High level Architecture

03 : Autonomous Trading Bot

Student ID	Name
23100011	Suleman Mahmood
23100197	Ahmed Tahir Shekhani
23100198	Ali Asghar
23100176	Syed Talal Hasan
23110345	Muhammad Ammar Ibrahim

Table of Contents

1. Introduction

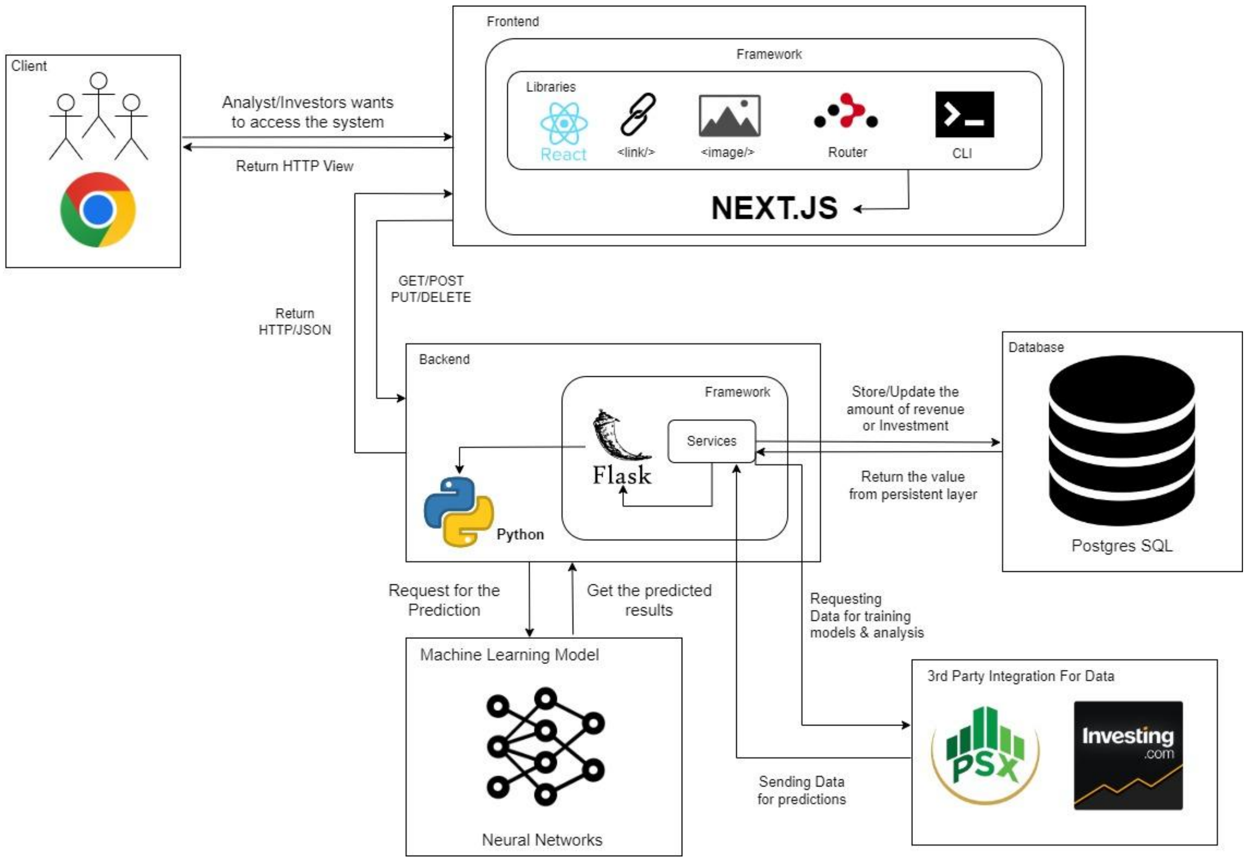
A web application with an autonomous trading bot instance that will trade to generate profitable returns on stocks. The bot will be trained on the PSX data. The bot's decision will be based on the concepts of game theory, mathematical models, financial techniques, and especially artificial intelligence. The primary web app will allow the user to provide the bot's configuration, which includes, target return, risk appetite, and duration of the instance.

