Chapter 03: Design

3.1 Chapter overview

3.2 Design Goals (quality attributes that are important to decide the architecture)

3.3 High level Design/ System Architecture Design

3.3.1 Architecture Diagram

- Can be drawn using tiered architecture or layered architecture

3.3.2 Discussion of tiers/ layers of the Architecture

3.4 Low-level Design/ System Design

3.4.1 Choice of design paradigm

Note: possible design paradigms => OOADM(object oriented analysis and design methodology / SSADM(structured systems analysis and design methodology)

[Depends of the programming paradigm you use.

Ex. Java -> OOADM

Structure programming -> SSADM]

3.5 Design Diagrams

3.5.1 Component Diagram (for both SSADM & OOADM)

Note: Possible design diagrams:

OOADM =>

Class diagram

Sequence Diagram

Other UML diagrams as applicable

SSADM =>

Data Flow Diagrams (Level 1 & Level 2 – only for the most important process identified in the level 1 DFD)

3.5.X1 Algorithmic Design (if you have come up with a novel algorithm- optional)

3.5.X2 Algorithmic Analysis

3.5.Y1 System Process Flow Chart (How the system will work form end to end. Use either a flow chart or an activity diagram (preferred) [object-oriented flow chart])

3.5.Z1 User Interface Design

1. low level fidelity wireframe diagram

2. high level fidelity prototype

Note: Any diagram that’s relevant to your project. At the same time don’t try to overdo by putting all possible diagrams

3.6 Chapter summary

Chapter 04: Initial Implementation

4.1 Chapter Overview

4.2 Technology Selection

You need to tell what your frontend is, your programing language, IDE, and all other things relevant to implementation

4.2.1 Technology Stack (frontend, middle tier, backend technologies)

4.2.2 Data-set Selection (only if you’re doing a data science project)

4.2.3 Development Frameworks – What you’ve chosen and why (can be justified in a tabular format)

4.2.4 Programming Languages – What you’ve chosen and why (tabular format is OK)

4.2.5 Libraries – What and why (tabular format is OK)

4.2.6 IDE – What and why (tabular format is OK)

4.2.7 Summary of Technology Selection (in a tabular form)

4.3 Implementation of the Core Functionality

- Take each functionality and put the code in image format

4.4 User Interface (either design in the design chapter or actual in implementation chapter. This is optional since prototype level; you’re not expected to have an UI.)

4.5 Chapter Summary

Example – Refer below

# CHAPTER 6: DESIGN

## 6.1 Chapter Overview

This chapter consists of the design decisions made to come up with a suitable architecture for implementation, based on the gathered requirements. High-level design, low-level design, design diagrams, UI wireframes have been used to convey how the design goals are expected to be achieved while discussing the reasoning for chosen design decisions.

## 6.2 Design Goals

Table 6.1: Design Goals of the proposed system

|  |  |
| --- | --- |
| **Design Goal** | **Description** |
| Performance | The recommendations matrix & opinion-mining data can be pre-processed and stored in memory to be used for recommendations. Since ensembled models are expected to be utilized, concurrency would be ideal to get the output from multiple models at the same time. This could cut down the processing time by 4-5 times (based on the number of models that are required to provide recommendations for the given input). |
| Correctness | The correctness & quality of the output should be of the highest possible level, utilizing all the available data. Explaining why a user is getting the proposed recommendation will ensure that the user isn’t misled into wrong purchase decisions. |
| Usability | Since the purpose of the system is to automate and make it easy for the user to explore NFTs, the usability of the system must be easy for users of all levels of expertise. |
| Scalability | The system may have to support many concurrent user requests in a production environment. The backend should be able to handle this. New data should be able to be added to the system with minimum effort. |
| Adaptability | Since the utilized Recommendation models may have to be altered based on the available data and user requirements in the future, these models should be able to be easily swapped out for new models while ensuring that the system won’t break in the process of upgrading, with minimum changes. |

## 6.3 High-Level Design

### 6.3.1 Tiered Architecture

The system’s architecture is depicted in the diagram below. The data, logic and presentation layers are organized in a three-tier architecture.

The research contribution in this system lies in data preprocessing of the *data tier*, recommendations models, and the recommendations diversifier of the *logic tier*.

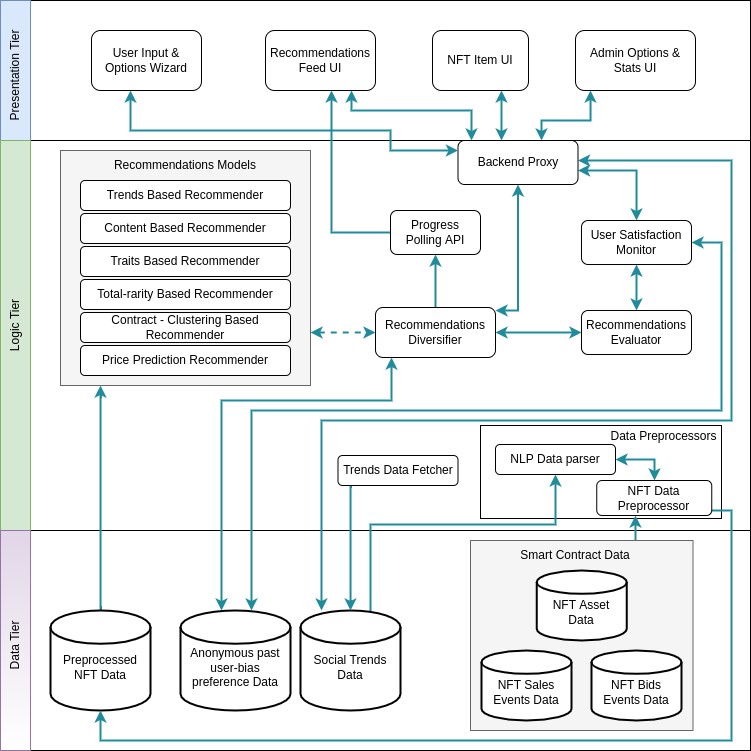


Figure 6.1: Three Tiered Architecture *(self-composed)*

While the entire architecture is represented in a modular approach for ease of understanding, several backend services are expected to work together in the fashion of a distributed microservices architecture when it comes to implementing the proposed architecture.

