**alamAPI: DEVELOPMENT OF STOCK MARKET PRICE TREND FORECASTING SYSTEM USING DYNAMIC MODE DECOMPOSITION AND**

**ARNAUD LEGOUX MOVING AVERAGE**

John Markton M. Olarte

Division of Physical Sciences and Mathematics, College of Arts and Sciences,

University of the Philippines Visayas, Miagao

CMSC 198.1: Special Problem I

Prof. Nilo C. Araneta

January 2023

**Chapter 3. Materials and Methods (Draft)**

This chapter will discuss the materials and methods that will be used for the development of the proposed system: alamAPI. Specifically, the following will be discussed in this chapter:

1. Development Tools and Software Requirements
2. System Diagrams
3. Hardware Requirements
4. Methodology; and
5. Gannt Chart

**Development Tools and Software Requirements**

The development of the alamAPI will use the following development tools and software requirements:

***Development Tools***

1. Visual Studio (VS) Code – This is a highly functional code editor, which will be used as the main development interface for the project.
2. MongoDB Compass – This is a graphical user interface used for the development and management of different MongoDB databases.
3. GitHub – This will serve as the code repository and version control system (using git) for the project.

***Software Requirements***

1. Python (version 3.11.0) – this will serve as the main programming language for the development of the different components of alamAPI, more specifically the following libraries will be used:

* For the development of the API and Database ODM
* FastAPI (version 0.85.0) – This is a library primarily used for building modern, fast, and high-performing web framework APIs (Tiangolo, n.d.). This will be utilized in the development of the project because of its (1) ease of utilization; (2) fast implementation; (3) high-performance (4) built-in robust API documentation (5) and high scalability.
* mongoengine (version 0.24.2) – This is a library developed as an Object-Document Mapper, which lets Python connect and work with MongoDB (MongoEngine, n.d.). This will be used in the alamAPI to connect the API endpoints to the MongoDB database.
* json (pre-installed) – This is a python library that can transform Python dictionary into json object, and vice versa. This will be used in the development of alamAPI for parsing and conversion of the data from the API and to the MongoDB database through an ODM.
* datetime (pre-installed) – This python library is used for creating a datatime object, which as the name suggests is an object that contains the date and time information. This will be used in the development to keep track with all the processes that is happening in the system through a date and time logs.
* os (pre-installed) – This is a python library that enables the user to do operations in the operating system such as creating directories, files, accessing operating system information, etc. This will be used to access the operating system’s environment variables, and to help in other OS-based functions.
* For the pre-processor (data collector):
* requests (version 2.28.1) – This library allows the user to create web requests to an external or internal servers. This will be used to connect and collect the current EOD market data from the third-party market historical data provider: EODHD.
* EODHD – A third-party market fundamental and historical data APIs provider (EODHD, n.d.)
* For the pre-processor (data-processor):
* pickle (pre-installed) – This allows an object to be saved and reloaded as a variable in Python, as such this will be used to save the machine learning developed and be utilized to process the new and updated data provided by the data collector.

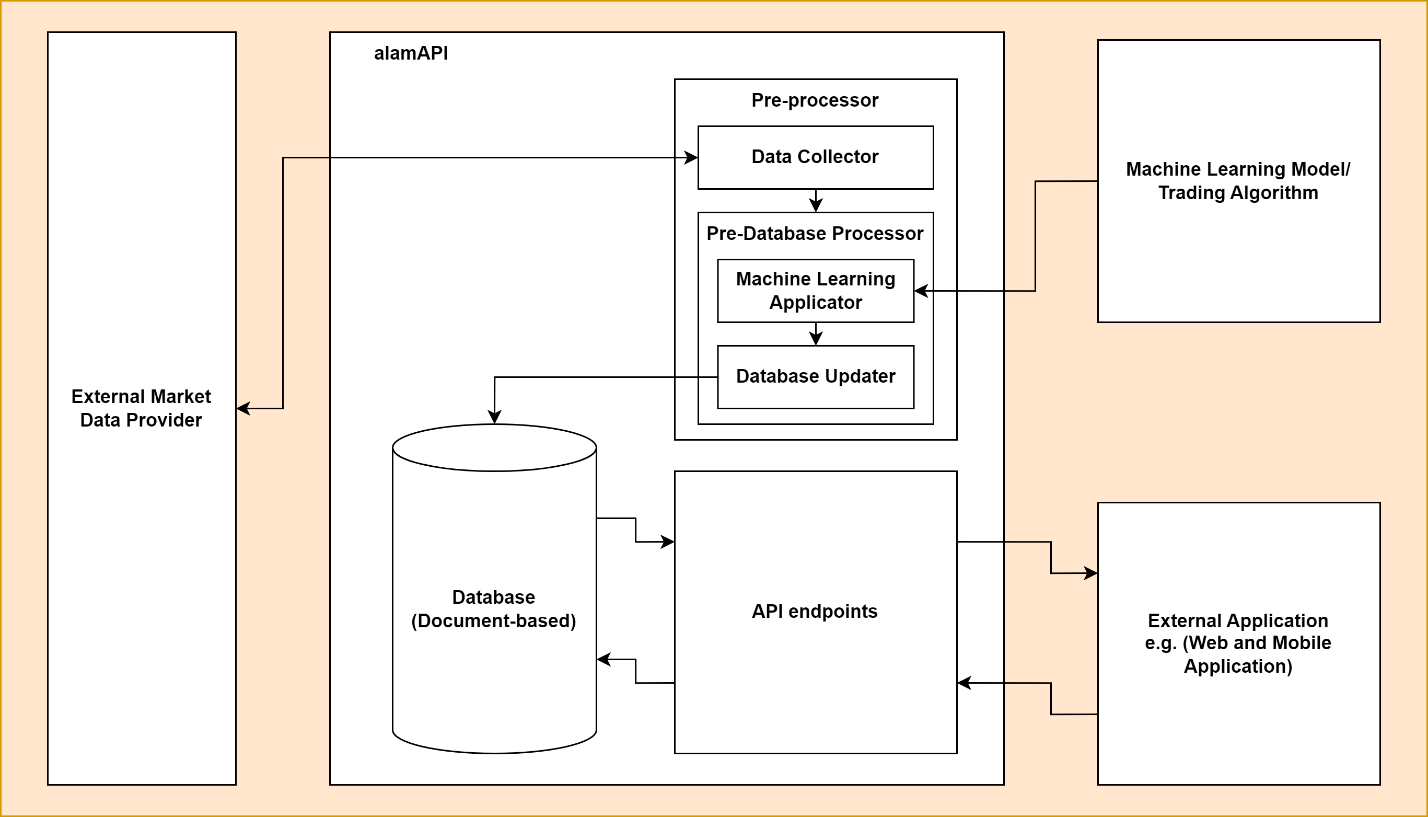
1. MongoDB – This will be used as the non-relational (document-based) database, that will hold the stock information, stocks to buy, and stocks to sell.
2. Jupyter Notebook – This will be used during the training and testing of the machine learning model that will be developed as part of the alamAPI.
3. CRON – A Linux-based scheduler. This will be used in the system to set a schedule for the historical data collection and processing for each market end-of-day (EOD) every 5 PM from Mondays to Fridays. Moreover, the scheduler is part of the pre-processor module of the system.
4. Docker – This is a very useful tool to creating containers, whereas a container contains a code and all its dependencies in one standard unit of software, which can be run in different machines regardless of its difference from the development machine used (Docker, n.d.). As such this will be used to create containers for each of the component of alamAPI, to enable it to run in different deployment machines.
5. Docker-compose – In order to run multiple containers at once, docker-compose will be used. This will be further discussed in the Container Diagram section of this chapter.

