Framework for Linguistic Applications Development(FLAD)

Group Members

C.M. Liyanage	13000632
P.A.D. Chandana	13000162
D.A.B.P. Dodangoda	13000349
G.D.D. Kanchana	13000535

Supervised by

Prof. K. P. Hewagamage

Dr. A. R. Weerasinghe

Mr. Viraj Welgama

Submitted in partial fulfillment of the requirements of the
B.Sc(Hons) in Software Engineering 4th Year Project (SCS4123)

University of Colombo School of Computing

Sri Lanka

May 22, 2018

Declaration

I certify that this dissertation does not incorporate, without acknowledgement, any material previously submitted for a degree or diploma in any university and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my dissertation, if accepted, be made available for photocopying and for interlibrary loans, and for the title and abstract to be made available to outside organizations.

Candidate Name:

.....

Signature of Candidate Date: .....

Candidate Name:

.....

Signature of Candidate Date: .....

Candidate Name:

.....

Signature of Candidate Date: .....

Candidate Name:

.....

Signature of Candidate Date: .....

This is to certify that this dissertation is based on the work of Mr. C.M. Liyanage, Mr. P.A.D. Chandana, Mr. D.A.B.P. Dodangoda, and Mr. G.D.D. Kanchana under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Supervisor Name:

.....

Signature of Supervisor Date: .....

Abstract

Natural Language Processing (NLP) is a main area emerged with the Artificial Intelligence. Text to Speech (TTS), Speech to Text (STT), Optical Character Recognition (OCR) and Language Translation are four main components of Natural Language Processing software applications which are also known as Linguistic components. There are many service providers currently available providing the services of the previously mentioned four linguistic components. Each linguistic component also has different vendors providing the service. These services can be online or offline. For an example, Google services such as Google Cloud Speech API, Google Cloud Vision API are online services. Offline services can be libraries such as Tesseract or stand-alone servers like MaryTTS which should be installed in machines. Therefore, these service providers maintain different interfaces to provide their services which makes a burden to the linguistic application developers. Developers who want to use different linguistic components in one application will have to face a lot of difficulties in many ways such as installing services, configuring services, etc. Developers who want to add more advanced functionalities to their applications by combining two or more linguistic components will have to face many major problems in integrating these components in the same application as these components are not in the same platform to be integrated. Considering all these problems we propose a framework which brings all the above mentioned linguistic components to a common platform and exposes the services as web services through REST APIs. Further, this framework addresses the issue of combining the services of linguistic components by exposing REST APIs for the complex services which are made up of services of two or more linguistic components. Therefore, developers only have to send requests to the REST APIs and handle the JSON responses sent by the system. Developers can create projects to access the services provided by the framework through the FLAD Console of the system. Developers can select the linguistic components and vendors of the selected linguistic components they need in their applications. When considering this framework from the technical aspect, expandable nature of this framework is an important achievement. This framework is designed in a such a way that a new linguistic component or a new instance of a linguistic component can be integrated to the framework with less overhead. However, when considering the overall nature of this framework it is clear that this will fill the gap between linguistic applications development and the linguistic application components which addresses the goal of this project that is to reduce the application development overhead through implementing a framework for linguistic application development.

Acknowledgement

First, we would like to be grateful to University of Colombo School of Computing for providing the opportunity to conduct a product based software engineering project under software engineering stream.

A very special gratitude goes to Prof. K.P. Hewagamage for giving us the seed idea of the project and supporting us as the main supervisor of the project. Advice and support given throughout the year were very helpful and important to make this project a success. We appreciate his valuable and constructive suggestions during the planning and development of this research work. His willingness to give his time so generously should be appreciated very much.

We would like to offer our gratitude to Dr. A.R. Weerasinghe and Mr. Viraj Welgama for supporting us throughout the project by giving advice and guidance to make this project a success.

Our special thanks are extended to the staff of language center at University of Colombo School of Computing for giving us the knowledge of natural language components and available Sinhala and Tamil linguistic applications.

We also appreciate the commitment and support given by MaryTTS GitHub community in solving the technical difficulties. The great support given by the active Google Developer community when solving technical difficulties related to Google Vision API and Google Speech API should be highly appreciated. We also appreciate the community support of StackOverflow when solving the programming difficulties.

We wish to acknowledge the great support of the highly committed team members who believed in team spirit and teamwork in achieving the final outcome of the project. The great collaboration and cooperation among the team members was the key to the success of the project.

Finally, we would like to offer our greatest appreciation to our parents in supporting us to beat unexpected difficulties in life and helping us to achieve our goal in the academic career in all sorts of ways.

Table of Contents

List of Figures

List of Tables

Acronyms

API	Application Programming Interface
DOM	Document Object Model
ESB	Enterprise Service Bus
FLAD	Framework for Linguistic Application Development
HMR	Hot Module Replacement
HTTP	Hypertext Transfer Protocol
JSON	Javascript Object Notation
MPA	Multiple Page Applications
NLP	Natural Language Processing
NPM	Node Package Manager
OCR	Optical Character Recognition
POM	Project Object Model
REST	Representational State Transfer
SPA	Single Page Applications
STT	Speech To Text
TTS	Text To Speech
UTF-8	Unicode Transformation Format-8

Chapter 1

Introduction

The current trend in computer science is to implement Artificial intelligence to mimic the human mind. Machines should be able to understand natural languages to achieve this goal. Natural language processing (NLP) is the ability of a computer program to understand human language as it is spoken.

Text To Speech (TTS), Speech To Text (STT), Optical Character Recognition (OCR) and Language Translation are the main application areas of NLP. Research groups in various universities and institutions have introduced various solutions/implementations in these fields.

With the rise of NLP field, people started using STT, TTS, OCR, and translations to increase human interaction in the computing applications. The majority of current applications are mobile-based hence acquiring audio utterances and images is an easy task. This has increased the usage of NLP in mobile applications. IBM Watson[1], Apple's Siri[2], Microsoft Cortana[3] and Google Assistant[4] are some of the leading smart assistants that uses NLP based applications.

The software applications which occupy NLP are called Linguistic applications. Continuously increasing mobile application industry handles a comparatively large part of linguistic applications. Mobile devices are not powerful enough to handle the processing of above linguistic components within itself. In order to achieve the best outcome of these applications, these services should be provided as third party services.

There are many software solutions currently available as services under each of these linguistic components with the different set of features and accuracy levels. Each of these services maintains their own application programming interfaces in order to provide their services. Due to this reason, there may be different appli-

[1]https://www.ibm.com/watson/

[2]https://www.apple.com/ios/siri/

[3]https://www.microsoft.com/en-us/windows/cortana

[4]https://assistant.google.com/

cation programming interfaces required to use when using linguistic components together. This complexity can be resolved by introducing a wrapper for these different application programming interfaces as a common application programming interface which is capable of communicating back and forth with child application programming interfaces in a flexible manner.

Framework based linguistic application development can be used to reduce the overall application development time and to improve the efficiency of applications. A framework for linguistic application development is a solution to handle the low-level configuration and integration of linguistic components and expose the functionality over well-defined application programming interface.

1.1 Motivation

The application development around linguistic components has become a necessity in current application development processes. The gap between the application development and integration, configuration of linguistic components has been a setback in the development of linguistic applications around the innovative ideas.

The unavailability of a standard method to utilize the services of linguistic components leads application developers to rewrite the code for integrating and configuring the linguistic components. This violates basic software engineering principles like code reuse. Further, the integration patterns used may not be optimal which leads to poor performance of applications. Lack of an architecture that provides the modularity, flexibility, and efficiency for linguistic application development is a dearth in the software engineering domain.

A framework which comprises an architecture that integrates all the linguistic components is a major requirement in this context. A framework enables developers to devote their time to meet software requirements rather than dealing with the more standard low-level details of providing a working system, thereby reducing overall development time.

For an example, a mobile application which helps the blind people to hear the content in a newspaper or a book can be simply developed using a framework. The developer does not need to do background study on the linguistic components or the methods of configuring and integrating them as the framework handles the low-level details related to linguistic components. Availability of a framework provides the best architecture to integrate linguistic components. It improves the performance and code reusability. It also simplifies the application code complexity. Further, the framework based development enforces the developer to follow best software engineering practices like low-coupling and high-cohesion through the use of well-

defined application programming interfaces.

The gap between the linguistic application development and the best integration of linguistic components can be filled by providing a framework for linguistic application development. The solution framework,

- enables rapid application development as the framework handles the low-level details of components.
- increases the application performance due to code reusability and internal architecture of the framework.
- reduces the complexity of application code because of application programming interfaces and low dependency between the application code and the code of the linguistic component.
- decreases the application size as the third party linguistic components libraries are implemented on framework side.

The design of a framework for linguistic application development supports the software engineering domain by addressing a commonly occurring issue in application development and support linguistic application developers all around the world.

1.2 Background

The key idea about the project is to integrate all the linguistic components Text To Speech (TTS), Optical Character Recognition (OCR), Speech To Text (STT) and language translation in a framework and expose their functionalities as a RESTful web service. Figure 1.1 shows the abstract view of the system. The framework facilitates to utilize the services simply using the REST API for linguistic application developers, without considering about the configuration, integration and other low-level details of linguistic components.

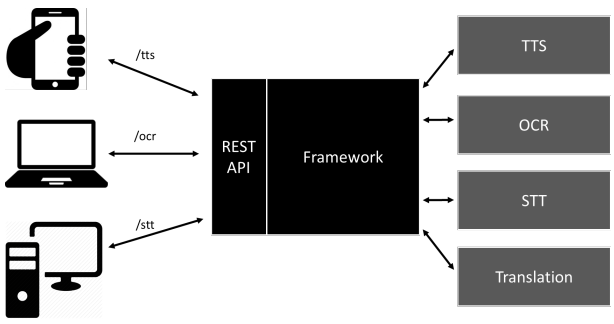

Figure 1.1: Abstract view of the system

The core of the framework is implemented using Java programming language and Jersey which is a RESTful web services framework used to implement the RESTful web services. The back end consists all the third party libraries and servers related to linguistic components which are configured and integrated into the core of the framework. The architecture of the framework is designed in a way to support the plug and play architectural model. The plug and play model enables the addition of new linguistic components to the framework easily without affecting initial architecture. The framework architecture was the main research area of the project as there were no prior linguistic application frameworks or similar systems that motivate the design of the architecture.

The plug and play model solves the issue of integrating heterogeneous technology into the framework. Available linguistic components are developed by many vendors using different programming languages and different technologies. For an example, the Google Speech API is a RESTful web service which exposes the STT service while the CMU Sphinx is an STT service which provides a standalone library. The framework should be able to integrate both the services irrespective of the vendor or technology used. Further, the framework should support the functionality of plugging a new STT component with minimal effort.

The framework is designed following software engineering best practices and different combinations of design patterns to achieve the ideal architecture for the framework. The detail description of the framework is explained in chapter 4.

1.3 Goal and Objectives

1.3.1 Goal

- Improve the development process of linguistic applications by reducing the overhead and overall development time by introducing a framework to support linguistic application developers.

1.3.2 Objectives

- Build a framework to expose TTS, STT, OCR, and language translation functionalities as web services which provides a uniform access point to all linguistic services.

- Provide a developer friendly application programming interface to ease off the integration of web services provided by FLAD.

- The architecture of the system should allow plugging new functionalities easily to the framework in order to address the future requirements.
- Provide friendly User Interfaces(UIs) for the easiness of end users.

1.4 Scope of the Project

The framework adheres to Representational State Transfer(REST) architectural constraints. The project consists of developing a RESTful web service which will provide four main web services such as,

- **Text To Speech Service** - When text is provided as input, the utterance of that text will be provided as output.
- **Speech To Text Service** - When an audio file of an utterance is provided as input, the text of that utterance will be provided as output.
- **Optical Character Recognition Service** - When an image file of a text is provided as input, the text of that image will be provided as output.
- **Translations Service** - When a word or a set of words is provided as input, it will be translated into specific language.

In addition to that, the system will provide five combined services. These services are combinations of main services. The combined functionality will act as a single web service.

- **OCR $\longrightarrow$ TTS** - OCR and TTS services are combined. The output text coming from the OCR service is fed as an input to TTS service. The output of TTS is sent as the response.
- **Translate $\longrightarrow$ TTS** - Translation and TTS services are combined. The output text coming from the Translations service is fed as an input to TTS service. The output of TTS is sent as the response.
- **STT $\longrightarrow$ Translate** - STT and Translation services are combined. The output text coming from the STT service is fed as an input to Translation service. The output of STT is sent as the response.
- **OCR $\longrightarrow$ Translate $\longrightarrow$ TTS** - OCR, Translation and TTS services are combined. The output text coming from the OCR service is fed as an input to Translation service. Then the output text coming from the translation service is fed to the TTS service. The output of TTS is sent as the response.

- **STT $\longrightarrow$ Translate $\longrightarrow$ TTS** - STT, Translation and TTS services are combined.The output text coming from the STT service is fed as an input to Translation service. Then the output text coming from the translation service is fed to the TTS service. The output of TTS is sent as the response.

The accuracy of the outputs depends on the used back end libraries. The overall performance of the framework relies on the input size, performance of used libraries, and speed of the internet connection. The framework handles the internal configurations to provide a web service from standalone libraries. Clients must have a user account so as to utilize the system. The framework will not have offline support.

The main deliverables of the project are,

1. A RESTful web service which exposes STT, TTS, OCR, translations and combined services as web services.

2. Client user interfaces to create and manage the projects to obtain the services from the framework.

3. Admin user interfaces to view the usage of the system and individual projects, manage the users and projects.

4. A User guide on how to use the framework.

1.5 Justification as Product based Project

The ultimate goal of this project is to build a framework of component-based plug and play architecture to which the four linguistic components (TTS, STT, OCR and Language Translation) can be plugged easily and expose their services to the application developers.

Application of software engineering techniques in this project begins from analyzing the project idea followed by writing the specification, design, and development. When analyzing the project idea it is important to find similar systems and analyze their pros and cons. However, there are no similar systems which facilitate the integration of above mentioned four linguistic components and reduce the overhead of configuring those components. It implies that this concept is a totally new one. Writing specification included laying out functional and nonfunctional requirements. To identify key users and user interactions that the system should provide, it was important to draw the use case diagram as a software engineering technique. Main users of this system are application developers and the FLAD administrators. Both application developers and FLAD administrators have different

use cases. Functional and non-functional requirements including the tasks of each user type of this system are mentioned and described in Chapter 3. Use of several design patterns when designing the system resulted arising a new design pattern to adapt the component based plug and play architecture. Class diagram of the system and further information about the design patterns used and details about the component based plug and play architecture are discussed in Chapter 4.

When developing the system, SCRUM of agile methodology is used as it was the most suitable development methodology for this project because of the iterative and incremental nature of the project. Further for the success of this project software engineering guidelines and principles were followed. More details on tools and technologies and development models used are discussed in Chapter 5.

Building a framework which supports component reusability and reduces application development overhead contributes to solve a commonly occurring problem in software engineering domain. The final product of the project is delivered as a fully functional framework for linguistic application development.Considering all these factors, this could be concluded as a product based software project.

Chapter 2

Background Study

2.1 Introduction

This chapter discusses current methods and techniques of linguistic application development mentioned in literature, technologies followed during the development of the project, architectural styles and design patterns which were used in the core of the framework. The background was studied to inherit the latest knowledge from the systems related to the context of this project.

Background study was mainly conducted related to framework development fundamentals, linguistic components, best software engineering principles and latest stable technologies which assist in framework development.

2.2 Software Framework

The software framework is an abstraction in which software providing generic functionality for a specific task and supports for the future requirements without changing existing architecture. A software framework provides a standard way to build and deploy applications on top of the robust architecture.It is a universal, reusable software environment that provides particular functionality as part of a larger software platform to facilitate the development of software applications, products, and solutions. Software frameworks may include support programs, compilers, code libraries, toolsets, and well-defined Application Programming Interfaces (APIs) that bring together all the different components to enable development of a project or a system.

The main purpose of the software framework is to improve the efficiency of creating new software. The software framework allows developers to concentrate on functional and non-functional requirements of their solution rather than considering the low-level configurations.

Frameworks have key distinguishing features.[1]

- Inversion of control - In a framework, unlike in libraries or in standard user applications, the overall program's flow of control is not dictated by the caller, but by the framework.
- Extensibility - A user can extend the framework - usually by selective overriding, or programmers can add specialized user code to provide specific functionality.
- Non-modifiable framework code - The framework code, in general, is not supposed to be modified, while accepting user-implemented extensions. In other words, users can extend the framework, but should not modify its code.

2.3 MaryTTS

Modular Architecture for Research on speech sYnthesis (Mary) TTS [2] is a java based multilingual Text-to-Speech Synthesis platform developed by German research center for artificial intelligence. Mary TTS is available as a standalone server which has to be downloaded to the local machine in order to get the service. The service is accessed through the interfaces defined by MaryTTS. MaryTTS is the widely used TTS service due to its support for different voices and extendability provided to include own locale. However, MaryTTS does not provide a simple web service which exposes the functionality. The Mary server is resource extensive and has a capacity of 60 megabytes. Mary implementation does not support JSON for data exchange. MaryTTS integration to FLAD provides a REST API endpoint to utilize TTS service using JSON support.

2.4 Kaldi Automatic Speech Recognition (ASR)

Kaldi is a toolkit for speech recognition written in C++ and licensed under the Apache License v2.0. Kaldi is intended for use by speech recognition researchers[3]. Kaldi provides a library, command line program and scripts for acoustic modeling. The API of Kaldi is command line interface. Kaldi has a high accuracy and an active community with rich documentation support. However, The CPU and RAM consumption of Kaldi ASR is high. It has a deep learning curve and needs a prior knowledge of natural language processing. The configuration overhead is relatively high with Kaldi ASR.

2.5 HTK Speech Recognition Toolkit.

HTK (Hidden Markov Model Toolkit)[4] is a portable toolkit primarily used for speech recognition researchers. HTK was originally developed at the Machine Intelligence Laboratory (formerly known as the Speech Vision and Robotics Group) of the Cambridge University Engineering Department (CUED). This toolkit is written using C programming language. The API of HTK is command line interface. HTK toolkit has a great documentation and a simple configuration task. However, the accuracy level is low due to slow training and processing. There is no up to date development over this toolkit and development has been deprecated.

2.6 CMU Sphinx Speech to Text

CMU Sphinx is a group of speech recognition systems developed at Carnegie Mellon University [5]. These include a series of speech recognizers (Sphinx 2 - 4) and an acoustic model trainer (SphinxTrain). Sphinx 4 is the current version of Sphinx. It is developed using the Java programming language. Sphinx 4 can be downloaded as a JAR file. The resource consumption is average in CMU Sphinx and can be embedded as a dependency in a Java Maven project. The accuracy of the output is average for a given input.

Flad should handle concurrent requests for endpoint services, therefore, the most efficient and cost-effective library should be used. So that we have used CMU Sphinx in FLAD. (see Appendix A for the comparison of STT libraries.)

The users of FLAD can obtain the STT service provided by CMU Sphinx as a web service. Therefore the users don't have to bear any configuration cost. The storage is saved as there is no need for the users to download the Sphinx library to their devices.

2.7 Google Cloud Speech API

Google Cloud Speech API is a powerful a speech recognition tool that enables developers to convert audio to text. This API recognizes over 110 languages and variants. Google Speech service uses deep learning neural network algorithms and accuracy improves over time. Google Cloud Speech API has 3 references [6].

1. Google Cloud Speech API Client Libraries
2. Cloud Speech REST API
3. Cloud Speech RPC API

In FLAD, the system uses the Cloud Speech REST API which performs only synchronous speech recognition. Synchronous Speech Recognition returns the recognized text for short audio (less than 1 minute) in the response as soon as it is processed. Audio content can be sent directly to the Cloud Speech API through a JSON request, or the Cloud Speech API can process audio content that already resides in Google Cloud Storage. If the audio content is sent directly to the Cloud Speech API through a JSON request it has to be sent as a Base64-encoded audio content in the JSON body.

However, Google Speech API is a paid service and priced monthly based on the amount of audio successfully processed by the service. (see Appendix B.1 for pricing details). Synchronous scheme only supports audio streams with one channel. There is a limitation of an audio file to be less than one minute in synchronous processing.

The framework uses Google Speech API integration and exposes a simple REST API endpoint. The users have no configuration overhead since the framework internally handles authentication and configuration of Google services.

2.8 Tesseract OCR

Tesseract is an OCR engine for various operating systems. It is free software released under the Apache License, Version 2, and development has been sponsored by Google since 2006. Tesseract is considered as the most accurate open-source OCR engines available. Tesseract has the ability to recognize more than 100 languages out of the box and it can be trained to recognize other languages [7].

Tesseract library which is available on GitHub facilitates the different set of features of OCR to support the development of OCR based applications. Tesseract library which is known as Libtesseract is written in C/C++. However Tesseract uses wrappers to wrap that library to be used in other languages like Java, .NET.

Tesseract does not have any REST API to expose its services as web services. Due to this reason, it is necessary to download and build the source when developing a mobile application which has the OCR functionality with Tesseract engine. The size and the performance overhead of the Tesseract library will be added to the application if it is included in the application. In the point of mobile application development, the size of the mobile application is a vital factor.

2.9 Google Cloud Vision API

Google OCR is under the Google Cloud Vision API which can detect and extract textual content from images. Text detection and Document text detection are

the main annotation features that support OCR [8]. Google OCR is able to do automatic language detection and has a REST API to support the services. The accuracy of Google OCR is high and it supports a wide variety of image formats like JPEG, PNG8, PNG24, GIF, RAW, ICO, and BMP.

However, it is a paid service (see Appendix B.2 for the detailed price list) and no direct image upload is possible. The image has to be encoded in the Base64 encoding scheme and has a limitation of file size to be less than four megabytes. The configuration overhead is high and requires image pre-processing before sending in RAW image files.

2.10 Google Translation API

Google Cloud Translation API translates text between thousands of language pairs [9]. Translation API enables programmers to get the translation service via a REST API. It supports more than hundred languages and capable of detecting the source language automatically. However, it is a paid service and input text has to be UTF-8 encoded for better accuracy.

After doing the initial background study on different linguistic components then we focused on designing an architecture for the core of the framework which should support the integration of any linguistic component along with the plug and play model.

2.11 WSO2 ESB

The component architecture of the WSO2 ESB [10] is built on the Apache Synapse project, which is built using the Apache Axis2 project. The initial foundation for the framework was inspired by the WS02 ESB architecture. ESB supports web services and support for XML. Message transformation feature leads to the introduction of Message Builder and Formatter component of FLAD. Logging and monitoring features in the ESB and integration patterns in WSO2 ESB leads to the foundation of the architecture in the FLAD framework.

2.12 Google Cloud Console

Google Cloud Console has a vast range of design thinking methods and approaches as one of the best cloud consoles which are available on the internet. It handles a comparatively large set of APIs, application users, and applications. It has user-friendly data arrangement methods and user interface approaches to facilitates the

most enhanced usability experience in order to handle a large number of users. Front end of the Console is built following the Google's Material design and it's concepts. This increased the front end usability of the system since the Google's Material design is designed to enhance the experience of end users by considering different aspects of the human-computer interactions.

2.13 REpresentational State Transfer (REST)

The term representational state transfer was presented and characterized in the year 2000 by Roy Fielding in his doctoral paper [11]. REST is the current trend in web services architecture. A web service that is based on REST technology is called a RESTful web service/API.

REST is an architectural style. It is lightweight than SOAP (Simple Object Access Protocol) style [12]. As REST uses less bandwidth it is preferred to use over the internet. The REST architectural style describes six constraints [13].

2.13.1 Uniform Interface

This defines the interface between clients and servers. It decouples the architecture which enables each part to evolve independently. The four guiding principles of the uniform interface are:

Resource-Based

Individual resources are identified in requests using URIs (Uniform Resource Identifiers) as resource identifiers. The resources and their representations are separated. For example the server does not send its pure database records rather a well formatted JSON response.

Manipulation of Resources Through Representations

When a client holds a representation of a resource, including any metadata attached, it has enough information to modify or delete the resource on the server, provided it has permission to do so.

Self-descriptive Messages

Each message includes enough information to describe how to process the message.

Hypermedia as the Engine of Application State (HATEOAS)

The hypermedia (or hyperlinks within hypertext) is body content, response codes, and response headers. In REST the application state is delivered via hypermedia to clients and from clients.

2.13.2 Stateless

The necessary state to handle the request is contained within the request itself, whether as part of the URI, query-string parameters, body, or headers. In web programming, the sessions are used to maintain the state across multiple HTTP requests. But in REST, the client's request must contain all information required for the server to process the request. For example, in each request, the client's unique identification key is sent to the server rather storing the key in session. The stateless property enables greater scalability since the server does not have to maintain, update or communicate that session state.

2.13.3 Cacheable

The responses can be defined as cacheable so that the clients can cache them. Well-managed caching policy can improve scalability and performance.

2.13.4 Client-Server

The clients are separated from servers. Their communication happens by means of the uniform interface. Clients are not concerned with internal data storage of the server and servers are not concerned with the user interface or user state. This separation improves portability of client code and scalability of the server. Server code and client code can be developed and replaced independently as long as the interface is not altered.

