UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

COS301 Capstone Project

Team Syntactic Sugar

Jargon Sentiment Analysis
Software Requirements Specification

Team Members
Graeme Coetzee
Christiaan Nel
Ethan Lindeman
Kevin Coetzee
Herbert Magaya

Client
Compiax

October 7, 2019

Contat email: syntacticsugar9@gmail.com

Contents

List of Figures

1 Introduction

1.1 Purpose

The purpose of this document is to present a detailed description of the Jargon Sentiment Analysis System. It will explain the purpose and features of the system, the interfaces of the system, what the system will do, the constraints under which it must operate and how the system will react to external stimuli. This document is intended for both the stakeholders and the developers of the system and will be proposed to the client, COMPIAX, for its approval.

1.2 Scope

The Jargon Sentiment Analysis system is a system enabling users to create projects specifying which topics to search for, and other additional configurations such as blacklisted or white-listed words. When users choose to run these projects, they start collecting information from a public source of opinion based data, such as Twitter. The system then analyzes this data, utilizing an Artificial Neural Network, classifying it into sentiment ratings (positive to negative). Users will then be able to interpret this result as a final opinion overview, as well as view more detailed information regarding the original data-set.

1.3 Definitions, Acronyms, and Abbreviations

NN - Artificial Neural Network, this refers to a processing framework that roughly resembles the workings of the human brain. It can utilize different different machine learning algorithms to process complex data inputs.[1] In the Jargon Sentiment Analysis system, this is what primarily will be used to analyse and classify sentiments.

