Architectural Thinking for Intelligent Systems Assignment 9
Team 11

1. Select an architectural style and apply it to your system?

   We plan to use Client/Server architectural style to develop our system. By default, any modern Intelligent Virtual Assistant system utilizes the power of centralized compute/data storage clusters to handle all the complex and large scale computational tasks and then the final results could easily be accessed/viewed through a rich client. In our system, we would use dedicated compute/data storage servers (GCP, AWS) to handle operations requested by the rich mobile client (Android/IOS).

2. What makes this style particularly appropriate for your purpose? List and discuss relevant constraints and forces that motivated your decision for this style?

   An Intelligent Assistant system running on a mobile device would often have restricted design and implementation issues like **limited Processing power (max 2.2 GHz on 4 cores)**, **memory (RAM)** and **data storage**. Hence, the client-server architecture for such system would be very apt and economical. Now, in the ages of **containerization** & **hybridized cloud infrastructures** (AWS), scalability would never be a problem too. The cloud (servers), provides a platform to Extract, Pre-process, Transform & Load massive amounts of data and also provides an option to train complex Natural Language Processing models on high performance GPUs which drastically reduces the training time of the system.

   The major **Constraints** that motivated our decision are:

   - **Data Processing & Storage:** As the training data becomes larger day by day, we won't be able to store everything in such small storage space and really tough to pre-process/condition the data before sending it to the learning model. Which invariably leads to a very heavy application which most of the people won't be able to use on their smartphones at all!

   - **Computational/Training Time:** Although there is a possibility of deploying deep learning models in the client device (smartphone) itself using 'TF Lite'. It is not recommendable as training a machine learning model on just 4 cores at a max of 2.3 GHz clock speed would lead to very long training time (typically days). And most of the times, these computations are often run of CPUs in a handled device (as GPUs are less powerful in a smartphone) which works in a serial order of instruction execution, leading to infinite wait times for other crucial processes.

The major **Forces** that motivated our decision are:

- **The training time of Deep Learning model should be low:** High performance GPUs in the compute clusters in the cloud (Servers) would be ideal to train our Deep Neural Networks fasters and also process massive amounts of data instead of running the same in a smartphone (client). Because of this reason, we have opted Client/Server architecture style

- **Wait times for the response must be low:** Whenever our system encounters new questions/data (not in the training samples), the model has to generalize accurately & quickly. User cannot wait more than 2-3 mins just for the system to reply back to 1 query. Thus, more computational power of the cloud (server) would result in quicker & better responses (hyperparameter tuning)