The reason for following a microservices architecture is to allow the system to scale while ensuring that points of failure can be easily recognized and taken care of separately. The distributed nature of the system is expected to be seen in the connection between the numerous Recommendations Models and the Recommendations Diversifier. These combined through output pipelines will act as an Ensebled Recommendations System. Although the system will be capable of distributing the load at this point, the expectation with the prototype is to run this in a single machine.

The purpose of each module that is represented in the above architecture is described below.

### Data Tier

1. Smart Contract Data - Data that is retrieved from Blockchain Smart Contracts. For convenience purposes, the data is fetched from the OpenSea API. Contains all the available data of each NFT.
   1. NFT Asset Data - All the content of each NFT.
   2. NFT Sales Events Data - Past sales data from NFT trading. (c) NFT Bids Events Data - All the current bids of each NFT.
2. Social Trends Data - Data gathered from social trends sites (Twitter, news sites, etc.)
3. Anonymous past user-bias preference Data - Each user’s preferred bias is stored anonymously. This can be identified by a user’s selection based on their requirement or based on the feedback received for each recommendation. This can be a temporary data store that can be cleared once the user session has ended.

### Logic Tier

1. Data Preprocessors - The preprocessing code required to modify/ extract required data that is usable for recommendations from all the available data.
   1. NLP Data parser - Responsible for extracting all the required data from what was collected through data mining techniques.
   2. NFT Data Preprocessor - Used to modify and separate data that can be utilized from smart contracts and processed trends data.
2. Recommendations Models - The various models that are used to provide recommendations based on identified diverse data points.
3. Recommendations Diversifier - The module that combines the recommendations produced by all the Recommendations Models, considering the bias.
4. User Satisfaction Monitor - The feedback received by users will be filtered and updated through this module, to update the moving bias while preserving user anonymity,
5. Recommendations Evaluator - The module that evaluates the user’s satisfaction with the recommendations produced, to separately identify under-performing & high-performing models.
6. Progress Polling API - The web-polling API that will be used to update the progress of recommendations generation in the frontend.
7. Backend Proxy - The interface that exposes the backend services to the frontend.
8. Trends Data Fetcher - Fetch global trends data from social APIs or by scanning through news websites.

### Presentation Tier (Client Tier)

1. User Input & Options Wizard - The UI that is presented to the user to enter the desired NFT(s) to be considered to recommendations as well as desired parameters and data-points (for advanced users).
2. Recommendations Feed UI - The UI that will show all the recommendations generated for a user. This will be similar to a home page on Youtube/ any other social network.
3. NFT Item UI - The UI that will show a chosen NFT with its data and recommendations.
4. Admin Options & Stats UI - The UI that will be exposed to a system Admin, allowing him to view the stats such as the general bias of the system. This will have options to define the data sources to be used for trends based recommendations and to adjust the bias.

## 6.4 System Design

### 6.4.1 Choice of the Design Paradigm

Although the author was very tempted to use OOAD (Object-Oriented Analysis and Design) to build the prototype due to the ease of extendability and further development of the system, the decision was made to use **SSADM (Structured Systems Analysis and Design Method)** based on the following factors.

* The project’s core research component is inclined towards Data Science. Therefore, it doesn’t gain a noticeable benefit by using Object Oriented approaches.
* The programming languages that are expected to be used for implementation don’t support OOP by nature.
* Ease of implementation of an MVP (Minimum Viable Product) for demonstrating the research application using the prototype.
* The time constraint of having to implement & document research within the time span of 10 months.

### 6.4.2 Data Flow Diagram

The Level 1 Data Flow Diagram presented below provides a more extensive breakdown of the components of the Context Diagram that was presented in the SRS.

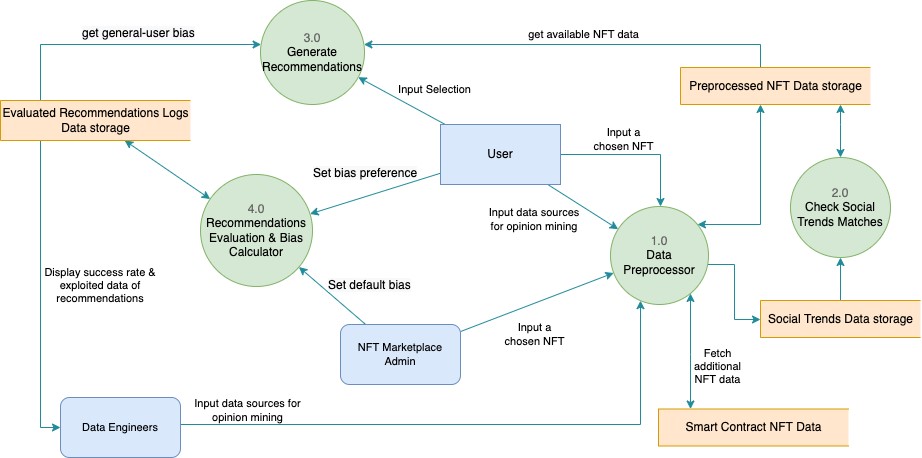


Figure 6.2: Data Flow Diagram - Level 1 *(self-composed)*

The Level 2 Data Flow Diagram presented below provides a more extensive breakdown of the components of the above Level 1 Data Flow Diagram.