2 Information Flow

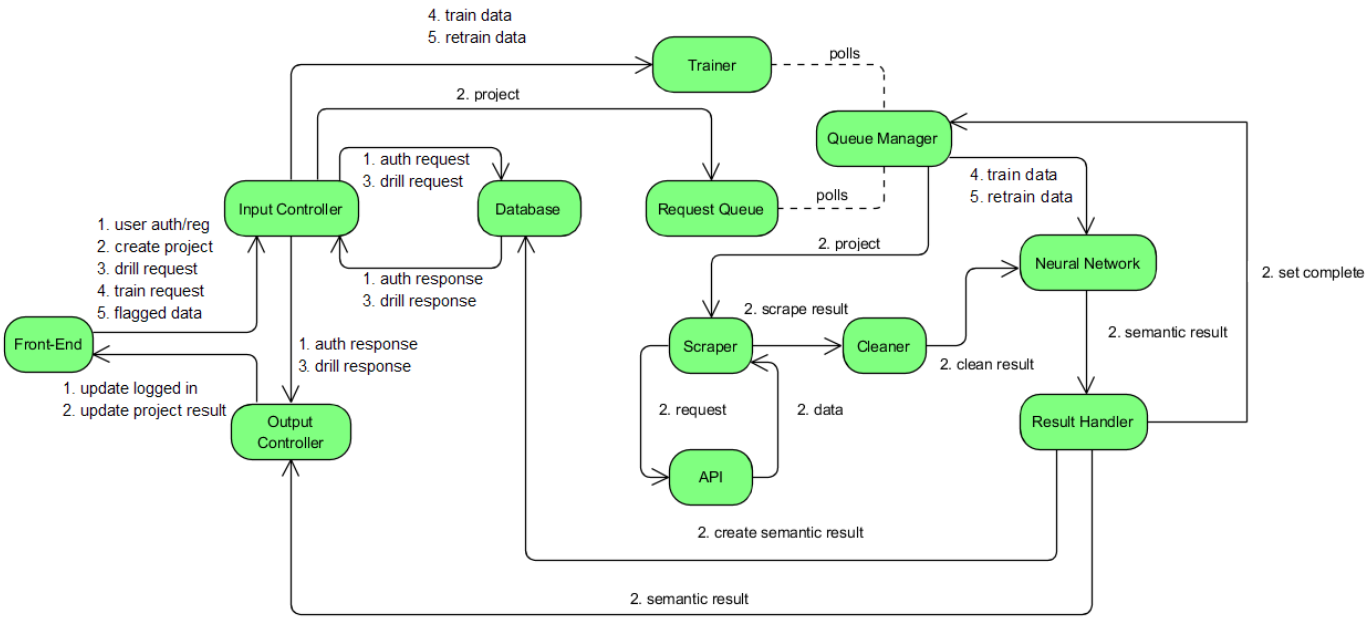

Figure 1: High-Level Information Flow Diagram

3 Domain Model

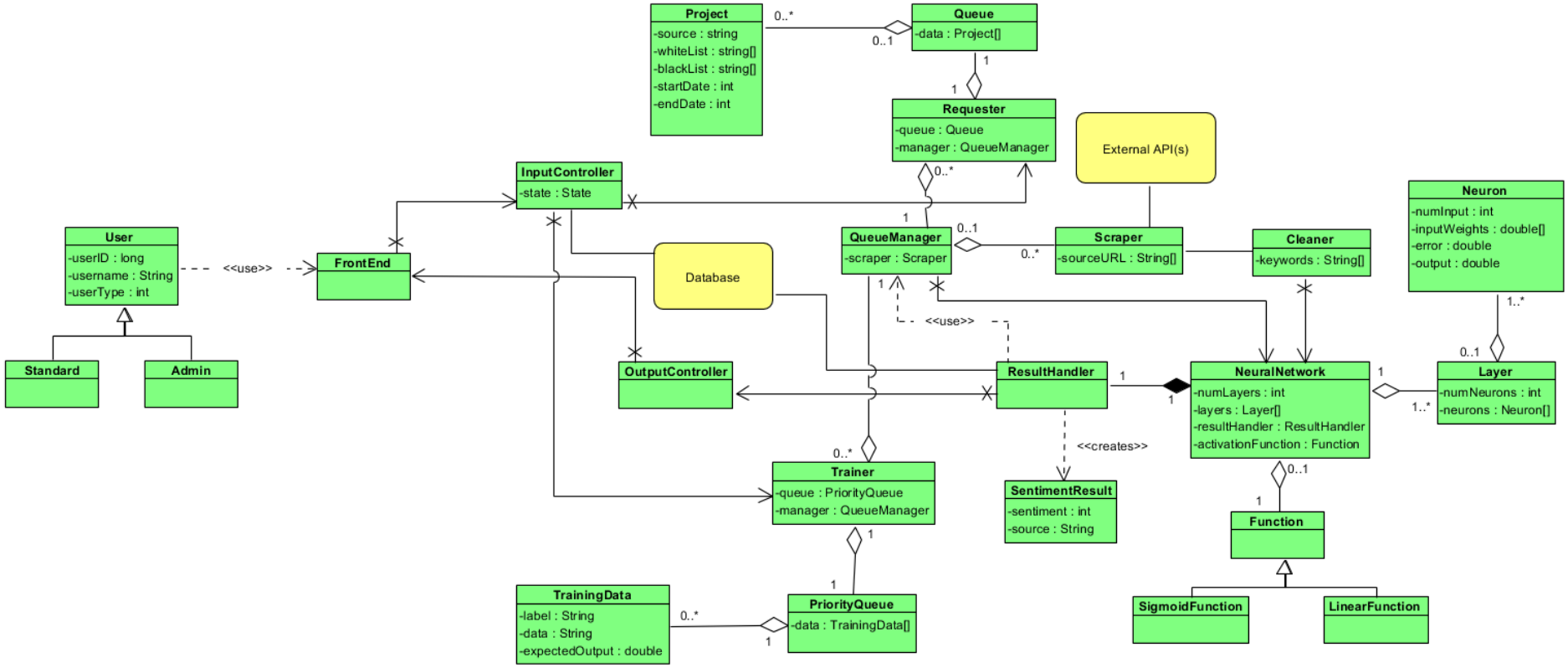

Figure 2: High-Level Domain Model

4 User Characteristics

- Researchers - This sentiment analysis system will at its core be a research tool for determining public opinion on different topics, and therefor the core users will be educated, professional researchers working for different companies and corporations, especially market researchers. These users will use the system to create projects and gauge opinions based on the topics and configurations chosen in these projects, or aid their management staff and higher-ups by setting up and running research projects, leaving the result interpretation up to their management staff.
- Management - Management or high-up personnel will use the end results the system produces as useful information in decision making on matters concerning brand management and product development. Their main use case for the system will be the interpretation of the system's result sets to make educated decisions.
- Administrators - Administrative users will include members of COMPIAX and the Syntactic Sugar team. These users are more knowledgeable on the topics of NNs, and will additionally make use of the system to retrain the NN with new or revised data.