2.13.5 Layered System

The system can have layers. Intermediary servers can be placed along the way to improve system scalability by enabling load-balancing and by providing shared caches. The layered system may enforce security policies.

2.13.6 Code on Demand

This is an optional constraint. Servers can transfer logic to the clients to execute. For example, the compiled components such as Java applets and client-side scripts

such as JavaScript can be transferred to a client to customize the functionality of a client. If a service violates any other constraint, rather than the optional constraint, it cannot strictly be referred to as RESTful.

2.14 REST API Design approaches

The design of an Application Programming Interface (API) has different phases and decisions to consider in order to make the usability and robustness of the API. API has to be maintained over the time to improve the efficiency of consuming. An API is a collection of routines, protocols, and tools for building software applications. It includes inputs, outputs, operations, and types as characteristics of an API. One must consider modularity to be the most important factor when starting to construct the architecture of the API so that the functionality is separated from the implementation.

APIs have a number of decisions during the architectural design process, and these decisions explain the semantics of the API [14]. These decisions include REST, requests and responses, resources, URL's, endpoints, HTTP verbs.

2.14.1 HTTP verbs

GET

The GET request is used to retrieve a representation of a resource. This can be a collection of a resource or a single item.

POST

The POST request is a new request that the web server accepts with the entity that is enclosed and identified by the URI. It is used to create subordinate resources, which explains it is a subordinate to some other resource. This method is not considered to be a safe method as making identical requests will likely result in two resources containing the same information.

PUT

The PUT request is used to create subordinate resources, which in essence means it is a subordinate to some other resource. This method is not considered to be a safe method as it modifies (or creates) state on the server, but it is idempotent. In other words, calling the same PUT request two times to create or update a resource will not create a new resource and the system still has the same state as it did with the first call.

DELETE

The DELETE request is fairly straightforward. It is used to delete a resource that is identified by a URI. The request will delete a resource, and repeatedly calling DELETE on that resource will always end up the same.

There are other HTTP verbs such as CONNECT, PATCH, OPTIONS, TRACE. In this background study, the main focus is given for the above four main verbs.[15]

2.14.2 Safe Methods

Methods that are considered to be safe are those that have no side effects and do not change the state of the server. When being considered not to have side effects, this explains that there is not much harm that can be done other than the likes of logging, caching, and other small harmless effects.

2.14.3 Semantic Versioning

Versioning is an important decision-making process that must take into consideration what state the API will be in when changes occur.

MAJOR version when making incompatible API changes,
MINOR version when adding functionality in a backward-compatible manner, and PATCH version when making backward-compatible bug fixes [16].

2.15 Authentication

Authentication handling can make a considerable difference in the design of the API. One of the most known and used authentication frameworks used these days is OAuth 2.0 [1]. This framework enables a third-party application to obtain limited access to an HTTP service.

Using OAuth 2.0 to access google API

Google Sign-In manages the OAuth 2.0 flow and token lifecycle, simplifying the integration with Google APIs. Google supports common OAuth 2.0 scenarios such as those for web server installed and client-side applications.

[1]https://oauth.net/2/

2.16 JavaScript Object Notation(JSON)

JSON is a lightweight data interchange format. It is a language-independent collection of key-value pairs [17]. It has become the state of art data interchange format. It is self-descriptive and human-readable than XML messages. Different languages treat JSON message as an object, record, struct, dictionary, hash table, keyed list, or associative array [18].

JSON has become a popular alternative to XML for various reasons.

- JSON is less verbose than XML.
- JSON data model's structure matches the real data. Hence it is easier to interpret and is predictable.
- JSON can be encoded to objects and decoded from objects. Most of the popular languages have JSON parsers.

In the FLAD framework, JSON has been used as the data interchange format in the requests and responses. Requests handling and response generating became easy because of the flexibility of JSON format.

Listing 2.1: Sample JSON message to obtain the TTS service from FLAD

```
{
    "inputText":"We have a clear sky",
    "inputType": "TEXT",
    "outputType": "AUDIO",
    "outputFormat": "WAVE_FILE",
    "locale":"en_US",
    "voice": "cmu-slt-hsmm en_US female hmm",
    "features": {}
}
```

2.17 React

React is a JavaScript library for building Interactive user interfaces. It is maintained by Facebook and it has a large community backup. Web applications developed using React does not reload the whole web page when the state changes. It re-renders only the modified area of the webpage. This mechanism improves speed, simplicity, and scalability.

2.17.1 Notable features

One-way data flow

React uses immutable values in its components. These values are called Properties (or props). Props are stored in a global store. React component does not modify the props that are passed to React components directly, Instead, it uses callback functions to modify the props in global store. The most used global store with React is Redux.

Virtual DOM

Virtual DOM is an in-memory representation of Real DOM. It is lightweight and detached from the browser-specific implementation details. This is the reason for achieving the faster loading time in React. In addition to that React uses below mentioned properties in order to update the virtual DOM.

- Efficient diff algorithm
- Batched update operations
- Efficient update of subtree only
- Uses observable instead of dirty checking to detect change

JSX

JSX is a preprocessor step that adds XML syntax to JavaScript. React can be used without JSX but JSX makes React a lot more elegant. Just like XML, JSX tags have a tag name, attributes, and children. If an attribute value is enclosed in quotes, the value is a string. Otherwise, wrap the value in braces and the value is the enclosed JavaScript expression. [19] [20]

2.18 Redux

Redux is a predictable state container for JavaScript apps. As the requirements for JavaScript single-page applications have become increasingly complicated, Redux provides more flexible and manageable way of handling data by reducing the complexities with states. This state can include server responses and cached data, as well as locally created data that has not yet been persisted to the server. Redux can be explained using three fundamental concepts.

2.18.1 Single source of truth

The state of the whole application is stored in an object tree within a single store. This makes it easy to create universal apps, as the state from the server can be serialized and hydrated into the client with no extra coding effort. A single state tree also makes it easier to debug or inspect an application.

2.18.2 State is read-only

The State is read-only explains the only way to change the state is to emit an action, an object describing what happened. This ensures that neither the views nor the network callbacks will ever write directly to the state. Instead, this should express an intent to transform the state. Because all the changes are centralized and happen one by one in a strict order, there are no subtle race conditions to watch out for.

2.18.3 Changes are made with pure functions

Redux uses Reducers which are just pure functions that take the previous state and an action, and return the next state. In addition to that, it returns new state objects, instead of mutating the previous state. [21]

2.19 Summary

This chapter discussed important facts found in literature which are related to the problem that is addressed and the solution that is proposed, including information on linguistic components that are currently available, REST API design approaches, etc. Further, important details on React and Redux which is used in front end of the system is discussed in this chapter.

Chapter 3

Functional and Non-functional Requirements

3.1 Functional requirements

3.1.1 Main users of the product

- **End-user: Linguistic application developer**
  The intended users of the product are software engineers who develop linguistic applications. The end users should have experiences in using RESTful web services.

- **Admin: The person who manages the framework**
  Admin can use the Management UI to manage the functionality of the framework.

3.1.2 End user's functional requirements

- Register with the system.

- Login to the system.

- Create, View, Edit, Delete projects.

- Select the services to use such as text to speech, speech to text, optical character recognition and translations.

- Select which libraries want to use as the back end of the service, for example if client selects to use speech to text service then he/she can decide whether to use a library such as Sphinx or to use Google speech to text cloud service.

- Get a private token. This token is used by the system to validate the user.

- Get TTS/STT/OCR/Translation services.
- Combine the TTS/STT/OCR/Translation services to get complex web service.
- View statistics of their own projects.

3.1.3 Admin's functional requirements

- Login to the system.
- Logout from the system.
- View all the projects.
- Block projects.
- View statistics of the usage of the system.
- Connect, remove endpoints to the framework. (endpoints are instances of TTS, OCR, STT and Translation services)
- Define sequences (sequence is a logical arrangement of mediators, more details about sequences are explained under the system design)
- View logs.

3.2 Non-Functional requirements

3.2.1 Performance

The system should provide a fast response time to the end user. The system should be able to produce high throughput utilizing fewer computer resources. The performance of API calls depends on the available bandwidth and on the performance of third-party linguistic services. However, the system provides optimization techniques like sequences and caching to improve the performance.

3.2.2 Scalability

Scalability is the ability to accommodate the growth of the requests. The system should support horizontal scalability and the system design is modeled to support horizontal scalability. (Multilayered architecture and component architecture of the system is discussed in Chapter 4 considering scalability). Functional scalability is the ability to enhance the system by adding new functionality at minimal effort.

(The plug and play model discussed in Chapter 4 facilitates functional scalability). Scalability is a key concern in the system as the framework should operate even in high request growth.

3.2.3 Availability

The degree to which the system or subsystems are operable. The system runs on server environments which is available 24/7. The availability is crucial in this system as the framework exposes the functionality as a web service.

3.2.4 Reliability

Reliability is the ability of the system to function at a specified moment providing intended results. The framework should provide a reliable service via service endpoints.

3.2.5 Maintainability

The framework should maximize efficiency, reliability, and security which are always improved by maintaining the system. New requirements can be introduced to the system because of maintainability. Maintainability helps to prevent unexpected working conditions. The framework should be designed and implemented considering maintainability.

3.2.6 Security

Since the framework provides services on a user basis, the system should validate the requests and ensure security. The users should only have access to the services purchased by them which brings security as a key concern of the system.

3.2.7 Data Integrity

Data integrity and consistency is considered to solve the heterogeneity service problem in the framework. The application developer should be provided with a consistent data format for requests and responses. (The implementation details in Chapter 5 explains the JSON structures which enforce the data integrity of the system)

3.2.8 Usability

Usability is the degree to which the system can be used by specified users to achieve qualified objectives with efficiency and satisfaction. The system consists of two parts main as FLAD console and FLAD core. The usability is taken as a key concern in designing the FLAD console which is discussed in Chapter 4 under design considerations.

3.2.9 Interoperability

The framework functionality is provided as a web service which can be accessed using any device and any platform. The system should work on both Linux or Windows server environments.

The exact achievement of the non-functional requirements are discussed in Chapter 6 under evaluation of the system.

Chapter 4

Design Consideration

4.1 Introduction

This chapter explains how the framework‘s design is implemented using latest technologies, following best software engineering principles. The entire system is mainly divided into two subsystems as front end FLAD Console and back end FLAD core. Different architectural designs are applied to the subsystems as the functional and nonfunctional requirements are different in two subsystems.

The front end FLAD console is of two parts, one part is where the end user application developer creates a service API application in our framework specifying all the services expected to obtain from the system. The second part is designed for framework administrators to manage, monitor the FLAD system.

The front end system is deployed on a separate server and independent of the back end core framework. The front end system runs on a node server and designed using React framework and Redux data flow architecture.

The back end system is the core of the framework which is a Java EE project that runs on a Jetty server. The core framework integrates all the linguistic components together and exposes a REST API which is implemented using Jersey according to JAX RS specifications. In addition to that back end core supports for plugging different linguistic services into the framework without changing the initial system design. The front end system of FLAD interacts with the back end framework via web service calls and the back end interacts with the data layer which consists of a MongoDB database management system.

4.2 System design and architecture of the FLAD back end core framework

The main architecture of FLAD core is divided into three sub architectures considering different functionalities of the system. Different architectures are required to cater different core functionalities supplied by the back end and thus divided into three sub architectures as,

1. Communication architecture
2. Structure architecture
3. Deployment architecture

The segregation of main system architecture into sub architectures reduces the complexity of analyzing the problem according to the functional and nonfunctional requirements. It is necessary to introduce different architectures to the different sections to achieve the desired outcomes. Figure 4.1 shows the main architecture segregation and the specific architectural styles used in each category.

The main research area of the project was to introduce best architectural styles into the framework. The background study guided into selecting the Service Oriented Architecture (SOA) as the communication architecture, Component Architecture for the structure architecture and Client-Server Architecture as the deployment architecture.

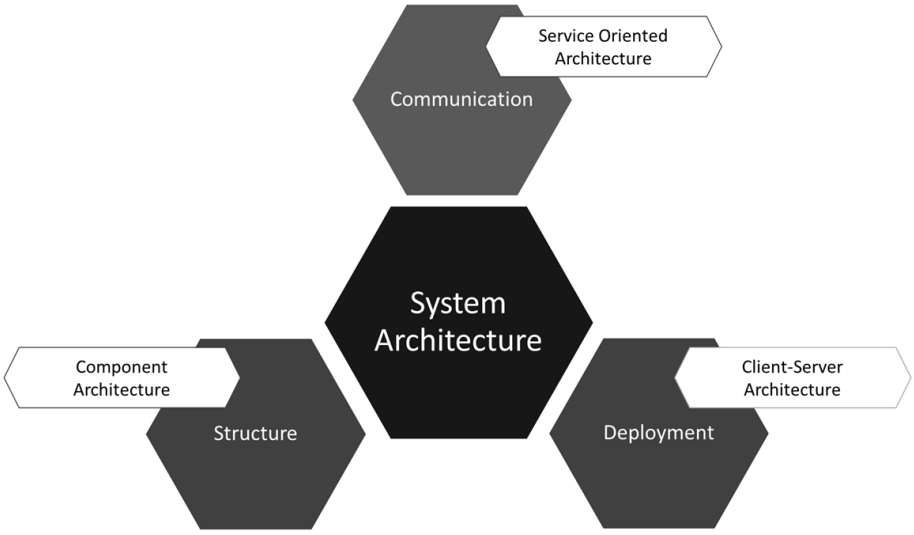

Figure 4.1: System architecture of FLAD

4.2.1 Communication architecture

The communication architecture is implemented using Service Oriented Architecture (SOA). SOA enables the communication of services of different application components through a communication protocol. According to SOA definition, a exposed service should contain four properties [22]. These properties are followed in the implementation of FLAD.

1. **The service should logically represents a business activity with a specified outcome**
   FLAD endpoints serve their specific services and it directly represents the business value of the service. For an example /tts endpoint directly represents the TTS service entity and results the intended outcome.

2. **The service is self-contained**
   FLAD provides simple services which are self-contained and independent. Everything that is required by the service is separated from other services and exposed via the REST API.

3. **Service is a black box for its customers**
   The service provided to the customers just gives the intended response to the customer without exposing the interior logic. The logic is handled inside the framework and the user simply gets the service according to the preferences selected in the creation of service API project using FLAD Console.

4. **It may consist of other underlying services**
   The FLAD complex services which are sequences, validations, message builder and formatter use other underlying services internally when executing.

FLAD implements services in a loosely coupled manner and modular approach which lean towards the SOA architectural pattern. REST architectural style reflects the SOA in the framework.This enables exposing functional building blocks accessible over HTTP providing the solution to represent new components or standalone systems. SOA based systems allow the utilization of services independently of development technologies and platforms such as Java, C++ and etc. Also by embracing SOA style motivates the use of well-defined interfaces and supports the component-based architecture which is the selected structural architecture of FLAD.

4.2.2 Component architecture

The main structure of the project is laid on top of the component architecture which is highly compatible with SOA. The definition of the different components in the

system and binding of them was part of the major research area of the project. The component-based architecture should be able to support the plug and play model of the system where any linguistic component can be added or removed from the system with minimal effect to the back end framework and this should support the extendability of the framework.Figure 4.2 shows the components defined in our framework which provide the fundamental functionalities by communicating via well-defined interfaces.

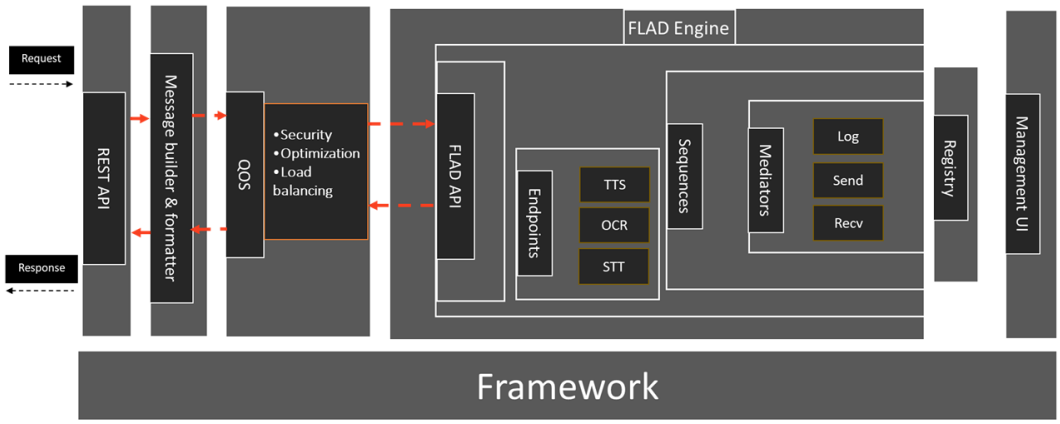

Figure 4.2: Component architecture of FLAD

The component architecture of FLAD is described using an example, Assume an application developer wants to use the STT service from the framework.

- First, the user makes a request to **/stt** endpoint in the **REST API** with the specified JSON payload.
- Then the user's request has to be parsed with a JSON parser and convert to the internal data representation using the **Message builder and formatter**.
- After parsing the payload, it is sent to the **QOS components** for validation and optimizations before sending to the real endpoint service. In an STT request, the mono stream of the audio file is checked and if the audio is on stereo it is converted to a mono stream and then encoded using Base64 encoding scheme.
- After preprocessing the request payload, it is sent to the internal FLAD API written in Java. The STT request is validated first and the STT instance is invoked dynamically with the help of factory and abstract factory design patterns. If the user has selected Google STT service in project creation phase then it has to invoke the Google STT instance and if the user has selected any other STT service it is dynamically detected with the help of data layer and then the initiation takes place.

- The FLAD internal API decides which endpoint to invoke and endpoints shown in Figure 4.2 are instances of STT, TTS, Trans or OCR. For example, an endpoint in the framework related to this example is Google speech API.

- Another important component is the **Sequence** component which is a combination of different send and receive calls of endpoints. For example **/seq2** in our framework invokes the sequence of (OCR $\longrightarrow$ Translation $\longrightarrow$ TTS). This sequence is capable of reading the content of an image and output the speech in a specified translation. The content of a German signboard can be heard in English using the above sequence.

- **Mediators** are the basic functional units in the framework. Currently, there are three mediators in the framework as Send, Receive and Logging. Send mediator is responsible for sending messages between different components of the framework and the Receive mediators are responsible for dealing with the return values of the components. The Log mediator logs every action in the framework to support monitoring.

REST API

Rest API exposes the resources and functionalities of the framework and enables basic HTTP protocols to utilize the framework services.

Message builders and formatters

Message builder is used to process incoming and outgoing payload data to XML or JSON. There are two basic functional units as message parsers and formatters. The message formatter is used to build the outgoing response from the message back into its original format. Message parsers and builders convert the payload data to required data formats of endpoints.

QoS component

The Quality of Service (QoS) component implements security, load balancing, and optimization. Basic message preprocessing is done with this component.

Endpoints

An endpoint defines an external destination for a message. An endpoint can connect to any external service after configuring it with any attributes or semantics needed for communicating with that service. TTS endpoint, OCR endpoint, STT endpoint and Translation endpoint are the basic endpoint types of the framework.

Mediators

Mediators are individual processing units that perform a specific function, such as sending, transforming, logging or filtering messages.

Sequences

A sequence is a set of mediators organized into a logical flow. The concept of sequence is an optimization technique used to reduce the successive incoming and outgoing calls to the framework and improve performance.

Registry

A registry is a content store and metadata repository. The framework cache required data in primary memory for fast retrieval. For example, the database connections are pooled after the first request so any request made after gets the advantage of fast retrieval of data and factory objects are cached in the Registry component after first initialization using the Service Locator design pattern.

Management UI and FLAD Console

The Management console provides a Graphical User Interface (GUI) that allows the framework administrator to monitor and manage the framework functionalities. The FLAD Console is the management UI which is implemented in React and deployed in a NodeJS server. FLAD Console enables users to specify required services through a project creation.

4.2.3 Deployment architecture

Figure 4.3 shows the distributed client-server architecture which is the deployment architecture of FLAD system. Distributed application structure, support the future scalability and load balancing in presence of high request growth. The framework's functionality exposed in a way to be utilized by any client globally and server can be replicated with the future expansion according to this model. The use of distributed client-server architecture enables four services of the framework to be distributed. The front end FLAD console service, back end framework core, MongoDB database and each endpoint service can be distributed using this model with the help of well-defined interfaces.

The front end FLAD console is deployed in a separate Node JS server and can be deployed in a different server. It communicates with back end server through REST API calls in a loosely coupled manner. The back end is deployed in a java based Jetty server and is independent to the front end server. Both the front end

server and back end server communicate over well-defined interfaces. The database server can be deployed on several servers and currently, the MongoDB server is on the same server machine where the back end framework server is deployed. The deployment of each linguistic components in separate servers is possible according to this model. For example, MaryTTS server is running on a different server and can be scaled up with the huge growth of requests.

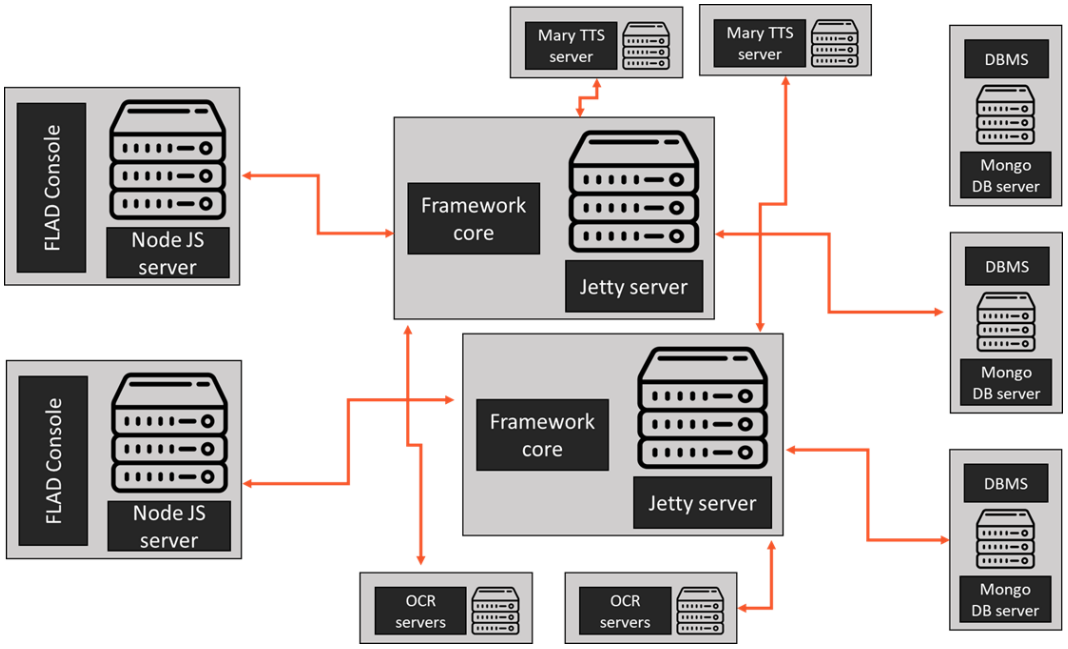

Figure 4.3: Distributed deployment of FLAD

4.2.4 The multi-layered architecture of FLAD

The layered architecture enables to understand the system in a concise way where all the core functional layers of the system are identified. Figure 4.4 shows the multi-layered architecture of the FLAD system. This representation allows easy understanding of the idea behind the scalability and extensibility of the system.

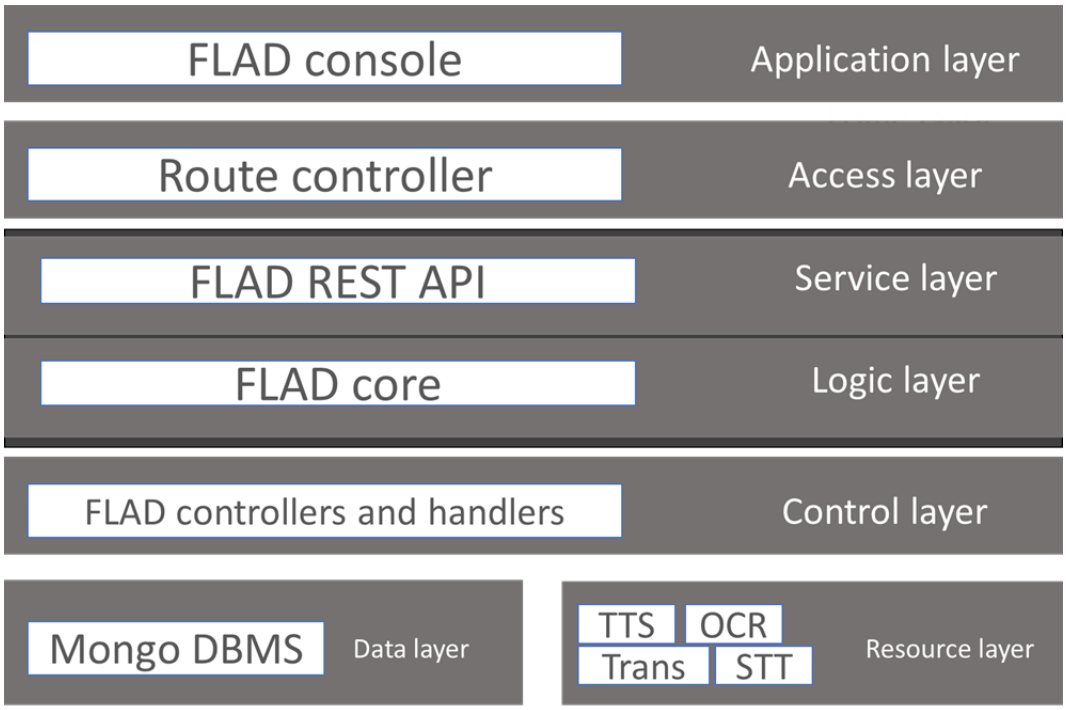

Figure 4.4: Multi-layered architecture of FLAD

The multi-layered architecture was designed from scratch according to the needs of the system. The application layer, service layer, logic layer, data layer and resource layer can be replicated and distributed. The modularity among the components within a layer was the key concern when developing the layered architecture. The modularity was achieved by the use of component-based architecture in each layer. The well-defined interfaces allow the independent layers to communicate using message parsing.

