# Infosys Internship 5.0

**Title: NSE Automated Report Downloader**

**Developer: [MARUKURTHI YAGNA PRIYA]**

**Introduction**

The Automated NSE Report Downloader is a Python-based solution designed to streamline the process of fetching daily stock reports from the National Stock Exchange (NSE). This tool addresses challenges like manual effort, data redundancy, and potential errors by automating report retrieval, validation, and organization. With features like error handling, retry mechanisms, and duplicate detection, it ensures that data is consistently accurate and readily available.

The project incorporates a user-friendly interface for managing settings and viewing logs, making it suitable for both individual traders and financial institutions. By eliminating the need for manual intervention and reducing the risk of errors, this tool empowers users to focus on analyzing market trends and making informed decisions.

**Objective**

* Automate the downloading, validation, and logging of NSE daily reports to ensure efficiency and accuracy.

**Significance**

* Eliminates manual intervention in downloading reports.
* Reduces errors caused by human oversight.
* Ensures timely updates for better decision-making and reporting.

**Project Scope**

Included:

* Automated downloading of NSE reports at scheduled intervals.
* Validation of downloaded files to ensure integrity.
* Retry mechanism for failed downloads.
* User-friendly log viewing and search functionality.

Excluded:

* Analysis or visualization of the downloaded data.
* Integration with external APIs for data processing or reporting.

Constraints:

* Dependency on the NSE website's availability and structure.
* File formats limited to those provided by the NSE (e.g., CSV, DAT).

**Requirements**

Functional Requirements:

* Schedule daily downloads at a specific time.
* Validate file existence to avoid duplicates.
* Retry downloads in case of failures (up to a maximum retry count).
* Maintain structured logs with timestamps, actions, and statuses.

Non-Functional Requirements:

* Reliability ensured through robust error handling and retries.
* Timely execution of scheduled tasks.
* Intuitive web interface for user interaction.

**Technical Stack**

* Programming Language: Python
* Frameworks/Libraries:
  + Selenium: For web scraping and interaction with the NSE website.
  + Flask: For building the user interface.
  + schedule: For task automation.
* Tools: ChromeDriver, Python logging module.

**Architecture/Design**

**System Architecture:**

The project follows a modular design, comprising:

* Downloader Module: Handles report fetching and validation.
* Scheduler Module: Executes downloads at scheduled times and manages retries.
* Logger Module: Captures all actions and errors in a structured log file.
* Web Interface: Displays logs and allows configuration of settings.

**Key Design Decisions:**

* Retry Mechanism: Ensures reports are downloaded despite intermittent network issues.
* Duplicate Handling: Prevents overwriting of already downloaded reports.
* Scheduled Execution: Automates recurring tasks without manual triggers.



Fig: 1(Block Diagram)

**Development**

* Technologies Used: Python, Flask, Selenium.
* Challenges Encountered:
* Handling incomplete or interrupted downloads.
* Avoiding duplicate downloads through file validation.
* Managing web scraping dependencies like ChromeDriver.
* Best Practices Followed:
* Modular and reusable code structure.
* Comprehensive error handling and detailed logging.
* Input validation for user configurations.

**Testing**

* Approach:
* Unit Testing: Tested individual functions like `fetch\_nse\_report` and `is\_zip\_already\_downloaded`.
* Integration Testing: Verified seamless interaction between modules (scheduler, downloader, logger).
  + System Testing: Simulated end-to-end scenarios, including network failures and retries.
* Results:
* Successfully handled edge cases like duplicate files and failed retries.
* Verified logging accuracy for all actions and errors.

**Deployment**

* **Environment Setup:**
* Install dependencies with `pip install -r requirements.txt`.
* Ensure ChromeDriver is installed and accessible.
* **Steps:** 
  + Run the application using `python app.py`.
  + Access the web interface for configurations and log monitoring.
* **Automation:** 
  + Scheduling tasks using the `schedule` library for daily execution.

**User Guide**

1. Open the application in a web browser.

2. Configure settings such as date range and file format.

3. Monitor activity logs for download status and troubleshoot issues.

**Troubleshooting Tips:**

* Update ChromeDriver if web scraping fails due to browser version mismatches.
* Check logs for specific error messages if downloads fail.