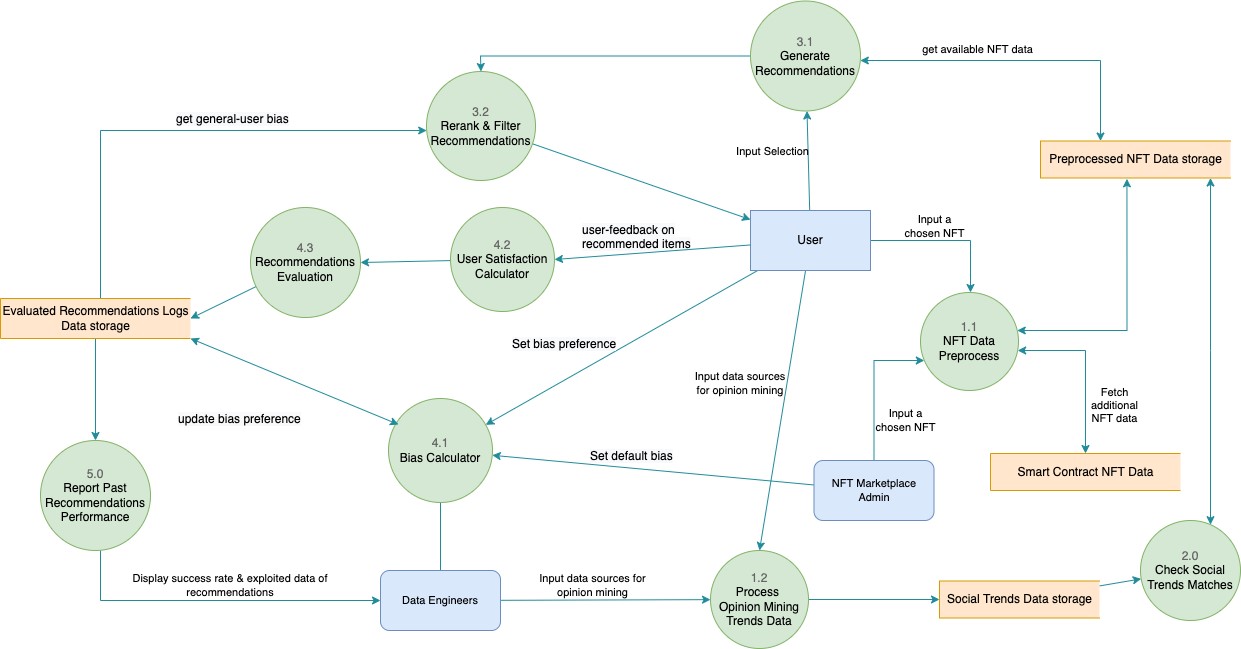


Figure 6.3: Data Flow Diagram - Level 2 *(self-composed)*

### 6.4.3 Algorithm Design

When studying available data in the system, it was identified that cross-collection NFTs cannot be recommended using the same concepts & data points followed for inter-collection matches. Therefore, multiple algorithms were considered to get a diverse set of recommendations.

### Infusing trends matches into Recommendations

The equation composed below is designed to be used to calculate the total trends score for an item. The methods of utilization of this score for recommendations have been discussed following the breakdown of the equation.

*𝑖𝑠* h *𝑤*  *𝑡*  i

*𝑁*

*𝑖*

*𝑠*

=

1

*𝑘*

*𝑘*

*𝑤*

=

1

*𝑠*

*𝑐*

*𝑣𝑡,𝑐*

*𝑀𝑒𝑑*

(

*𝑇*

*𝑣𝑡*

)

*𝑚𝑢*

(

*𝜇*

+

*𝑛*

*𝑚*

)

Í Í

*𝑇𝑡𝑠,𝑖* =(6.1)

*𝑁*

*𝑖𝑠*

Equation for social trend-match score for recommendations *(self-composed)*

*𝑇𝑡𝑠,𝑖* - Total trends score for one item

*𝑁𝑖𝑠* - Total number of information sources

*𝑖𝑠* - Source of information

*𝑘𝑤* - Number of keywords in the current item

*𝑠𝑐* - Sentiment score surrounding chosen trend content

*𝑚* - Match value, a Boolean used to check if the current evaluated content contains the chosen trend to be matched against.

*𝑢* - User priority, used to check the current user’s interest in the chosen trend. This is 1 by default

*𝑡𝑣𝑡,𝑐* - Tweet volume at this moment in time of the chosen content

*𝑀𝑒𝑑*(*𝑇𝑣𝑡*) - Median Tweet volume at this moment in time *𝜇* - Constant, set to 0.1 to avoid division by 0 error for today’s trends *𝑛𝑚* - Number of days between the current day & the day of the trend.

The following equation extracts the calculation of the impact score of the chosen trend (*𝑖𝑡*), as described above. Twitter data has been taken as the example source here. The data source can be even an internet forum.

*𝑖𝑡* = *𝑡𝑣𝑡,𝑐* (6.2)

*𝑀𝑒𝑑*(*𝑇𝑣𝑡*)

Equation for the calculation of the impact score of a chosen trend *(self-composed)*

For trends that don’t have a measurable volume, *𝑡𝑣𝑡,𝑐* can be taken as (*𝑇𝑣𝑡𝑚𝑖𝑛* − 1) to give it the lowest possible value, or as *𝑀𝑒𝑑*(*𝑇𝑣𝑡*) to omit the impact score all-together.

The algorithm, *𝑇𝑡𝑠,𝑖* can be applied to inter-collection recommendations as well, if each NFT in the collection has unique names and descriptions. Using unique traits didn’t seem to make sense for comparison with this algorithm, but it may be valid if it can be proved that the traits can be matched with trends data.

The Total trends score for one item calculated above can either be taken for recommendations as to the top N items or as an absolute similarity match with other chosen items’ trends scores.

The beauty of this equation is that it isn’t necessarily required to be applied for only NFT recommendations. It can be used to enhance any content-based recommendations model. It can be seen as another way of infusing collaborative filtering, without the collection of user-specific data by the platform that integrates the presented Recommendations Architecture.