4.2.5 Core Component architecture of FLAD

This section describes the design patterns used in the core component architecture and the decision points which lead to use them. The design patterns are applied in both the front end FLAD console and the back end framework. Applied design patterns are integrated in order to overcome the commonly occurring issues in the system.

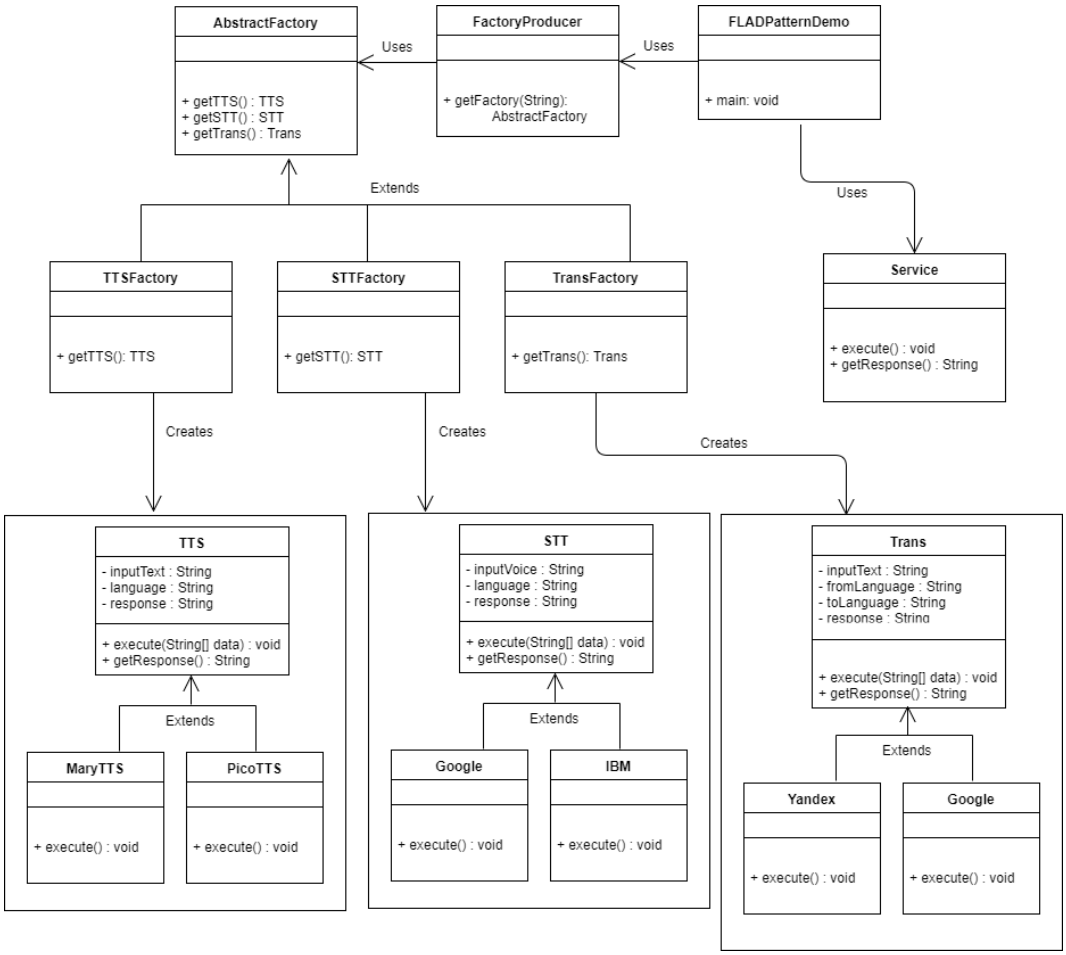

Figure 4.5: Class diagram of FLAD core

Figure 4.5 shows the class diagram architecture of the system. The structure of FLAD system design is based on Component Architecture. This diagram consists of service components and factories that generate above components and relation of their particular instances. Abstract Factory Pattern, Builder Pattern, and Command Pattern integrated in order to achieve the plug and play architecture.

Abstract Factory Pattern involves installing and instantiating each component in the environment. Factory Producer provides interface for the Abstract Factory to communicate with front view, In this context, it will be the REST API which accesses by users.

Builder Pattern uses when the components or subcomponents in the architecture dependant on each other. When adding a new service such as MaryTTS, Google or IBM, system will create an object of the given service and inject it to their parent objects. As shown in figure 4.5, the parent objects are the service initiators named TTS, STT, OCR, and Trans. Each of this service objects depend on these service initiators.

Command Pattern gives components a consistent entry point, allowing them

to be readily swapped in and out. Each of service initiators feeds service objects assigned to them through execute method.

This also follows dependency injection when passing of a dependency to a dependent service object that would use it. All the service objects located at the bottom of the diagram designed as a dependant to the service initiator objects.

These plugged service objects will execute as single services on their own, but on the inside of the structure, they are coupled with service objects which are generated using the particular factory. When new service introduces, it is required to link that service to the relevant service initiators.

The Factory and Abstract Factory design patterns are combined in the core of the system to achieve the plug and play model. This abstract factory design pattern further facilitates the dynamic initiation of services at runtime in a flexible manner.

The Abstract factory class delegates the responsibility of object instantiation to TTSFactory, STTFactory, OCRFactory, and TransFactory. Those Factories use inheritance for the object instantiation of real service dynamically. These factories are the key components of achieving the plug and play model for the framework. Figure 4.5 shows how inheritance and encapsulation are used in the class diagram. The component class is the base class of each integrated service component. It defines all the common features and behaviors that expected to retrieve from particular instances which are yet to plug into the architecture. The execute method which is defined as public describes the internal logic of each instance. These component instances are the actual integrated service classes which interact directly with parent service components in order to provide their services to the upper level. Figure 4.5 contains MaryTTS and PicoTTS, as TTS subclasses. These instances may have their own additional features. Basically, each instance supposed to provide minimum required features defined in parent components. All the internal logic of each instance describes in the execute method which is designed to invoke dynamically.

These steps should be followed to add a new component to the system,

- Define a class with the name of the new component.
- Inherit the superclass which the new component belongs to.
- Override the execute method and write the interior logic specific to the component.
- Add the entry name in the related Factory class.

The exact implementation details and logical flow is discussed in detail in chapter 5. This approach leads the minimal change to the framework functionality and

this can be further simplified using XML to define the name of the new component at runtime using Management UI. However, writing of the implementation logic in the execute method of the component is unavoidable as different linguistic services provide different APIs. For an example, MaryTTS and Tesseract OCR are standalone apps and the logic needs to be manually coded in the execute method.

Service Locator Pattern

Service locator pattern caches the objects in creation in order to achieve performance enhancement. The use of this pattern enabled to cache the object references after it's first creation into separate HashMaps in Java. The caching is done at the Cache class as shown in Figure 4.6. The service locator first looks at the cache and if there is no reference in the hash maps then creates the objects and cache it before sending. This improves the performance of the object creation time via caching the references in the Registry component of the framework by avoiding object creation at each request.

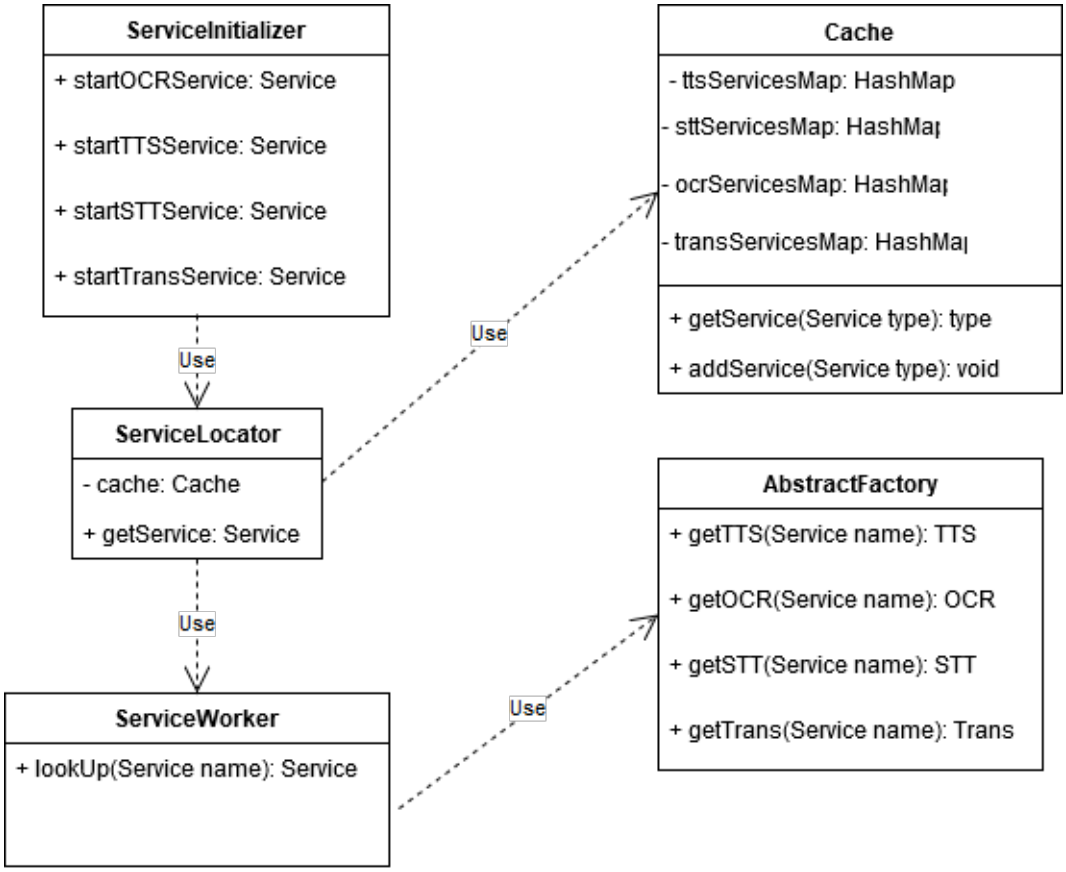

Figure 4.6: Service locator pattern of FLAD core

Singleton design pattern

The singleton design pattern is introduced for single object creation in database connection class. MongoDB database connection pool is handled inside the MongoDB database server. The connection pool size is a variable and it can be set a maximum number of connections according to the requirements. Inside the implementation of database connection class, the database connection is defined to be a singleton object to decrease the resource consumption and increase application performance.

Apart from the major design considerations explained above, every minor design aspects are also considered in implementing the framework. Java collections which are natively thread safe has been used in the implementation processes. For example, Vectors has been used instead of ArrayLists in implementations. HashMap which is not thread safe were synchronized using programming implementations in some situations. This system architecture deals with high concurrency and design considerations introduced to support concurrency issues as well.

4.3 System design and architecture of the FLAD front end Application

FLAD front end consists of FLAD Console and FLAD Admin Panel. Both of these main components are designed using several front end engineering concepts.

ReactJS and Redux are used in the development process of FLAD front end. FLAD front end design based on the components which are identified as the basic building block of React application. Every React component is like a small system that operates on its own. It has its own state, input, and output.

React components pass data between components as props. Prop is defined as a property of React component's element which can fetch and use inside the component logic. The whole React tree may have a store object which is accessible by every component.

React has two main concepts when considering the component design perspective. The first is Presentational components and the second is Container components. Both of these components are designed for specific tasks. FLAD front end contains several presentational and container components which will be discussed in chapter 5. These components define part of the application and available to use anywhere inside the application. This improves the reusability of the application.

Presentational components concern about how things look. These components do not have dependencies on the rest of the application, such as Redux actions or stores, they just handle the data displaying aspect.

Container components concern about how things work. These components may contain both presentational and container components inside but usually, do not have any DOM markup of their own except for some wrapping divs, and never have any styles. Container components have the ability to provide the data and behaviour to presentational or other container components. This type of components often considered as stateful, as they tend to serve as data sources [23].

This above design consideration provides better separation of concerns. It makes easy to understand the application and UI by writing components in this approach. This design techniques also provide better reusability. It is designed to use the same presentational component with completely different state sources, and turn those into separate container components that can be further reused.

4.3.1 Component Composition

The design of the front end is built using React components. These components can contain other presentational components or container components. This takes the design process of the front end application into a completely simplified architecture by providing the combined components structure. The initial step of designing the component hierarchy of FLAD is deviding the UI into a set of components. The basic component separation of FLAD Console is displayed in Figure 4.7.

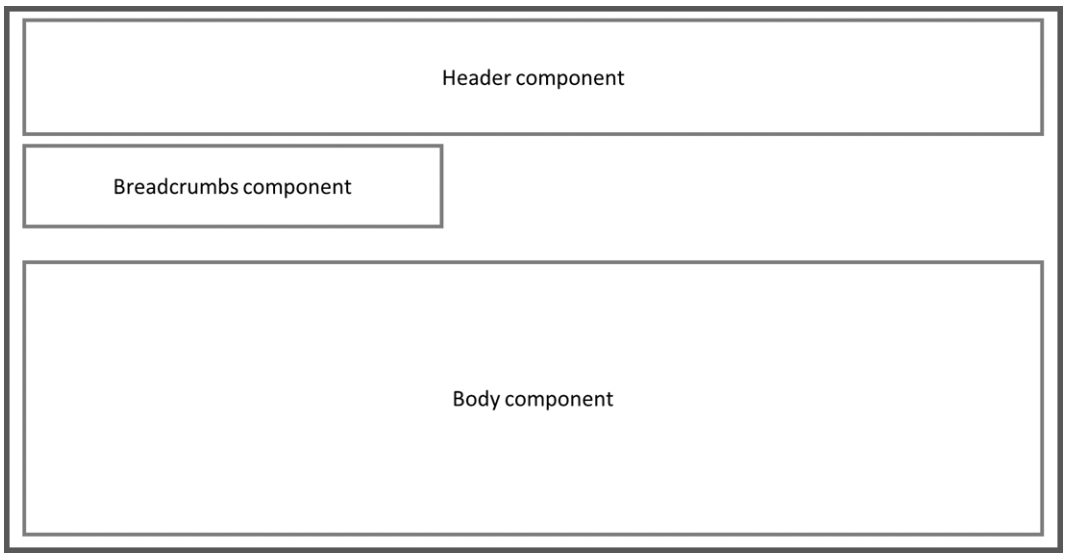

Figure 4.7: Basic component separation of FLAD Console

App component is created on top of the above components. Other child components will be pushed into React component tree based on the routes assigned by the React Router. The simplified React element structure can also be displayed as below.

Listing 4.1: React element structure of FLAD

```
<App>
  <HeaderComponent>
    <BreadcrumbsComponent>...</BreadcrumbsComponent>
    <BodyComponent>...</BodyComponent>
  </Header>
</App>
```

4.3.2 Higher-order components

High-order components are similar to decorator design pattern. It is wrapping a component and attaching some new functionalities or props to it [24].These components add functionality to another object dynamically while they are at the rendering state. It can be used to enhance the behaviour of the component without requiring the author to reopen its class. The higher-order component allows achieving the control on the input and the data which is required to send as props.

4.3.3 Dependency Injection

The components involved in front end design have dependencies. A proper management of these dependencies is critical for the success of the project. React has reusable and testable UI components. This type of components do not require the dependency injection directly.

High-order components handle the activity of passing data into the child components representing the behaviour of dependency injection. FLAD passes from simple strings to action methods to the child components from high-order components in order to keep the modularized and organized structure of the front end design.

4.3.4 File Hierarchy

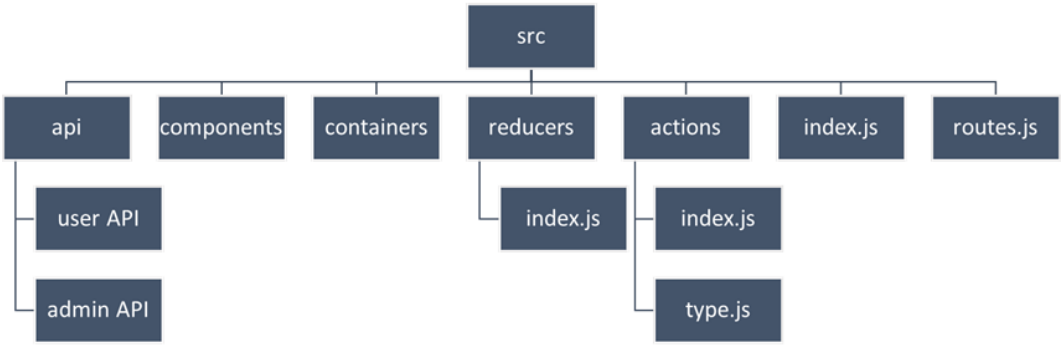

Figure 4.8: FLAD front end file hierarchy

FLAD front end file hierarchy follows the above structure in Figure 4.8. This structure supports the modularity of React components defined in the components and the containers folders. The purpose of actions, components containers and reducers is discussed in chapter 5. The API is the additional area which is included in this structure. It defines the functions which are capable of fetching the data from the FLAD back end REST API. These functions are triggered in actions and dispatch the fetched data into the reducers.

There is another file hierarchy available for the React application development. This second structure is based on components. It consists action, reducer and component design in a separate folder in order to improve the code visibility and efficient component rendering. This is more suitable for the simple application which uses few number of components. The front end component designing of FLAD uses several child components at once in each container due to the complexity of the component structure. FLAD suits the selected file structure shown above than this component based file structure due to the complexity of component structure.

4.3.5 Unidirectional data flow

FLAD's data flow is designed based on the Redux data flow structure which is strictly unidirectional data flow. This means that all data in an application follows the same lifecycle pattern, making the logic of FLAD more predictable and easier to understand. It also encourages data normalization and does not allow to end up with independent and multiple copies of same data that are unaware of each other. The data flow structure of FLAD is displayed on Figure 4.9.

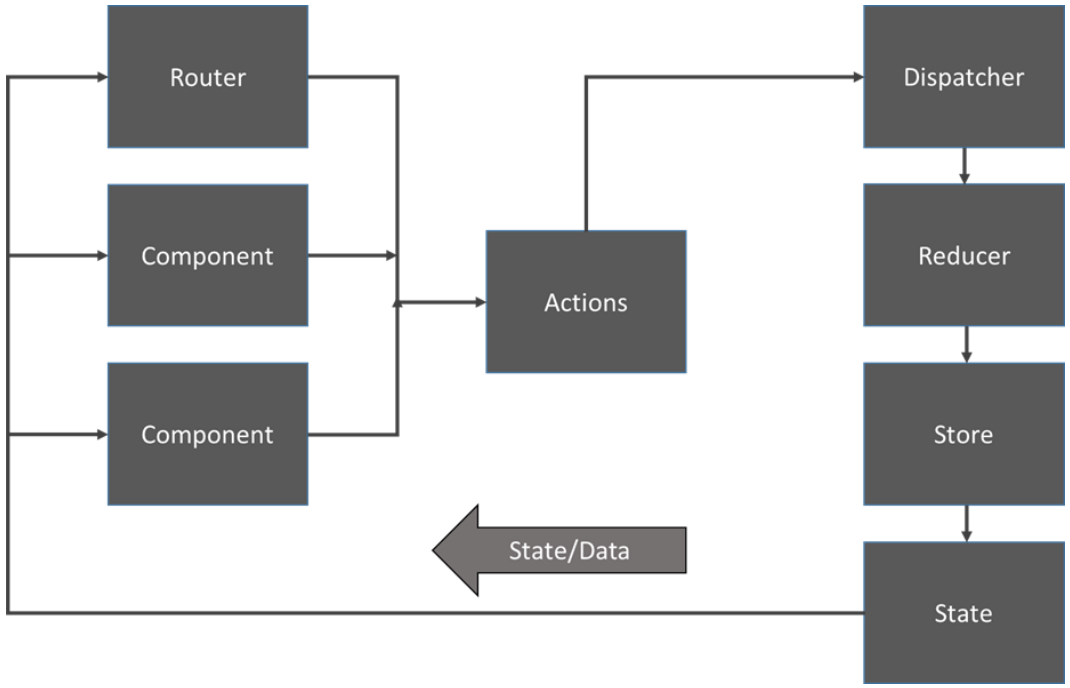

Figure 4.9: The data flow structure of FLAD

React components contains several actions which are capable of dispatching the behaviour of intended action to the reducers. This binding is based on the action types defined in the application. The action types of creating a new project action are below.

```
export const ADD_PROJECT_START = 'ADD_PROJECT_START'
export const ADD_PROJECT_ERROR = 'ADD_PROJECT_ERROR'
export const ADD_PROJECT_SUCCESS = 'ADD_PROJECT_SUCCESS'
```

Each action has a start, error, and success as its main states. These states define the state of the application while rendering occurs. This also supports the front end view manipulation process. Once an action is triggered the state of action is transformed into the start state. This state user waits until the response is received. This state of the response can be either success or error. The view will be re-rendered depending on this state. This method also improves the debugging process of the FLAD.

4.3.6 Usage of ESLint

JavaScript is an interpreted language and it does not provide details about the errors yet to occur until the time of running the program and it hits the error-prone code. This behaviour leads wastage of time and hampers productivity. ESLint introduces the solution to reduce this problem to some extent by providing various error tracking methods. In addition to that ESLint fetches the mistakes by enforcing consistent standards and best practices [25].

The ESLint rule set used in the FLAD can be found in the appendix.

4.4 Summary

This chapter discussed how the front end which is the FLAD Console and the Admin Panel and the back end which is the core of the system is designed. It is mentioned here how back end is divided in to Communication, Structure and Deployment architectures considering different functionalities of the system. Further, the system design and architecture of front end of the system that consists of FLAD Console and Admin Panel is discussed in this chapter.

Chapter 5

Implementation

5.1 Introduction

This chapter discusses how the solution is provided, development approaches, tools, technologies, and decisions taken in implementing the FLAD system. The technology stack and tools used throughout the entire development process is discussed. This chapter is mainly divided into two sections. The first section describes the overall process to show the approach for our solution and the next section describes back end and front end development details.

5.2 Overall process

FLAD console provides the entry point to the application developers to utilize the services provided by the framework. Developers who are interested in the development of the linguistic applications required to grant the authentication via Google Sign-In to access the features of FLAD Console. After being authenticated, users will be able to create a new service API project and configure it by specifying the linguistic services they require in the project or view existing service API projects created previously. The successful service API project submissions will redirect to the view page of the newly created application. Here, users will be provided with application secret key and application Id of service API project. Application secret key and its implementation on FLAD is discussed in Section 5.3.3. In addition to that, users will be provided with a generated documentation for the selected service API project.

Users are given the Access Token which is made using the combination of Application Id and Application Secret Key. This Access Token is required to access the created project using REST API calls. A POST request with the following format should be sent to obtain an Access Token.

Listing 5.1: A POST request to obtain the Access Token

```
Request URL : /api/v1/token
Request body:
{
    "appId" : "sample-tts-app-ljxxxxswx-xxx-sxbnz",
    "appSecret": "hxxxoisxxxxpdf34ds"
}
```

The system will provide an access token in a JSON response as follows with the status code of 200 for a successful request.

Listing 5.2: Response with the Access Token

```
Response body:
{
    "access_token" : "eXXeXAiOXXXXXiLCJhbGciOiJIXXXXJ1",
    "expiresIn" : 20,
    "appSecret" : "7fd1593XXXXX78019XXXXXf142f94XXXXXX"
}
```

Applications which are created only with basic functionalities without adding sequences will be provided with at most four main endpoints. Only the endpoints for the selected services will be activated for each service API project.

All the service requests are required to provide the Access token of the user in the header of the HTTP request. Exact request, response JSON formats are shown in Appendix C under linguistic service names.

The obtained Access Token will get expired after completing the defined number of requests at the obtaining of the Access Token. The response will be provided after the expiration of Access Token as follows with the status code of 401 Unauthorized.