With recent advancements in deep learning frameworks and access to faster gpus, training complex models that can predict on time series data has opened new avenues to explore stock market trading. We plan on using models that have a memory component in them, such as LSTM (Long Short Term Memory) to make predictions and trades on the stock market.

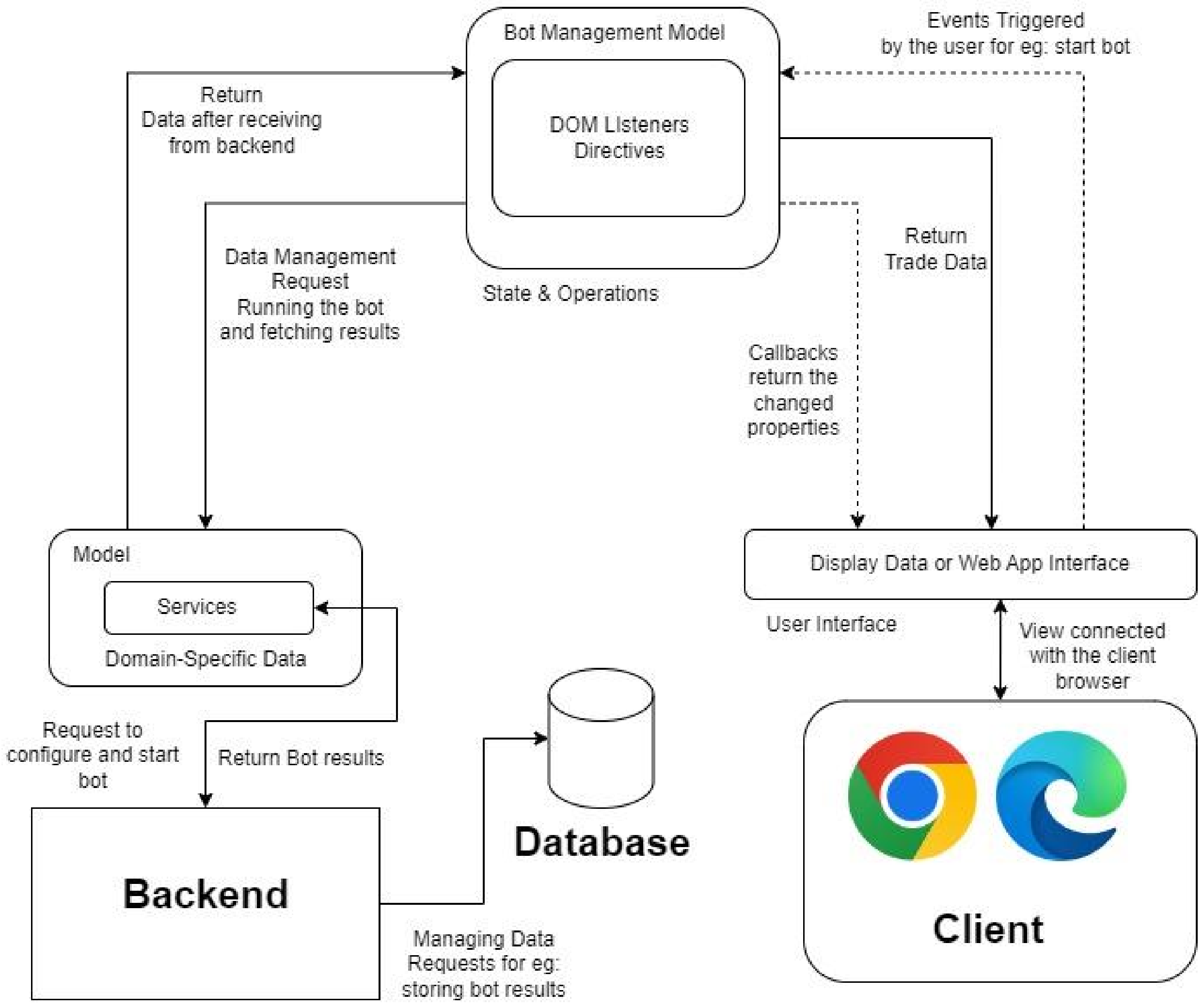
The overall objective for the application would be to achieve the return target provided by the analyst while configuring the bot and minimize loss according to the risk factor provided. The potential users of this application would be trade analysts or managers who will use the bot to run its instances according to their requirements.