### Recommendations based on Rarity

Õ*𝑁𝑡* 1

*𝑇𝑟,𝑡* = *𝑐𝑡* (6.3)

*𝑡*=1 *𝑇𝑁*

Equation for the calculation of the total trait rarity score of an NFT *((rarity.tools, 2021b),*

*(rarity.tools, 2021a))*

*𝑇𝑟,𝑡* - Total rarity of a trait

*𝑁𝑡* - Total number of traits in the NFT

*𝑐𝑡* - Trait count of the chosen trait (number of occurrences in the collection) *𝑇𝑁* - Total supply of NFTs in the collection

The absolute difference between the total rarities is calculated when an NFT from a collection is chosen. The lowest scoring items are recommended to the user. This gives the NFTs that may be as closely valuable as the initially chosen NFT. This allows recommending NFTs that don’t have unique content descriptions.

Furthermore, the traits are fed into a Content-based Recommendations Model to get NFTs with the most similar traits to be recommended.

### Varying Bias for Recommendations Diversifier

Finally, all these recommendations produced by algorithmic models had to be presented to the user suitably. Instead of going with a weighted bias which was recommended by the experts that were interviewed, it was decided to make this bias variable with time.

The reason for opting for this in contrast to having pre-trained weights & biases using a Neural Network architecture that Amazon successfully attempted with its recent Autoencoder (Larry, 2019) DL model was to allow a more optimized output, without having to retrain the model. Another reason to opt for this method was due to the lack of user data to identify the most optimum weights or to train a DL model.

The calculation of this bias draws concepts from Reinforcement learning techniques.

h *𝑔 𝑏𝑝,𝑠* i

*𝑛*

*𝑖*

=

0

(

*𝛼*

+

*𝑛*

*𝑚*

)

Í

*𝐵𝑤,𝑝* =(6.4)

*𝑁*

*𝑛𝑔*

Equation for the calculation of the recommendations bias in combining outputs in ensembled models *(self-composed)*

*𝐵𝑤,𝑝* - Default Bias weighting for a chosen pipeline that recommendations are given from

*𝑏𝑝,𝑠* - Successful bias selection for a chosen pipeline for the last n days *𝛼* - Constant, set to 0.001 to avoid division by 0 error for today’s bias selections *𝑛𝑚* - Number of days away from the current day.

*𝑛𝑔* - Grouped days (Eg: 1 day, 7 days, 1 month, 3 months, 6 months, 1 year)

*𝑁𝑛𝑔* - Total number of grouped days considered

### Applying Bias Push

When presenting recommendations, the author decided to allow a system admin to be capable of suggesting a push towards a preferred direction to allow the bias to be altered.

*𝐵𝑐,𝑝* = *𝑏𝑙,𝑝* + *𝐵𝑤,𝑝* − *𝑏𝑎,𝑝* (6.5)

Equation for the calculation of the recommendations bias in combining outputs in ensembled models *(self-composed)*

*𝐵𝑐,𝑝* - Current bias of a chosen recommendations pipeline

*𝑏𝑙,𝑝* - Last applied user bias for the chosen recommendations pipeline. This can be 0 or null

*𝐵𝑤,𝑝* - Default bias of a chosen recommendations pipeline

*𝑏𝑎,𝑝* - Admin suggested bias of a chosen recommendations pipeline

The above bias will be applied only to users who haven’t chosen a preferred bias. It can be applied to users who have chosen the bias as well, but it is suggested to be applied after initially showing recommendations to the user using their requested bias.

#### 6.4.4 UI Design

UI wireframes will be designed and added before implementing the UI of the MVP (Minimum Viable Product) that will be created over the following weeks. Since the core research component didn’t require a UI, this design was not necessary for this submission.

#### 6.4.5 System Process Flow Chart

The algorithm’s flow and decision structures are depicted in the flowchart below. It explains a significant proportion of the system since the expected implementation is primarily procedural.

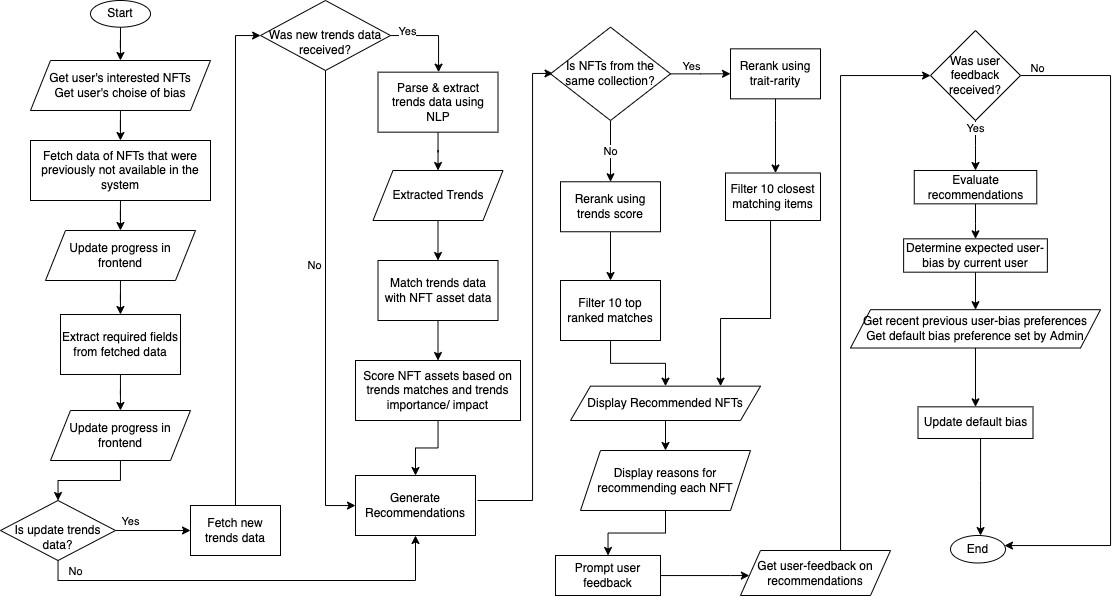


Figure 6.4: System Process Flow Chart*(self-composed)*

The process followed to generate Trend-based Recommendations has been displayed in a simplified process flow diagram in *Trends based Recommendations Process Flowchart (selfcomposed)* of *Appendix C - Design*.

