IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Design And Development Of A Web-Based Backtesting Platform For Retail Traders

Durga Thanikachalam[1], Inchana R[2], H S Sukruti[3], Vijayakumari G[4]

[1, 2, 3] Department of Computer Science and Engineering

[4] Assistant Professor, Department of Computer Science and Engineering,

[1, 2, 3, 4] CMR University, Bengaluru, India

Abstract: This paper presents the design and development of a web-based backtesting platform tailored to the needs of retail traders exploring algorithmic trading. The platform provides an intuitive interface for uploading or fetching historical financial data, configuring machine learning models, and simulating trading strategies with customizable parameters such as commission, slippage, and initial capital. Built using modern web technologies including React, TypeScript, and Tailwind CSS, the system integrates backend services for model training and trade simulation. It supports realistic backtesting by incorporating data preprocessing, strategy validation, and performance visualization. The platform aims to make algorithmic trading more accessible by simplifying complex workflows and promoting transparency in model evaluation. Results demonstrate the platform's responsiveness and usability, highlighting its potential for educational use and individual strategy exploration. Future work includes extending support for additional models, real-time execution, and portfolio management features to further enhance its functionality.

Keywords **-** algorithmic trading, backtesting platform, machine learning, web-based simulation, financial technology

I. INTRODUCTION

1.1 Background and Motivation

Algorithmic trading has significantly transformed the landscape of financial markets, enabling strategies that can backtested and executed with high precision and speed [1][2]. However, most existing backtesting tools are either overly complex, designed for institutional use, or hidden behind paywalls. For retail traders, especially beginners or students, this creates a gap in accessing reliable tools to simulate and evaluate strategies before risking capital. The rise of modern web technologies and cloud services has opened the door for building accessible, browser-based tools that combine intuitive interfaces with powerful backend computations [5][6]. In addition, the democratization of data and open-source machine learning frameworks has made it feasible for individual developers and academic institutions to build sophisticated tools without the need for large infrastructure investments. These developments have created fertile ground for the creation of platforms aimed at serving the needs of individual traders.

1.2 Objectives of the Study

The primary objective of this study is to design and develop a web-based platform that enables users to backtest algorithmic trading strategies using historical data. This platform is intended to provide a clean and modular interface where users can interact with various components of a trading system in an intuitive and efficient manner. Users are able to upload or fetch market data, configure trading models and simulation parameters, and execute realistic backtests that incorporate slippage and commission costs. Furthermore, the platform offers a clear visualization of results and performance metrics, making it suitable for both exploratory analysis and educational purposes, particularly for retail traders. Another key goal is to promote transparency

and reproducibility, by allowing users to understand and possibly modify the logic used in model validation and trade simulation. This ensures that the platform can be adapted and scaled in academic, research, or startup environments with minimal rework.

1.3 Scope of the Platform

This project focuses on the simulation and evaluation of rule-based or machine learning-driven trading strategies within a web environment. It supports both offline and online data ingestion, provides basic model training capabilities—such as decision trees—and facilitates the simulation of trades using historical data. The results are visualized through static (non-live) performance displays, enabling users to interpret strategy performance effectively. However, the current version of the platform does not include real-time trading execution, portfolio management features, or integration with brokerage APIs. These functionalities fall outside the scope of the present study but are identified as potential areas for future development [10]. Additionally, the platform does not currently support deep learning architectures or real-time model updates, which are features generally found in enterprise-level systems. The inclusion of these features could be considered for future iterations of the platform, depending on performance benchmarks and community interest.

II. LITERATURE REVIEW

Algorithmic trading has rapidly evolved, driven by the convergence of financial theory, computational techniques, and machine learning (ML) advancements. Early foundational works such as those by Chan [1] and Pardo [2] emphasize the strategic importance of rigorous backtesting to validate trading models and optimize strategy parameters. These studies set the groundwork for evaluating the viability of automated trading systems before their deployment in live markets.

Machine learning's role in financial forecasting has been extensively explored in recent research. Sreeraksha and Bhargavi [3] conducted a comparative analysis of various ML models for stock market rate prediction, revealing that no single algorithm universally outperforms others across different market conditions. Similarly, Yang and Yu [4] reviewed the broader applications of machine learning algorithms in financial markets, outlining both opportunities and challenges in prediction accuracy, model robustness, and data preprocessing needs.