Listing 5.3: Response for expired Access Token

```
Response body:
{
    "status" : "false",
    "msg" :"access token expired",
}
```

The client application should capture the response and request again to obtain the renewed Access Token when the Access Token is expired.

5.3 The back end development

The back end framework is implemented in Java EE and Maven as the software project management and comprehension tool. Java EE is considered to be robust and secure for building enterprise applications. Java supports pure object-oriented concepts and Java EE provide great support for web services. The complete web service support and flexibility was a key point for selection java for the back end development. Java gives the freedom to use a wide variety of servers like Apache Tomcat, WildFly, Jetty server, Glassfish server and etc. Jetty web server is the main web server the system and it is a Java Servlet container developed by Eclipse foundation.

The back end server machine currently running three main servers which are resource extensive. In the development phase, the requirement of a full-featured lightweight server that can be embedded and handles many concurrent requests was a key concern. Jetty server consumes raw resources from the server machine and can be embedded in Maven Project Object Model (POM). Jetty is more developer friendly and supports many developer options and commands. It can be deployed, launched and restarted quickly than other servers like Apache Tomcat or Wildfly server.

The main reasons for using Jetty over a well-known Tomcat server are,

- Jetty is lightweight than Tomcat and consumes fewer resources.
- Jetty is Embeddable which means we can add jetty as a regular dependency to the project. But in Tomcat, we put our application into the server container.
- Jetty can be very quickly launched and restarted. So that it is suitable for a development environment where the server needs to be restarted frequently.
- Jetty has very low maintenance cost over Tomcat which requires more configuration.

The Java EE web API supports JAX-WS to develop SOAP-based web service and JAX-RS API supports RESTful web service. FLAD uses the JAX-RS specification for the REST API implementation. Jersey framework which follows JAX-RS specifications has been used to implement the REST API in our framework. Jersey framework has great documentation, community support and it is developer friendly than other frameworks like Restlet or Apache CXF and it supports a wide range of annotations needed to handle the HTTP requests and responses.

Maven project management tool was used mainly for the dependency management and as the build tool of the project. Maven helps the inclusion of necessary libraries needed for the framework and building and compilation of the project.

Many libraries and frameworks are used in the project to ensure the best performance and robustness of the system and the individual tasks of those libraries are discussed in relevant sections later in this chapter.

5.3.1 Message builder and formatter implementation

Message builder and formatter is one of the main components in the component architecture of the framework as shown in Figure 4.2. The message builder and formatter is a message parser to and from the REST API of the framework.

The framework supports endpoint for the services developed by many vendors which accept different inputs to process and return different outputs. For example, Tesseract OCR needs a Java data object containing the image details while Google Vision API expects a JSON object. In this example Message Builder and Formatter enables a common input format for all the OCR services and defines a common format for all the responses output by the OCR components. This heterogeneity of inputs and outputs of all the linguistic components are handled by the message builder and formatter component.

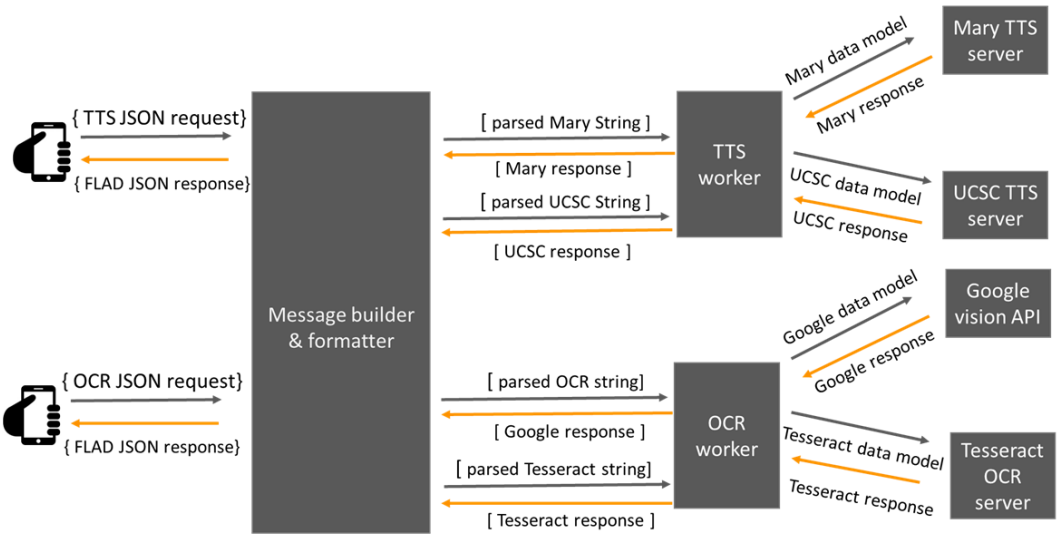

Figure 5.1: Function of Message Builder and Formatter

The requests made to the REST API of the framework is sent to the Message builder and formatter for parsing and formatting. According to Figure 5.1 the JSON body of the request is sent to the Message builder and Formatter component when a TTS request comes to the REST API. The key functionalities of Message builder and Formatter include building data model for all the endpoint services and building a common data model for the responses, creating FLAD custom error messages and JSON parsing.

This component is implemented with the help of GSON[1] library which is an open source Java library to serialize and deserialize Java objects to (and from) JSON.

GSON library is developed by Google and it allows easy conversion of Java objects into their JSON representation and JSON representation to an equivalent Java object. There are other JSON parsers available but they don't fully support Java generics. The framework needed some basic functionalities that are supported by GSON where other similar libraries do not support. FLAD framework is implemented with many custom Java classes with Java collections which needed to be parsed to JSON in one go. For an example, the conversion of a HashMap representation to a JSON object representation is complex with other JSON parsing libraries, but GSON offers some simple parsing techniques which made implementation easy. Further, some of the GSON features which helped in the development of the framework are,

- Provide simple toJson() and fromJson() methods to convert Java objects to JSON and vice-versa
- Allow pre-existing unmodifiable objects to be converted to and from JSON
- Extensive support of Java Generics
- Allow custom representations for objects
- Support arbitrarily complex objects (with deep inheritance hierarchies and extensive use of generic types)

The Message builder is designed to convert the request JSON body into a predefined Java data representation with help of GSON library. The request JSON for the TTS request is different from the request for Translation request. (see Appendix C for JSON request body representations). The mapping of each request JSON to it exact endpoint service is done in this component. An example is shown is Figure 5.2 where MaryTTS server is expecting a query string having all the attributes in the URL string. This component converts the JSON body structure to the required query string and builds the response sent by MaryTTS server to a JSON response before sending to the application user.

[1]https://github.com/google/gson

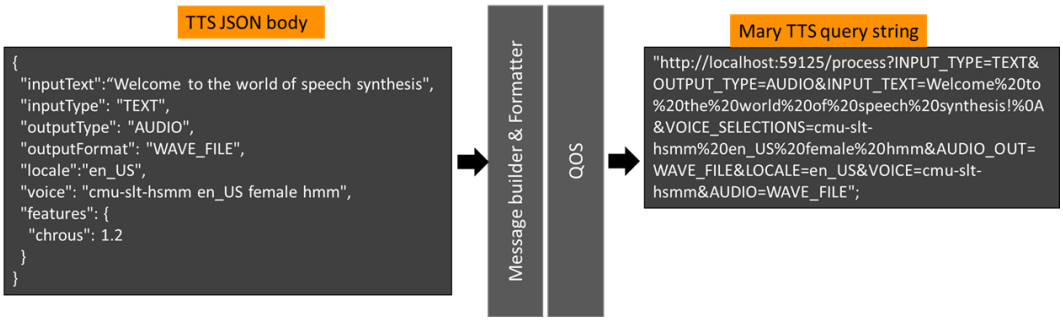

Figure 5.2: MaryTTS message conversion

Mary TTS query string contains UTF-8 encoded strings which are created by the Quality of Service (QoS) layer before sending to the Mary server. The exact functionalities of QoS layer are discussed later in this chapter.

Another main function of this component is to manipulate JSON objects which are the external data representation of the system. FLAD back end contains implemented wrappers around the GSON API and customizes the JSON manipulation to meet the framework's need for this component.

5.3.2 Quality of Service component implementation

QoS component is responsible to add security and optimizations to the framework's functionalities. Any incoming request should go through the validation process first for security reasons and reliability of services.

Basic functionalities of QoS component are,

- Validation (token generation and validation)
- Optimization
- Base64 encoding (for OCR and STT components)
- UTF-8 encoding (for translation and TTS components)

Validator class is implemented inside the QoS component to validate the user requests. This validator class uses a Java library implementation of JSON Web Tokens (JWT)[2]. JWT is used to generate tokens which are used in the validation process of requests in the framework. JWT implementation uses several algorithms to generate a secure web token. RSA512, HMAC512, and ECDSA512 are most commonly available algorithms for use and JWT allows the easy selection of an algorithm in the process. The framework implementation, JWT uses HMAC512 (Hash-based Message Authentication Code) as the algorithm which is faster than

[2]https://github.com/auth0/java-jwt

RSA512. JWT is the commonly used web token generating tool used currently and provides a robust service.

5.3.3 JSON web token generation process in FLAD

FLAD end user; the application developer has two Ids after creating a service API project in the FLAD console.

- Google Id - The identification number given by Google when Google authentication is done.
- Project Id - Automatically generated project Id when a user creates a project in the FLAD console.

The token request is processed in the Validator class and generates a token using the provided two keys and return the token to the user. The user has to keep the web token securely and can use it as the validation key in the future requests. Figure 5.3 shows the token generation process in the framework.

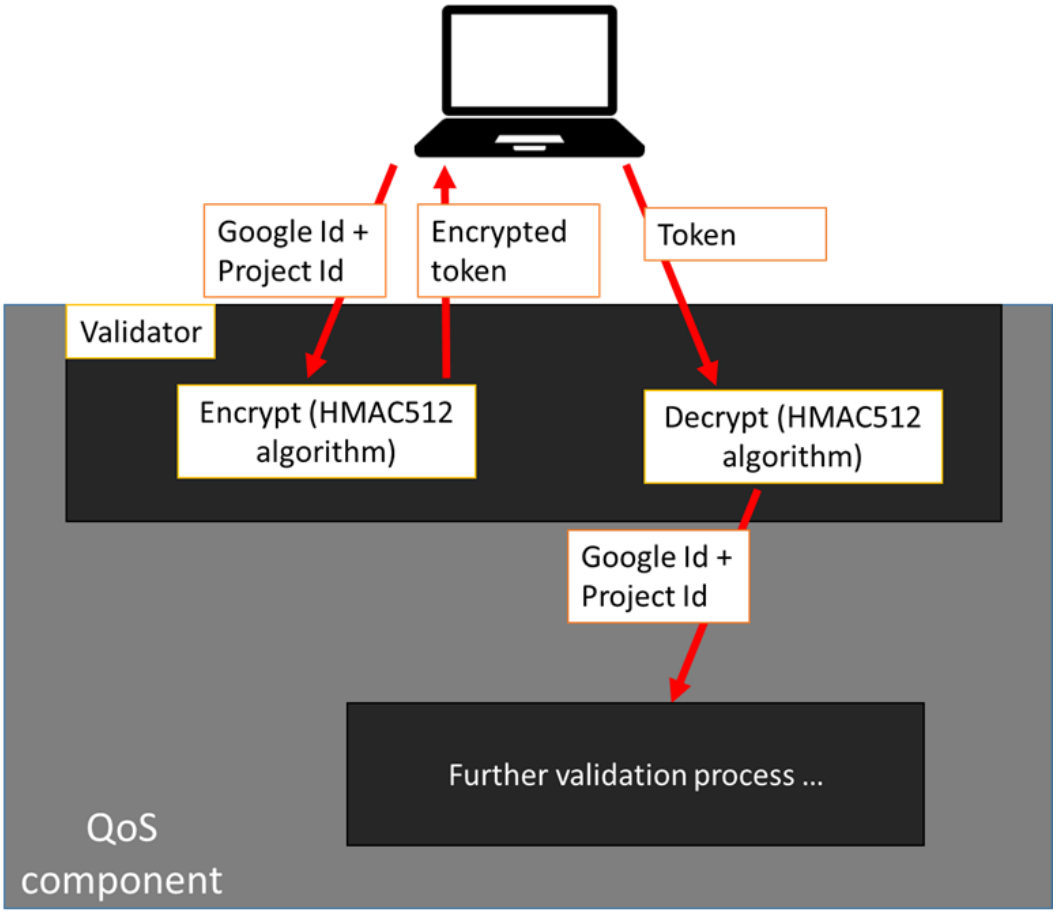

Figure 5.3: Token generation process in FLAD

Second functionality of the validation process is to validate the user request by examining the submitted token. The request validation happens on several levels. Figure 5.4 shows how the request validation happens for simple services. The implementation logic is different when validating complex services such as sequences.
According to the Figure 5.4,

1. The Google Id is used to recognize the validity of the user using a database lookup. User details are stored in a collection of MongoDB. (see Appendix D for database document structures)

2. If the user is a registered user, next the project of the user is identified using the project Id decrypted by the JWT decrypt method. The user can create many projects in the FLAD system and the relevant project is located from the project collection of the user.

3. The specified project contains all the endpoints related to the services selected by the application user in project creation phase using the front end FLAD console. The exact project creation process is discussed later in this chapter. The project of the user is checked for the purchased endpoints. If the user has made a request to an endpoint which has not been purchased the validation fails by returning 403 error code with status forbidden.

4. If all validation steps succeed, the service instance specified in the user project is invoked.

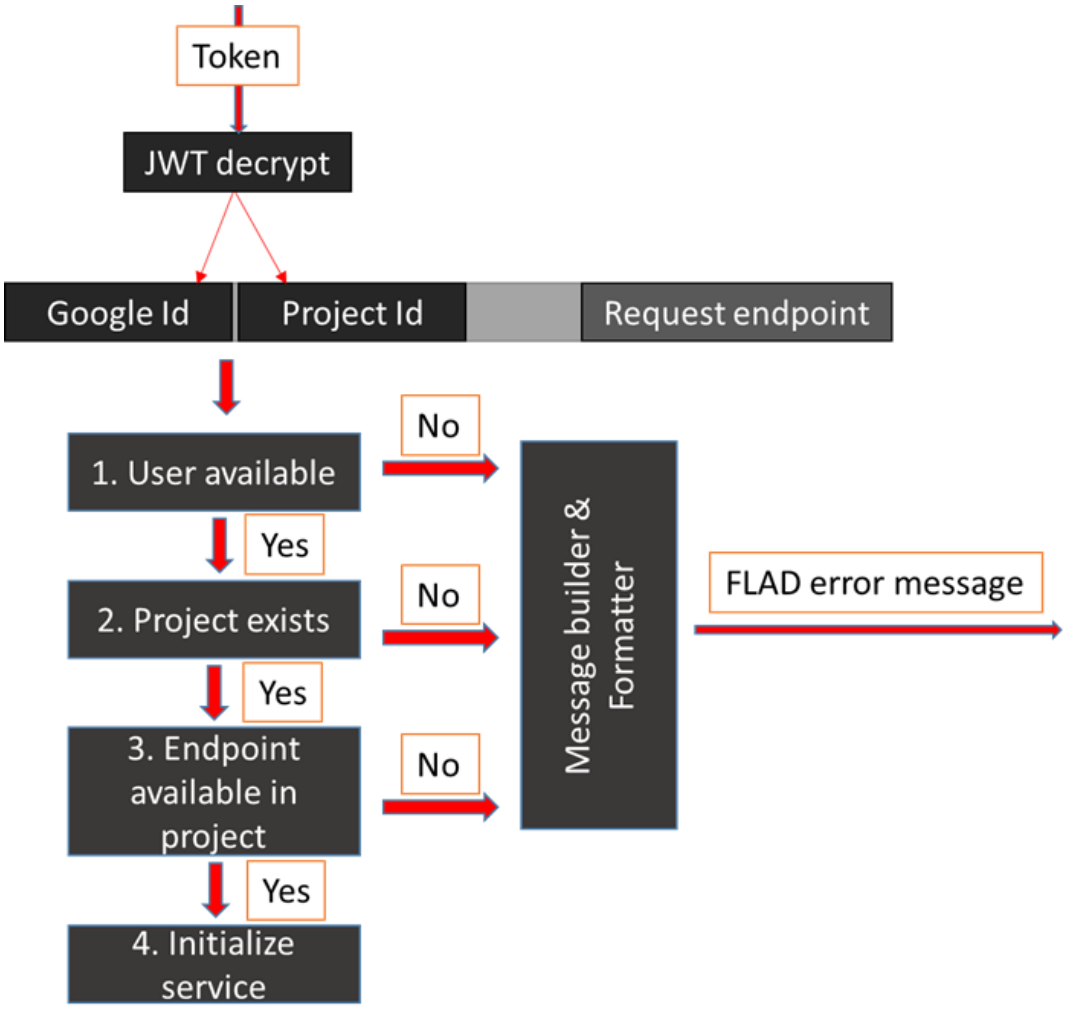

Figure 5.4: FLAD validation for simple service

The other functionality of QoS component is to do optimization in the framework workflow. The use of Service Locator design pattern to provide more efficient service is done at QoS component. The exact explanation of Service Locator design pattern is described in section 4.2.5. The Service Locator design pattern is used to cache the objects in HashMaps in the first creation of the objects and the subsequent requests utilize already initiated objects without creating them which is a time-consuming task in java. The object reuse and caching are done using QoS component.

Meta-data required for the functionality of the framework is stored in the database. The meta-data is fetched from the database when the back end server starts and after the first fetch, the values are cached in Java collections like HashMaps, Vectors for future use. This prevents costly database lookups on subsequent calls.The reason of using HashMaps over other collection is the fast lookup provided by the collection. This avoids number of iterations which is time-consuming and improve the performance drastically.

As in some instances, concurrency issue related to the HashMaps are handled externally using synchronization blocks in Java to avoid unexpected behavior. This adds reliability to the system.

5.3.4 TTS component implementation

TTS implementation of the framework is done using MaryTTS which is an open source java based multilingual Text-to-Speech Synthesis platform. MaryTTS is the widely used TTS component which has a huge community support on Github. MaryTTS has no REST API and it includes a standalone server which needs to be run separately on the server machine and access via service interfaces.

The solution provided by the framework is to expose a simple REST API to the application user while handling the complex configuration and interface calls in the back end. The MaryTTS server is deployed on port 59125 in the server machine and the incoming requests are redirected to the process method of MaryTTS API with required parameters.

Listing 5.4: Input TTS JSON request body

```
{
  "inputText":"We have a clear sky",
  "inputType": "TEXT",
  "outputType": "AUDIO",
  "outputFormat": "WAVE_FILE",
  "locale":"en_US",
  "voice": "cmu-slt-hsmm en_US female hmm",
  "features": {  }
}
```

Listing 5.4 shows the JSON request which is expected in a tts request. This approach simplifies the complex query string expected by MaryTTS server and provides the end user a more programmer friendly JSON structure. These structures are discussed in the FLAD endpoints documentation which is given after a project creation for the developer use. The response is created by the Message builder component as shown in Listing 5.5.

Listing 5.5 shows the response of a tts request. The “response” contains the public URL to the audio file. In the response “executionTime” is the overall time taken to process the request which is 2957 milliseconds.The execution time is highly depended on the bandwidth of the connection.

MaryTTS execution time = MaryTTS local process time + upload time to the cloud

Listing 5.5: TTS JSON response body

```
{
    "other": [
        {
            "executionTime": "3711",
            "locale": "en_US"
        }
    ],
    "response":"https://www.googleapis.com/download/
       storage/v1/b/samplebookstorebucket/o/maryTTSAudio.
       wav?generation=1514361245171530&alt=media",
    "processedAT": "2017-12-27_13-24-05"
}
```

Figure 5.5 shows the implementation approach of the TTS services. Only the relevant components are shown in this for the simplicity of the diagram.

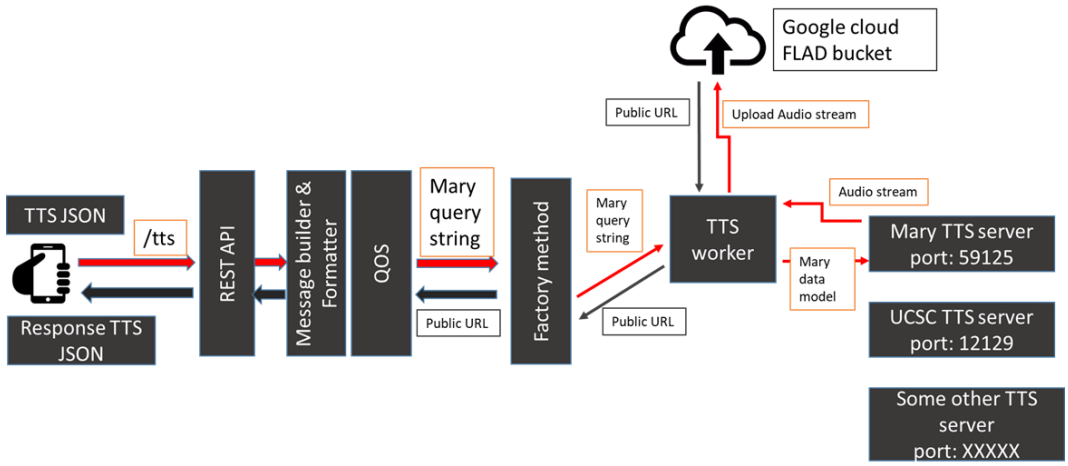

Figure 5.5: Implementation approach of the TTS services

- The factory design pattern is used to find the relevant TTS service and the TTS worker builds the data model relevant to the back end service.
- MaryTTS server returns an audio stream with a successful request.
- TTS worker uploads the audio stream to the preconfigured Google cloud purchased by FLAD framework.
- After a successful upload, the framework requests a publicly accessible URL for the uploaded audio file and returns it to the application user.

- The application user can use the public URL to access the processed audio file.

There are two approaches available when dealing with TTS responses. TTS components return an audio stream to the calling root. One approach is to create an audio file (MP3 or WAV) using the audio stream and store in a public storage. Then send the publicly accessible URL to the user. The second approach is to stream the audio directly to the user where the user can take direct actions over the stream.

5.3.5 OCR component implementation

The framework provides two OCR services to the application user. The following section discusses the tesseract OCR implementation and Google Vision API implementation separately.

Tesseract-OCR Implementation

Tesseract OCR is a package that contains an OCR engine called libtesseract. Tesseract has unicode (UTF-8) support, and can recognize more than 100 languages. Tesseract supports various output formats including plain-text, hocr(html), pdf, etc. Developers can use libtesseract C or C++ API to build applications. Though the libtesseract is written in C/C++ there are wrappers for the libtesseract library for other languages.

FLAD is developed using Java as the programming language and tess4J java wrapper of tesseract has been used to implement the Tesseract component.

Execution time = Tesseract OCR processing time + upload time.

Google Cloud Vision API implementation

Google OCR or Google Cloud Vision API is the second OCR implementation of the framework. It is not a free service and the service has been purchased under the FLAD account.

Listing 5.6: JSON structure of the OCR request

```
{
    "imageFileName":"german.png"
}
```

The OCR endpoint of FLAD is expecting a POST request and Listing 5.6 shows the JSON structure of the POST request. “imageFileName” field contains to the name of the image which is to be subjected to optical character recognition.

The request is sent to the Tesseract engine or Google Cloud Vision API depending on the user preferences. The JSON response of a ocr request is shown in Listing 5.7.

Listing 5.7: JSON structure of the OCR response

```
{
    "other": [
        {
            "executionTime": "3481",
            "locale": "de"
        }
    ],
    "response": "Hast du das Brechern meines Herzens
       gehrt?",
    "processedAT": "2017-12-27_13-18-48"
}
```

In Listing 5.7 “locale” is the language detected in the image content and the “response” contains the content of the image.

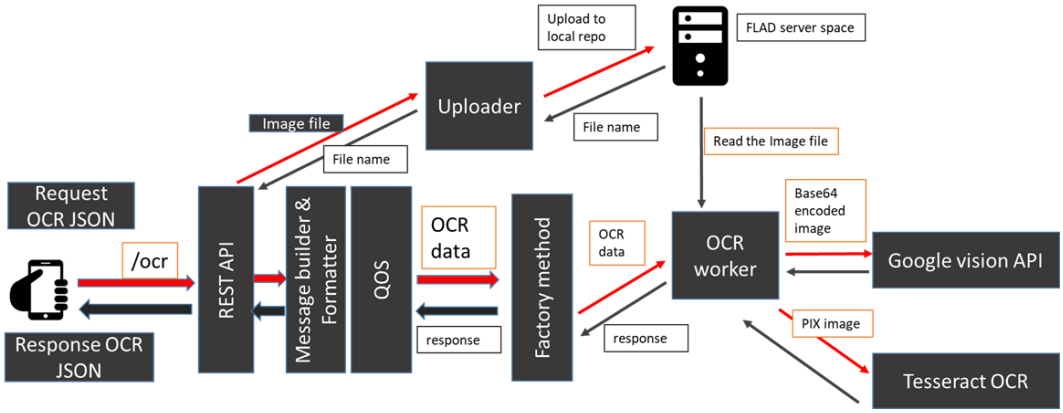

Figure 5.6: Implementation approach of the OCR services

Figure 5.6 shows the flow of the OCR process. First, the image file is uploaded to the server using the /upload endpoint that provided by the FLAD. The request is made to a standard format by the Message builder and Formatter. The request is sent to the OCR worker through the factory method. OCR worker reads the image from the FLAD server space and executes the OCR service. If the user is using Tesseract OCR, then the image is converted to a pix image and it processed using the Tesseract engine. If the user is using the Google Vision API’s service then the image is converted Base64-encoded format and it is processed using the Google OCR which also contains the Tesseract engine. The response which is either generated from Google Vision API or Tesseract is sent to the OCR worker

followed by factory method and Message builder and formatter. Message builder and formatter create a JSON response as shown in the Figure 5.6.
Execution time = Google vision API access time + OCR processing time + upload time.