**Conclusion**

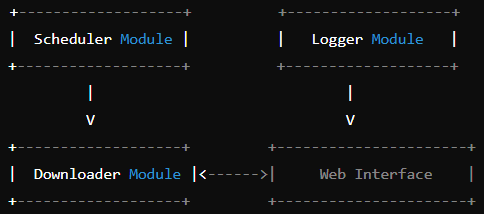
The “NSE Automated Report Downloader” meets its objective of automating the report-fetching process effectively. Its robust scheduling, error handling, and user interface make it a reliable tool for regular use.

**Future Improvements:**

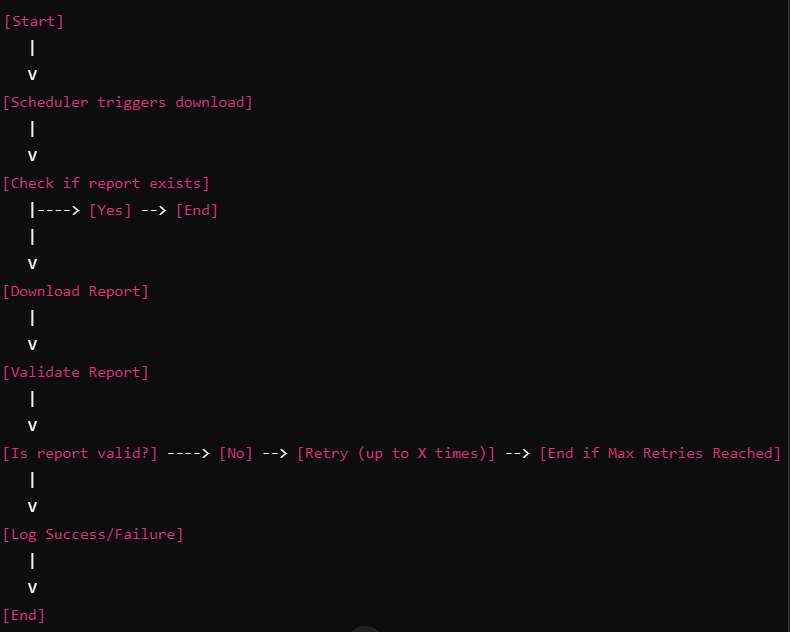
* Incorporate data analysis and visualization features.
* Implement cloud storage for secure access to reports.

**Appendices**

* **Diagrams**:
  + **System Architecture Diagram**: This diagram illustrates the modular design of the application. It highlights the major components of the system, such as the Downloader, Scheduler, Logger, and Web Interface modules, and shows how they interact with each other.



* + **Download Process Flowchart**: This flowchart shows the sequence of operations involved in downloading a report. It covers checking if a report already exists, handling failures with retries, and logging actions.



* **References**:
  1. **NSE Documentation**
     1. **URL**: [NSE India Official Website](https://www.nseindia.com)
     2. Provides documentation on accessing and downloading stock market reports and other data. The specific structure of URLs and parameters used for report downloads can be found in the site's documentation.
  2. **Selenium Documentation**
     1. **URL**: https://www.selenium.dev/documentation/en/
     2. Selenium is used for automating web interactions, such as fetching reports from the NSE website. This documentation covers the use of the Selenium WebDriver for browser automation, including the installation and setup process.
  3. **Flask Documentation**
     1. **URL**: https://flask.palletsprojects.com/en/2.2.x/
     2. Flask is used to create the web interface for the project. This documentation provides guidance on building web applications, handling HTTP requests, and creating dynamic content in Flask.
  4. **schedule Library Documentation**
     1. **URL**: [Schedule Python Library](https://schedule.readthedocs.io/en/stable/)
     2. The schedule library is used to automate the execution of periodic tasks, such as downloading the NSE reports at specified intervals.
  5. **Python Logging Documentation**
     1. **URL**: [Python Logging](https://docs.python.org/3/library/logging.html)
     2. Python’s built-in logging library is used to capture activities, actions, and errors throughout the project. This documentation provides details on how to configure and use logging in Python for debugging and tracking system actions.
  6. **ChromeDriver Documentation**
     1. **URL**: https://sites.google.com/a/chromium.org/chromedriver/
     2. ChromeDriver is used with Selenium to control the Chrome browser. This reference provides installation and usage instructions, including compatibility details with different versions of Chrome.
  7. **Retry Decorator (Custom Code)**
     1. **URL**: [Python functools.wraps](https://docs.python.org/3/library/functools.html#functools.wraps)
     2. This reference explains how to use the functools.wraps decorator for creating retry logic in Python. This feature ensures that failed operations can be retried, reducing the impact of intermittent network issues.