We may refer to *researchers* and *management users* as *regular users*.

5 Architectural Design

The Jargon project shall be implemented using the **Microservices** architectural style.

5.1 Microservices

5.1.1 Definition

Microservices is an architectural pattern that consists of abstracting each business function of your system into it's own separate service/application. This server can be run on an entirely different server if need be. Services interact with one another through the use of TCP/IP and HTTP requests. This means that each service will have it's own API.

5.1.2 How does this differ from Service-Oriented Architecture?

Simply put, Microservices are application specific and are a higher level of granularity compared to Service-Oriented Architecture. Service-Oriented Architecture considers how services are structured and interact on an *enterprise* level whereas Microservices is more focused on how different services and functions within a single application are structured and interacting.

5.2 Benefits of Microservices

There are many benefits to using the Microservices architecture, to list a few:

- **Better scalability:** Since each microservice runs independently, it is possible to scale a single service as we want without affecting the entire system in the process.
- **Better failure tolerance:** The fact that each service is isolated means that if one service were to fail it would not take down the entire system in the process.
- **Easier to debug/fix:** Since each service is independent from other services and is representative of a particular function of the system, if there is an error in the logic of the system with respect to that function, then we can easily inspect just that service alone in order to find the bug. Thus we have narrowed our window for failure.
- **Easier to deploy:** It is possible to containerize each service into its own Docker image, allowing you to easily deploy to any server as you wish by just installing Docker and then running your Docker-Compose file. Docker also integrates well with Kubernetes in order to orchestrate your containers and scale the services as you wish.
- **Language agnostic services:** Since services communicate with each other via HTTP and their respective APIs, this means that each service can be written in whatever language the developer would prefer, as it only needs to interface with the API. This allows for a large amount of flexibility in development and means each service can be fully optimized using the best language that supports its function.
- **Large increase in development speed:** Microservices lends itself to being really easy to extend and scale, due to the isolation of its services this means that an organization can place teams on separate services and only worry about integrating them once development has been completed. This vastly improves productivity of developers since there's no dependencies to worry about when developing your particular service. This is a large reason as to why Microservices has been so widely adopted in modern industry.

- **Isolated resources:** Since each service is isolated, it can run on its own independent server as well. This means that we can have a distributed network where each node on the network runs one service, meaning that each service can have its own resource pool to utilize, rather than needing to fight with other services for resource utilization.

5.3 Challenges of Microservices

Despite the various benefits, we should still consider the challenges. To list them:

- **Additional infrastructure required:** Due to the vast size of the solutions created with this architectural style, the testing, review and deployment cycle is quite complicated. This means that an automated testing/deployment framework/tool is required in order to optimize this process.
- **HTTP communication may be unsafe:** Since each microservice communicates with TCP/IP and HTTP, this leaves room for external forces to attempt to break the system or steal data. Thus developers need to ensure that all data is sent across a secure and encrypted channel.
- **Slight performance drop:** This is due to microservices needing to make requests across a network rather than directly to a sub-module if it were monolithic.
- **Higher collaboration among developers:** Due to the nature of the system design, each team needs to have higher collaboration with other teams in order to appropriately develop their services to interact correctly. This can be a large overhead in large corporations.

5.4 Our Reason for Microservices

5.4.1 Jargon System Design Goals

The Jargon Sentiment Analysis system should be:

- Easily extendable for future development by Compiax.
- Scalable to prevent bottlenecking in the neural network analysis.
- Able to integrate with the python-based Neural Network.
- Deployable to Compiax-based servers.
- Able to function despite server faults.