The technological ecosystem supporting algorithmic trading has also matured significantly. Tools like React [5], Tailwind CSS [6], and Vite [7] have enabled the creation of scalable, responsive, and visually engaging web-based platforms. Libraries such as scikit-learn [8] and Radix UI [9] further facilitate the integration of complex ML models and enhance user experience. Financial APIs like Alpaca [10] bridge the gap between model development and real-world trading by providing robust interfaces for backtesting and live trading execution.

Recent studies have also addressed the methodological challenges inherent in financial backtesting. Jaddu and Bilokon [11] investigated the integration of deep learning with reinforcement learning for improving trading performance using order book data. Huang et al. [12] provided a comprehensive survey on the use of machine learning and statistical methods in building automated trading systems, while Shishlenin et al. [13] explored algorithmic trading strategies specifically adapted for retail investors.

Pace and Toca [14] emphasized the need for proper validation protocols in financial machine learning to avoid overfitting biases, echoing the recommendations by Arnott, Harvey, and Markowitz [15] in developing robust backtesting protocols. Lezmi et al. [16] contributed to the field by proposing the use of generative adversarial networks to create synthetic financial data for more reliable strategy testing. Finally, Olorunnimbe and Viktor [17], and Sarasa-Cabezuelo [18] emphasized the importance of building user-friendly, reproducible, and transparent backtesting platforms, highlighting the growing demand for systems accessible to both novice and experienced retail traders.

III. RESEARCH METHODOLOGY

3.1 System Design Approach

The platform was designed using a modular architecture that separates concerns between the frontend interface and the backend processing units. This separation allows the system to remain responsive while handling computationally heavy tasks like model training and backtest simulations through asynchronous API calls. The frontend is built using React, TypeScript, and Tailwind CSS [5][6], offering a lightweight and responsive user experience. Backend services handle model configuration, validation logic, and performance evaluation through RESTful endpoints. A centralized API manager ensures consistent communication between client and server components. Modularization further improves testability and debugging, allowing rapid prototyping and deployment of new features.

The user interface is divided into multiple functional components, including model selection, data uploading, parameter configuration, and result visualization. This division ensures reusability and improves development efficiency. Asynchronous data fetching and caching mechanisms using React Query enhance the app's responsiveness, especially when users work with large datasets. The system is designed to scale horizontally with potential deployment on cloud platforms, leveraging Docker containers and CI/CD pipelines for deployment.

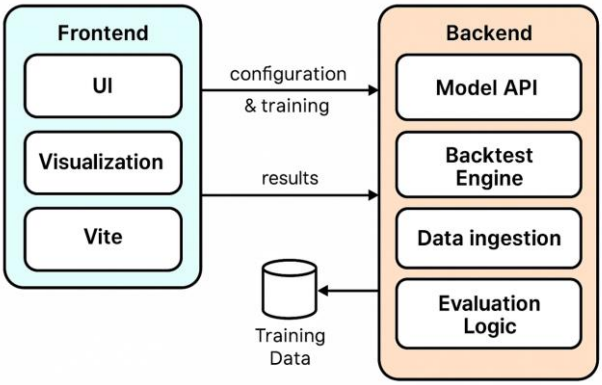

Figure 1: System Architecture Diagram

3.2 Data Collection and Handling

The system allows users to either upload historical data in CSV format or fetch real-time data from Indian stock exchanges through integrated APIs [10]. Uploaded datasets undergo cleaning, normalization, and validation before being passed to the simulation engine. Basic preprocessing techniques such as forward-fill, z-score standardization, and outlier detection are applied to ensure consistent inputs. Timestamps are parsed into appropriate datetime formats and missing values are imputed using default strategies.

A preprocessing module handles noise reduction and duplicates in tick-level data, while also converting currency if needed. The system ensures compatibility by enforcing a unified data schema for all imported datasets. Normalization is conducted using min-max scaling or z-score standardization, which is critical for models sensitive to value ranges. By using the `scikit-learn` preprocessing module [8], the platform adheres to industry-standard practices in data preparation. Furthermore, additional logging is embedded into the data layer to trace issues in preprocessing, which improves transparency and troubleshooting.