Technical details:
The project's tech stack would be Next.js for frontend web application, Flask for backend server, and PostgreSQL for our persistent storage. The application would follow three-tier architecture with a repository pattern for the persistent layer, models, and command layer for modifying the state.

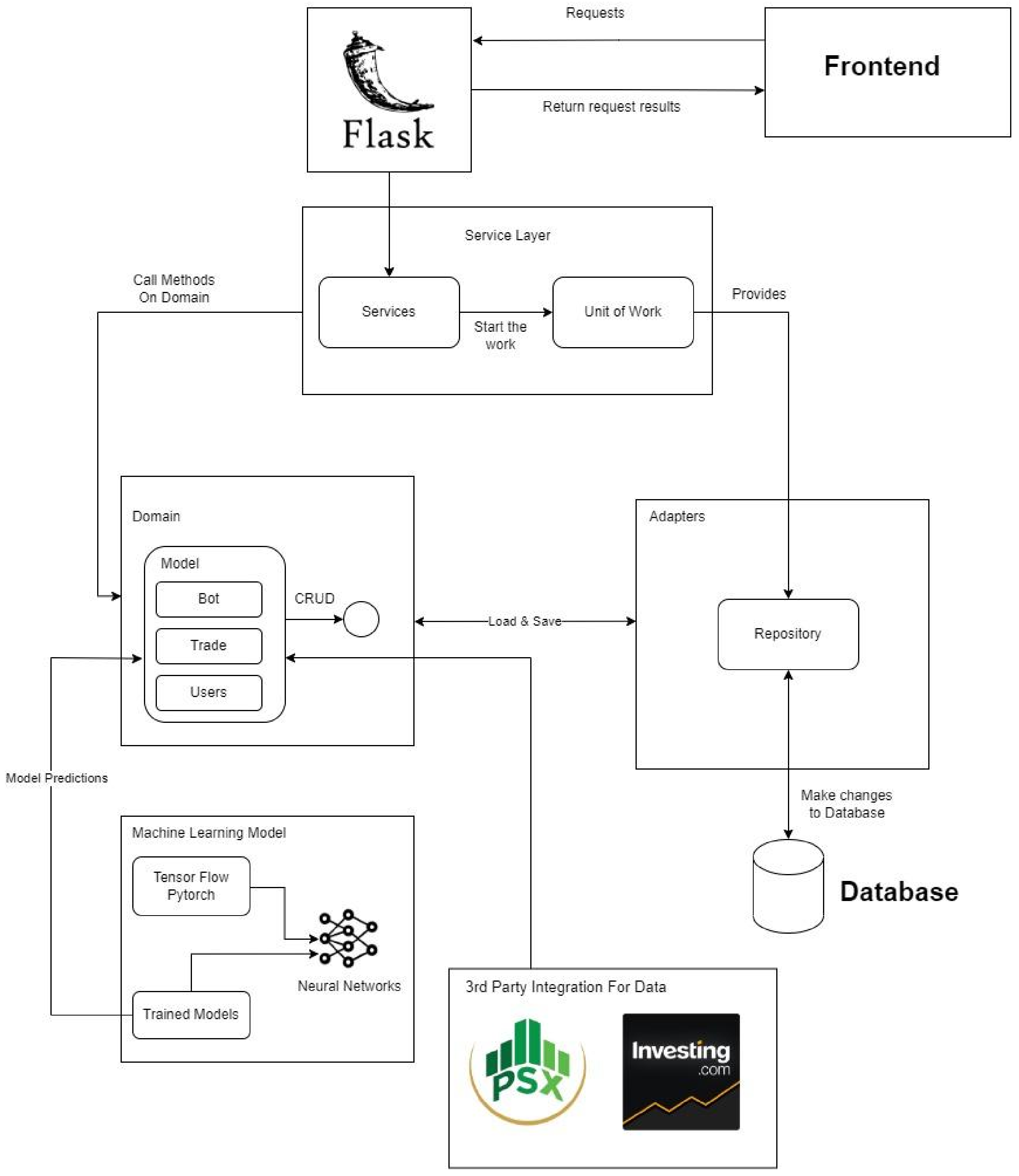
1.1 Architecture Diagram


Frontend
Framework
Libraries
React
<link/>
<image/>
Router
CLI
NEXT.JS
Client
Analyst/Investors wants to access the system
Return HTTP View
GET/POST PUT/DELETE
Return HTTP/JSON
Backend
Framework
Flask
Services
Python
Database
Store/Update the amount of revenue or Investment
Return the value from persistent layer
Postgres SQL
Request for the Prediction
Get the predicted results
Requesting Data for training models & analysis
Machine Learning Model
Neural Networks
3rd Party Integration For Data
PSX
Investing.com
Sending Data for predictions

Frontend - MVVM Architecture



Backend Architecture based on Domain Driven Design



1.2 Architecture Description

The project is based on three-tier architecture, front-end, back-end and database. The front-end is using MVVM design pattern, and we are using Next.js framework, which does server side rendering for better user experience. Front-end communicates with back-end through api calls, and we are using a python based flask framework in the back-end which integrates postgre-sql database, machine learning models and 3rd party services such as investing.com and PSX. The client sends a request to the frontend that is based on next.js framework, where the Dom listener is the View Model and the Client request is the view. The services are connected to the backend and it is the controller in this model.

The Backend uses the domain driven design (DDD). This enables the backend to load micro services and connect them. Whenever a command is received by the flask API it checks if it is a view or a service request. If it is a micro service request, the domain of the micro service is loaded and the changes are made and then sent to the user. The repository is connected to the database.