**System Diagrams**

In this chapter, the appropriate system diagrams will be shown and discussed. This shall help in the understanding of the system’s features, data flow, and processes. Whereas all the diagrams can be viewed in full resolution, using the GitHub repository, provided in the author’s note at the title page.

***Top-Level Overview Diagram of the alamAPI and Its Interactions to External Systems***

The figure below shows the top-level overview of the alamAPI and its interactions to any third-party or external applications.



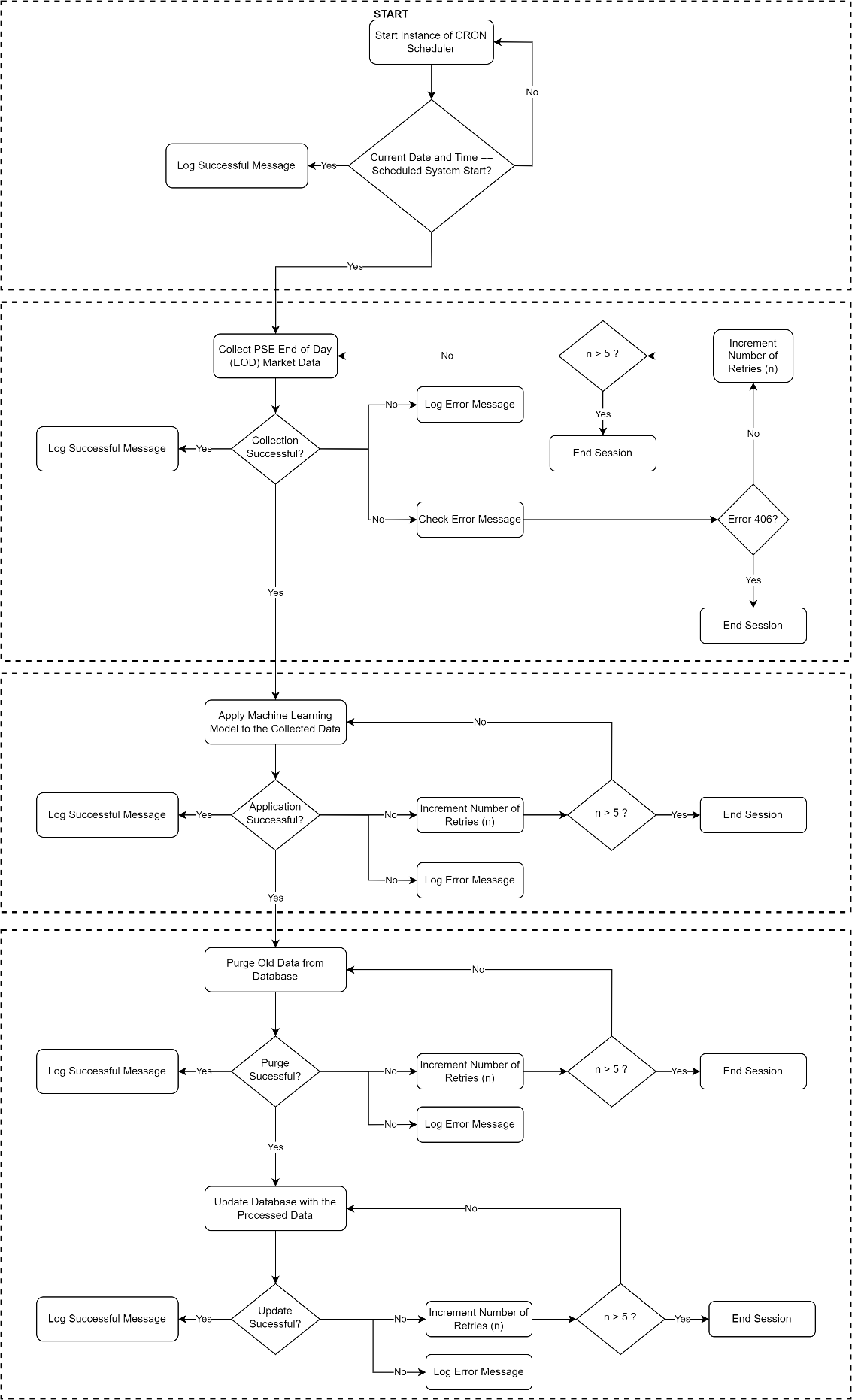
*Figure 1. Top-Level Overview of the alamAPI and Interactions with External Applications/Systems*

As shown from the figure above, the alamAPI is connected to three external entities: (1) External Market Data Provider, which provides the system with the needed historical market data; (2) Machine Learning Model or Trading Algorithm, in the case of this special problem, a machine learning model will be developed and will be utilized by the system, however as previously discussed the system is created to accept any other machine learning model or proprietary trading algorithms that other developers may or want to develop in the future; and (3) External Application, which can be a web-based or mobile-based application.

On the middle of the diagram the alamAPI is observed to have three main components, namely, (1) Pre-processor, which is further divided into sub-components: (a) Data Collector, which collects the data from the external market data provider; (b) Pre-Database Processor, which processes the historical market data collected by applying the developed machine learning model and sending it to the database updater module; (2) Database, which is running on MongoDB a document-based and non-relational database; finally, the database is connected to the (3) API endpoints which processes the request and responses of the system to any external application connected to the API via a network.

***Process Flow Diagram***

Below is the figure of the Process Flow Diagram of the alamAPI. The diagram shows the different processes that the system will undergo once it has been deployed in the server.



*Figure 2. Full Overview of the Process Flow Diagram for the alamAPI*

To better view and understand the flow of the processes, we can divide the discussions per components in the diagram.

***Scheduler*.** Using CRON, a Linux-based scheduler, a scheduled task is provided to the server running the system. Since, the system will be containerized in a Linux System, the scheduler will run once the instance of the Docker Engine is running on the server system, which can be of any operating system. Then, if the current date and time of the contained system matches the scheduled date and time from CRON, it will log that the scheduled task has started, otherwise, it will not do anything and will check again for the current date and time.

The consequent processes in this process flow diagram will run after the scheduled task is initiated. Wherein the schedule task will run everyday from Mondays to Fridays, every 5:00 P.M. And the whole process can be seen in the figure below.

Diagram

Description automatically generated

*Figure 3. Process Flow Diagram of the Scheduler (CRON)*

***Data Collector*.** This is the first task that the scheduled activity will do, which is simply to connect to the historical market data provider and collect the historical market data that is updated for the current date.

Wherein, if the collection is successful, it will log that the system has successfully connected and collected the updated historical market data for that day and will proceed to use the collected data to the Machine Learning Model.

Otherwise, it will log the error, and it will check the error message. Wherein, if the error message shows “Error 406” of “Payment Needed”, then the scheduled task will end in this section. This is also the reason why the end of each process ends in logging the activities of the system, so that the maintainers of the system can easily pin-point the problem to be fixed during the actual deployment of the alamAPI. Moreover, if the error is anything else, then the system will retry to collect the data for a maximum of five tries, and if it still encounters an error during the retry window, the session will also end.