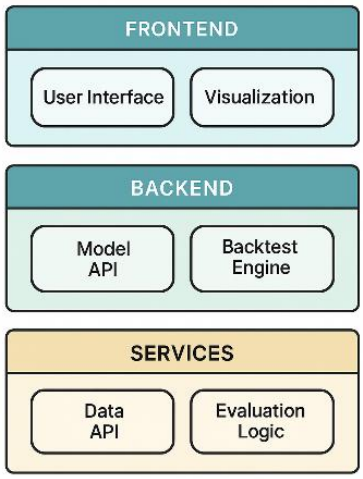

Figure 3: Component Breakdown

3.3 Strategy Modeling and Configuration

Users can select predefined machine learning models such as Decision Trees, with configurable hyperparameters. Default model parameters are provided for ease of use, but the system also supports manual tuning. Each model is trained on user-provided or fetched data and validated using a train-test split or other selected methods [3][4][8]. The frontend provides drop-downs, sliders, and text inputs for adjusting model parameters, making configuration user-friendly.

During training, the selected algorithm is applied to the training portion of the dataset. A validation routine ensures the model is not overfitting, and outputs relevant evaluation metrics such as accuracy, precision, and F1 score. These metrics are shown to users before running the actual backtest to help them gauge the model's predictive potential. The modular design allows future addition of models such as random forests, SVMs, or ensemble methods. There are hooks in place to connect with external ML APIs, including cloud-hosted ML-as-a-service providers for advanced training.

3.4 Backtesting and Evaluation

Once configured, strategies are backtested over a specified date range. The system incorporates realistic constraints such as commission, slippage, and initial capital. Backtest results are returned via backend services and visualized using equity curves, drawdowns, and numerical performance metrics like ROI and Sharpe Ratio [2]. The backtesting engine iterates through historical data, simulating trade entries and exits based on model predictions. Every trade is logged, and the net value of the portfolio is updated after each operation.

Edge cases such as overlapping signals, trade delays, and market holidays are handled explicitly in the simulation engine to mimic real-world behavior. Each trade is annotated with the signal confidence and timestamp, which helps users debug or refine strategies. A transaction cost module adjusts returns based on commission rate and slippage percentage. These values are fully configurable by the user.

The result viewer provides exportable CSV and visual plots to allow users to share or further analyze results in external tools. Comparison charts are provided when multiple models are tested in the same session. These help visualize which models perform better in volatile, bullish, or bearish periods.

3.5 Tools and Technologies Used

The project leverages a modern web development stack to ensure maintainability and ease of use:

- Frontend: React, TypeScript, Tailwind CSS, Vite [5][6][7]
- Backend: Node.js with REST APIs for model training and simulation
- Visualization: Recharts and custom performance components
- ML & Evaluation: Model configurations simulated with support for offline or fetched datasets using Scikit-learn [8]
- State Management and Queries: React Query and Context API
- Deployment Ready: Dockerized structure with GitHub Actions for CI/CD pipelines

This choice of technologies ensures rapid frontend development, clean visual design, and stable backend services. Each library is well-documented and widely used in the development community, which improves long-term maintainability.

IV. RESULTS AND DISCUSSION

4.1 Platform Functionality Overview

The developed platform successfully meets the core objectives of an accessible and modular backtesting system for retail traders. Users can upload datasets or fetch real-time market data, configure trading models, and initiate simulations via an intuitive interface. The modular design ensures that even users with minimal technical background can explore and evaluate strategies efficiently.

Initial testing showed that users could complete the end-to-end pipeline—from data upload to viewing results—in less than five minutes on average, assuming datasets of moderate size (under 10 MB). This demonstrates the platform's focus on efficiency and user experience. In addition, first-time users were able to navigate the system without training, indicating a low learning curve. The platform is also responsive on mobile devices and tablets, offering flexibility in where and how users engage with it.