## 6.5 Chapter Summary

The design, architectural aspects, and the flow of the project and novel author-designed algorithms were documented in this chapter followed by the expected UI wireframes to be implemented for the end-users interaction with the system.

# CHAPTER 7: IMPLEMENTATION

## 7.1 Chapter Overview

This chapter explains the core implementation of the research prototype together with the technologies, languages & supporting tools used for development of the prototype, with reasoning to the choice of each selection.

## 7.2 Technology Selection

### 7.2.1 Technology Stack

The technologies that were used to implement the prototype at each layer are shown below.

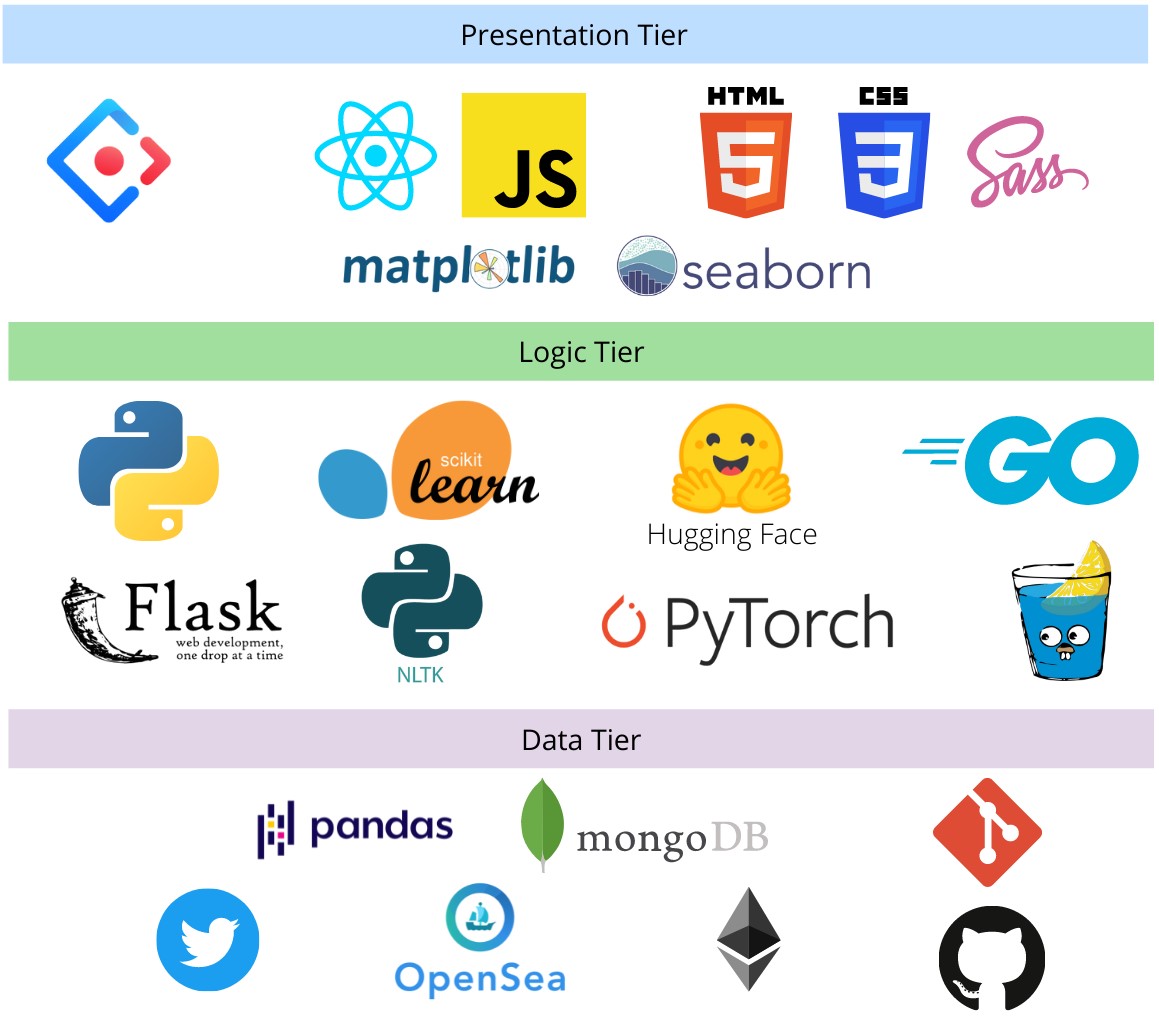


Figure 7.1: Technology Stack *(self-composed)*

**Linux** will be the default choice for development since of the ease of support for multiple development tools and performance benefits. MacOS/ Windows will be used for research documentation & study purposes.

The rest of the choices in the above tech-stack have been explained in the following sections.

### 7.2.2 Data Selection

Being a data science project at the core, it was important to choose the best possible sources of data to gather sufficient data for analysis & produce the best possible recommendations. The data requirements identified were,

1. NFT asset data
2. Global trends data
3. NFT Smart Contract data
4. NFT events (sales) data
5. NFT bids data

Since the main technological research gap to be addressed was with the integration of global trends into content based recommendations, this was given a higher priority at first. These data requirements were sourced from the following sources and heavily pre-processed there after to create a usable dataset for data analysis.