The flow of processes discussed above can be observed in the figure provided below.

Diagram

Description automatically generated

*Figure 4. Process Flow Diagram of the Data Collector*

***Machine Learning Model Application*.** In this process the developed machine learning model/algorithm will use the current historical market data collected to predict the future trend of the stock market and decide whether that stock should be bought or sold for the next market day.

Wherein if the application of the machine learning model is successful, then the system will log the success of the operation and proceeds into updating the database.

Otherwise, it will log the error, and will retry the operation for a maximum of five times. Once after the five retries is unsuccessful, then the system will end the session at this stage.

The flow of processes discussed above can be observed in the figure provided below.

Diagram

Description automatically generated

*Figure 5. Process Flow Diagram of the Machine Learning Model Application*

***Database Updater*.** This process flow will purge the old content of the database, and once successful it will update the database with the new documents created from the previous process.

The flow of processes discussed above can be observed in the figure provided below.

Diagram

Description automatically generated

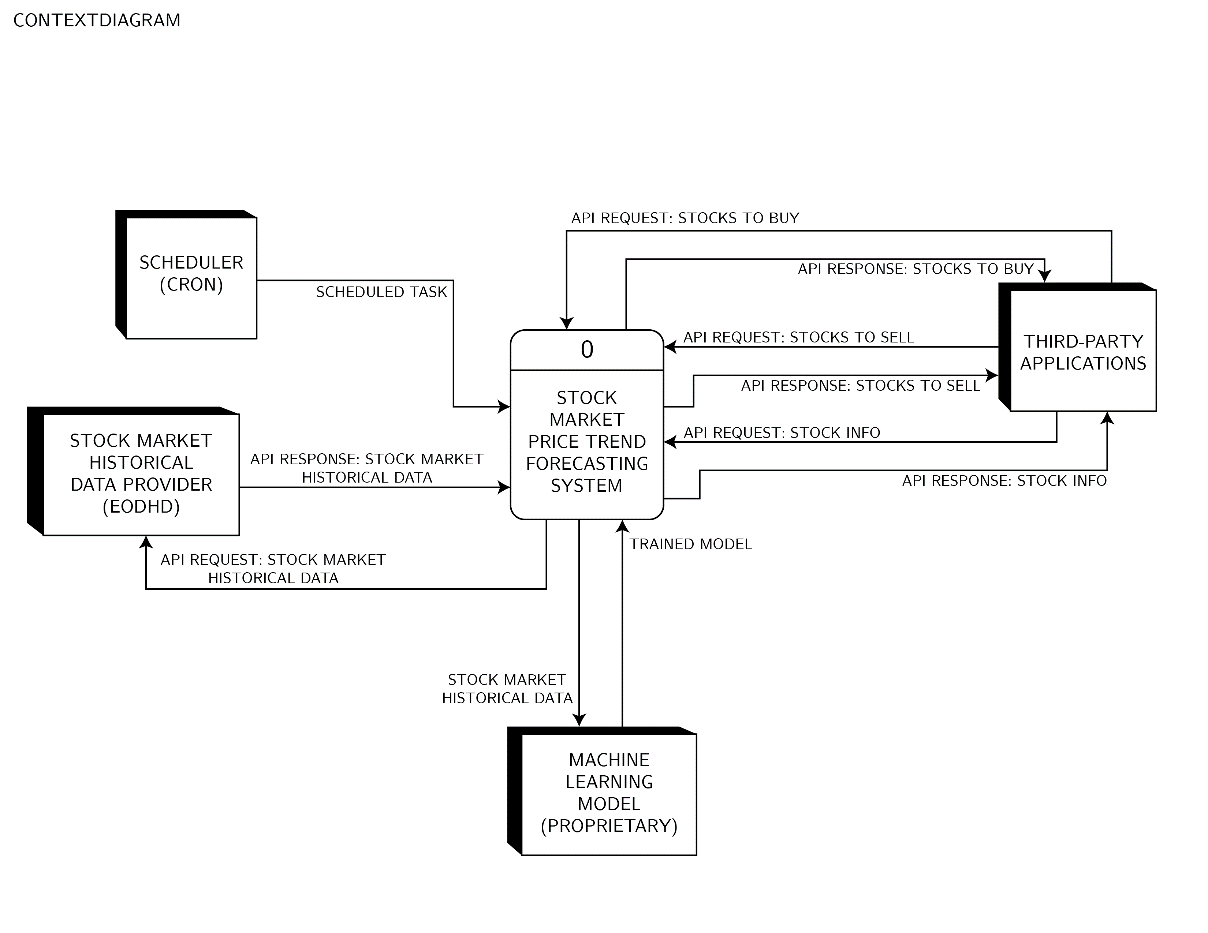
*Figure 6. Process Flow Diagram of the Database Updater*

***Data-Flow Diagram (DFD)***

In this section the DFD of the alamAPI, specifically the Stock Market Price Trend Forecasting System will be presented and discussed. A data-flow diagram will help to better understand how the processes works, and how data flows from one process to another. This is specifically important as it shows the overview of the security of the data by showing how it can be accessed. In the case of alamAPI, the data that can only be accessed publicly is the listed stock to buy and sell, as well as the stock information as provided in its database, and as allowed by the API endpoints.

Moreover, the DFD paradigm used in the diagrams presented in this section follows the Gane-Sarson DFD symbols, which utilizes four basic symbols: (1) Entity / External Entity; (2) Data Flow; (3) Process; and (4) Data Store (Visual Paradigm Online, n.d.)

**Context Diagram.** First, let us start by showing the overview of the whole process by showing the context diagram of the system itself as process 0, which can be seen in the provided figure below.