5.4.2 Justification

Meeting requirements
From the above mentioned requirements it's clear as to why we think microservices is the architectural style best suited to our system. It solves all of the above-mentioned requirements. Isolated services allow us to create scalable, language agnostic functions, that can be containerized for easy deployment and extended in a later period of time.
Client satisfaction
Our client, Compiax, has requested that we make use of a microservices architecture as they currently use this architectural style in their development and thus it would make it easier to integrate with their current systems.

End-user satisfaction
We believe that in using microservices, it will allow us to properly optimize our system and extend it such that we can provide the best User Experience to our end-users. Scalability will vastly improve the responsiveness of our system and allow for faults to be virtually unnoticeable. Downtime will also not be noticed through the use of a continuous deployment methodology.
Team satisfaction
Last but not least, we believe that in implementing a microservices architecture, it will prove to be an amazing learning opportunity for our group as a whole, and further prepare us for working in industry. This is largely due to the fact that microservices is a fairly complicated architectural style and is also currently one of the leading styles utilized in industry. w

5.4.3 Countering the challenges

Of course, we didn't only consider the benefits of microservices when making our decision, but also the challenges, and how detrimental they would be to our system, and how we could counter them. The following is how we plan to counter the above-mentioned challenges in section 5.3.

- Our system makes use of Travis CI in order to automate the testing process. We also plan to use Docker to containerize our application and will host it on Google Cloud, where each new Docker build will be pushed to the Cloud server, thus providing us with continuous deployment.
- Sensitive data shall be encrypted before being sent across microservices, we will most likely make use of the HTTPS protocol when making requests.
- The perfomance drop from network communication is negligible to our system.
- Since we are not a large organization made out of a variety of differing teams, we do not face a problem when it comes to collaboration. Collaboration in our team is also further improved through the use of project management tools such as Trello and Slack.