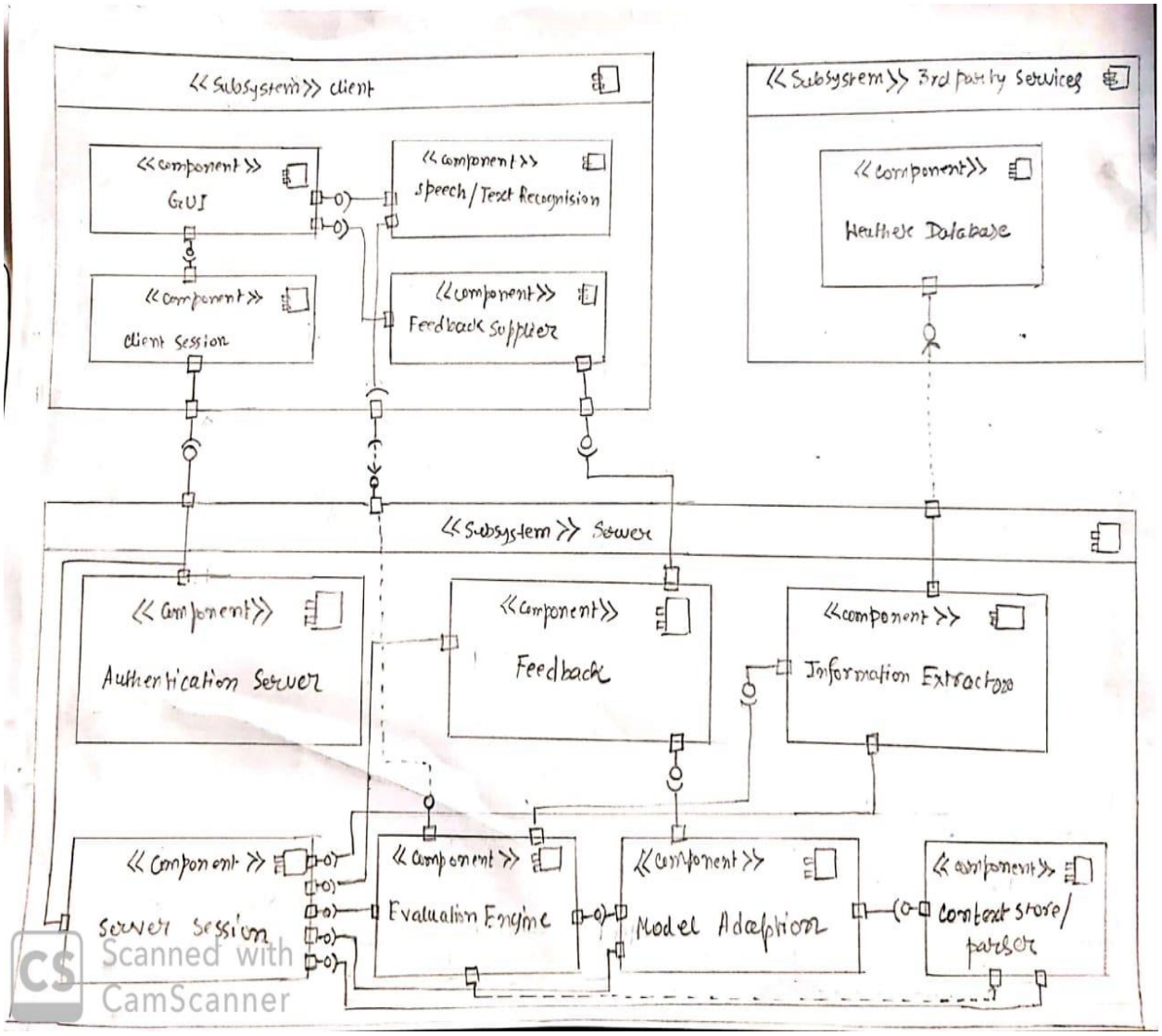
- **Keep implementation efforts low:** Because of the modularity of our system, it is easy to implement & delegate computationally intensive tasks like data pre-processing and model training to the cloud (server) and other less intensive tasks to the rich client in the smartphone (client)

- **System should adhere to privacy regulations:** Recently, this is becoming a major force while deciding the architecture of the system. We can implement strong security standards like **Secure Sockets Layer** (SSL) which establishes encrypted link between a server and a client, **Transport Layer Security** (TLS) etc. to prevent Data Leakages, Identity Theft and so on!

  Thus, Client/Server architecture performs better than any other architectural style for our system in terms of **Computational Times/Data Storage/Privacy**.

(Please turn the page over)

3. Create a package or component diagram to visualize the top-level architectural style of the system.



4. Refine and modularize the architecture by defining additional system structures and relationships at a second level of detail. Document how functional and non-functional requirements relate to the individual structures?

We have already modularized most of the functions in our system to achieve high cohesion. But, when we refine this at a second level of detail, we get the below refined modules in our system:

a. Speech/Text Recognition Engine
   - Text Recognition (sub modules)
   - Speech Recognition

b. Query Evaluation Engine

c. Context Store/Parser
d. Information Extractor
    - Textual Info Extractor
    - Audio Info Extractor
    - Visual Info Extractor
e. Model Adaption
f. Feedback Interface

Let's understand how Functional & Non-Functional requirements relate to the above modules/structures.

Text & Speech Recognition modules:

- **Functional:** The module should recognize & collect natural language input from user input
- **No Functional:** The word accuracy rate should be greater than 80%

Query Evaluation Engine:

- **Functional:** This module has to pre-process the data such as **Data Cleansing, Stop Word Removal, Stemming, Lemmatization** etc. Also process the input data into different Word Embedding
- **No Functional:** The processed strings should not have **any** Stop Words, Words with suffixes etc.