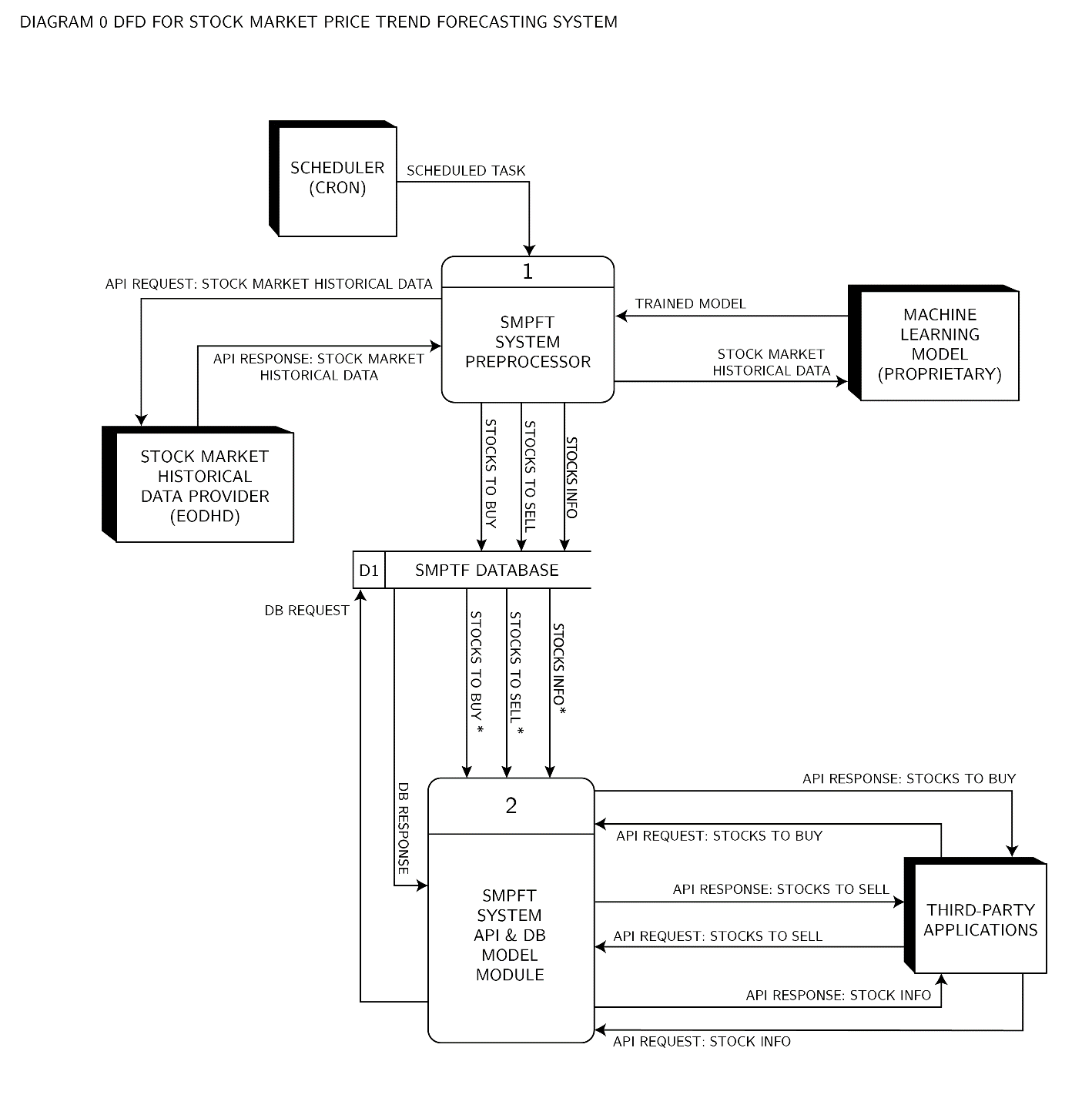


*Figure 7. Context Diagram of the Stock Market Price Trend Forecasting System*

The above figure shows the root process (process 0), which is the underlying system of the alamAPI itself: Stock Market Price Trend Forecasting (SMPTF) System which is connected to four external entities: (1) Scheduler, which will be provided by CRON; (2) Stock Market Historical Data Provider, which will be EODHD; (3) Machine Learning Model, which will be developed along side the development of the system; and (4) Third-Party Application, which will also be developed in the conduct of this special problem as the test application for accessing, testing, and showcase of the features of alamAPI.

Moreover, all the necessary data flow lines can also be observed: (1) Scheduled task, which is the committed task on schedule as indicated in the CRON application; (2) API Request: Stock Market Historical Data, which is the request information passed by the root process to the historical market data provider ; (3) API Response: Stock Market Historical Data, which is the data passed by the historical market data provider to the process after accepting its request. (4) Trained Model, this is the object class from the Machine Learning model that will be developed and used by the system; (5) Stock Market Historical Data, as the name suggests this is the historical market data which will also be used to train and improve the machine learning model; (6) API Request: Stocks to Buy, which is the data passed from the third-party application to the root process to request for which stocks are in the Buy document of the database; (7) API Response: Stocks to Buy, upon the request of the third-party application, the root process will process the request to the API and sends back the list of stocks to buy to the requester; (8) API Request: Stocks to Sell, which is the data passed from the third-party application to the root process to request for which stocks are in the Sell document of the database; (9) API Response: Stocks to Sell, upon the request of the third-party application, the root process will process the request to the API and sends back the list of stocks to sell to the requester; (10) API Request: Stock Info, which is the data passed from the third-party application to the root process to request for the general information about a particular stock, this will be further discussed in the Object-Document Mapped (ODM) diagram; and (11) API Response: Stock Info, upon the request of the third-party application, the root process will process the request to the API and sends back the information of the stock based on what was requested.

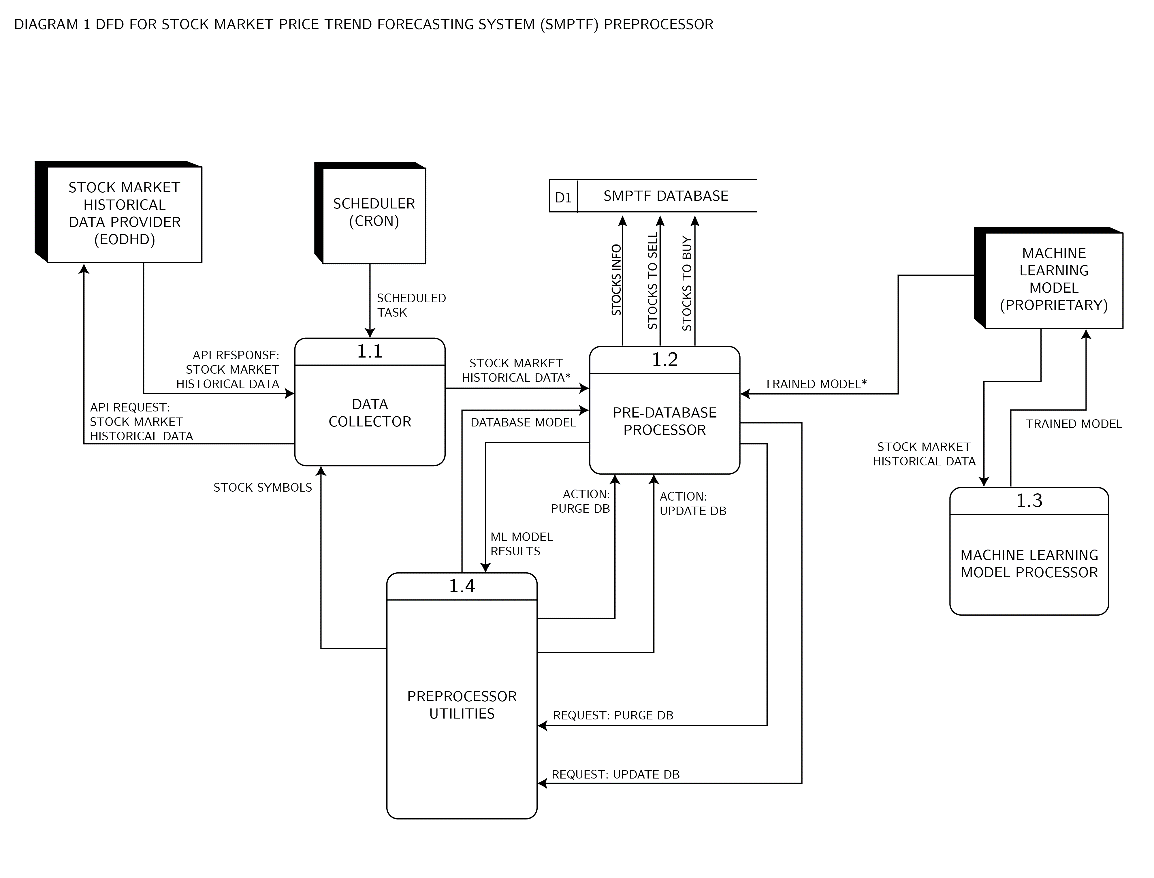
**DFD of Diagram 0.** To better understand how each data going in-and-out of the root process, is being processed, it is essential that we look inside the inner workings of the root process, which is shown in the DFD of Diagram 0, as provided in the figure below.