6 Deployment Model

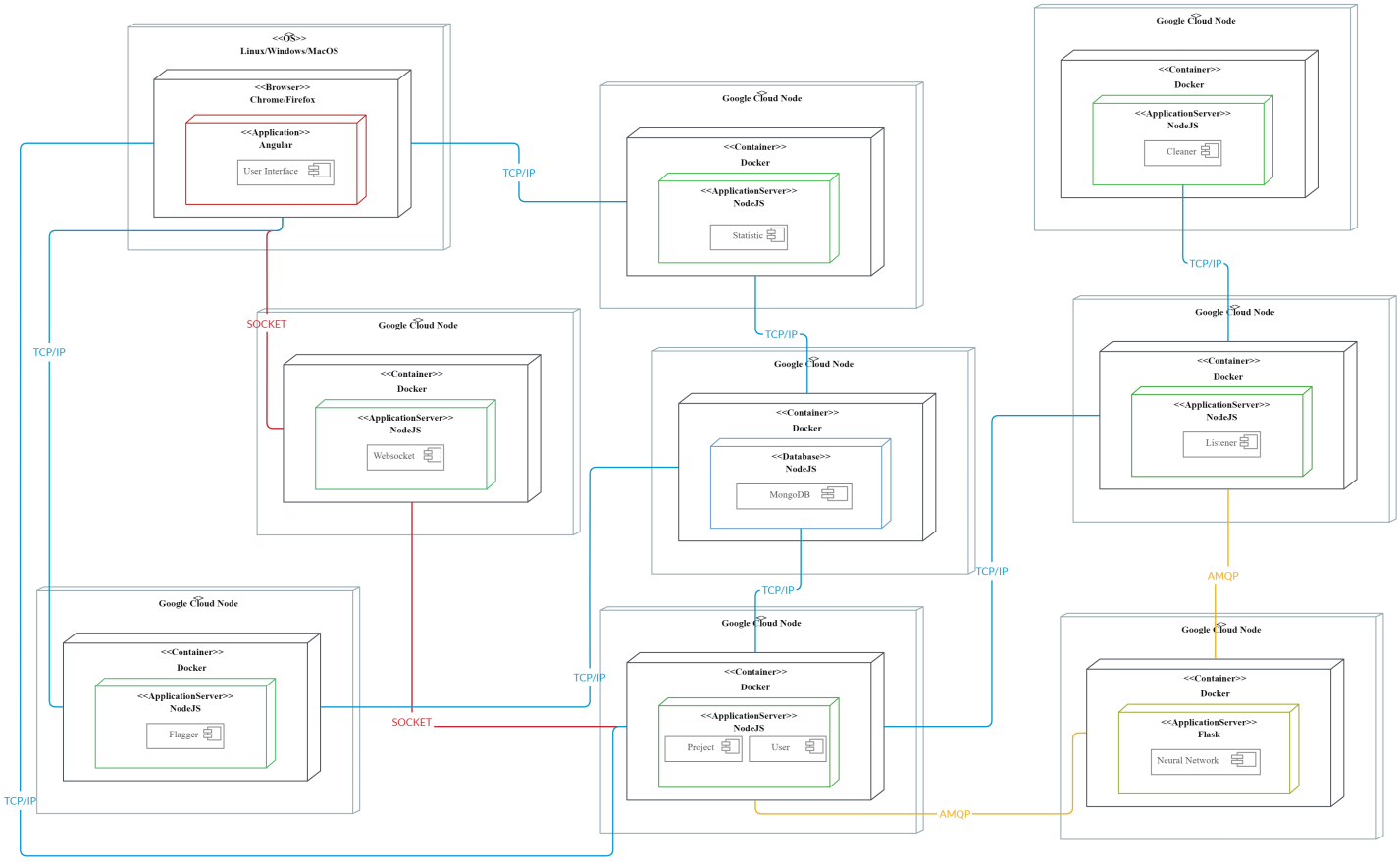

Figure 3: Deployment model

7 Functional Requirements

R1. Users should be able to interact with the system through a web interface frontend.
R2. Users should be able to register/delete their account.
- **R2.1.** Users should be able to edit/update their account details.
- **R2.2.** Regular users should be able to delete their own account.
- **R2.3.** Admin users should be able to delete any regular user's account.

R3. Users should be able to log in and out to the application.
R4. The system should restrict further application functionality to logged-in users.
R5. Users should be able to create multiple projects and configurations.
- **R5.1.** Users should be able to queue projects.
- **R5.2.** Configurations should allow the user to specify options to change the brand to search on.

R5.3. Configurations should allow the user to specify options to determine when to start/stop streaming data and words that are whitelisted or blacklisted.

R6. Users should be able to compare completed projects.

R7. Users should be able to search for completed and in porgress projects.

R8. The system should be able to stream data (needed by a project) in real-time from a source such as Twitter.

R9. The system should be able to perform sentiment analysis on the data (of a project).

R9.1. The sentiment analysis subsystem should give the sentiment result of a project a rating that can be either positive, negative or neutral.

R9.2. The sentiment analysis subsystem should make use of a neural network to perform the sentiment analysis.

R9.3. The sentiment analysis subsystem neural network model should be able to retrain itself.

R9.4. The sentiment analysis subsystem neural network model should be able to be manually trained/trained by an admin user.

R9.5. Admin users should be able to provide the sentiment analysis subsystem with training data, which the neural network will use to train itself.

R9.6. Users should be able to provide the sentiment analysis subsystem with training data by means of flagging/modifying the results of a project.

R10. Each subsystem should be able to log its activities.

R10.1. Regular users should be able to view logs of activities, directly related to their operations.

R10.2. Admin users should be able to view activity logs of both the entire system in addition to those directly related to their operations.

R10.3. Users should be able to view logs recorded between any valid given timeframe.

R10.4. Users should be able to backup logs.

R10.5. Users should be able to download log backup files.

R10.5.1. Users should be able to download log backup files to the device which they are interfacing through.

R10.5.2. Users should be able to download log backup files to a cloud storage service of their choosing.