5.3.6 Translation component implementation

Google Translation is used as the translation endpoint of the framework. The Google Translation is a paid service and the service is purchased under FLAD account for testing purposes. The translation endpoint is simplified to expect a POST request with a simple JSON body as shown in Listing 5.8.

Listing 5.8: JSON structure of the Translation request

```
{
        "inputText": "Hello How are you?",
        "target": "sp"
}
```

The phrase to be translated is denoted by "inputText" and the target language which the text should be translated to is specified by "target". Here the request is to translate "Hello how are you?" to Spanish.

The request is sent to the REST API of Google Translation and the response is created by the Message builder component as shown in Listing 5.9.

Listing 5.9: JSON structure of the Translation response

```
{
    "other": [
        {
            "executionTime": "185",
            "locale": "en"
        }
    ],
    "response": "Hola como estas"
    "processedAT": "2017-12-27_13-27-00"
}
```

The locale shown in Listing 5.9 is "en" since the detected input language which is English and "executionTime" is how much time taken to process the request which is 687 milliseconds.

Execution time = Google Translate access time + Translation processing time.

5.3.7 STT component implementation.

Two speech to text endpoints are initially provided by the FLAD system. One is using CMU Sphinx library and the other one is using Google Cloud Speech API. Users can select the preferred STT endpoint in the service API project creation phase.

CMU Sphinx implementation

CMU Sphinx has been selected as the STT service because it consumes fewer resources than other free STT libraries and it is easy to use. CMU Sphinx library can be added as a dependency to the project using Maven. As it is developed using Java, we can communicate with the library using Java functions. The audio file expected by CMU service has to be preprocessed by the QoS component. QoS component converts the uploaded audio file to a WAV file with Mono channel having a sample rate of 16000 Hz.

Google Cloud Speech API

Google Cloud Speech API provides a web service to convert speech to text. FLAD gets the Google Speech API's service through their exposed web service. Google accepts Base64-encoded audio files.

The framework exposes STT service as a REST API so that the user does not have to do complex configuration and installation task. The user can select the preferred STT endpoint for their application. They can change the used STT endpoint to another in the future easily because FLAD handles the configuration part internally and exposed a friendly and consistent API.

FLAD's STT endpoint accepts a POST request from users which contains a simple JSON body as shown in Listing 5.10.

Listing 5.10: JSON structure of the STT request

```
{
      "audioFileName":"test3.wav"
}
```

When FLAD receives the request it uploads the specified audio file to the server. The uploaded file is passed to CMU Sphinx library or Google Speech API according to the user preference. The response that is sent by CMU Sphinx library or Google Speech API is formatted using Message builder and Formatter is shown in Listing 5.11.

Listing 5.11: JSON structure of the STT response

```
{
    "other": [
        {
            "executionTime": "5099",
            "confidence": "0.986863"
        }
    ],
    "response": "hello how are you",
    "processedAT": "2017-12-27_13-29-04"
}
```

The response contains "confidence" which the STT returns as the confidence value and the "response" is the output text of the content of the audio file.

Execution time = STT processing time + Audio upload time

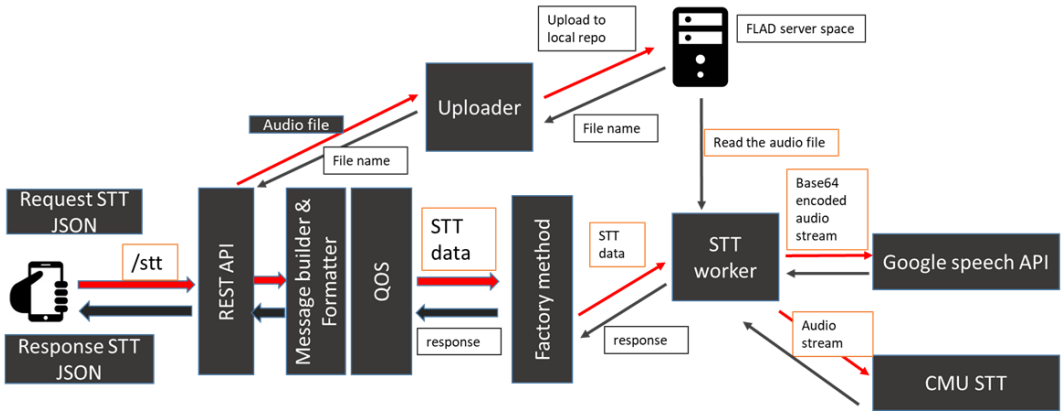

Figure 5.7: Implementation approach of the STT services

Only the most relevant components are shown in Figure 5.7 for the simplicity of the diagram. As the first step, the audio file is uploaded to the server. The name of the audio file is sent back to FLAD core system. Using the factory design pattern the relevant STT service is dynamically invoked and the STT worker builds the data model relevant to the back end service. STT endpoint returns the response text after completion of processing the audio file. The response text is obtained by Message builder and Formatter. It adds the necessary fields and headers and builds the HTTP response. The REST API sends the response to the relevant client.

5.3.8 The sequence implementation

Sequences are a serial combination of several simple linguistic endpoint services. Sequences are introduced to reduce the overhead of sequential web service calls

needed to perform a complex linguistic application need.

For an example, Assume a user wants to read the content of a German signboard and get the English translation. Then the application developer does not need to send two separate calls to the framework as one for OCR and one for Translation, Instead, he can perform the request using the sequence number which can be obtained after adding relevant sequence to the service API project to perform that task in a single web service call.

There are six possible sequences defined in the framework which can be made using the four linguistic components STT, TTS, OCR, and Translation. In FLAD, there are six pre-defined different JSON requests to make the sequence execution easy.

The user can select which sequences he needs for the application in the application creation process in FLAD console. The user can send a POST request with the required JSON input to http://localhost:8080/api/v1/seq/[seqNo] with the relevant sequence number.

Sequence 0 (OCR $\longrightarrow$ TTS)

This sequence takes an image as the input and produce content of the image as a speech.

Listing 5.12: JSON structure of the Sequence 0 request

```
{
    "ocr":{
        "imageFileName":"german.png"
    },
    "tts":{
        "locale":"en_US",
        "voice": "cmu-slt-hsmm en_US female hmm"
    }
}
```

As shown in Listing 5.12 it gives the audio output in English voice of the input image file.

Sequence 1 (OCR $\longrightarrow$ Trans)

This sequence converts the content of an image to a given language.

Listing 5.13: JSON structure of the Sequence 1 request

```
{
    "ocr":{
        "imageFileName":"german.png"
    },
    "trans":{
            "target": "en"
    }
}
```

This request converts the content in the german image file to English and output the English content.

Sequence 2 (OCR $\longrightarrow$ Trans $\longrightarrow$ TTS)

This sequence converts the content of an image and output the speech in that translated language.

Listing 5.14: JSON structure of the Sequence 2 request

```
{
    "ocr":{
        "imageFileName":"german.png"
        },
    "trans":{
        "target": "en"
        },
    "tts":{
        "locale":"en_US",
        "voice": "cmu-slt-hsmm en_US female hmm"
    }
}
```

Sequence 3 (Trans $\longrightarrow$ TTS)

This sequence translates a given string to a specified language and gives its speech in the translated language.

Listing 5.15: JSON structure of the Sequence 3 request

```
{
    "trans":{
        "q": "Hej det har ar kul och gldje. Livet ar
           inte bra",
        "target": "en"
    },
    "tts":{
        "locale":"en_US",
        "voice": "cmu-slt-hsmm en_US female hmm"
    }
}
```

Sequence 4 (STT $\longrightarrow$ Trans)

This translate the speech to a given language and output the translated text.

Listing 5.16: JSON structure of the Sequence 4 request

```
{
    "stt": {
        "audioFileName": "test3.wav"
    },
    "trans": {
        "target": "si"
  }
}
```

Sequence 5 (STT $\longrightarrow$ Trans $\longrightarrow$ TTS)

This translate the speech to a given language and output the translated text back as speech in translated language.

Listing 5.17: JSON structure of the Sequence 5 request

```
{
    "stt": {
    "audioFileName": "test3.wav"
  },
  "trans": {
    "target": "de"
  },
  "tts": {
    "locale": "en_US",
    "voice": "cmu-slt-hsmm en_US female hmm"
  }
}
```

5.3.9 Log mediator component

Log mediator is implemented to perform all the logging of the transactions in the framework. Apache Log4j2[3] has been used to implement the logging mediator. Apache Log4j is a Java-based logging utility. The reasons for using Log4j over some other frameworks like Logback are, Improved performance, Automatic Reloading of Configurations, and Advanced Filtering facility provided by Log4j [26].

5.3.10 Error handling

Clients use REST API to communicate with the web service via HTTP requests and responses. Hence the information delivered to and from the server should be self-descriptive. For example, the access token must be included in all the requests send to FLAD. If the access token is missing in the request header the system should provide an error message informing the access token is required in the request. When the client gets this descriptive error message he can identify the mistake and quickly resolve. The error message should be self-descriptive so that the client will be able to resolve the issues related to the requests.

In addition, a REST API should use standard HTTP error codes in the responses. These error codes are helpful for the client to identify the server errors and client errors. The exact error codes used by FLAD is shown in the Appendix E.

The FLAD system implementation uses custom exception handling mechanisms to define exceptions in the workflow. These custom exception handlers provide

[3]https://logging.apache.org/log4j/2.x/

descriptive error messages customized to the nature of the exception. ExceptionMapper of Jersey library has been used to define custom exceptions in the REST API. All the exceptions are defined in a one Java package with self-descriptive messages about the error. The exception handler generates the error message and the relevant HTTP error code. This data is passed to Message Builder and Formatter and it creates the JSON response by adding the error message in the response body and error code in the response header. Then the HTTP response is passed to the client.

Listing 5.18 shows an error response related to an invalid request. The user has issued a request to /stt endpoint with correct request structure but, the user has not picked the STT service from FLAD system.

Listing 5.18: An error response related to an invalid request

```
{
    "message": "Not eligible for the service",
    "status": "Forbidden",
    "errorCode": "403"
}
```

5.4 The front end development

The current trend in web development industry is developing Single Page Applications(SPA) over Multiple Page Applications(MPA). Single-Page Applications (SPAs) are Web apps that load a single HTML page and dynamically update that page as the user interacts with the app.
SPA advantages over MPA [27],

- Faster page loading times.
- Improved user experience because the data is loading in the background from server.
- No need to write the code to render pages on the server.
- Decoupling of front end and back end development.
- Simplified mobile development; you can reuse the same back end for web application and native mobile application.

By considering the advantages of SPA, the front end is designed as a single page application. The front end was developed separately from the back end. The communication between front end and back end happens via REST API calls.

In the field of front end development, many JavaScript frameworks emerge day by day. Angular JS, React JS, and Vue JS are the mostly used front end frameworks. The most suitable framework for our project was selected by considering the advantages and disadvantages of each framework. (See Appendix Ffor the comparison.

By considering the advantages of SPA, the front end is designed as a single page application. The front end was developed separately from the back end. The communication between front end and back end happens via REST API calls.

In the field of front end development, many JavaScript frameworks emerge day by day. Angular JS, React JS, and Vue JS are the mostly used front end frameworks. The most suitable framework was selected by considering the advantages and disadvantages of each framework. See appendix for the comparison.By considering the performance, developer friendliness, and group members' previous experience we decided to use React as the front end framework.

React JS is a front end development framework developed and maintained by Facebook. More details about React is discussed in Chapter 2 under background study. React runs on NodeJs server. Node Package Manager(NPM) is used to install and maintain the packages. Webpack, Babel, React hot loader, and Nodemon are some necessary tools used for React development.

Webpack takes modules with dependencies and generates static assets representing those modules. This technique keeps initial loading time to a minimum. Different modules can be independently implemented as Webpack combines them into static files automatically [28]. Babel is a JavaScript compiler. It compiles React JSX into browser-compatible JavaScript [29]. React Hot Loader is a plugin that allows React components to be live reloaded without the loss of state. It works with Webpack and other bundlers that support both Hot Module Replacement (HMR) and Babel plugins [28]. Nodemon is a utility that will monitor for any changes in your source and automatically restart your server and it is perfect for development environments [30].

Google Material-UI is used to implement the user interface and its components. It provides React components that implement Google's Material Design [31]. Material UI is one of the first UI kits for React. It provides a grid system to develop mobile-friendly user interfaces.

Maintaining the application state is difficult in a single page application. Redux JS is used to maintain state in SPA. Redux is a predictable state container for JavaScript apps [21]. The state is stored in Redux Store.

A minimal React template project available on Github is used to initialize the front end implementation. React has many dependencies need to be installed, using a pre-configured template reduces the configuration cost.

The front end was developed in two phases. In the first phase, the client functionalities were developed. In the second phase, the administrator functionalities were developed. Next section briefly discusses how the client's functionalities were implemented and the tools that were used.

5.4.1 How front end communicates with the back end

The front end of FLAD runs on Node JS which is based on an event-driven non-blocking I/O model. It means the operations do not wait for I/O to complete. So that when front end sends a request to the back end to get data or put data, the thread in execution does not wait until it gets the response. Instead the thread in execution handovers the request to a worker thread and execute the next instruction. To handle this situation Redux Async Actions [4] is used with Redux Thunk middelware[5].

5.4.2 How Redux is used

The application state is stored in a JavaScript plain object. This object cannot be modified directly. This limitation helps to avoid bugs as different parts of the code cannot change the state directly from different places.

An action is used to modify the state.The action is also a JavaScript object which describes what happened. JavaScript functions are used to connect the actions with the state. In Redux these functions are called Reducers. A part of the application state is shown in Figure 5.8. The image shows project details of the project named "test".

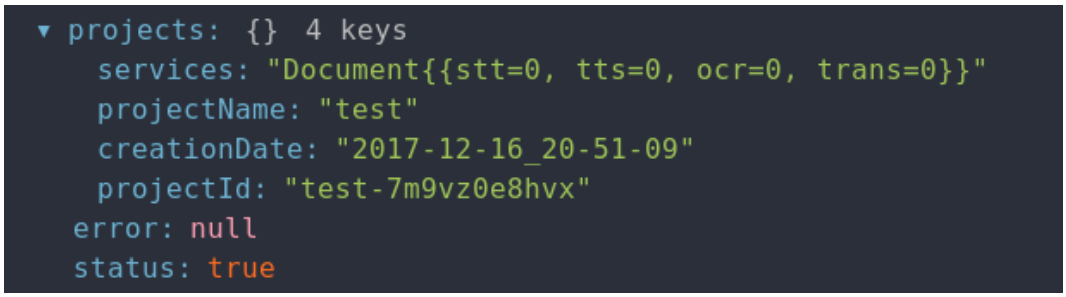

Figure 5.8: A part of Redux application state

Assume this project needs to be edited. Then,

1. The User enters the data to "Edit Project" form and clicks Submit.

[4]https://redux.js.org/docs/advanced/AsyncActions.html
[5]https://github.com/gaearon/redux-thunk

2. When Submit is clicked, edit project function is called with the project data as parameters.

3. Edit project function dispatches three Redux actions.

4. First “EDIT PROJECT START” action is dispatched. This action modifies Redux state by setting the fetching attribute to True. The fetching attribute is used to identify that a back end API call is going to start. We use this attribute to show the user that the system is processing the submitted data.

5. front end sends a POST request to back end API with project data that needs to be edited using a JavaScript promise.

6. back end gets the data and updates the relevant project in the database. Then it sends a response to the front end with the new project data.

7. When the front end receives the success response with the data, ”EDIT PROJECT SUCCESS” action is called. This action modifies Redux state by setting ”fetching” attribute to False and ”fetched” attribute to ”True”. It also updates project data in the Redux store to the latest data received from the back end. This action also sets ”notifications” attribute in the state to ”Project Successfully Edited”. It triggers a pop up in the user interface to indicate the user the project has successfully edited.

8. If the project's edit request failed, the promise catches that error and dispatches the ”PROJECT EDIT FAILED” action. This action sets the ”error” attribute to the error message received from the back end. This action also sets ”notifications” attribute in the state to ”Project Editing failed”. It triggers a pop up in the user interface to indicate the user the project edit request has failed.

The user experience and performance of the front end is improved with the help of Redux. It has also eased the development as it acts as the single source of truth.

5.4.3 How the application is structured

The application is structured and refactored to make the maintenance easy. The accepted React-Redux application directory structure is used in the front end. It has helped to write clean code. The directory structure is as follows,

- actions/ - All the Redux actions are defined in this directory.

- api/ - The functions used to call back end api are defined in this directory.

- components/ - All the UI components are implemented in this directory. These components act as Views. We have split different UI components into different files such as NotificationsComponent, AppBarComponent, and EditProjectFormComponent. These components have their own local state and functions to manipulate the local state.
- containers/ - The files in containers directory have the business logic. It renders the relevant UI components according to the business logic. The objects in Redux state is used in containers and the functions to call Redux actions are also defined in containers.
- reducers/ - The reducers which binds Redux actions with Redux state are defined in this directory.
- styles/ - scss style files are stored here.
- index.js - This file served as the entry point to the application.
- /routes.js - All the routes/paths and the containers should be rendered in each route is defined in this file. This file is also used to check authorization.

5.4.4 How authorization is handled

The user should have permission to access the different routes/paths of the applications. React Higher-Order Components (HOC) have been used to implement authorization. Briefly, a higher-order component is a function that takes a component and returns a new component based on the logic defined in the function [32].

The attributes “isLoggedin” and “isAdmin” in Redux state are used to implement authorization. When the user successfully logged in ”isLoggedIn” flag is set to True. The authorization component checks this flag when the user attempts to access the routes/paths of front end UI. The authentication HOC renders the relevant UI component if the ”isLoggedIn” flag is True otherwise it renders the welcome page. When the user logs out, ”isLoggedIn” flag is set to False then the authentication HOC redirects the user to welcome page.

When admin user logs in “isLoggedIn” and “isAdmin” flag is set to True and admin can access the admin routes.The authorization is handled within the topmost component (in routes.js) to ensure the security of the application.

5.4.5 Authentication Component

The overhead of authentication functionality was avoided by using Google Sign-In. Google Sign-In is a secure authentication system that reduces the overhead of login for users, by enabling them to sign in with their Google account with same account they already used with Gmail, Play, Google+, and other Google services [33]. It has increased the ease of use as the users can register with the system by signing in to their Google account. Users do not have to provide their personal details to our system. It will increase the usability of FLAD. It provides a one click authentication for the user which make the system more user-friendly.

When a user login to the system for the first time, his email address, user image url and Google Id is saved in the MongoDB database. This Google Id is used to uniquely identify the user.

A "LoggedIn" flag is used to identify whether a user has logged in or not. If a user successfully logs in to the systems, his authentication details are saved in Redux application state and "LoggedIn" flag is set to True.

The application state is stored on web browser's local storage to make the system more available. So that even if a user accidentally closes the browser he/she can resume to the same state when he/she opens the browser again. The system deletes browser's local storage when the user logs out and set the "LoggedIn" tag to false. This redirects the user to the Welcome page.

5.4.6 Project Component

Different UI components have been used for Project View, Create, Edit and Delete functionalities. Their operations are handled in relevant containers. See Appendix G for UIs.

5.4.7 SummaryBox Component

SummaryBox is a reusable dynamic React component implemented for Admin-Container of the FLAD front end. This component supports for dynamic icons, text, colors and functions which is passing through props. In addition to that this an is atomic react component in FLAD which does not use another custom react components inside this component. SummaryBox uses the higher-order component created by withStyles to inject an array of styles into the DOM as CSS.(see Appendix G.4)

5.4.8 EnhancedTable Component

Enhanced Table Component is the primary large data display method in FLAD Admin container. This dynamic component has been used in several places such as users, projects, requests in FLAD Admin panel. This component consists of EnhancedToolbar component and TableBody component. These above components render separately in order to provide the completeness of the EnhancedTable component and its actions.

EnhancedToolbar component behaves under two states, search state and actions state. The list of the data in the TableBoby component changes according to the change of the searchString in the search state as shown in the Appendix G.5. Next, action state contains the actions which can be performed under each table item. Appendix G.6 shows the view change of this state. These actions are sent from the Parent container of each Table according to the type of the data that is being used. System actions for users data and project data are below.
Actions for users data

- View user profile
- View projects
- Ban users

Actions for project data

- View project
- View user
- Pause project

The above actions are dynamic and all actions should be passed in the format below.

Listing 5.19: Actions format

```
{
    'id': 'action-1',
    'text': 'View Project',
    'icon': 'eye',
    'action': this.handleViewUser.bind(this)
}
```

The attributes of this format are,

- id : for the handling process of iterative elements.

- text: the text to display when mouse over on action icon as tool tip text.
- icon: the icon which should be displayed as icon in the action bar. These icons are imported from the material-ui icons package.
- Action: the action should be performed when user click on the action.

5.4.9 UsageChart Component

This chart is designed to display the requests handled by the FLAD system within a timeframe. (see Appendix G.7). UsageChart component contains material UI build-in elements for the container usage and recharts[6].

5.4.10 Breadcrumbs Component

Breadcrumbs component provides the information about the current location of the user within the FLAD Console. This component is designed using chips which is a Material UI component. The data which should be converted into breadcrumbs representation should be passed as prop named data as in the format below. This array will be used in the rendering of breadcrumbs component.(see Appendix G.8).

Listing 5.20: Data array passed to Breadcrumb

```
[
    [1, 'Home', '/admin'],
    [2, 'Users', '/admin/users']
];
```

5.5 Summary

This chapter described the solution provided along with the development approaches, tools, technologies and decisions taken in implementing the system. This chapter also discussed the overall process of the system which describes how a user should use the system. Further, this chapter discussed how the linguistic components are integrated in this system and how the system provides its services to the users. In addition to that, some other important components such as Message Builder and Formatter Component and Quality of Service Component are discussed in this chapter. This chapter also discussed the front end implementation including how front end communicates with the back end of the system.

[6]https://github.com/recharts/recharts

Chapter 6

Evaluation

6.1 Introduction

This chapter explains the achieved goal and objective using qualitative and quantitative measurements. The latter part of the chapter contains the testing process and the limitations of the project. The FLAD system evaluation was conducted under three main section. Architecture wise evaluation, Service wise evaluation, and Software wise evaluation.

6.2 Software wise evaluation

6.2.1 FLAD console loading time

FLAD console is designed using REACT framework and the evaluation is done considering the rendering time of components and the memory consumption.
Testing environment,

- The internet download bandwidth = 8Mbps
- The upload bandwidth = 5Mbps
- Average Ping = 70ms
- Firefox Quantum web browser

Figure 6.1 shows the rendering time in milliseconds with the increase of number of components.

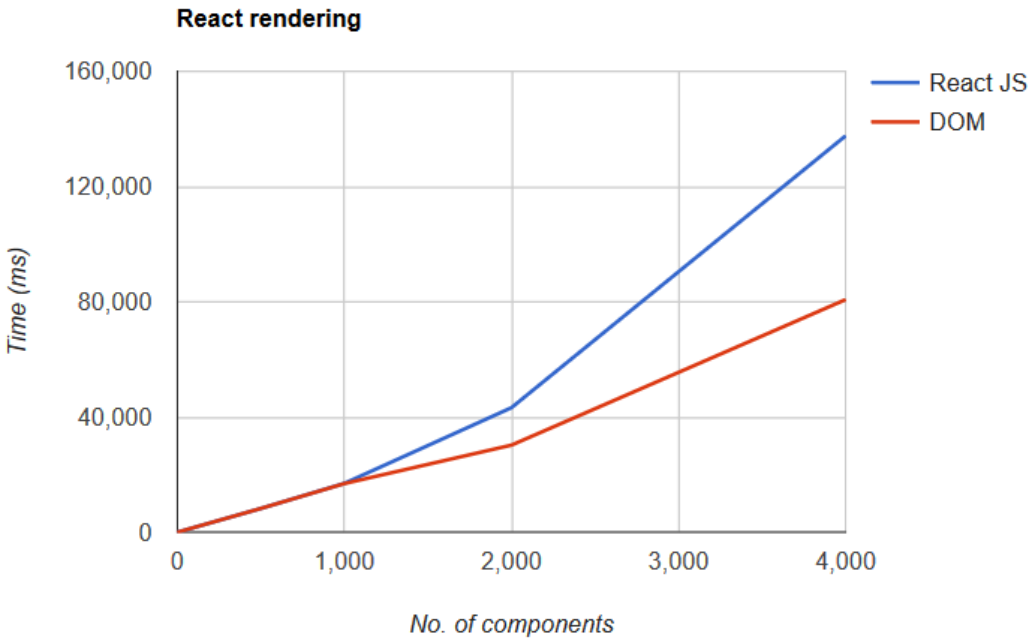

Figure 6.1: The rendering time of normal DOM vs React

The evaluation was carried out using normal DOM (Document Object Model) and React JS. It is clear that when number of components are higher than 1000, React JS has to perform twice the work than DOM version. The reason is that React JS has to deal with both the virtual DOM and the actual DOM to render components.