*Figure 8. DFD of Diagram 0*

From the figure above, the root process, has two main processes: (1) SMPFT System Pre-Processor, which is the system’s data processing unit; and (2) SMPFT System API and DB Model Module, which processes the API requests and responses, as well as the database of the system.

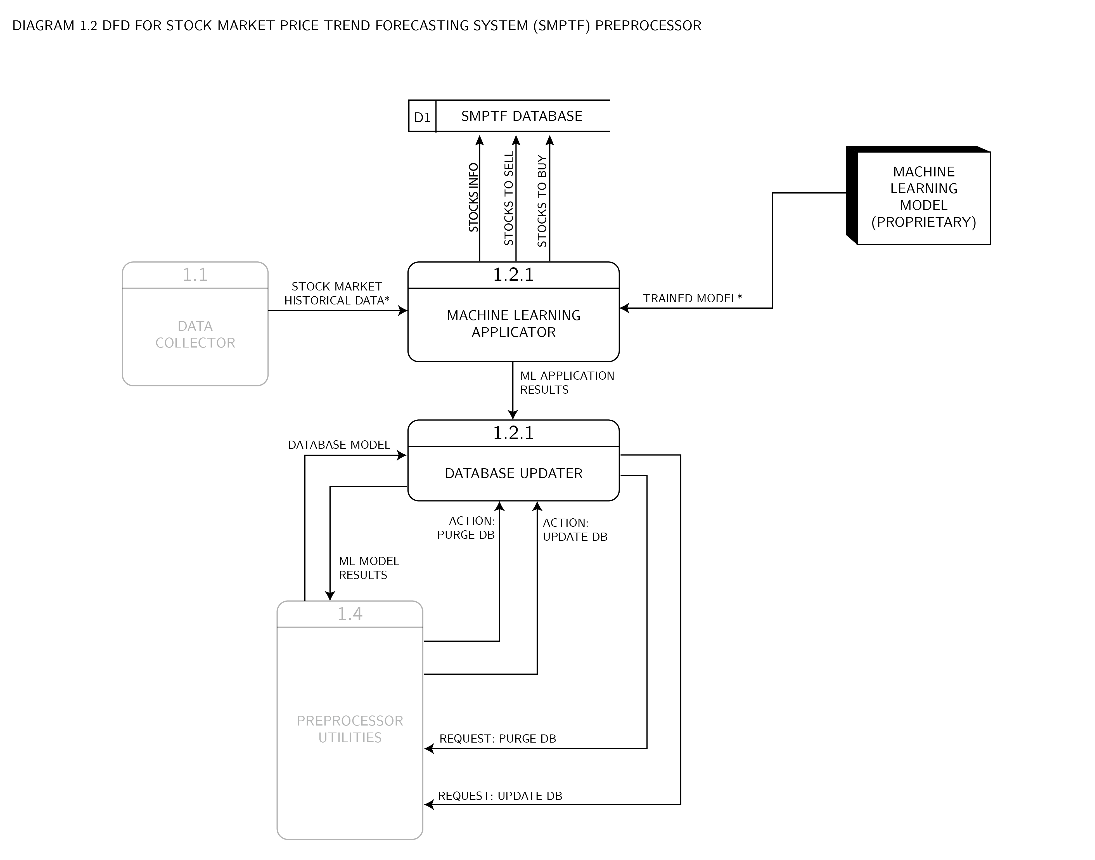
**DFD of Diagram 1.** To better understand the internal workings of the Process 1, it will be useful to check the DFD of that process, which is provided in the figure below.

**

*Figure 9. DFD of Diagram 1*

From the figure shown above, it can be observed that Process 1 is composed of 4 internal processes, namely, (1) Data Collector, which is the main process responsible for collecting the historical market data; (2) Pre-Database Processor, which is the processes that the collected data goes into before being sent to the system’s database, the internal processes of this process will be further discuss in the succeeding part of this section; (3) Machine Learning Model Processor, which is the training module or process for the machine learning model that is used to externally train the machine learning model that will be used by the system in the pre-database processor; and (4) Preprocessor Utilities, this will be the processes the process any system utilities such as the database actions, database models, and stores temporary data and system variables.

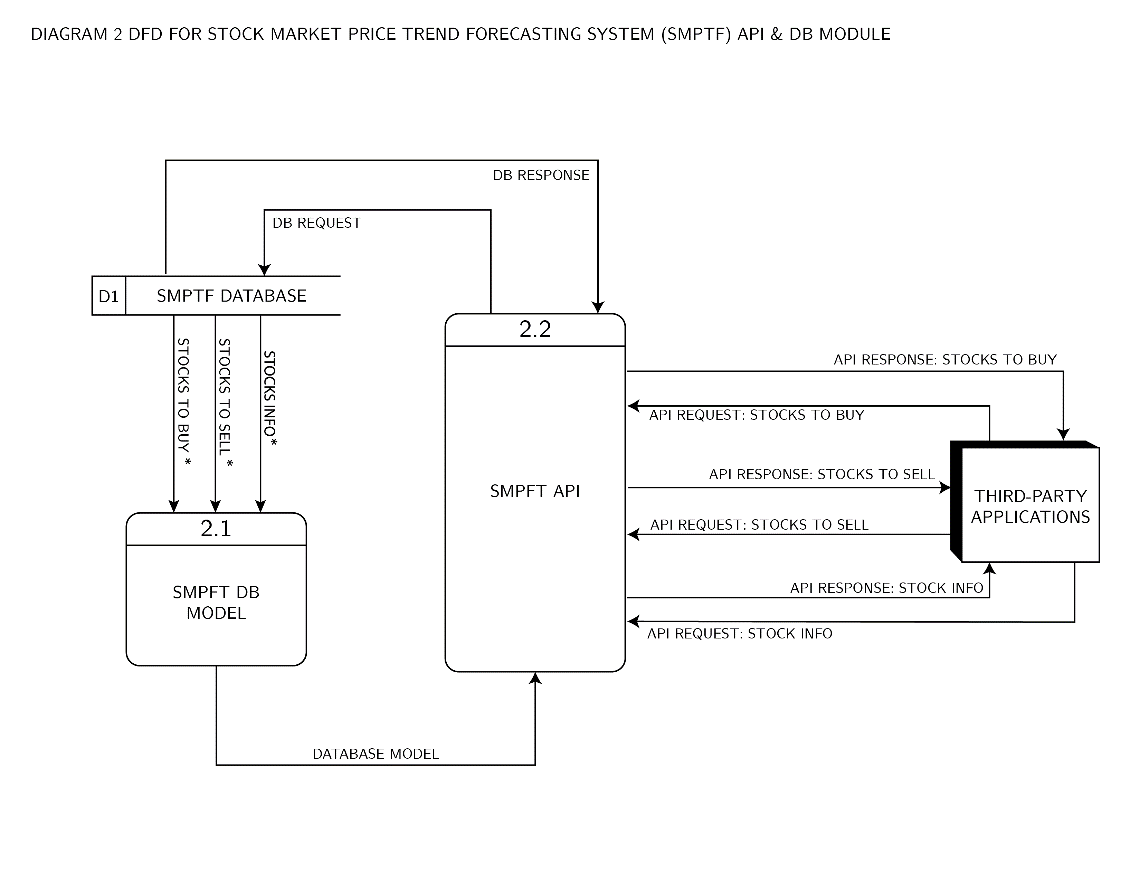
***DFD of Diagram 1.2.*** This shows the processes inside the process 1.2: Pre-database Processor.