8 Use Cases

- UC1.1 Register Account (Actor: User, Subsystem: Accounts)
- UC1.2 Login (Actor: User, Subsystem: Accounts)
- UC1.3 Logout (Actor: User, Subsystem: Accounts)
- UC1.4 Edit Account Details (Actor: User, Subsystem: Accounts)
- UC1.5 Remove User (Actor: Admin, Subsystem: Accounts)
- UC2.1 Train Neural Network (Actor: Admin, Subsystem: Neural Network)
- UC2.2 Analyse Data (Actor: System, Subsystem: Neural Network)
- UC2.3 Flag Items (Actor: User, Subsystem: Neural Network)
- UC2.4 Retrain Network (Actor: User, Subsystem: Neural Network)
- UC3.1 Create Project (Actor: User, Subsystem: Project)
- UC3.2 Edit Project (Actor: User, Subsystem: Project)
- UC3.3 Delete Project (Actor: User, Subsystem: Project)
- UC3.4 View Project Results (Actor: User, Subsystem: Project)
- UC3.5 Drill Down Results (Actor: User, Subsystem: Project)

- UC3.6 Compare Projects (Actor: User, Subsystem: Project)
- UC3.7 Search Project (Actor: User, Subsystem: Project)
- UC4.1 Create Log (Actor: System, Subsystem: Logs)
- UC4.2 View Logs (Actor: Admin, Subsystem: Logs)
- UC5.1 Submit Error Report (Actor: User, Subsystem: Support)
- UC5.2 Resolve Error Report (Actor: Admin, Subsystem: Support)
- UC5.3 Submit Bug Report (Actor: User, Subsystem: Support)

9 Subsystems

- Project subsystem
- Accounts subsystem
- Logs subsystem
- Neural Network subsystem
- Support subsystem

Figure 4: Use case diagrams

10 Quality Requirements

Performance

Q1 System response time should be limited to within 3 seconds of a user action. Frequent response delays will be logged and any discrepancies in response time will be analysed and fixed.
Justification: User experience is crucial to the app as it is intended for use by Market Researchers.

Security

Q2 The system will be implemented with access control. Users can only view the projects of other users, only admins have full read/write access. This will be done through the use of third party software, additionally, users may submit error reports in the case of any unexpected behaviour.

Justification: The system may potentially have rather sensitive information for each user and we would like to protect their privacy.

Q3 Each user will have a maximum of 3 failed login attempts before their account is temporarily suspended. More than 10 failed attempts will result in a frozen account.

Justification: This is a measure to prevent unauthorized access to a user's account. Further providing information privacy. We believe that no user would find our platform appealing if they cannot trust us with their data.

Safety

Q4 Logs need to be kept in order to keep track of system actions performed in the case of a system breach. Data will be periodically backed up every 2 weeks to prevent data loss. Logs will be kept for a maximum of 1 year before being archived to improve server drive usage.

Justification: This is required in order for us to improve our user experience by identifying the most utilized system functionality. It also provide a great way of finding faults.

Availability

Q5 The neural network needs to have a downtime of less than 1%. Downtime will be tracked and if it's above 1% then optimization measures will be put into place.

Justification: If the network were to go down during analysis, it could potentially corrupt the dataset or analysis, thus it is vital that we prevent this from occurring.

Q6 The system downtime is measured and the number of failures per month will be tracked. These will be monitored daily and we aim to have less than 0.1% risk of downtime.

Justification: Any risks of downtime are a risk to our entire system, thus they must be identified as soon as possible and rectified.

Q7 We aim to have zero downtime when adding a new server to the system as well as when deploying new code. This shall be done using Kubernetes and a continuous deployment tool.

Justification: When scaling our system to accommodate fluctuating requests, we want the process to be seamless, we cannot afford to stop any currently running analysis as it could potentially corrupt the entire data set.

Reliability

Q8 The neural network must be trained so as not to produce too many incorrect results. Users will have the ability to flag incorrect results in order to retrain the network. A count of these incorrect results will be kept in order for the system admin to notice any discrepancies in the network. We aim to have a failure rate of less than 3%.

Justification: The core function of our system is to provide a sentiment analysis, thus if we do not make it produce reliable results, we have made a less than desired system. This quality requirement is *vital* for our system.