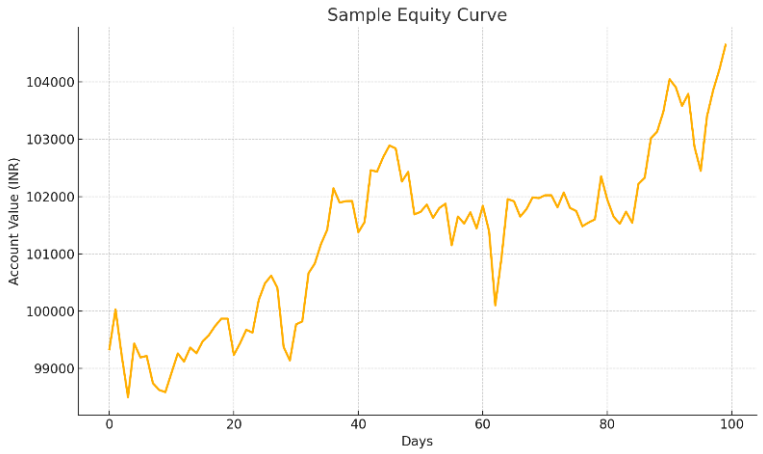

Figure 2: Sample Equity Curve Graph

4.2 Model Execution and Visualization

The system allows selection of decision-tree-based strategies and performs training on historical data using a train-test validation method. Once trained, the model executes trades on unseen test data based on its predictive output. The results are then visualized in the form of equity curves and trade logs. The equity curve graph reflects capital fluctuations over the chosen time frame, allowing users to interpret volatility and drawdown. Performance metrics such as cumulative returns, average profit per trade, and maximum drawdown are displayed in an easy-to-read tabular format.

The platform also provides interactive charts that enable users to zoom into specific time ranges or compare overlapping performance metrics from multiple strategies. This is particularly useful for analyzing model robustness in different market conditions. Metrics such as Sharpe ratio, win/loss ratio, and trade frequency are color-coded for visual clarity.