Figure 6.2 shows the memory consumption of React JS with the increase of number of components.

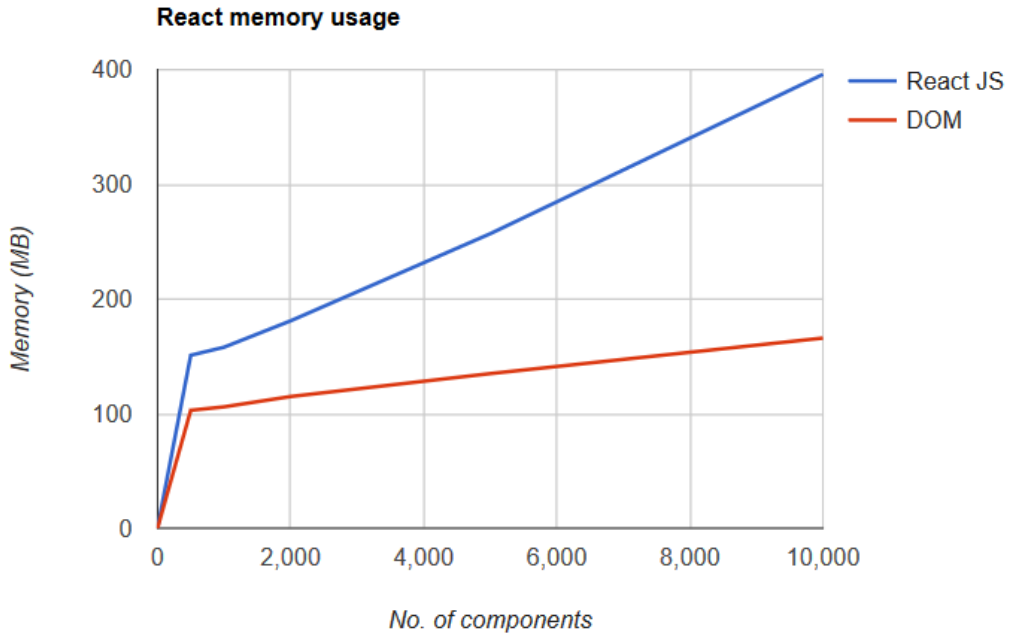

Figure 6.2: The memory consumption of normal DOM vs React

React JS consumes more memory because of the virtual DOM manipulation.

6.2.2 Completeness of the scope and objectives

The goal of the project was to improve the development process of linguistic applications by reducing the overhead and overall development time through introducing a framework to support linguistic application developers. The FLAD system is the final output of the project which contains the framework and some additional features like FLAD console. The FLAD approach has decreased the development time of application developers due to easy project creation logic provided by FLAD console and more programmer friendly REST API of the framework services. The summary of evaluation form given to application developers about FLAD console is given in Appendix H.

All the objectives mentioned in Section 1.3.2 have been achieved by the FLAD system. FLAD has achieved the functional requirements and non-functional requirements with the design and implementation approaches.

- **Performance**

  The performance of the framework is improved with many design and implementation approaches. The use of service locator design pattern enables fast retrieval of objects from cache rather than initiating them in each request. The concept of sequences has improved the overall response time of

complex services. Table 6.1 shows the exact quantitative measurements of complex services of FLAD and the efficiency gained using sequences. The use of efficient Java collections like HashMaps over ConcurrentHashMaps and SynchornizedHashMaps makes fast retrieval of values without overhead of object locking.

- **Scalability**

  The design of the FLAD supports scalability of the system. The layered architecture and component-based architecture (described in Chapter 4) allows the replication of servers in presence of high concurrent users. The FLAD system supports horizontal scalability and functional scalability due to the plug and play model. System supports horizontal scalability in database layer by using sharding technique in MongoDB. In the current stage sharding is not used in the project.

- **Availability**

  The FLAD system runs on a server which is available at all times. The server replication can make the system highly available without any downtime. If the system needs a full restart, Jetty server takes approximately 20 seconds to reboot. Thus availability can be computed considering both uptime and downtime of the system using the following equation.

$$Availability = \frac{FLAD[uptime]}{(FLAD[uptime] + FLAD[downtime])}$$

  Server replication can reduce the downtime and increase the availability. However, server replication is considered to be a future development. (all the future developments are discussed in Chapter 7).

- **Reliability**

  System reliability is considered as the combination of framework reliability and the reliability of third party linguistic components. The framework REST API and endpoint services are tested for the reliability (the testing process is discussed in Section 6.6). The reliability test showed that the intended outcome is always produced in service API calls. The FLAD system is not responsible for the reliability of the third party linguistic components.

- **Maintainability**

  A framework is always built considering the extendability and maintainability. The design and the implementation of the framework allow future developments easy. New features can be easily added to the functionality of the

framework because of the loose coupling of components. The well-defined interfaces facilitate the easy maintainability of the system.

- **Security**
  Security is ensured in many ways. The token generation is done at the back end to ensure the security and the token is always enforced to send in the HTTP header. Each request is validated using the Validator component in the framework to prevent unauthorized activities. The user cannot send requests to endpoint services which he/she has not purchased by the FLAD system and user cannot make requests to admin routes or developer routes which are only accessible for FLAD administrators.

- **Data Integrity**
  The framework provides a uniform data format across API calls. The request JSON structure is self descriptive and programmer friendly. The response JSONs for all service calls are in the same format which makes data consistent.

- **Usability**
  The FLAD console has made project creation easy and efficient which allows programmers to reduce the development overhead and provide the service APIs quickly. The evaluation done for FLAD console is shown in Appendix H. The uniform JSON formats made the programmers to develop applications faster.

- **Interoperability**
  The framework functionality is provided as a web service which can be accessed using any device and any platform. This makes the system interoperable. FLAD is built using Java which is platform independent and can be easily deployed in Linux or Windows servers.

Best combination of service instances which provides efficient responding time for sequences

The sequences have proved the optimization of the service workflow. They are created to improve the efficiency by reducing the number of stateless HTTP calls. Table 6.1 shows the quantitative results of the instances with sequence enabled and disabled. The quantitative evaluation of the request, response times are highly depended on the back end service instance and the internet bandwidth. All the endpoint service are implemented to interact with JSON and this approach improves message conversion overheads and improves the efficiency of sequences. The evaluation was done using the following services.

- OCR-Google Vision API, STT-Google Speech API, Translation-Google translation and TTS-MaryTTS.
- The internet download bandwidth = 8Mbps.
- The upload bandwidth = 5Mbps.
- Average Ping = 70ms.

$$Efficiency = \left(1 - \frac{time\ with\ sequence}{time\ without\ sequence}\right) * 100$$

Table 6.1: Efficiency of sequences

Sequence	**With sequence(ms)**	**Without sequence(ms)**	**Efficiency**
Sequence 0	6853.86	7266.58	5.6797%
Sequence 1	2446.44	4088.34	40.1606%
Sequence 2	5996.73	8254.23	27.3496%
Sequence 3	5010.23	5153.54	2.7808%
Sequence 4	5186.37	5312.19	2.3685%
Sequence 5	7969.16	9378.08	15.0235%

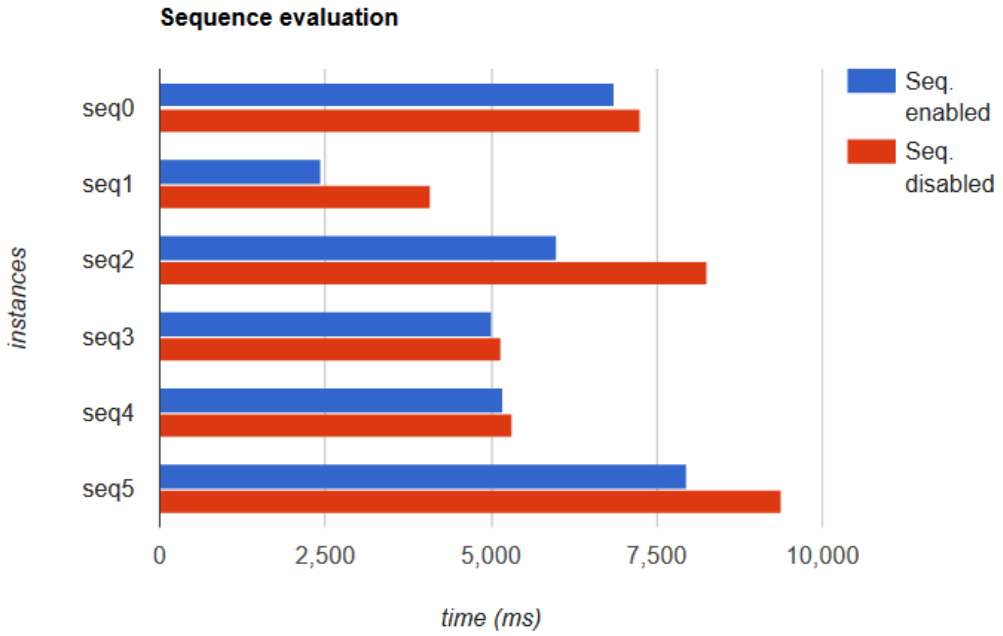

Figure 6.3: The graphical view of sequence evaluation

The graphical view of sequence evaluation is shown in Figure 6.3, the lower is better. According to this graph it is clear that sequences respond faster than independant simple services.This evaluation concludes that the expected optimization was achieved using sequence implementation.

6.3 Architecture wise evaluation

We have designed and implemented the architecture of FLAD to easily plug new services and new endpoints to the existing services with the least code modifications.

6.3.1 Effort of adding new service instances to the existing services

The objective was to design an architecture that facilitates adding new services to the framework with minimal changes. According to the architecture when adding a new service instance for an existing service there are only four changes done to the system. These changes do not need a manual server restart. This prevents the interruption of services exposed by the FLAD system.

- Add a class inheriting the service class with the new instance name inside the relevant service package.
- Override the execute method and implement the logic.
- Add an entry in the Service factory class with the new instance name for dynamic invocation.
- Add a label name in Endpoint Operation class for FLAD console display.

6.3.2 Effort of unplugging a existing component

Plug and Play architectural model facilitates removal of an existing component with minimal changes to the system. When unplugging a component there are only two minor changes done to the code structure. These changes do not need a manual server restart. This prevents the interruption of services exposed by the FLAD system.

- Remove the entry in the Service factory class with the instance name that needs removal.
- Delete the label name in Endpoint Operation class.

6.3.3 Effect of adding new services to the FLAD system (overhead of modification of the code)

The architecture supports the addition of new linguistic services other than STT, OCR, TTS, and Translation to the system. This needs only five changes to the framework. The addition of a new linguistic service triggers a manual server restart and on average takes 10 seconds to reboot the FLAD system in a Linux server environment.

- Implement the new service class by inheriting the Service base class provided by the framework.
- Add a getService method in the Abstract Factory class.
- Implement a new Factory class for the new service.
- Add an entry in the Service Initializer class to expose the new service.
- Create a new REST API endpoint in the ServiceRouterController class.

6.4 Server wise evaluation

6.4.1 Evaluation of plugged services

The pluggable components are used for the testing of the framework in the development phase. The plugged services impose restrictions which are discussed in Chapter 7 under the limitations of the project. The plugged services are not evaluated under restrictive environment. For the development of the framework, the services which are standalone is preferred as they processed outputs relatively faster than the services which provided a REST API. The extra HTTP request redirection causes some delay in the result and the internet bandwidth becomes a bottleneck in the performance.

The services which provide JSON support performs better than the services which expect a different data format as the const for JSON parsing takes time. Any service developed in any language other than Java can be plugged into the framework unless they provide a Java wrapper or a web service. We did not find any issue with the services which are not compatible with the framework as all most all the services support either a web service or a Java wrapper because of the popularity of Java.

6.4.2 Latency of the requests

The latency is highly dependent on the bandwidth of the internet connection. The evaluation was done using the following services and conditions.

$$Latency = T1 + processing\ time + [upload\ time] + T2.$$

T1 - Time taken by request to reach framework
T2 - Time taken by response to reach the client
Note: upload time is not relevant to some services.

- Sequences were evaluated using OCR-Google Vision API, STT-Google Speech API, Translation-Google translation and TTS-MaryTTS.
- The internet download bandwidth = 8Mbps
- The upload bandwidth = 5Mbps
- Average Ping = 70ms

Table 6.2: Latency of simple services

Service	**Latency (ms)**
Google OCR	3100.69
Mary TTS	2446.44
Google Translate	0987.65
Google Speech API	4224.54
Tesseract OCR	3211.21
CMU Sphinx STT	6945.51

Table 6.3: Latency of Complex services (Sequences)

Sequence Name	**Latency (ms)**
Seq0	6853.86
Seq1	2446.44
Seq2	5996.73
Seq3	6710.23
Seq4	5186.37
Seq5	7969.16

6.4.3 Requests per period of time which can handle by the FLAD (How many concurrent requests can handle)

MongoDB

Testing environment,

- Linux Ubuntu 16.04 (64bit)
- MongoDB 3.6
- 8 GB RAM
- Intel core i5 2.5 GHz

MongoDB performance is evaluated under default configuration settings and through a connection pool of size 100. In order to test the performance in concurrent requests several threads were created and each thread executes a sequential series of operations by making calls to the database interface layer to load the database (the loading phase) and to execute the workload (the transaction phase). Read, Write operations were issued in two separate loops and measured the latency against the number of operations per second.

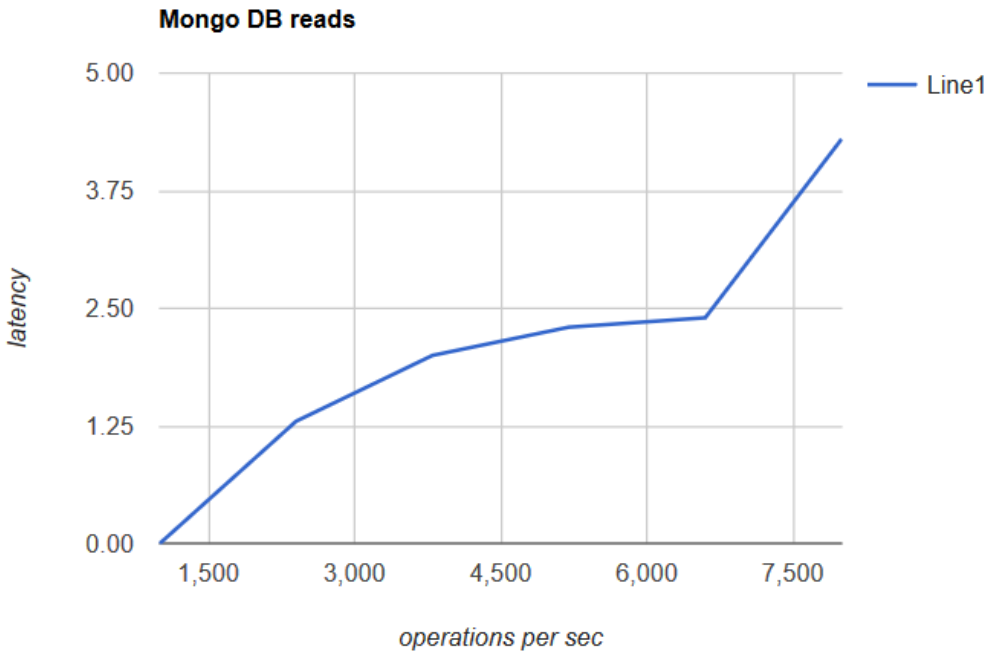

Figure 6.4: MongoDB performance evaluation of read operation

Figure 6.4 shows how the latency increases with the increase of unit operations. Under default settings maximum of 3000 concurrent users are recommended.

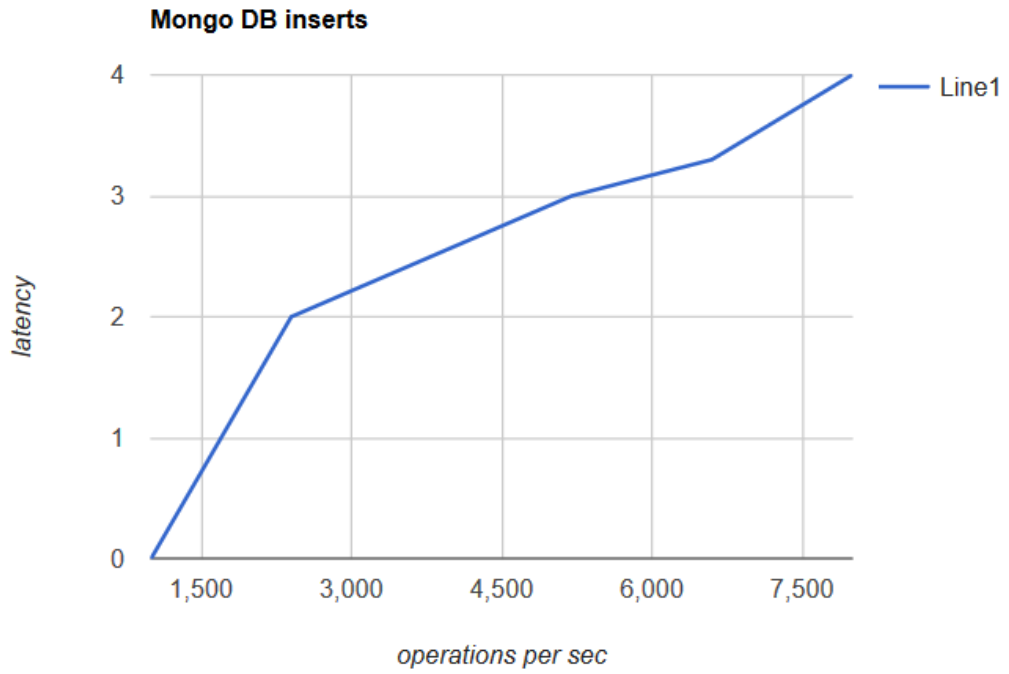

Figure 6.5: MongoDB performance evaluation of write operation

Figure 6.5 shows how the latency increases with the increase of write unit operations. Under default settings maximum of 2000 concurrent users are recommended in write operations.

Jetty server

Testing environment,

- Linux Ubuntu 16.04 (64bit)
- MongoDB 3.6
- 8 GB RAM
- Intel core i5 2.5 GHz

Evaluation of Jetty server was done by sending simple requests to process a JSON request. The evaluation criteria was done on 3 aspects request waiting time, request handling time and throughput. Figure 6.6 shows the request waiting time against concurrent requests. The request handling time of Jetty is shown in Figure 6.7. Throughput of the server with the increase of concurrent requests is shown in Figure 6.8.

Table 6.4: Jetty performance evaluation

Concurrent Requests	**Requests Waiting Time**	**Requests Handling Time**	**Throughput**
1	6.391	6.391	156.48
5	11.484	2.297	435.37
10	19.063	1.906	524.59
20	25.625	1.281	780.49
40	0.797	31.875	1254.9
60	6.578	394.688	152.02
80	5.563	445	179.78
100	1.781	178.125	561.4
200	6.984	1396.875	143.18
300	3.109	932.813	321.61
400	6.531	2612.813	153.11

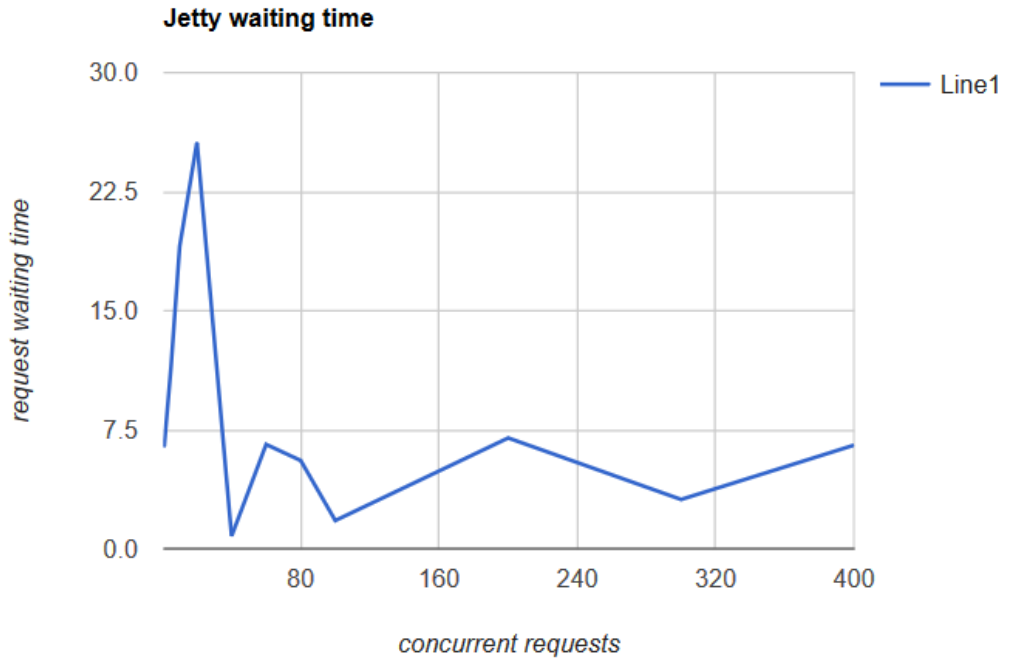

Figure 6.6: Jetty waiting time

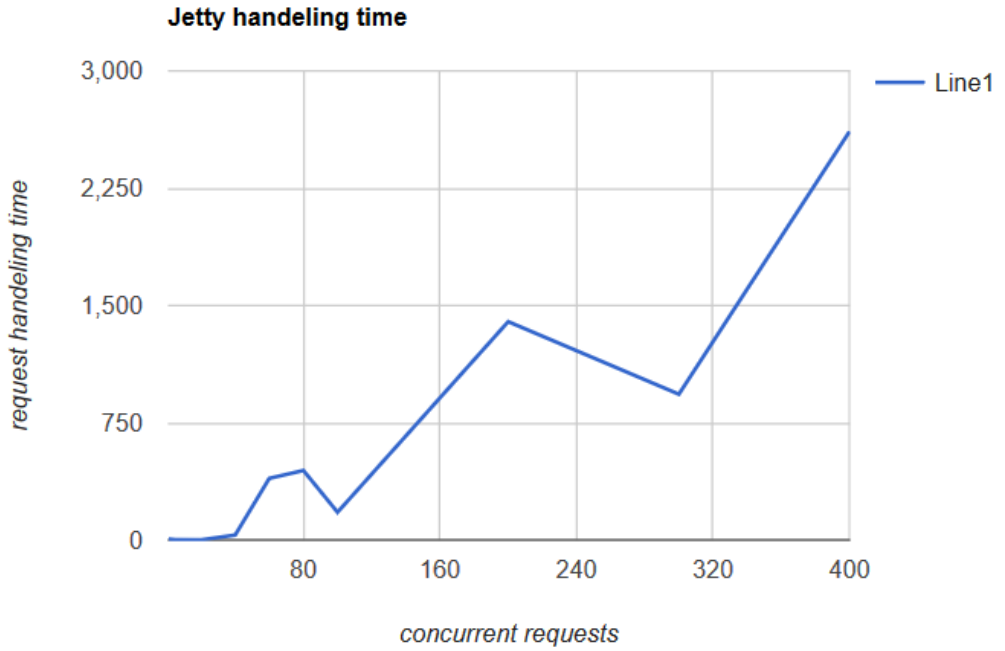

Figure 6.7: Jetty handling time

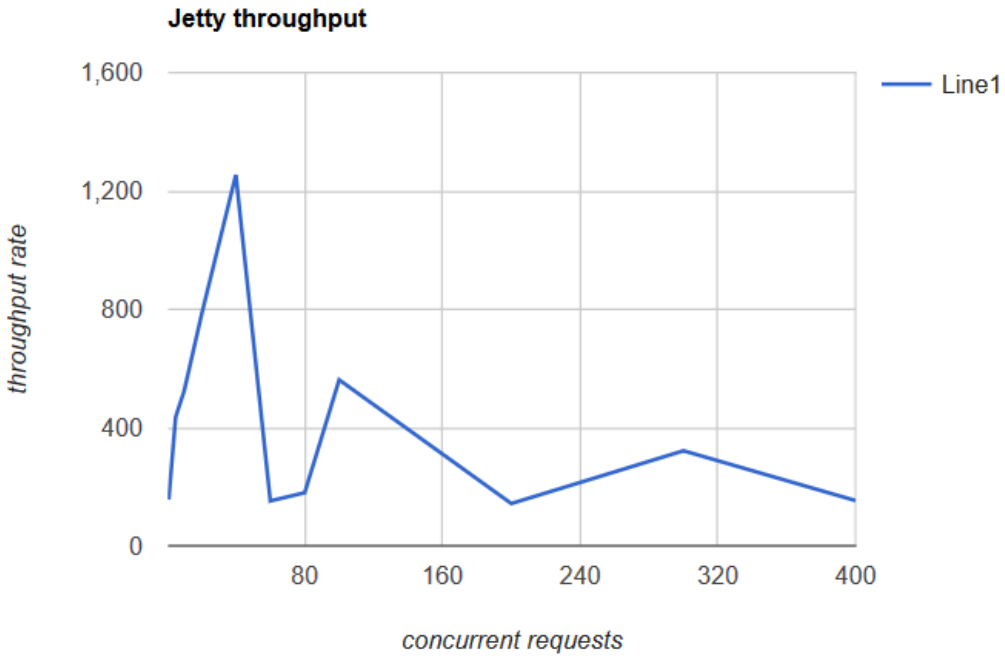

Figure 6.8: Jetty throughput

6.5 Proof of Concept

A mobile application was developed along with the framework by the development team as a proof of concept. FLAD is a new concept where services of both single linguistic components and combined linguistic components are taken to a common platform. Details about the mobile application development are not included in the previous chapters of this thesis, as it is not directly related with the framework development, but it is about using the services provided by the framework. In the Evaluation chapter, it is better to mention about the mobile application development as the development team experienced a true convenience in building an application using the REST APIs provided by the framework.