*Figure 10. DFD of Diagram 1.2*

As previously discussed, the pre-database processor consists of processes inside that processes the data before it will be eventually sent to the database of the system. Wherein the process and data flow is shown in the figure above. Namely: (1) Machine Learning Applicator, which applies the trained machine learning model to the collected data; and (2) Database Updater, which processes the document outputs from the Machine Learning Applicator process, to be used in the database of the system.

**DFD of Diagram 2.** The final diagram will show the inner processes of the Process 2: SMPFT System API and DB Model Module.



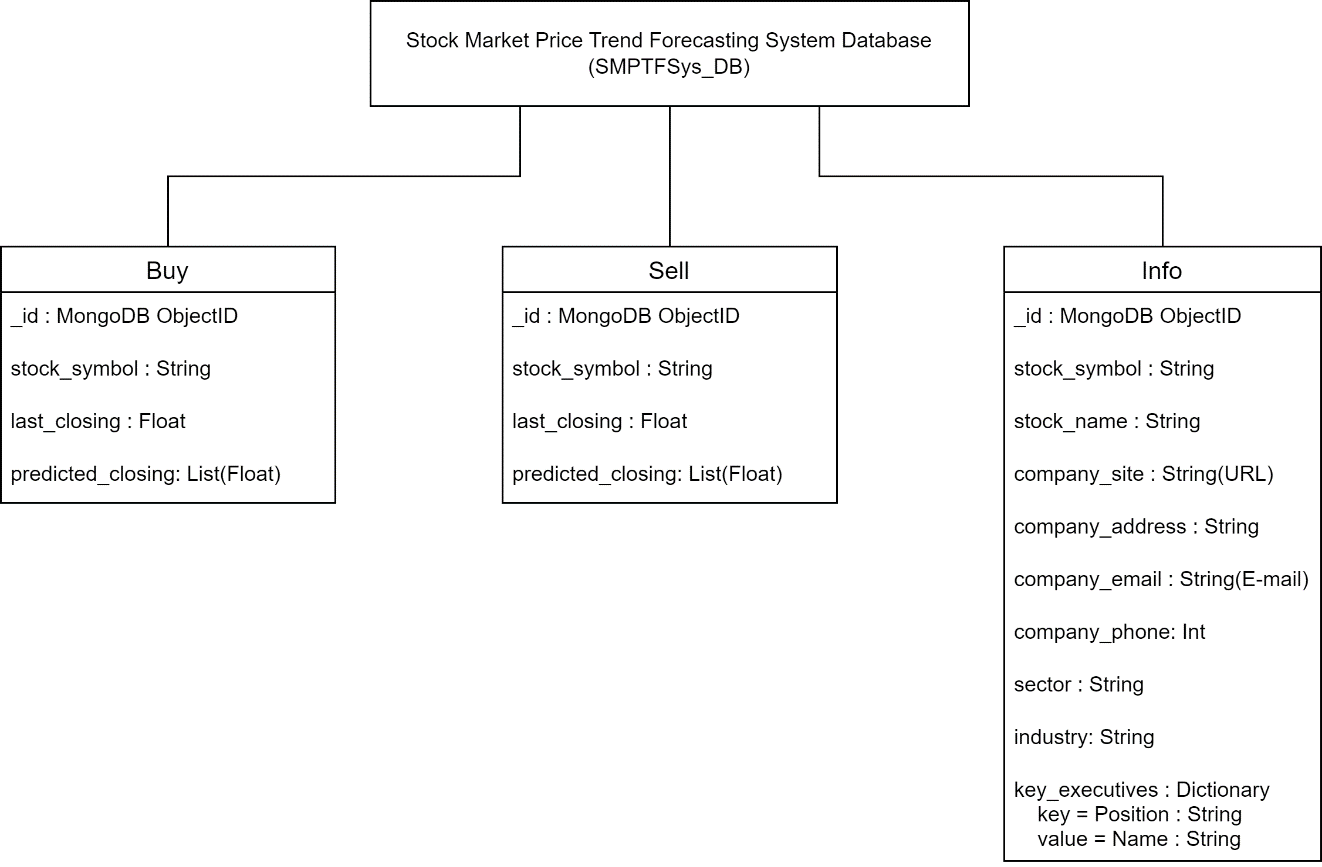
*Figure 11. DFD of Diagram 2*

The figure above shows two internal processes of the Process 2, namely, (1) SMPFT DB Model, which the database model that is used to process and connect MongoDB to the Python program; and (2) SMPFT API, which is composed of the API endpoints that processes the requests and response to and from the system, respectively.

***Object Document Mapper (ODM) Diagram***

Since the database that will be developed for the system will be a non-relational, hence an Object Document Mapper (ODM) diagram is shown in this section rather than an Entity Relationship Diagram (ERD).

The ODM diagram is as shown on the figure below.



*Figure 12. ODM Diagram for SMPTF System Database (SMPTFSys\_DB)*

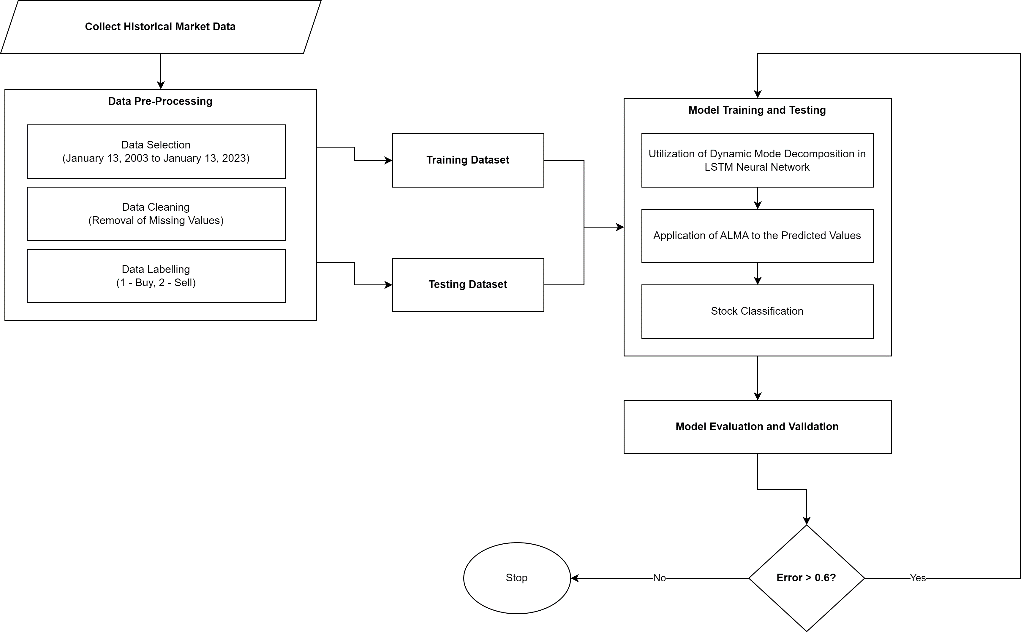
As shown from the figure above, the SMPTFSys\_DB will be the collection name of the non-relational database of the system. Wherein it will be composed of three documents with the list items following this convention: “item name”: “item type”. Note that each document are their own separate entities, hence the database is called non-relational, as the documents are not in any way related to each other.