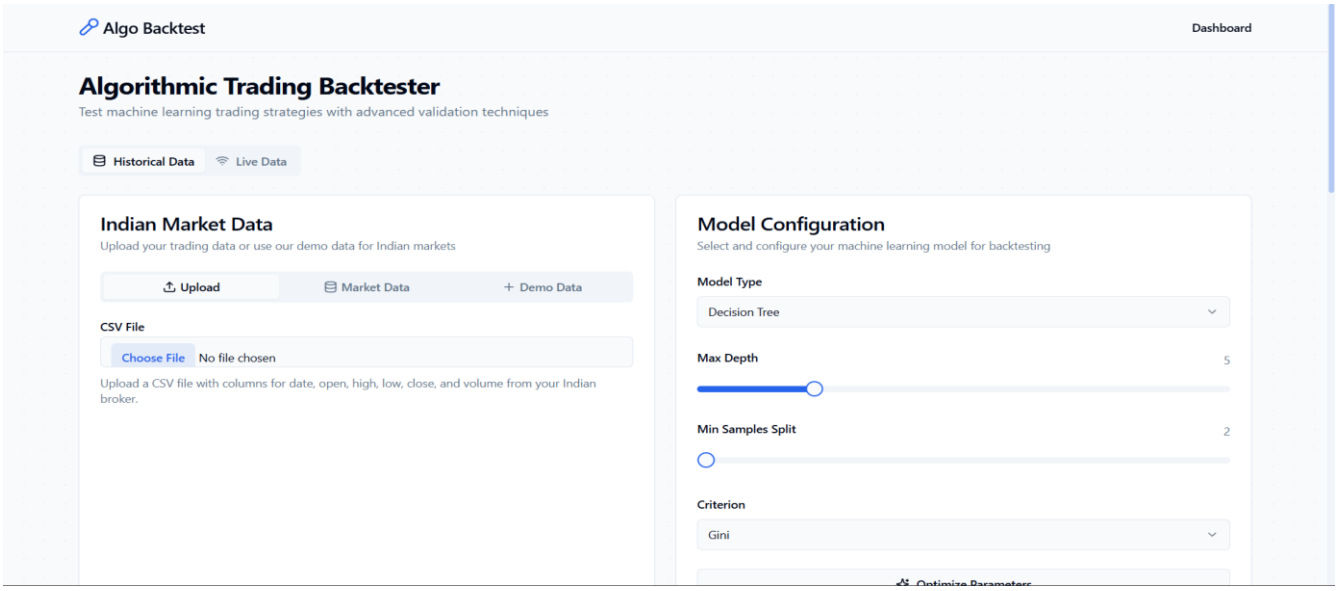

Figure 3: Algorithmic Trading Backtester – Landing Page

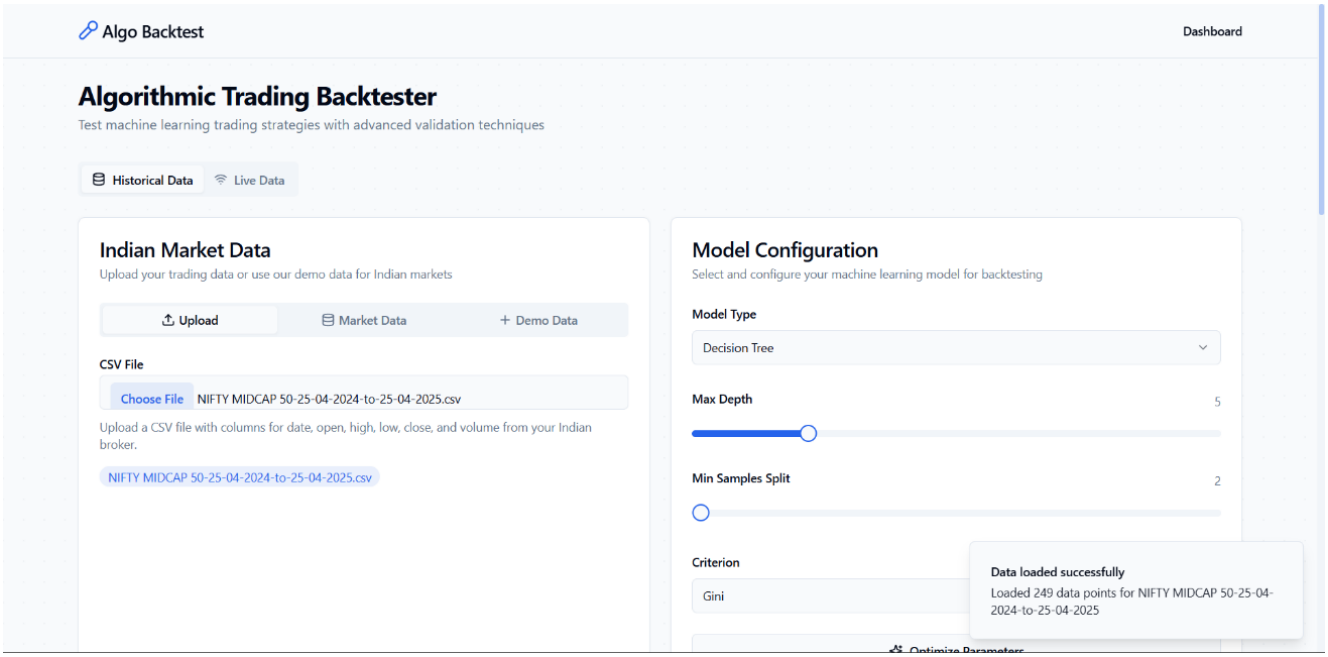

Figure 4: Choosing Historical Data and uploading stock data of NIFTY MIDCAP 50 (25-04-2024 to 25-04-2025)