This mobile application is used in the testing process of the framework as well. Test types such as performance testing and stress testing (mentioned in Table 6.5) of the framework are done by providing this mobile application to some user groups and allowing them to use functionalities of the application. Some user interfaces of the mobile application are shown in Appendix I.

6.6 Testing Process

Table 6.5: Test Plan

Test type	**Reason for using**	**Method of usage**	**Tools**
Compatibility Testing	The framework should run on Windows servers or Linux servers.	The system is compiled and built on both Linux and Windows operating systems.	Native OS platforms
Functional Testing	Need to check all the functionalities of the framework are running as intended	Running tests using a test automation tool to test all the endpoints in all router controllers Black box testing	TestNG
Integration Testing	To check inter module compatibility. To see how different pluggable components work each other.	Testing all the sequences which integrates all the services. Testing all different endpoint services with dynamic behaviour change. Black box testing	TestNG
Performance Testing	How the framework perform in different conditions. To see how use of sequences improve the performance.	Check how system perform at normal, peak, and exceptional load conditions.	Black box testing
Stress Testing	How system handles when there are large number of concurrent requests.	Testing how multiple concurrent requests are handled by the FLAD server, MongoDB server and individual linguistic services.	Jmeter, The Grinder
System Testing	To see how the whole framework behaves altogether.	Use a mobile application as a proof of concept and test all the service endpoints.	Manually calling the REST API of the framework

Table 6.6: Test Plan

Test type	**Reason for using**	**Method of usage**	**Tools**
Unit Testing	Each function and module is tested with JUnit.	Use maven like build tools and create a test plan for each code module.	JUnit 5
Web Service Testing	Test the REST API calls and response codes	Check all available services under DevRouter Controller, Admin Router Controller and Service Router Controller	Soapui, SOAtest
Usability Testing	To test if the FLAD system has minimized the application development overhead through FLAD console project creation logics and service API calls	Developing some applications as a proof of concept and allow users to create FLAD service APIs in FLAD console to test the usability of the system.	Xamarin based mobile application, FLAD console
Recovery Testing	To check if the different modules can perform independently. What happens when a particular component crash and how the system will manage.	Crash components manually and test if the system can recover automatically.	Manually

The FLAD system consists three main subsystems as FLAD console, FLAD back end and independant linguistic services. Initial testing was done separately on the subsystems and the whole system was tested under system testing. Above Table 6.5 shows the test plan performed on the FLAD system.

Front end FLAD console evaluation was done using a feedback form given to a selected computer science students (see Appendix H.1 for the evaluation form) and the results are depicted in the bar graph in Figure 6.9.

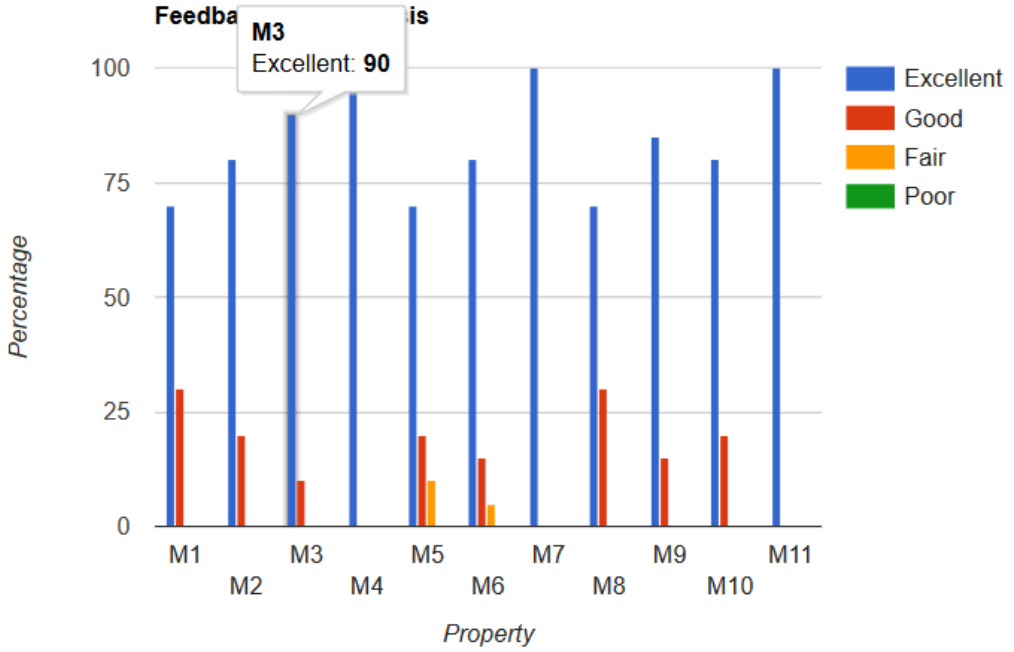

Figure 6.9: Analysis of front end evaluation

M1 - Colors, M2 - Font, M3 - Font size, M4 - Font color, M5 - Placement of components (structure of the UI), M6 - Interactiveness, M7 - Meaningfulness of terms used, M8 - Self-descriptiveness of icons, M9 - Overall design, M10 - Ease of use, M11 - Documentation

6.7 Summary

This chapter discussed how this system is evaluated under three main sections as software wise, architectural wise and service wise. The evaluation is done using qualitative and quantitative measures. System's response times for all the services in milliseconds and efficiency of those services are also included in this chapter. This chapter also mentioned how user-friendliness of the system's front end is evaluated using an evaluation form. In addition to these things, the testing process done using several test types such as unit testing, stress testing, etc. is discussed in this chapter.

Chapter 7

Conclusion

In this study, it is identified that there is a considerable gap between linguistic applications development and the linguistic components. There is an actual overhead in integrating the linguistic components within a linguistic application. It becomes more complicated when developers try to combine two or more linguistic components together to get a complex service.

This study includes a well-focused background study covering several linguistic components including their pros and cons, configuring overheads, etc. Background study also includes REST API design approaches that are studied to use in this system and component architecture of the WSO2 Enterprise Service Bus. Further, it includes the study of React and Redux which is used in front end development of the system.

System design is facilitated with three architectures that serve three main components of the system. Service Oriented Architecture serves the communication components while Component architecture and Client-Server architecture serves structure component and deployment component respectively.

System implementation is done in two main streams as back end development and front end development. Back end development holds implementation of REST APIs to provide services of the framework. Further, the back end development includes integration and combination of linguistic components to the framework. Front end development includes implementation of the FLAD Console which provides the users to create projects with the services they need from this system and the Admin Panel that provides admins to manage the system.

Both back end and front end of the system are evaluated using quantitative and qualitative measures. This system is evaluated as software-wise, architectural-wise and service-wise. Front end is evaluated by considering user-friendliness of the FLAD Console as the parameter and getting the feedback from a sample set of users.

7.1 Limitations

Limitations of the project fall under the limitation of linguistic components and limitation of server environments.

Limitations of server environments

- In windows server environments UTF-8 encoding scheme is not supported for the incoming requests handled by Jetty server.
- In windows server environment all the server has to be manually run without a script. Linux environment supports script loading of server starts.

Limitations of linguistic components

The framework is not responsible for the limitation of the services provided by third party linguistic components. These limitations are applied only when the specified linguistic components are selected as the endpoint service.
Google STT limitations,

- No direct audio file is upload possible need to convert to Base64 encoding scheme.
- Only mono audio files are supported under the synchronous scheme.
- Synchronous speech recognition returns text for only short audio (less than 1 min)
- Asynchronous requests limits the audio requests to less than 180 minutes
- Streaming requests have a limit of audio clips less than 1 minute.
- Phrases per requests are limited to 500.
- Requests per 100 seconds are limited to only 500. (refer Appendix J.1)

Mary TTS limitations,

- Mary TTS supports only limited number of voices and dialects. For an example, Sinhala voices are not supported. As a result, if the user inputs a Sinhala text to MaryTTS the result will not be output.
- Text inputs higher than 500 characters takes more than 3 minutes to render as audio. The limitation lies with MaryTTS processing and not with the framework.

Google OCR limitations,

- Google has a limit of the image size of 4MB for OCR service.
- 8MB limit size for a request.
- Requests per minute is limited to 600.
- Images per month is limited to 20,000,000.

Google Translation limitation,

- Characters per day is limited to 2,000,000 characters.(see Appendix K.1)
- Some language translation does not provide correct translations.

7.2 Future Work

7.2.1 Support for stream API

Currently, the framework does not handle streaming of audio files in the API. Audio streaming can be provided as a future update in TTS and STT requests. Limitations in Jersey framework lead to the postponement of streaming feature in the framework.

7.2.2 Bulk OCR processing

The framework only allows single image upload per a request in OCR endpoint. Bulk OCR processing functionality can be implemented in future updates so that the OCR for large PDF documents can be done through the framework.

7.2.3 Load balancing

Manual load balancing is a complex task in the framework as framework support distribution of several components. The framework can be moved to a Nginx server and can achieve load balancing in the future,

7.2.4 Use of external environment files

The configuration of the framework is handled in the internal code segments and the external env files make it easy to manipulate the servers easily. All the metadata related to server configuration can be embedded in an external file and can be used for easy code modifications.

7.2.5 Enhancing plug and play

The idea of representing the state of a component using XML can be used to make plugging new components easily into the framework. FLAD console admin panel can be used to dynamically add and remove components to the framework.

7.2.6 Server replication

Currently, the development process was carried out using single server environment. But the architecture of the system is design in a way to support the distribution of servers through replication which is considered as a future development.

7.2.7 Introducing a revenue model

The framework is developed using some proprietary linguistic services like Google translation, Google Speech API, and Google Vision API. The application developers need to purchase these service from FLAD in order to use these services. The revenue generation model is not currently introduced in the system and planned to include in the FLAD console in project creation phase.

The project's goal and objectives were achieved with the great commitment of the four team members from the start of the project and the individual contributions are specified in Appendix L, Appendix M, Appendix N and Appendix O.

References

[1] D. Riehle, "Framework design: A role modeling approach." *Ph.D. Thesis*, 2000.

[2] Marytts, "marytts/marytts." [Online]. Available: https://github.com/marytts/marytts/wiki/MaryInterface

[3] "What is kaldi?" [Online]. Available: http://kaldi-asr.org/doc/about.html

[4] "Htk speech recognition toolkit." [Online]. Available: http://htk.eng.cam.ac.uk/

[5] N. Shmyrev, "Cmusphinx open source speech recognition." [Online]. Available: https://cmusphinx.github.io/

[6] "Speech api - speech recognition — google cloud platform." [Online]. Available: https://cloud.google.com/speech/

[7] tesseract ocr, "tesseract-ocr/tesseract," Dec 2017. [Online]. Available: https://github.com/tesseract-ocr/tesseract.git

[8] "Vision api - image content analysis — google cloud platform." [Online]. Available: https://cloud.google.com/vision/

[9] "google cloud translation api documentation — translation api — google cloud platform." [Online]. Available: https://cloud.google.com/translate/docs/

[10] "Wso2 documentation." [Online]. Available: https://docs.wso2.com/display/ESB490/Architecture

[11] R. T. Fielding, "Architectural styles and the design of network-based software architectures," *Doctoral dissertation, University of California, Irvine*, 2000.

[12] "What is rest (representational state transfer)? - definition from whatis.com." [Online]. Available: http://searchmicroservices.techtarget.com/definition/REST-representational-state-transfer

[13] T. F. P. eCollege, "What is rest?" [Online]. Available: http://www.restapitutorial.com/lessons/whatisrest.html

[14] Codementor, "Rest api tutorial: How to design a sustainable web api." [Online]. Available: https://www.codementor.io/rest/tutorial/rest-api-design-best-practices-strategy

[15] T. F. P. eCollege, "Using http methods for restful services." [Online]. Available: http://www.restapitutorial.com/lessons/httpmethods.html

[16] T. Preston-Werner, "Semantic versioning 2.0.0." [Online]. Available: https://semver.org/

[17] "Json," Dec 2017. [Online]. Available: https://en.wikipedia.org/wiki/JSON

[18] "Introducing json." [Online]. Available: https://www.json.org/

[19] "React (javascript library)," Dec 2017. [Online]. Available: https://en.wikipedia.org/wiki/React_(JavaScript_library)

[20] "React tutorial — build with react js." [Online]. Available: http://buildwithreact.com/tutorial/jsx

[21] "Read me." [Online]. Available: https://redux.js.org/

[22] "Service-oriented architecture standards." [Online]. Available: http://www.opengroup.org/standards/soa

[23] "Design principles." [Online]. Available: https://reactjs.org/docs/design-principles.html

[24] R. Dodson, "Javascript design patterns: Decorator," Dec 2014. [Online]. Available: http://robdodson.me/javascript-design-patterns-decorator/

[25] Yannickcr, "yannickcr/eslint-plugin-react," Dec 2017. [Online]. Available: https://github.com/yannickcr/eslint-plugin-react

[26] "Apache log4j 2." [Online]. Available: https://logging.apache.org/log4j/2.x/

[27] S. Shimanovsky, "Multi page web applications vs. single page web applications." [Online]. Available: http://www.eikospartners.com/blog/multi-page-web-applications-vs.-single-page-web-applications

[28] [Online]. Available: https://webpack.github.io/docs/what-is-webpack.html

[29] "Babel · the compiler for writing next generation javascript." [Online]. Available: https://babeljs.io/

[30] “nodemon reload, automatically.” [Online]. Available: https://nodemon.io/

[31] “Material-ui.” [Online]. Available: http://www.material-ui.com/

[32] “Higher-order components.” [Online]. Available: https://reactjs.org/docs/higher-order-components.html

[33] [Online]. Available: https://developers.google.com/identity/

Appendices

Appendix A

Table A.1: Comparison of Speech To Text libraries

#	**Kaldi**	**CMU Sphinx**	**HTK toolkit**	**Google STT cloud service**
Accuracy	high	medium	low	high
Speed	medium	medium	low	high
Computational cost	Very high	medium	medium	-
Configuration cost	Very high	low	low	low
Ease of use	low	high	low	high
Freely available	yes	yes	yes	no

Appendix B

Table B.1: Google speech API pricing

Monthly usage	**Price per 15 seconds***
0 - 60 minutes	free
61 - 1,000,000 minutes**	$0.006

Table B.2: Google OCR pricing

Feature	**First 1000 units/month**	**Units 1001 - 5,000,000 / month**	**Units 5,000,001 - 20,000,000 / month**
Label Detection	Free	$1.50	$1.00
Text Detection	Free	$1.50	$0.60
Safe Search (explicit content) Detection	Free	Free with Label Detection, or $1.50	Free with Label Detection, or $0.60
Facial Detection	Free	$1.50	$0.60
Landmark Detection	Free	$1.50	$0.60
Logo Detection	Free	$1.50	$0.60
Image Properties	Free	$1.50	$0.60
Crop Hints	Free	Free with Image Properties, or $1.50	Free with Image Properties, or $0.60
Web Detection	Free	$3.50	Contact Google for more information
Document Text Detection	Free	$3.50	Contact Google for more information

Appendix C

Listing C.1: Request and Response to obtain TTS service

```
Request URL : GET /api/v1/tts
Request body:
{
    "inputText":"We have a clear sky",
    "inputType": "TEXT",
    "outputType": "AUDIO",
    "outputFormat": "WAVE_FILE",
    "locale":"en_US",
    "voice": "cmu-slt-hsmm en_US female hmm",
    "features": {}
}

Response body:
{
    "other": [
         {
            "executionTime": "3711",
            "locale": "en_US"
         }
    ],
    "response":"https://www.googleapis.com/download/
       storage/v1/b/fladbuc ket/o/maryTTSAudio.wav?
       generation=1514361245171530&alt=media",
    "processedAT": "2017-12-27_13-24-05"
}
```

Listing C.2: Request and Response to obtain STT service

```
Request URL : POST /api/v1/stt
Request body:
{
     "audioFileName":"german.wav"
}

Response body:
{
    "other": [
        {
            "executionTime": "5099",
            "confidence": "0.986863"
        }
    ],
    "response": "hello how are you",
    "processedAT": "2017-12-27_13-29-04"
}
```

Listing C.3: Request and Response to obtain Translation service

```
Request URL : POST /api/v1/trans
Request body:
{
    "inputText": "Hello how are you"
    "target": "sp"
}

Response body:
{
    "other": [
        {
            "executionTime": "185",
            "locale": "en"
        }
    ],
    "response": "Hola como estas",
    "processedAT": "2017-12-27_13-27-00"
}
```

Listing C.4: Request and Response to obtain OCR service

```
Request URL : POST /api/v1/ocr
Request body:
{
    "imageFileName":"german.png"
}

Response body:
{
    "other": [
        {
            "executionTime": "3481",
            "locale": "de"
        }
    ],
    "response": "Hast du das Brechern meines Herzens
       gehort? ",
    "processedAT": "2017-12-27_13-18-48"
}
```

Appendix D

Listing D.1: MongoDB User document

```
{
    "_id" : ObjectId("59b90237b717d531c0d15cc3"),
    "googleId" : "112XXXXX1857XXXXX",
    "imageUrl" : "https://lh5.googleusercontent.com/-
       UuOGBRIMdM/c2W_SBvPE2Y/s96-c/photo.jpg",
    "email" : "dhanushka787@gmail.com",
    "name" : "Dhanushka Chandana",
    "givenName" : "Dhanushka",
    "familyName" : "Chandana",
    "status" : "1",
    "creationDate" : NumberLong(1510425000000),
    "updatedDate" : NumberLong(1513017000000)
}
```

Listing D.2: MongoDB Project document

```
{
    "_id" : ObjectId("5998558341bad2f809098799"),
    "projectName" : "test-web1",
    "projectId" : "PROJ1",
    "googleId" : "112XXXXX1857XXXXX",
    "clientSecret" : "443ffdvbXXXXXXX",
    "creationDate" : NumberLong(1510252200000),
    "updatedDate" : NumberLong(1512844200000),
    "services" : {
        "tts" : "mary-tts",
        "stt" : "cmu"
    },
    "sequences" : {
            "seq1" : [
                    "OCR",
                    "TTS"
            ],
            "seq2" : [
                    "STT",
                    "Trans",
                    "TTS"
            ]
    },
    "status" : "0"
}
```

Listing D.3: MongoDB Project document

```
{
    "_id" : ObjectId("5a2971add877e324b0793f15"),
    "requestId" : "123tt456",
    "type" : "stt",
    "startTimestamp" : NumberLong(1507349700000),
    "endTimestamp" : NumberLong(1507349715000),
    "projectId" : "sample-tts-app-bouwoxcsp3r",
    "status" : "success"
}
```

Appendix E

Table E.1: FLAD error codes

Error code	**Status**	**Reason**
200	OK	For successful transactions
230	Non-Authoritative Information	Try to access sensitive information
307	Temporary Redirect	Incase if the request is redirected
401	Unauthorized	Accessing non authorized route points
400	Bad Request	Request body is not in specified format
403	Forbidden	Accessing non purchased services or admin services
404	Not Found	Requested service is unavailable
408	Request Timeout	3rd party services hangs
413	Request Entity Too Large	Image size or Audio size exceeds limits
429	Too Many Requests	Processing request limit reached
500	Internal Server Error	3rd party service crash or server crash
503	Service Unavailable	3rd party service is unavailable or try to access a deprecated service

Appendix F

Table F.1: Comparison of JavaScript front end development frameworks.

#	**Angular JS**	**Vue JS**	**React JS**
Github stars	33,300	77,000	84,000
Developer Satisfaction	65%	89%	92%
Learning cost	Medium	Low	High
Flexibility	Low	High	High
Library size	143K	23K	43K
Speed	Low	High	Medium
Memory allocation	High	Low	Medium
Member experience	Some	No	Good