* NFT asset, events, bids data - From the **OpenSea API**.
* Global trends data

**–** Twitter data - From **Twitter developer API**.

* Ethereum Smart Contract data - From Etherscan & OpenSea

All the data-points that could be used for recommendations and explored with iterative development, as a research. This iterative process took a long time since the APIs were rate limited. The gathered pre-processed datasets will be made available for public use for future researches.

### 7.2.3 Selection of development framework

Table 7.1: Selection of development framework

|  |  |
| --- | --- |
| **Framework** | **Justification for selection** |
| Gin Gonic | It’s extremely convenient to build APIs using Gin with Golang. It also has an easily debuggable log output & claims smashing performance (up to 40 times faster!) |
| Ant Design | The world’s second most popular React UI framework. Used in many industrial applications and has a wide range of components to match most UI requirements. Since it’s tree-shaking compatible, it will build only the components that are used. This reduces build time of the frontend. The CSS is easily customizable as well. |
| Flask | Easy to build APIs for Python. |

Although this is a data science project, all data science models utilized were built from scratch without the use of libraries, since doing so allowed the author to tweak the models at will.

### 7.2.4 Programming language

**Python** is the language that will be used to create the ML models. Python is an all-purpose language that has been used in many projects involving data science. It has a vast collection of supporting libraries that eases many data science related tasks.

For the API proxy it was decided to use **Golang**, which is statically typed language that attempts to resemble the performance of C. Golang will allow the application to support concurrency and multi-threaded communications while being extremely lightweight and fast. This will be used to avoid any bottlenecks that could occur at this point in the system, while potentially bolstering performance.

For the frontend, **JavaScript** was decided to be used to show dynamic content and allow a highly interactible & inviting user experience.

### 7.2.5 Libraries Utilized

Table 7.2: Libraries Utilized with justification for choices

|  |  |
| --- | --- |
| **Library** | **Justification for selection** |
| Pandas | Pandas dataframes allow a vast range of functionalities required for data analysis such as cleaning, transforming, filtering, sorting & manipulating of data |
| Scikit-learn | Used for vectorizing text and generate similarity matrices between items, for recommendations. |
| NLTK | Convenient to use for NLP data parsing, using the RAKE vectorizer. |
| Hugging Face  Transformers | Availability of pre-built high performance Open-source NLP Transformer models. A model that was built using Pytorch was chosen due to it’s speed. |
| Matplotlib &  Seaborn | Has almost any type of visualization method for data analysis. |
| React | A UI library that makes it easy to build interactive websites. It was important to develop an easily interactible frontend, since it will be the users’ point of interaction with the system. This was easily doable thanks to the vast array of capabilities offered by React. |

### 7.2.6 IDE’s Utilized

Table 7.3: IDEs Utilized with justification for choices

|  |  |
| --- | --- |
| **IDE** | **Justification for selection** |
| Google Colab | Convenience of trial & error of fetching data, building, testing ML models and ability to work across multiple devices with the cloud development environment. |
| VSCode | Extremely dynamic while being simple to use, yet powerful for front-end development with it’s extensions & code snippets. |
| Golang | Convenient syntax highlighting & auto-completion for Golang development. |

### 7.2.7 Summary of Technology selection

Table 7.4: Summary of Technology selection

|  |  |
| --- | --- |
| **Component** | **Tools** |
| Programming Languages | Python, Golang, JavaScript |
| Development Framework | Gin Gonic, Flask |
| UI Framework | Ant Design of React |
| Libraries | Pandas, Scikit-learn, NLTK, Matplotlib, React |
| IDE – Research | Google Colab |
| IDE – Product | VSCode, Golang |
| Version Control | Git, GitHub |
| Application hosting | Netlify, AWS |

## 7.3 Implementation of Core Functionalities

Since a Recommendations System’s ultimate goal is to reduce the amount of information overload and provide the user with the best possible options, it was essential to build a dataset to suit the expected requirements. Just throwing in all the data fetched from APIs into a DL wouldn’t give an expected successful recommendation. Therefore, the fetched data was heavily preprocessed.

### NFT Data Mining

Continuously being able to add new NFTs or even adding an initial set of NFTs should be possible in the system for users’ convenience. When doing so, we need to make sure that relevant information is extracted.

The data extraction is done to extract information required for recommendations, to view

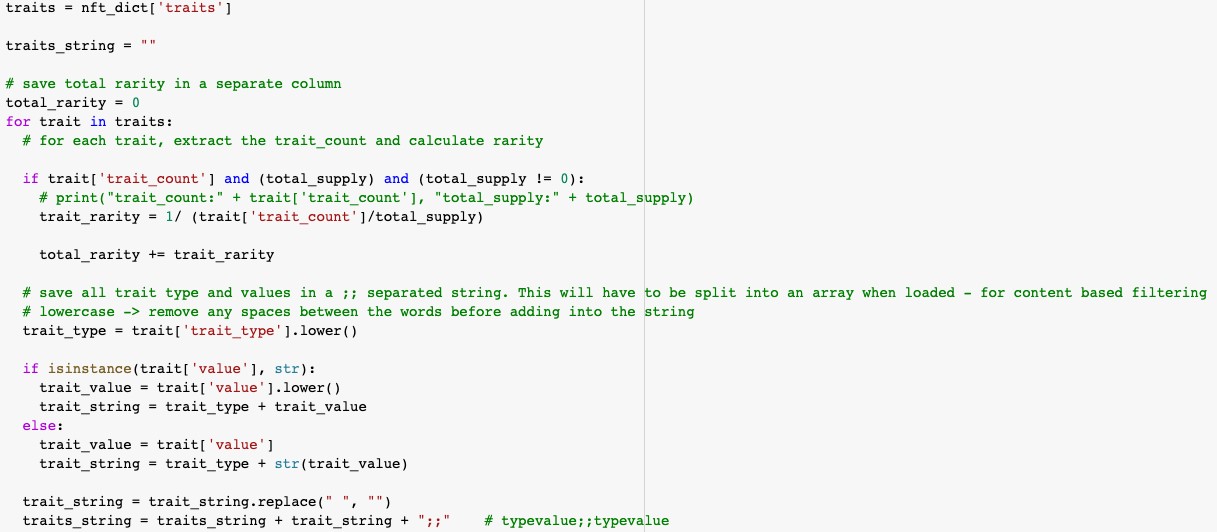


Figure 7.2: Implementation code segment: NFT data mining & preprocessing *(self-composed)* details of items & to save information for recommendation algorithms/ predictions that are potentially possible in the future.