Maintainability

Q9 The system must run over multiple servers to prevent one central point of failure. Server failures should be monitored and audit logs reviewed in the case of a failure. We aim to have a failure rate of lower than 1%.

Justification: System failure is not an option, therefore we want to do as much as we can to mitigate or prevent it.

Q10 The codebase will be modularized into roughly 4-5 different microservices to further allow for easier extension and maintenance. Bug reports may be submitted by users and we aim to have fewer than 1 bug report per month.

Justification: Compiax would like to utilize this project for the market researchers of Consulta, thus we find it imperative that we make the system as easily extendable and bug free as possible.

Usability

Q11 The system UI needs to be designed taking UX into account. User experience surveys shall be issued to the users in order to determine the overall experience. We aim to have 95% user satisfaction.

Justification: Our end-users are very important to us, and if they do not want to use our system, that reflects poorly on both us and our client, Compiax. Therefore we wish to make the user experience as polished as possible.

Q12 The system needs to be intuitive to use and self-explaining. Usability tests shall be issued to users in order to determine where the system needs improvement. We aim to have 100% task completion in the usability tests.

Justification: Our system needs to be easily understood and usable so as to retain our end-users.

Scalability

Q13 The system needs to be able to run at least 5 or more Twitter listeners in parallel for different projects. We aim to have zero bottlenecking into the neural network. This shall be done using Kubernetes and Docker.

Justification: Our system needs to be fast, efficient and perform well. This is because it ties directly into the user experience and end-user retension. Our client Compiax also we requested that we optimize the system as much as possible.

11 Constraints

11.1 System Constraints

- Copyright and Privacy laws dictate that the system shall only collect and store personal information that is required for the minimum functioning of the system. No further information may be collected from the users of the system without explicit consent from the user.
- The system must make use of a Neural Network or some form of intelligent agent approach in order to perform the sentiment analysis.
- The system is currently limited to the Twitter API request limit with respect to data aggregation limits.
- Currently the only social platform that our system supports for sentiment analysis is Twitter.

11.2 Hardware Constraints

It should be noted that these minimum hardware constraints are with respect to the server-side devices.

- Stable internet connection
- Mid-level CPU
- Mid-level GPU
- Cloud server
- Storage
- 4GB RAM

12 Technology

The following techonologies will be/are utilized in the development and function of the Jargon Sentiment Analysis system:

- MongoDB database.
- Angular 7 for frontend web application.
- Python and PyTorch for the Neural Network.
- Docker and Kubernetes for containerization and orchestration respectively.
- Mocha and Chai for Unit and Integration testing.
- NodeJS for the servers and APIs.
- JavaScript for the Listeners.

13 Trace-ability Matrix

	Project subsystem	Accounts subsystem	Logs subsystem	Neural Network subsystem	Support subsystem
R.1.					X
R.2.		X			
R.2.1.		X			
R.2.2.		X			
R.2.3.		X			
R.3.		X			
R.4.		X			
R.5.	X				
R.5.1.	X				
R.5.2.	X				
R.5.3.	X				
R.6.	X				X
R.7.	X				X
R.8.				X	
R.9.				X	
R.9.1.				X	
R.9.2.				X	
R.9.3.				X	
R.9.4.		X		X	
R.9.5.		X		X	
R.9.6.				X	
R.10.			X		
R.10.1.		X	X		
R.10.2.		X	X		
R.10.3.			X		
R.10.4.			X		
R.10.5.			X		
R.10.5.1.			X		
R.10.5.2.			X		

Table 1: Traceability matrix mapping subsystems to functional requirements

	Project subsystem	Accounts subsystem	Logs subsystem	Neural Network subsystem	Support subsystem
Q1					X
Q2	X	X			X
Q3		X			
Q4			X		
Q5				X	
Q6			X		
Q7			X		X
Q8				X	
Q9			X		X
Q10			X		
Q11			X		X
Q12					X
Q13				X	

Table 2: Traceability matrix mapping subsystems to quality requirements