1.3 Justification of the Architecture

Backend

DDD (Domain Driven Design) allows us to architect our backend server in terms of microservices. Our implementation involves various iterations of strategies with which the bot makes its decisions so DDD helps us to create pure models in python which will be able to process and make decisions. The architecture does not depend on the existence of any specific library for a specific implementation. This allows us to use such frameworks as tools, rather than having to cram our system into the limited constraints. It also allows us to write testable code and follow TDD (Test driven development) and write unit test cases for the domain model, repository model and write some E2E tests to test complete functionality. Writing unit tests for the domain model and commands is very fast since they don't require any implementation details for the databases. FakeRepositories and Unit of work allows abstractions and dependency inversion principle allows this architecture to be independent of its inner layers. This brings more confidence into our code and prevents code that is testable and verifiable. Since we are also working in the domain of finance involving real money, quality code that does what it is intended to do is of the highest importance. This architecture is also Independent of UI which can change very easily, without changing the rest of the system. Our business rules are not bound to any choice of the database. Independent of any external factors since layers aren't dependent on outside layers and they can easily be swapped.

Frontend

For frontend, Next Js is used which is based on MV-VM Architecture. Our above given architecture for frontend used MV-VM as base design. Next JS helps in server-side rendering, along with client side with optimized routing system. It makes the code further re-usable. It makes fully interactive, performant and highly dynamic websites. Particularly MVVM, also flattens the dependencies. It means that the focus would be on single function where the function is separated from the interruption of other dependencies.