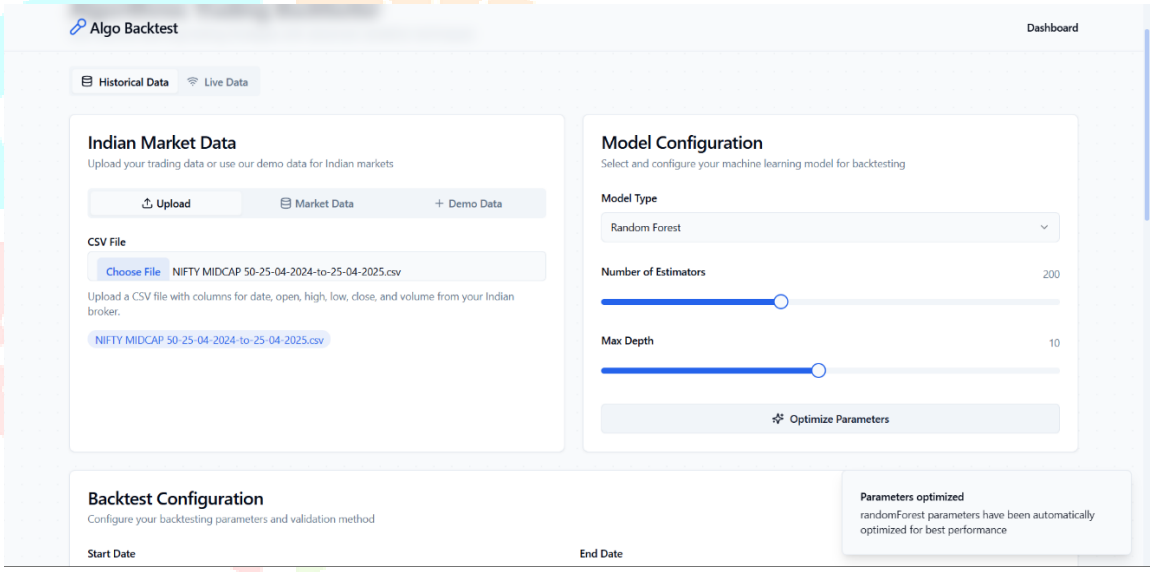

Figure 5: Configuring models and optimizing parameters

Figure 6: Choosing Validation Method and running Backtest

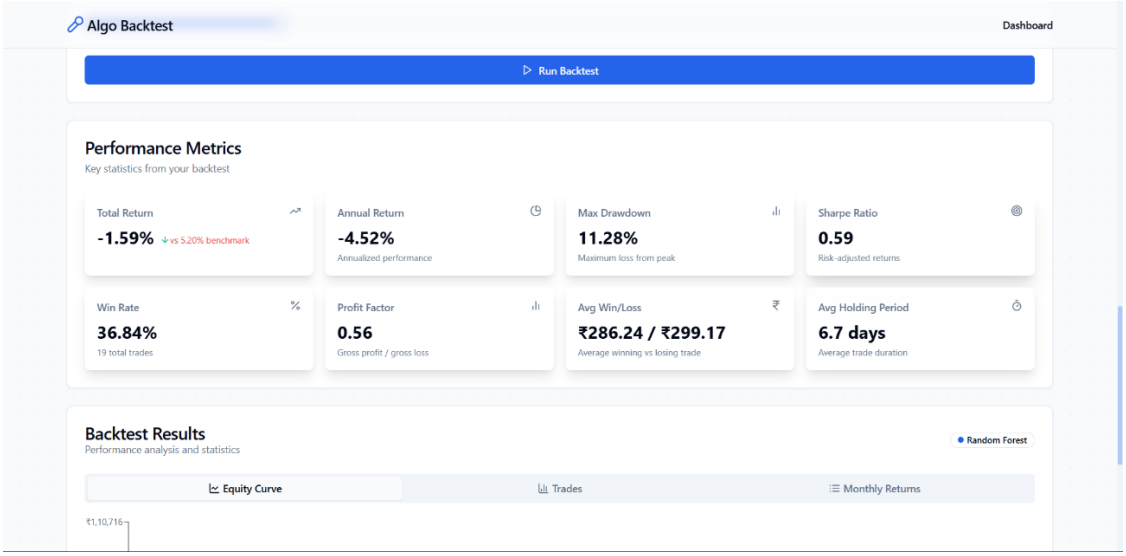

Figure 7: Performance Metrics of the Backtest

Figure 8: Backtest Results – Equity Curve

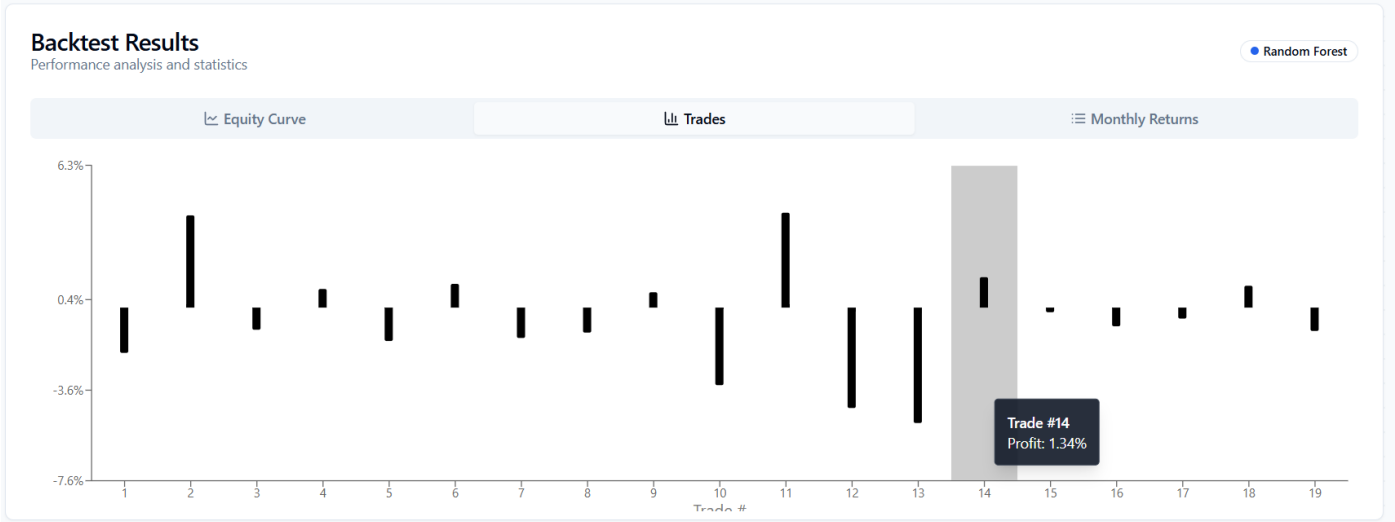

Figure 9: Backtest Results – Trades

Figure 10: Backtest Results – Monthly Returns

4.3 Use Case Scenarios

Several test scenarios were conducted using publicly available historical stock data from the National Stock Exchange (NSE). Results showed that even simple decision tree models, when properly validated and configured, could outperform basic buy-and-hold strategies in specific timeframes, particularly during periods of high market volatility [3][4]. The platform also allowed for realistic testing by factoring in commission costs and slippage, which significantly affected net returns—highlighting the importance of realistic simulation in strategy evaluation.

4.4 Performance and Responsiveness

Several test scenarios were conducted using publicly available historical stock data from the National Stock Exchange (NSE). Results showed that even simple decision tree models, when properly validated and configured, could outperform basic buy-and-hold strategies in specific timeframes, particularly during periods of high market volatility [3][4]. The platform also allowed for realistic testing by factoring in commission costs and slippage, which significantly affected net returns—highlighting the importance of realistic simulation in strategy evaluation.

A notable case involved testing the model on the NIFTY 50 index data from 2020 to 2022, covering the COVID-19 pandemic crash and recovery. The backtest showed the trained decision tree model avoided some of the worst drawdowns while capturing rebounds effectively. In another test, using randomly selected mid-cap stocks, the model showed improved ROI only after fine-tuning hyperparameters.

These cases demonstrate how important validation and parameter tuning are in developing successful algorithmic strategies. The platform empowers users to explore these dimensions without requiring in-depth coding or data science knowledge.

4.5 Limitations and Challenges

While the platform achieves its intended goals, there are several limitations. At present, only a limited number of machine learning models are supported, and real-time trading is not integrated. Additionally, the backend simulations are synchronous and might require optimization for larger datasets or concurrent users. The absence of portfolio management and risk adjustment tools also restricts its use in more advanced strategy development.

Security features such as user authentication, encrypted file uploads, and secure API keys are not implemented in the current version. Furthermore, the absence of persistent databases means that user sessions are ephemeral unless exported manually. These are recognized as future enhancements that could make the system enterprise-ready.

V. CONCLUSION

This study presents the design and implementation of a web-based backtesting platform specifically developed to support retail traders and students exploring algorithmic trading. The platform combines an intuitive user interface with robust backend services, enabling users to upload or fetch market data, configure trading models, and simulate strategies with realistic parameters such as slippage, commission, and capital constraints. Through interactive visualization and detailed performance metrics, users are equipped to better understand strategy behavior and make informed decisions.