Context Store/Parser:

- **Functional:** Previous user inputs/queries to the system should be captured and stored
- **Non Functional:** Input from past 7 days should be stored

Text/Video/Audio Information Extractor:

- **Functional:** Based on the query requirement, retrieve corresponding information from 3rd party sources
- **Non Functional:** Requested information should be retrieved in real-time (perhaps less than 2 seconds)

Model Adaption:

- **Functional:** The Deep Neural Network/Machine Learning model should train on the set of user inputs/outputs collected throughout the time
- **Non Functional:** The model should have an accuracy/recall/precision of more than 75%

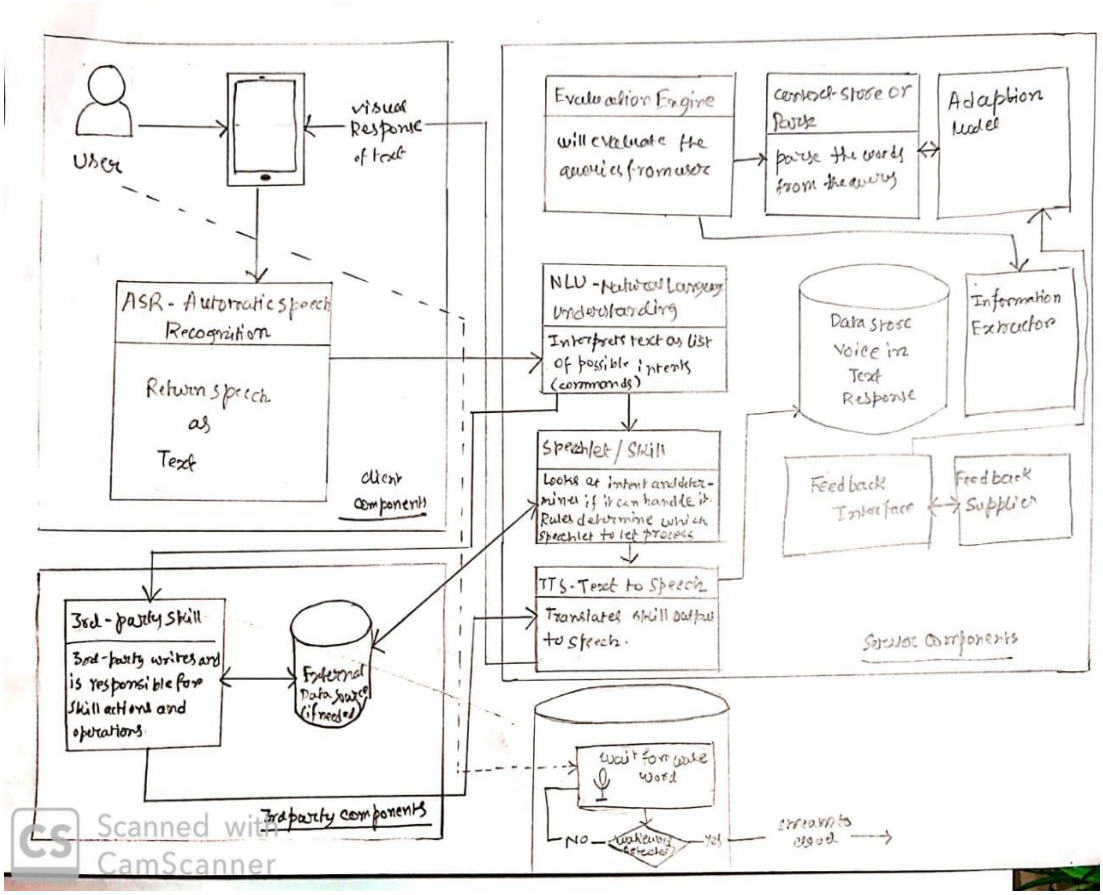
Feedback Interface:

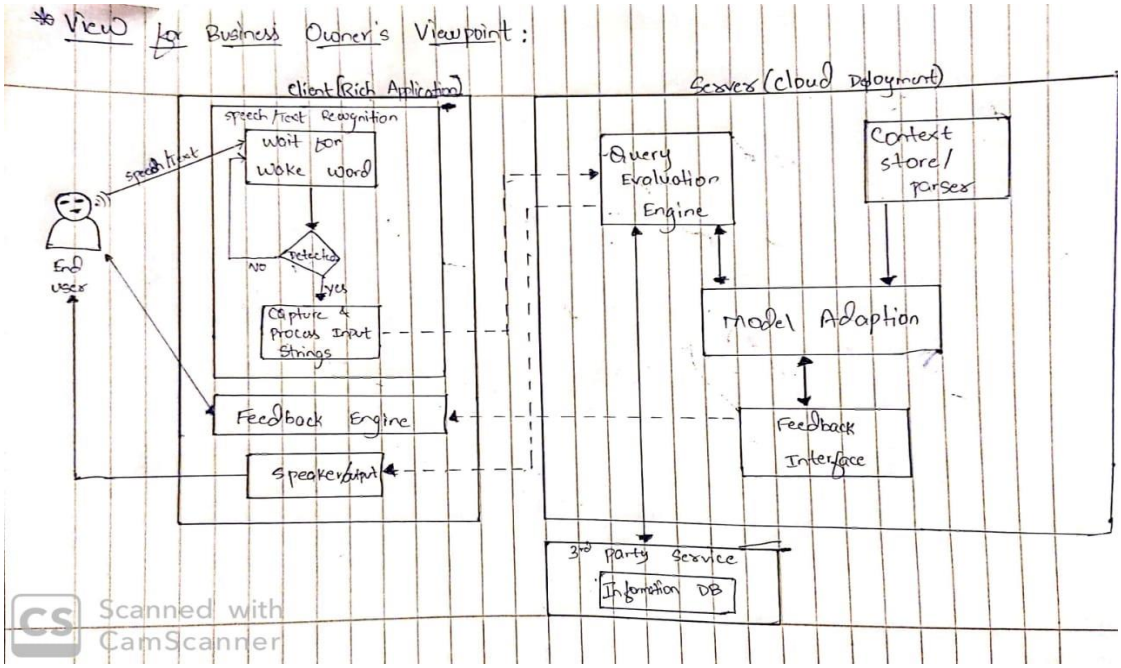
- **Functional:** The module should collect user/admins feedback on system's performance for a respective query
- **Non Functional:** The feedback must be accurately captured

5. Create two views to illustrate how the system architecture implements one specific functional and one non-functional requirement.

Functional Requirement relates to activation of application for taking user input: - The system will use only few particular wake words, such as "Alexa" or “Hey Google”. The system will have a call based wake word verification. The application will automatically activate its microphones without waiting for the wake word in multi-turn interactions and the system will not require the use of a wake word as part of the customer utterance while the customer will use it as touch initiated application and the microphone will be disabled until it listens to any wake word.

Non Functional requirement relates to how good the system performs the functionality implemented by the developer. For instance, the system should be able to wake up every time the wake word will be uttered. Like if the wake word is uttered 12th time, the system should be able to recognize wake words all the 12 times and should wake up all 12 times to serve the user.





6. Is there another architectural style for your system that would constitute a viable alternative? What makes this style inferior to the one you prefer?

   As per our understanding, '**Pipes & Filters**' architectural style could be a viable alternative. The below are the following reasons for not choosing this style over Client/Server:

   - As we already know in Pipes & Filters, subsequent errors from previous stages cannot be recognized and handled. So in our system, If there are possible defects/errors in the 'Data Pre-processing or 'Query Evaluation/Parse' stages, the same errors would propagate to 'Model Adaption' stages. Which would compound the **overall error** in processing the user's query. This results in **very poor performance/accuracy** of the system. Whereas, in Client/Server style, individual components can be evaluated & tweaked whenever it seems the components are generating erroneous results.
   - Developing a **Deep Neural Network for NLP** based on **Re-enforcement Learning with Context Sensitivity** is already a herculean task (highly complex!!) and takes a lot of time & effort just to design and debug the model. Combining this with other components such as Query Parsing Engine, Information Extractors, and Feedback Model etc. in a pipeline would make the system very **complex, confusing & highly difficult to implement**. As we all know, the more the complexity of the system is, the higher are the chances that the system would get riddled with bugs/errors. For an Intelligent System, **more errors would translate to worse predictive accuracy/performance**.

7. Review the principle of traceability. Have you implemented it in your architecture? How is it visible in the documents and views that you created so far?

   Yes, we have implemented the principle of traceability to improve the understandability of our system architecture. Since, the system is very modular, the development lifecycle of every module can be traced to its requirements easily. This is achieved by following strict measures such as **Uniform Naming conventions**, **Version Control**, **Self-Documentation** etc.

   The structures/module: 'Query Evaluation' and 'Model Adaption' are responsible for the fulfillment of the above decided Functional & Non Functional requirements. In the Documents & Views created so far, we can easily understand every aspect of individual modules such as the required functionality, required accuracy/precision of that respective module and so on.

   Using Traceability, if we map all the views for modules such as **Speech Recognition System, Query Evaluation Engine, Content Parser & Model Adaption etc.**, we see a clear description of the entire system and its overall required functionalities.

   In the first view, we can see the evolution of the Query answering/prediction feature over every iteration of development starting from the requirement phase. In the initial stages, there is no sensitivity towards the context of the query. But, as we progress towards the later stages of the lifecycle, the entire functionality with context sensitivity seems complete in compliance to the stakeholder's requirements.