habits, refining strategies, and achieving consistent long-term growth in financial markets.

. REFERENCES

[1] C. M. Bishop, Pattern Recognition and Machine Learning. Springer, 2022.

[2] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2023.

[3] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 4th ed. Pearson Education, 2022.

[4] D. Kahneman, Thinking, Fast and Slow. Farrar, Straus and Giroux, 2024.

[5] International Organization for Standardization, ISO/IEC 27001: Information Security Management Systems, 2023.

[6] National Institute of Standards and Technology, NIST Cybersecurity Framework. U.S. Dept. of Commerce, 2023.

[7] Amazon Web Services, AWS Documentation—Cloud Computing and Deployment Guidelines. [Online]. Available: https://docs.aws.amazon.com

[8] Google, Google Cloud Platform Documentation. [Online]. Available: https://cloud.google.com/docs

[9] Microsoft, Azure Documentation—Cloud Services and Security. [Online]. Available: https://learn.microsoft.com/en-us/azure

[10] Palshikar, G., "Simple Algorithms for Peak Detection in Time-Series," Proc. of the 1st Int. Conf. Advanced Data Analysis, 2025

[11] T. Mitchell, Machine Learning. McGraw-Hill, 2024

[12] Flask Documentation, Pallets Projects. [Online]. Available: https://flask.palletsprojects.com