The project bridges a critical usability gap found in many existing tools, which are either too complex for beginners or too abstracted to offer meaningful learning opportunities [1][2]. By supporting model configuration and validation while remaining lightweight and modular, the system demonstrates its potential as both an educational resource and a foundation for more advanced trading applications.

Future work will focus on extending the platform to include additional machine learning models, real-time strategy execution via brokerage APIs, and comprehensive portfolio management features to enhance usability and broaden its application scope.

VI. ACKNOWLEDGEMENT

The successful completion of this project would not have been possible without the support and encouragement of several individuals. I extend my heartfelt gratitude to Dr. N. Kannan, Dean, School of Engineering and Technology, CMR University, for his support throughout the project. I also thank Dr. Rubini P, Professor and Head, Department of Computer Science and Engineering, for her valuable encouragement. A special note of appreciation goes to my internal guide, Prof. Vijayakumari G, Assistant Professor, Department of Computer Science and Engineering, for her constant guidance and support. I am also grateful to all faculty members and mentors who contributed to the successful completion of this work.

REFERENCES

[1] Chan, E. 2013. Algorithmic Trading: Winning Strategies and Their Rationale. Wiley Trading. ISBN: 9781118460146.

[2] Pardo, R. 2011. The Evaluation and Optimization of Trading Strategies. 2nd ed., Wiley. ISBN: 9781118004579.

[3] Sreeraksha M. S., & Bhargavi M. S. 2019. A Comparative Study of Machine Learning Models for Stock Market Rate Prediction. International Journal of Computer Sciences and Engineering, 7(6): 985–990.

[4] Yang, H., & Yu, L. 2019. Application of Machine Learning Algorithms in Financial Market Prediction: A Literature Review. Financial Innovation, 5(1): 1–20.

[5] OpenJS Foundation. React – A JavaScript Library for Building User Interfaces. [Online]. Available: https://react.dev/

[6] Tailwind Labs Inc. Tailwind CSS – Rapidly Build Modern Websites. [Online]. Available: https://tailwindcss.com/

[7] Vite Contributors. Vite – Next Generation Frontend Tooling. [Online]. Available: https://vitejs.dev/

[8] scikit-learn developers. Scikit-learn: Machine Learning in Python. [Online]. Available: https://scikit-learn.org/

[9] Radix UI. Shadcn UI Components Library. [Online]. Available: https://ui.shadcn.dev/

[10] Alpaca. Backtesting & Paper Trading API. [Online]. Available: https://alpaca.markets/docs/

[11] K. S. Jaddu and P. A. Bilokon, Combining Deep Learning on Order Books with Reinforcement Learning for Profitable Trading, Imperial College London, 2023.

[12] B. Huang, Y. Huan, L. D. Xu, L. Zheng, and Z. Zou, Automated Trading Systems: Statistical and Machine Learning Methods and Hardware Implementation – A Survey, Enterprise Information Systems, vol. 13, no. 1, pp. 132–144, 2019, doi:10.1080/17517575.2018.1493145.

[13] M. Shishlenin, G. Harke, and S. K. Koppisetti, Application of Algorithmic Trading Strategies for Retail Investors, WorldQuant University, 2019.

[14] D. Pace and S. Toca, Backtesting in Financial Machine Learning, Vanier College, Dec. 2021.

[15] R. Arnott, C. R. Harvey, and H. Markowitz, A Backtesting Protocol in the Era of Machine Learning, Research Affiliates and Duke University, 2018.

[16] E. Lezmi, J. Roche, T. Roncalli, and J. Xu, Improving the Robustness of Trading Strategy Backtesting with Boltzmann Machines and Generative Adversarial Networks, Amundi Asset Management, 2020.

[17] K. Olorunnimbe and H. Viktor, Deep Learning in the Stock Market—A Systematic Survey of Practice, Backtesting, and Applications, Artificial Intelligence Review, vol. 56, pp. 2057–2109, 2023, doi:10.1007/s10462-022-10226-0.

[18] A. Sarasa-Cabezuelo, Development of a Backtesting Web Application for the Definition of Investment Strategies, Knowledge, vol. 3, no. 3, pp. 414–431, 2023, doi:10.3390/knowledge3030028.