The three documents are as follows:

1. Buy – this document will contain all the stocks that the machine learning model predicted and classified as a stock to buy. The diagram shown in Figure 12 also tells the path in which this document can be accessed, that is: MongoDB Instance 🡺 SMPTFSys\_DB 🡺 Buy. Wherein, information regarding the stocks can be accessed using the stock\_symbol, since the \_id is a private id.
2. Sell – this document will contain all the stock that the machine learning model predicted and classifies as a stock to sell. The diagram shown in Figure 12 also tells the path in which this document can be accessed, that is: MongoDB Instance 🡺 SMPTFSys\_DB 🡺 Sell. Wherein, information regarding the stocks can be accessed using the stock\_symbol, since the \_id is a private id.
3. Info – this document will contain the general and relevant information about a stock, or the general company information. The diagram shown in Figure 12 also tells the path in which this document can be accessed, that is: MongoDB Instance 🡺 SMPTFSys\_DB 🡺 Info. Wherein, information regarding the stocks can be accessed using the stock\_symbol, since the \_id is a private id.

***Machine Learning Model Diagram***

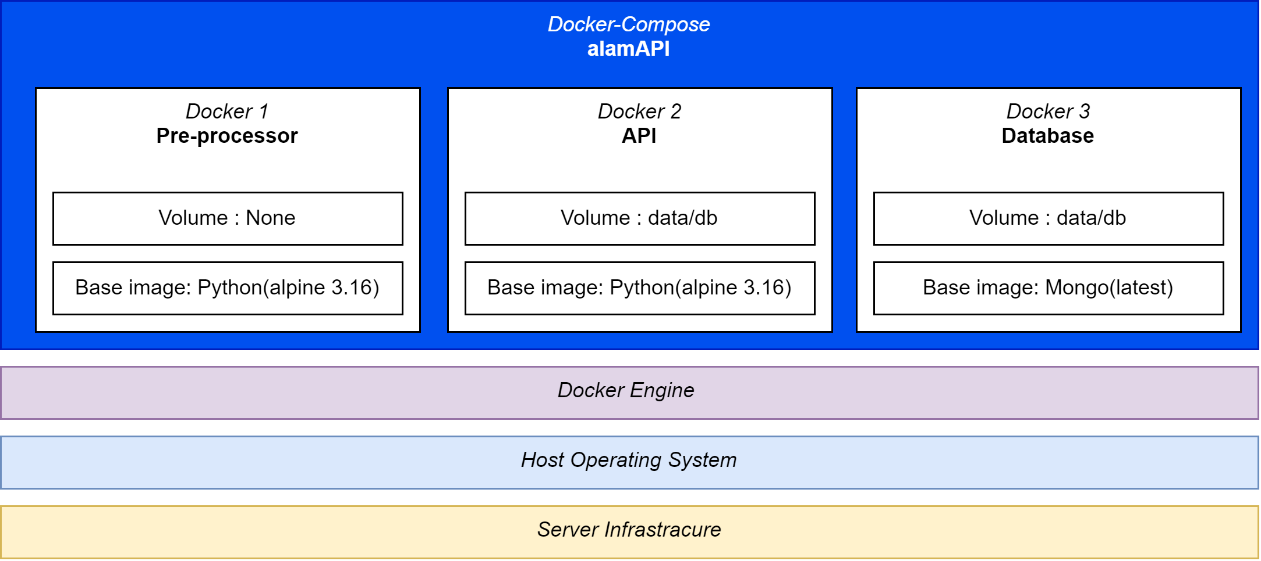
In this section, the process on how the machine learning model will be developed is shown by the figure provided below. Wherein, the process overview is based on the Fine-Tuned Support Vector Regression Model for Stock Predictions by Dash, Nguyen, Cengiz, and Sharma (2021).

**

*Figure 13. Overview of the Process of Machine Learning Model Development*

***Docker-Compose Layer Diagram***

In this section, the different layers of the docker-compose containers are shown based on the way it will be used in the deployment of the system. Moreover, this diagram is also based on the provided diagram in the Docker documentation, regarding containers. Note that in the diagram shown below, the lowest level is the” Server Infrastructure” and the highest level are the three Docker instances.

**

*Figure 14. Docker-Compose Layers Diagram*

**Hardware Requirements**

In this section, the hardware requirements will be discussed.

***For the Development of the alamAPI, and Training and Testing of the Machine Learning Model***

To develop the alamAPI and the underlying system that it utilizes, the project developer would be needing a laptop with at least the following minimum requirements:

1. A desktop class 4-core CPU running on 2GHz (minimum).
2. 16 GB of Random Access Memory (RAM). This is to ensure that multiple instances of programs can run efficiently in the system.
3. An up-to-date GPU with CUDA cores, this will be used specifically for faster training and testing of the machine learning model. Although if this is not available then a more powerful CPU maybe required.

***For the Development of the Test Application and System Testing***

1. A device that can connect to a network such as a smartphone. More preferably an Android smartphone as the test application that will be developed will run on Android devices. The specifications do not matter, if it can run a browser or the developed Android test application.

**Methodology**

This section of the Chapter 3 will be divided into two sections:

1. Software Development Process, wherein an Agile development will be discussed; and
2. Procedures, wherein the general procedures of development will be tackled.

***Software Development Process***

Due to the expected heavy time constraints of the development of the system, the author of this paper decided to follow an Agile Software Development Process, primarily it will be using Agile Sprints for an efficient time management during the whole software development process. Wherein the following are the list of Sprints and sub-activities that will be followed are shown in the Table below:

Table 1. Summary of Sprints and Target Activities

|  |  |  |
| --- | --- | --- |
| **SPRINT NUMBER** | **TARGET ACTIVITIES** | **ALLOTED TIME1** |
| 1 | **Main Activity:** System Planning and Evaluation  **Sub Activities:**   1. Topic Proposal 2. Drafting of Chapters 1 to 3 for the Special Problem Proposal. 3. System Architecture and User Requirement Analysis | **12 Weeks**  *Start: September 15, 2022*  *End: December 9, 2022* |
| 2 | **Main Activity:** System Prototyping  **Sub Activities:**   1. Build the different component of the alamAPI as indicated in the top-level overview diagram of the system, the following prototype will be developed: 2. API endpoints 3. Database 4. Pre-processor 5. Testing of the build prototype. This also include creating unit test cases for each component. 6. Initial Documentations, this will be done inside the GitHub repository. | **13 Weeks**  *Start: September 30, 2022*  *End: January 05, 2023* |
| 3 | **Main Activity:** Machine Learning Model Training, Testing, and Evaluation  **Sub Activities:**   1. Collection of Historical Data, outside the Data Collector module of the system. As the full data will be needed for each stock for the training, rather the 200-day only historical data. Whereas the last date on the market data should be January 13, 2023. 2. Development of the Machine Learning Model. This includes data standardization, data splitting, and data training. 3. Machine Learning model testing and evaluation. 4. Revision of Chapter 1-3, in preparation for the final paper submission. | **10 Weeks**  *Start: January 15, 2023*  *End: March 30, 2023* |
| 4 | **Main Activity:** Integration of Machine Learning Model to the alamAPI and Additional Data Collection  **Sub-Activities:**   1. Testing and Evaluation of alamAPI with the integration of the Machine Learning Model. 2. System Testing, this will be done to verify the functionality of the whole system, given a test deployment environment. Moreover, this will be done in a span of 4 weeks 3. Drafting of Chapter 4 and 5 | **6 Weeks**  *Start: March 31, 2023*  *End: May 12, 2023* |
| 52 | **Main Activity:** System Documentation  **Sub-Activities:**   1. Updating and Finalization of Documentations included in the GitHub Repository. 2. Writing of the results, discussions, conclusions, and recommendations for Chapter 4 – 5 3. Special problem paper revisions 4. Start the development of the test application (for showcasing of the system features) | ***6 Weeks***  *Start: April 14, 2023*  *End: May 26, 2023* |
| 62 | **Main Activity:** Preparation for Final Defense and System Presentation  **Sub Activities:**   1. Finalization of the mobile-based test application 2. Revisions and Finalization of the special problem paper. 3. Creation of presentation slide deck for the presentation of the special problem. | **3 Weeks**  *Start: May 27, 2023*  *End: June 17, 2023* |

1. *Start and End Dates are based on the University’s Academic Calendar and the Schedule provided by the Special Problem Adviser.*
2. *Sprints 5 and 6 are no longer part of the actual system development but is still included as a basis for the Gantt chart. Moreover, these activities can still be considered as part of the documentation process.*

From the given table above, it is shown that there is a total of 39 weeks; from September 15, 2022, to June 17, 2023, however it must be noted that an additional 1 week was added to each sprint’s allotted time to compensate for any unforeseen events during each sprint.

It should also be noted that Sprint 1 and Sprint 2 overlaps as the development of the prototype will start at Week 3, this will be possible as there will already be an initial system design to be followed, and any changes made during Sprint 1 can easily be adjusted to the creation of the prototype of the system in Sprint 2. This is also the case for Sprints 4 and 5, since their activities overlaps with each other, such that there are things in Sprint 4 that are unsupervised, hence, to better manage the time it is reasonable to start the activities of Sprint 5 along side the later parts of Sprint 4.

Moreover. The full details about the scheduling will be further discussed in the Gantt Chart of this chapter.

***Procedures***

In this section, the step-by-step procedures that will be followed in line with the development and testing of the system; alamAPI. Whereas the following are the procedures:

1. Designing of the System Architecture for alamAPI
2. Designing of Machine Learning Model
3. Development of System Prototype
4. Training, Testing, and Evaluation of the Machine Learning Model
5. Integration of the Machine Learning Model to the alamAPI
6. Initial testing for alamAPI, this shall also include any debugging, bug fixing, and code refactoring.
7. Pre-deployment testing, this testing phase includes the following tests that will be done for a one-month continuous system operation:

* Functional Testing, by monitoring the functionality of the alamAPI over 30 days and checking the success and error logs at the end of the given timeframe.
* Stress Testing, by creating ten-million artificial requests to the API everyday for 30 days.

1. Logging and summarization of results from all the prior tests conducted
2. Analysis and discussion of test data results.
3. Code Documentation
4. Maintenance, which will span beyond the time scope of the special problem.

**Gantt Chart**

Based on Table 1, the following figures for the Gantt Chart (created using TeamGantt) shows the software development schedule for the development of alamAPI. The Gantt Chart is divided into the different sprint to present the project scheduling. Moreover, a zoomed-out view of the whole Gantt Chart, will also be provided at the end of this section.

***Gantt Chart for Sprint 1***

The figure below shows the schedule of activities for Sprint 1. Wherein, it will start on September 15, 2022, and end on December 9, 2022.

A picture containing timeline

Description automatically generated

*Figure 15. Gantt Chart for Sprint 1: System Planning and Evaluation*

***Gantt Chart for Sprint 2***

The figure below shows the schedule of activities for Sprint 2. Which will start on September 30, 2022, and end on January 5, 2023.

*Chart

Description automatically generated*

*Figure 16. Gantt Chart for Sprint 2: System Prototyping*

***Gantt Chart for Sprint 3***

The figure below shows the schedule of activities for Sprint 3. Wherein, it will start on January 15, 2023, and end on March 30, 2023.

*Application

Description automatically generated with low confidence*

*Figure 17. Gantt Chart for Sprint 3: Machine Learning Model*

*Training, Testing, and Evaluation*

***Gantt Chart for Sprint 4***

The figure below shows the schedule of activities for Sprint 4. Which will run from March 31, 2023, until May 12, 2023.

*Chart

Description automatically generated*

*Figure 18. Gantt Chart for Sprint 4: Integration of Machine Learning Model*

*to the alamAPI and Additional Data Collection*

***Gantt Chart for Sprint 5.***

The figure below shows the schedule of activities for Sprint 5. Which will be from April 14, 2023, to May 12, 2023.

*Chart

Description automatically generated with medium confidence*

*Figure 19. Gantt Chart for Sprint 5: System Documentation*

***Gantt Chart for Sprint 6.***

The figure below shows the schedule of activities for the final sprint for the development of alamAPI. Which will be done from May 27, 2023, until June 17, 2023. However, it should be noted that the end day may change, depending on the scheduled final defense.

*Chart

Description automatically generated with low confidence*

*Figure 20. Gantt Chart for Sprint 6: Preparation for*

*Final Defense and System Presentation*

***Full Gantt Chart.***

To have an overview of the whole schedule of each Sprints, the full Gantt chart is provided below.

***Chart

Description automatically generated with medium confidence***

*Figure 21. Full Gantt Chart for the Development of alamAPI*

**References**

Dash, R. K., Nguyen, T. N., Cengiz, K., & Sharma, A. (2021). Fine-tuned support vector regression model for stock predictions. Neural Computing and Applications. doi: <https://doi.org/10.1007/s00521-021-05842-w>

Docker. (n.d.). *Use Containers to Build, Share and Run Your Applications*. Docker + WASM. Retrieved November 13, 2022, from <https://www.docker.com/resources/what-container/>

EODHD. (n.d.). *Financial APIs Documentation. Fundamental and Historical Data APIs*. EODHD. Retrieved November 13, 2022, from <https://eodhistoricaldata.com/financial-apis/>

MongoEngine. (n.d.). MongoEngine User Documentation. MongoEngine. Retrieved November 13, 2022, from <http://docs.mongoengine.org/>

Tiangolo. (n.d.). FastAPI Documentation. FastAPI. Retrieved November 13, 2022, from <https://fastapi.tiangolo.com/>

Visual Paradigm Online. (n.d.). *Gane-Sarson Data Flow Diagram Tutorial*. VisualParadigm Online. Retrieved November 15, 2022, from <https://online.visual-paradigm.com/knowledge/software-design/gane-sarson-dfd-tutorial/>