2. Risk Management

2.1 Potential Risks and Mitigation Strategies

Sr.	Risk Description	Mitigation Strategy
1.	Limited access to high performance GPUs to train models	Use colab pro or buy high performance GPUs
2.	Unable to access real time PSX API	Use freely available data by scraping the internet Pay for bloomberg subscription
3.	No experience in deploying a deep learning model	Collaboration with experts at 10 pearls to help in deployment of the model
4.	Backend unable to run multiple model instances simultaneously and cause delayed response exceeding minimum acceptable response time of 1 second	Simplify the model by using neural network compression strategies
5.	LSTM model unable to provide 70% accuracy on the available training data	Work on statistical models in parallel to deep learning solutions Acquire much larger amount of data than required for training the model
6.	Only 1 member in the team has experience working with deep learning frameworks	Thoroughly document the work done during development of model
7.	Deep learning model may be outdated too quickly	Re-train the model every 2 days
8.	Only one member of the development team has working knowledge of trading	Work with students of business school to get all requirements prior to the commencement of development
9.	Our development is dependent on all 5 members and if anyone stops the work might affect others	Divided the task into components and assigned the task to sub-groups so one can substitute the other in emergency.
10.	Course load from other courses might cause delays in development	Divide tasks equally with buffers in completion deadlines

3. Tools and Technologies

Frontend

Language
- Typescript 4.8.4
- ESLint 8.25.0

JavaScript library
- Next.js 17.0.2
- React.sj 18.2.0

CSS Framework
- Tailwind CSS 3.1.8
- DaisyUI 2.31.0

API library
- Axios 1.1.2

Deployment
- Vercel

Backend

Language
- Python 3.10.7
- PostgreSQL 15

Persistent storage
- PostgreSQL 15

Server
- Flask 2.2.2

Deployment
- Google Cloud’s App Engine flexible

Deep Learning Models:

Language

- Python 3.10.7

Frame work

- Pytorch and Tensorflow 1.12.1

Tool

- Google Colab

3. Hardware Requirements

FrontEnd Requirements:
(Basic)

1. Anyone of the following operating systems:
    a. Windows: 7 or newer
    b. MAC: OS X v10.7 or higher
    c. Linux: Ubuntu
2. Processor: Minimum 1 GHz; Recommended 2 GHz or more
3. Ethernet connection (LAN) OR a wireless adapter (Wi-Fi)
4. Hard Drive: Minimum 2 GB; Recommended 4 GB or more
5. Memory (RAM): Minimum 1 GB; Recommended 2 GB or above

Development Requirements:

1. Anyone of the following operating systems:
    a. Windows: 7 or newer
    b. MAC: OS X v10.7 or higher
    c. Linux: Ubuntu
2. Processor: Minimum 2 GHz; Recommended 4 GHz or more
3. Ethernet connection (LAN) OR a wireless adapter (Wi-Fi)
4. Hard Drive: Minimum 8 GB; Recommended 16 GB or more
5. Memory (RAM): Minimum 4 GB; Recommended 8 GB or above

BackendEnd Requirements (deployment):

1. Anyone of the following operating systems:
    a. Windows: 7 or newer
    b. MAC: OS X v10.7 or higher
    c. Linux: Ubuntu
2. Processor: Minimum 4 GHz; Recommended 8 GHz or more
3. Ethernet connection (LAN) OR a wireless adapter (Wi-Fi)
4. Hard Drive: Minimum 64 GB; Recommended 128 GB or more
5. Memory (RAM): Minimum 8 GB; Recommended 16 GB or above

1. Who Did What?

Name of the Team Member	Tasks done
Suleman Mahmood	Architecture justification, Tools and Technologies
Ahmed Tahir Shekhani	Architecture diagram, Architecture Description, Updated the risk management , reviewed the document
Syed Talal Hasan	Risk Management
Ali Asghar	Hardware Requirements, reviewed the risk management

2. Review checklist

Before submission of this deliverable, the team must perform an internal review. Each team member will review one or more sections of the deliverable.

Section Title	Reviewer Name(s)
Architecture diagram, Hardware Requirements	Suleman Mahmood
Tools and Technologies	Ahmed Tahir Shekhani
Hardware Requirements	Syed Talal Hasan
Risk Management	Ali Asghar