### Trait Content & Rarity Preprocessing, Vectorizing & Recommendations

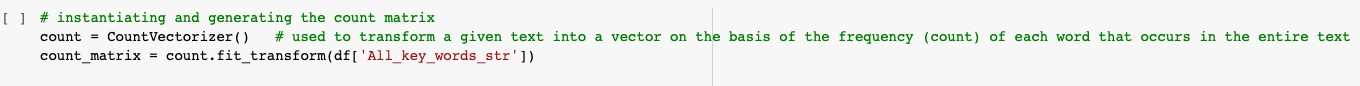


Figure 7.3: Implementation code segment: Content Vectorizer *(self-composed)*

A Count Vectorizer was used from the *scikit learn* library to vectorize all words, to be used for similarity matching. The reason for choosing the Count Vectorizer over a Tf-Idf Vectorizer was because Tf-Idf will give lower scores to more common words found in the dataset. Since our intent is to identify all the possible matches and primarily rank the content based results using global trends, it made more sense to go with a Count Vectorizer.



Figure 7.4: Implementation code segment: Generating the Cosine Similarity Matrix *(selfcomposed)*

A Cosine Similarity Matrix is then generated from the *scikit learn* library to identify all the matching words contained across all NFTs content. This generates the recommendation ahead of time.

The recommendation generation algorithms in Fig 7.5 were created to cater towards matching NFTs within a collection, since most of the major NFT-collections have comparatively more unique data in traits compared to descriptions. Trait rarity similarity was identified to be the best

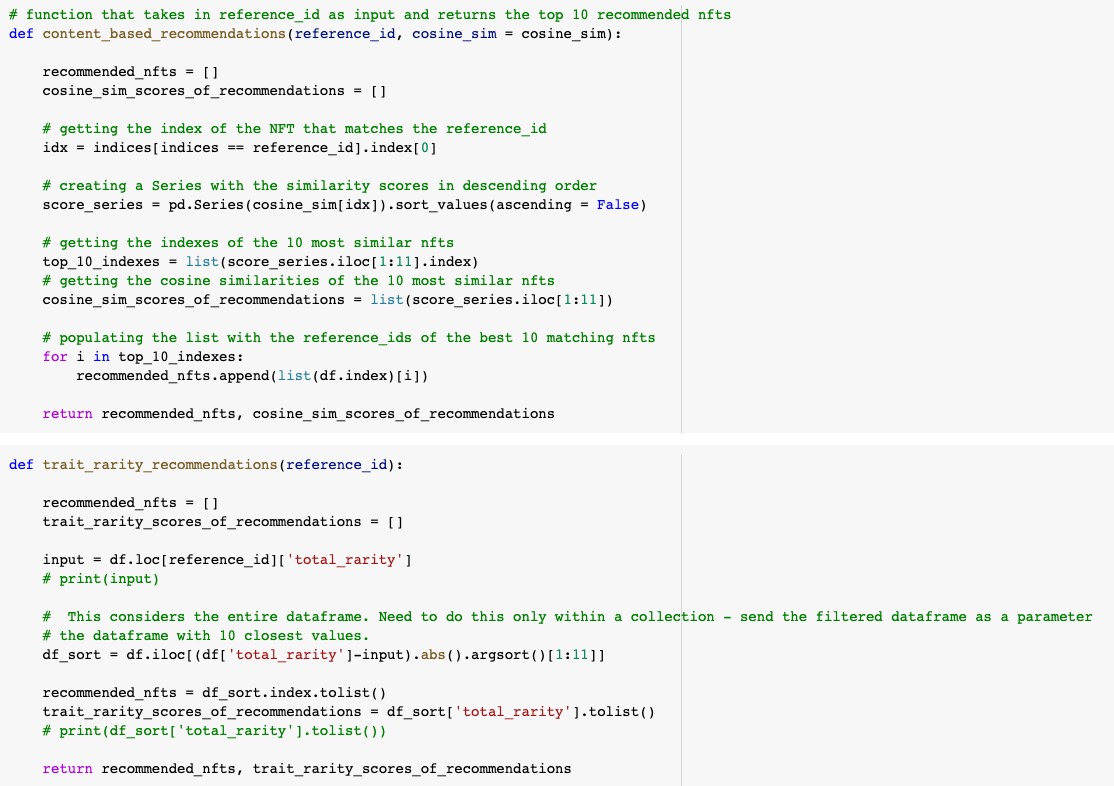


Figure 7.5: Implementation code segment: Generate Trait Rarity & Similarity Recommendations *(self-composed)*

way to identify total uniqueness which represents the value of each NFT. Although the calculation of total rarity was explored by *rarity tools* during the course of the research (rarity.tools, 2021a; rarity.tools, 2021b), recommending similar total rarities is a novel implementation in the application domain.

### Trends Sentiment Analysis, Preprocessing & Recommendations

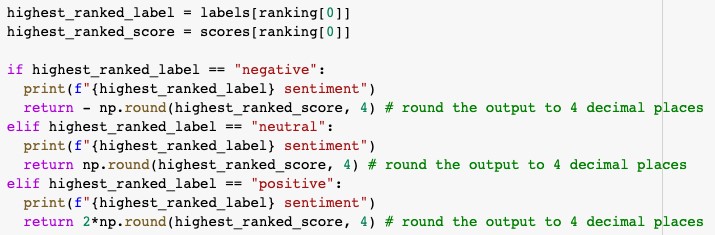


Figure 7.6: Implementation code segment: Tweet Sentiment Analysis *(self-composed)*

The output of the Sentiment analysis model was returned as negative for negative sentiment, as it is for neutral sentiment & multiplied by 2 for positive sentiment to make an impact on the rankings of the output produced.

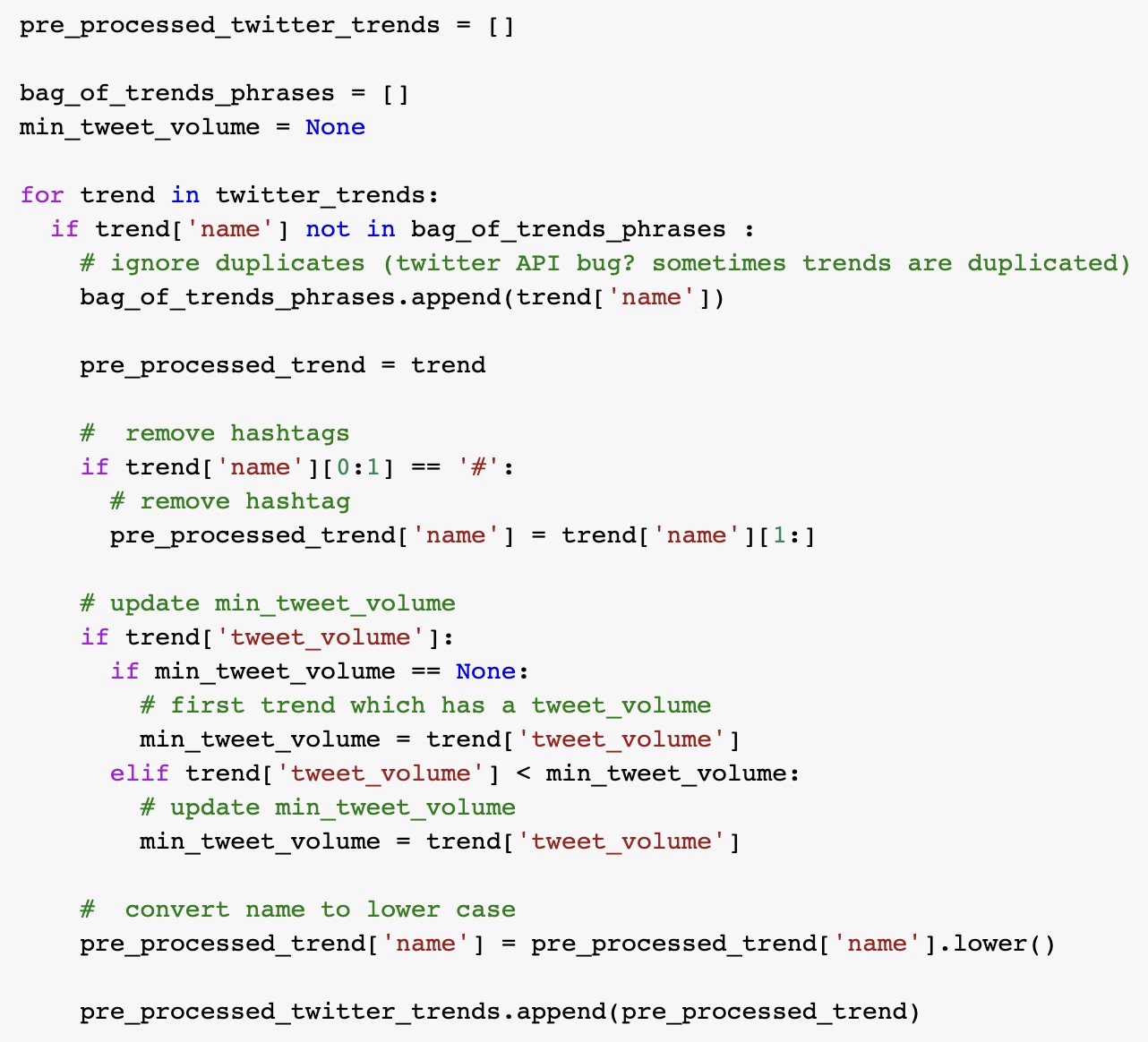


Figure 7.7: Implementation code segment: Preprocess Trends Data *(self-composed)*

The code segment in Fig 7.7 preprcoesses trends that are fetched from the live Twitter API.

The code segment in Fig 4 of *Appendix D - Implementation* assigns a tweet volume for trends with no volume & calculates the median Tweet volume which used to calculate the impact score of each trend.

The code segment in Fig 7.8 is used to calculate the trends score for each NFT and finally make trends-based recommendations.

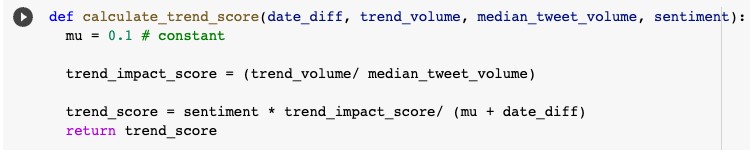


Figure 7.8: Implementation code segment: Calculating Trends Score *(self-composed)*

## 7.4 Testing & Evaluation Code of Models

All code snippets that were created for Testing & Evaluation purposes have been mentioned under *Appendix D - Implementation*



Figure 7.9: Implementation code segment: Calculating All Trends Scores *(self-composed)*

## 7.5 User Interface

The UI wireframes depicting the planned UI for the MVP (Minimum Viable Product) have been place in *Appendix D - Implementation*.

## 7.6 Chapter Summary

The chapter comprised of the technologies, languages & supporting tools utilized to implement the prototype developed as part of the research. Discussions accompany the code snippets and algorithms produced as part of core functionality. the UIs expectable in the Minimum Viable Porduct of the project have been presented.