Appendix G

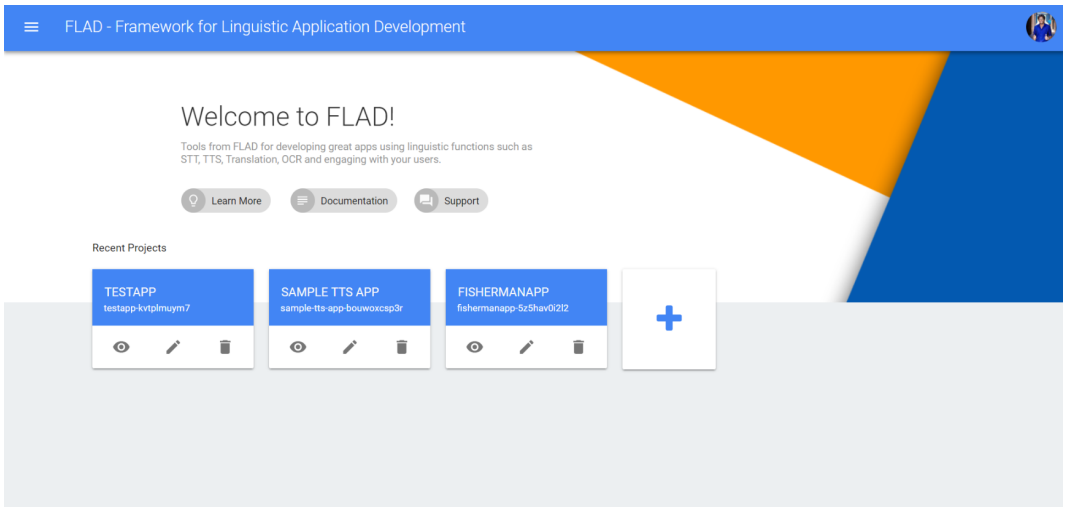

Figure G.1: User Dashboard

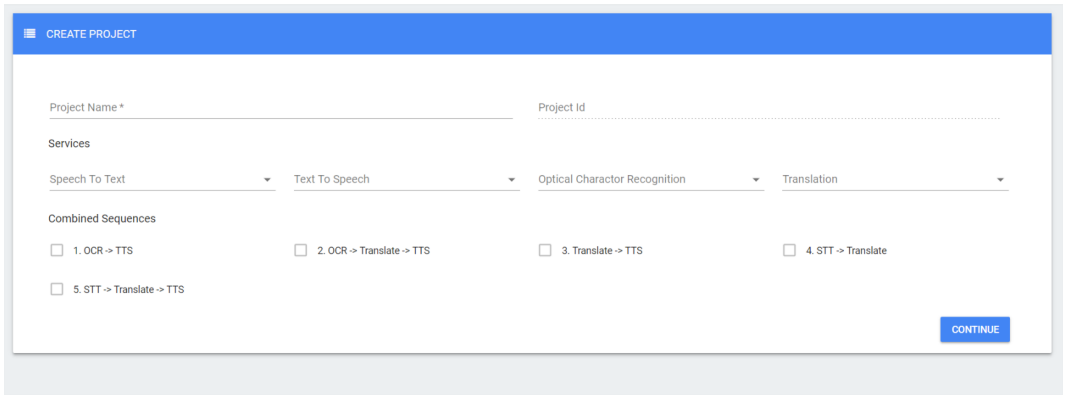

Figure G.2: Create Project Form

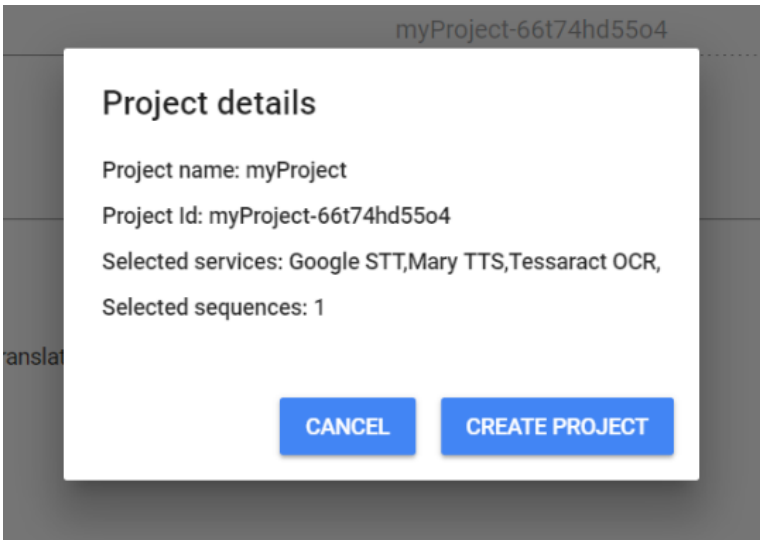

Figure G.3: View Project Details

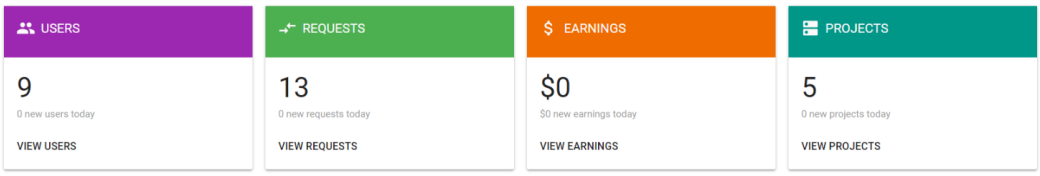

Figure G.4: Summary Box

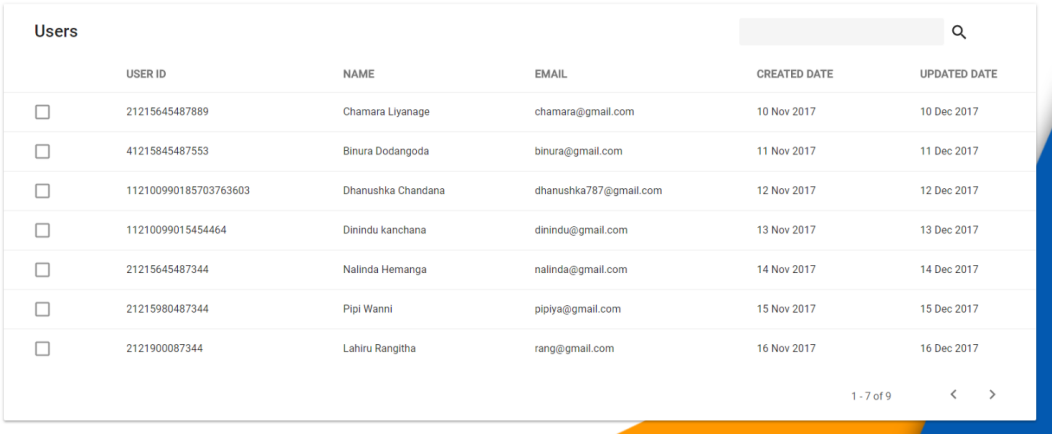

Figure G.5: Enhanced Table

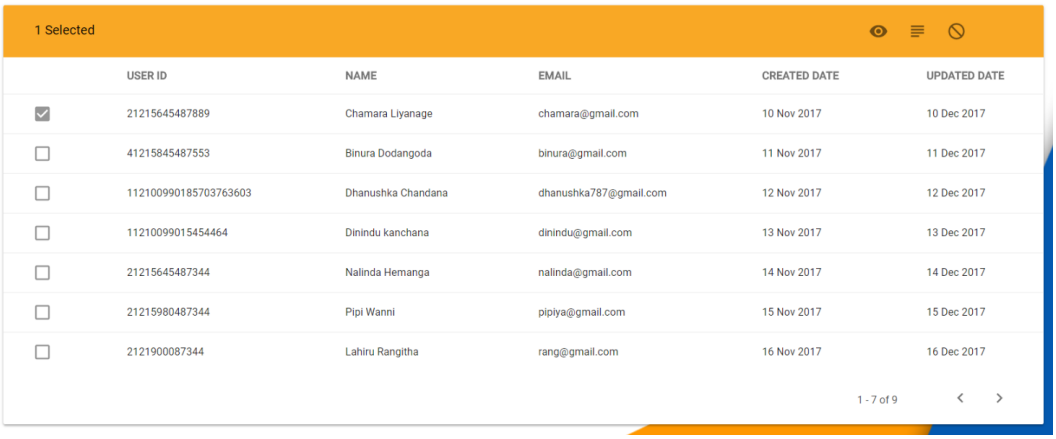

Figure G.6: Enhanced Table (Item selected state)

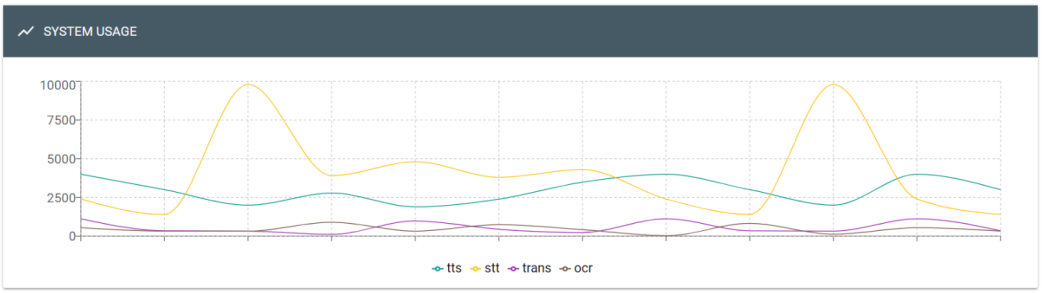

Figure G.7: Usage Chart

Figure G.8: Breadcrumbs

Appendix H

Table H.1: Front end evaluation form

	Excellent	**Good**	**Fair**	**Poor**
M1 - Colors				
M2 - Font				
M3 - Font size				
M4 - Font color				
M5 - Placement of components				
M6 - Interactivity				
M7 - Meaningfulness of terms used				
M8 - Self-descriptiveness of icons				
M9 - Overall design				
M10 - Ease of use				
M11 - Documentation				

Appendix I

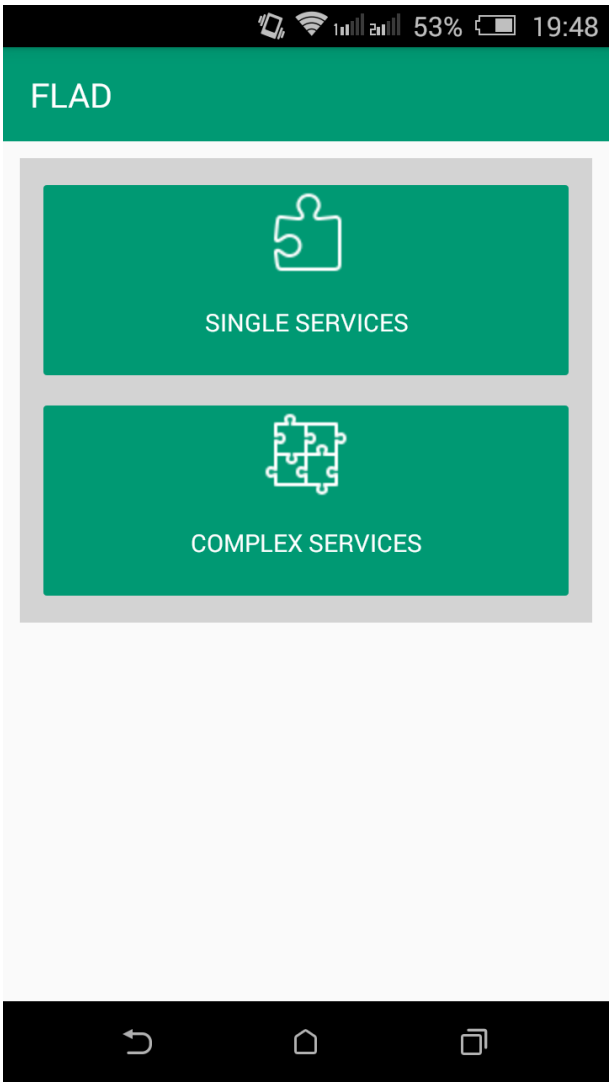

Figure I.1: Home Screen (Main Dashboard)

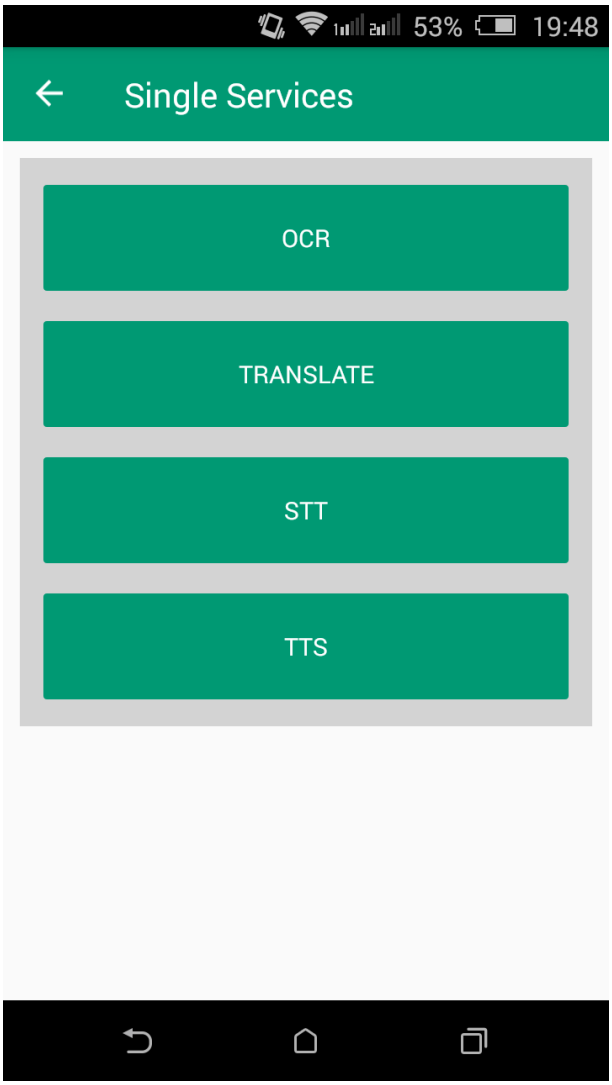

Figure I.2: Single Services Dashboard

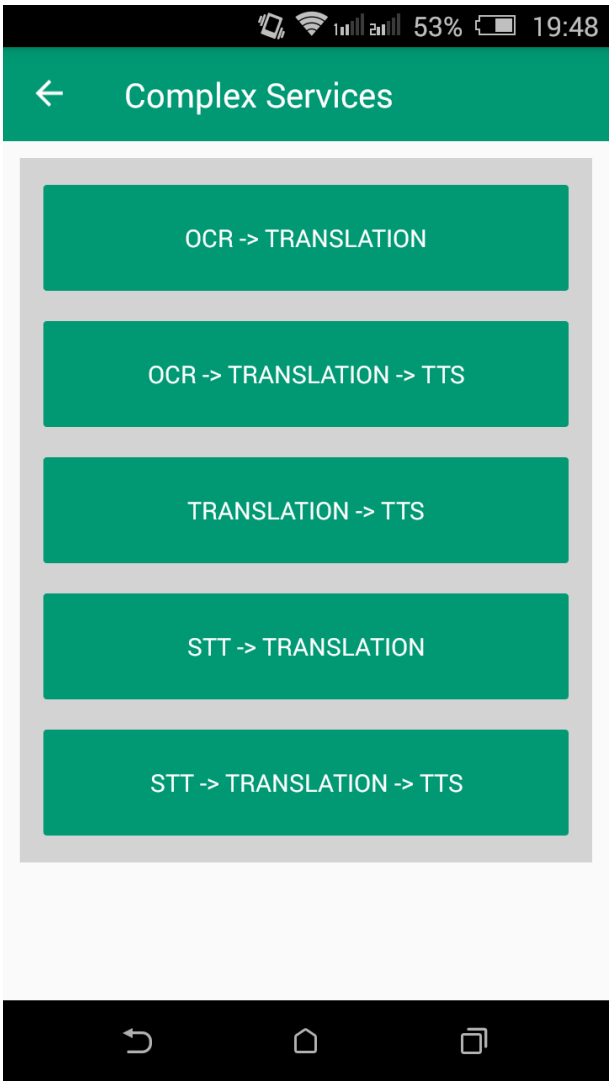

Figure I.3: Complex Services Dashboard

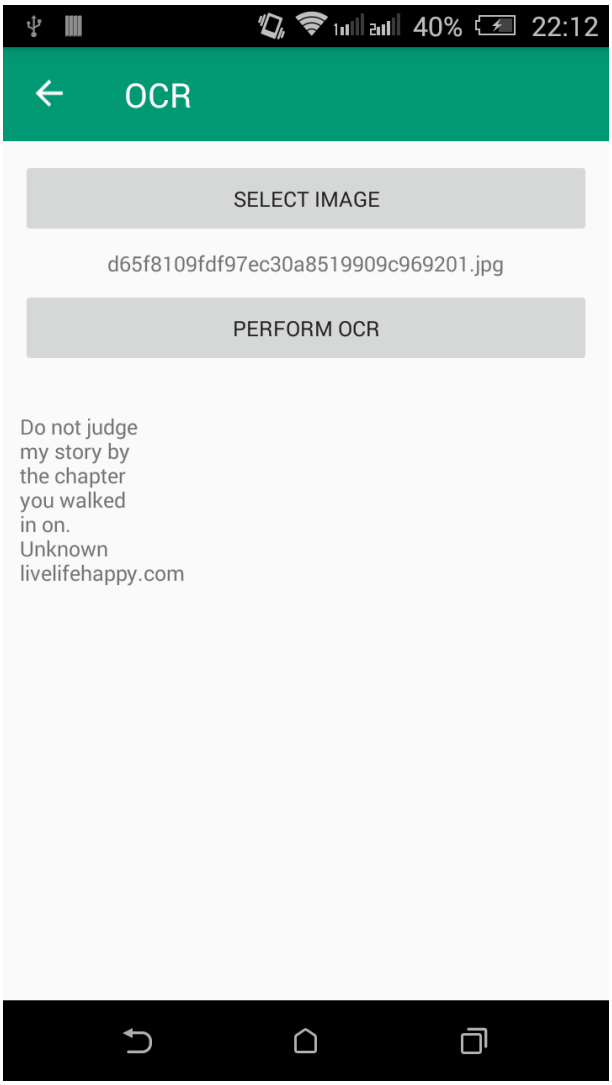

Figure I.4: OCR Screen after performing a OCR operation

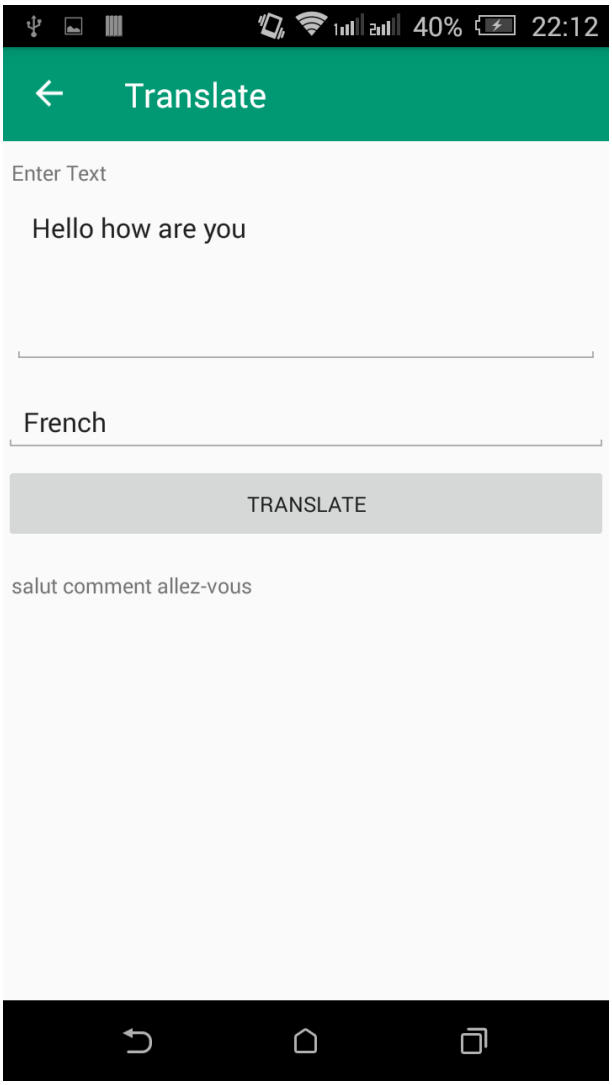

Figure I.5: Translate Screen after performing a English to French Translation

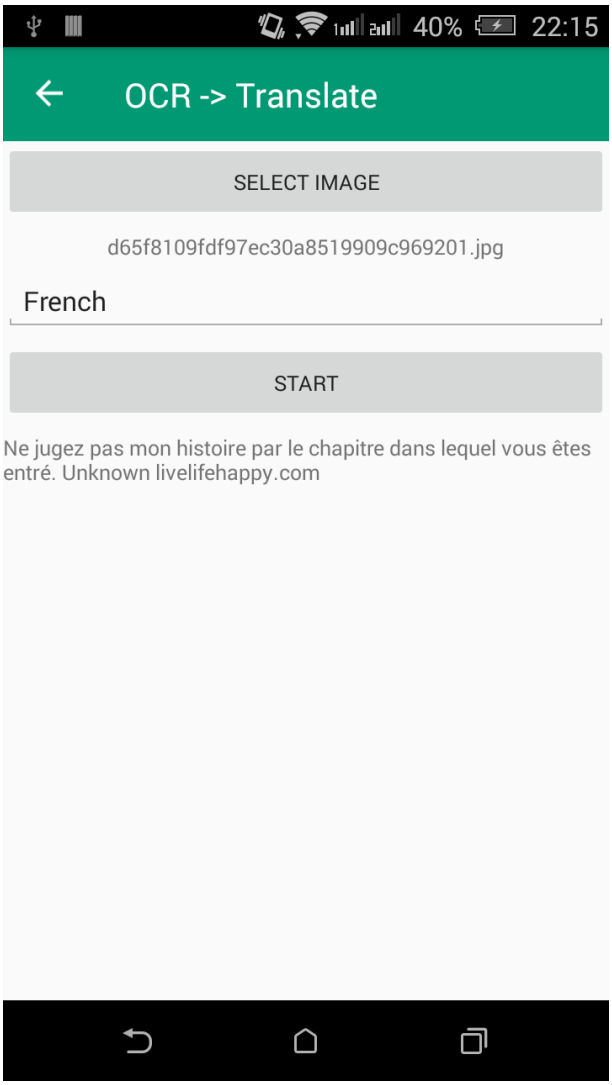

Figure I.6: OCR to Translate page after performing a OCR + Translate combined operation

Appendix J

Table J.1: Requests limits of Google STT

Type of Limit	**Usage Limit**
Requests per 100 seconds*	500
Processing per 100 seconds	10,800 seconds of audio
Processing per day	480 hours of audio
Phrases per request	500
Total characters per request	10,000
Characters per phrase	100

Appendix K

Table K.1: Requests limits of Google Translate

Content Quota	**Default**	**Maximum**	**Duration**	**Applies To**
Characters per day	2,000,000 characters	unlimited	day	project
Characters per 100 seconds per project	500,000 characters	500,000 characters	100 seconds	project
Characters per 100 seconds per project per user	100,000 characters	100,000 characters	100 seconds	user and project
Supported Languages Requests per 100 seconds	1000 requests	1000 requests	100 seconds	project

Appendix L

Contribution of C.M Liyanage

The main research area of the project was to find the best architecture for the framework which facilitates easy plug and play model. All the members did the background study on architecture and my part was to find the architecture for the core of the framework. After doing some background study I found that in order to provide an easy plug and play the functionalities of the framework should be considered as different components and designed the component architecture of the system. The component architecture is designed to meet the core functionalities of the framework with a solid foundation to the framework. I defined the components of the core of the system as explained in Chapter 4 under component architecture. The key functionalities of the framework were understood by analyzing the functional and non-functional requirements of the project. The fundamental components I defined were given a specific task in the framework. The component architecture emphasizes the separation of concerns with respect to the wide-ranging functionality in the framework. It can be considered as a reuse-based approach and helps in composing loosely coupled independent components into the system. The idea of expanding the framework was concerned when giving this approach. Components play a crucial role in the framework REST API component. The REST API can easily link all the necessary functionalities due to this approach.

I further improved the design of the framework by designing the multi-layered architecture to support scalability of the project considering the future requirements. The multilayered architecture is explained in Chapter 4. Each layer in the multi-layered architecture can be horizontally scaled with the introduction of well-defined interfaces.

After doing the part of the architectural design I was involved with the implementation of the back end core framework. When doing the implementations I integrated the Service Locator Design pattern to the framework with some additional custom features to make the system faster by caching object references. The

design pattern was implemented in the QoS layer of the framework. The framework is service oriented and the object initialization logic was handled through the Service Locator Design pattern which improved the performance of the back end services of the framework significantly. The design pattern application is explained under Chapter 4.

I implemented the Message builder and formatter component of the system. The implementation details are in Chapter 5. This component needed some JSON parsing libraries and the message conversion of the back end endpoint services. I defined some common data formats across the framework to give the solution to message heterogeneity of linguistic components through this components. The message builder is a key component in the component architecture which was custom built to meet the specific message passing pipeline efficient.

MaryTTS implementation was done by me in the back end framework. MaryTTS was selected as the test TTS service in the framework. This task needed the background knowledge gathered in MaryTTS library when doing some configurational changes. The Mary server was hosted in a separate port and plugged into the framework via the internal plugging mechanism. Two server versions of MaryTTS was requested from the MaryTTS Developers for Windows environment and Linux environment. MaryTTS had no service API exposed for public use. So, it was difficult to integrate to the framework. Finally, managed to invoke the necessary interfaces by studying the MaryTTS project at code level. The MaryTTS client code was analysed to find the exact query string of MaryTTS service. The Message builder and formatter component was used to parse Mary data as well. All the details are discussed in section 5.3.1.

The initial database integration was done by me and I integrated MongoDB database with the latest MongoDB driver and made the MongoDB connection singleton by considering the resource consumption. The MongoDB integration needed some background study on the configuration of MongoDB. I defined the initial UserOperations and Project Operations of the project. UserOperations and ProjectOperations are database handerls which handle user and project data respectively. Document structure for database Collection was defined initially to meet the requirements of the project.

Google translation was integrated into the framework as the testing translation component. I created the Google cloud console account under the FLAD project account name and purchased the API keys needed for the integration. By reading the documentation of Google Translation integration techniques I chose the REST API integration over SDK implementation. The exact implementation details are explained in section 5.3.6. The JSON structure expected by Google translation API was changed according to the JSON structure defined in the framework and

parsing was done using the Message builder and formatter methods.

When doing the development in the framework the front end FLAD console needs access to the back end framework. The Dev Router controller methods were introduced to provide the needed services to the front end console. getEndpioints, getSequences endpoints were implemented and exposed under Dev Router controller. The above mentioned endpoints are frequently accessed by the front end FLAD console so, I used in memory data structures like HashMaps to cache endpoint names and sequence metadata and prevented costly database lookups in subsequent requests. These kind of minor optimizations were done in code level implementations to provide an efficient service to the front end.

Sequence implementation (complex service) implementation was done in the back end. The implementation details are discussed in section 5.3.10. JSON message parsing was implemented as the main standard of message passing inside the framework. This enabled the easy communication needed among different linguistic components. The message conversion overhead was cutdown in the Sequence implementation.

The validator sub-component under QoS component was implemented for complex and simple service validations. The validation process is discussed in 5.3.2. The QoS methods such as Base64 encoding needed for audios and images were provided using this component.

Integration of Google Cloud Storage to the FLAD system was done in order to upload audio files generated with TTS components. For this integration, i had to use the SDK approach provided by Google over the REST API approach. The SDK approach needs extra configuration in server machine and I have installed the Cloud SDK and used the Google SDK Shell to configure the cloud storage bucket which acts as the repository for the FLAD project. An internal timer was defined to measure the exact execution time elapsed for endpoint services. Every JSON response output by the framework contains a field called execution time which is calculated by this timer.

There were many challenges I faced especially during the design and implementation phases of this project. As there were no prior similar systems like our framework all of the team members had to design the architectures from ground level. The component architecture and multi-layered architecture was designed with the help of team members and the knowledge gathered in background study. The architecture design phase needed great effort due to the lack of pre-existed solutions. When integrating MaryTTS server in our framework I faced a lot of technical difficulties. There was no documentation guideline provided for the integration and I had to contact the Mary Github community to get some technical knowledge. When implementing the Message builder component the parsing tech-

niques sometimes showed poor performance due to use of some libraries and methods. Changing some libraries and implementation logic with the help of other team members slow parsing issues which eventually made the component efficient. In Sequence implementation, there was spaghetti code structure due to its complexity which was resolved with the help of team members. Finally, Sequence component was refactored to provide a clean code implementation. All of us followed some specific design rules in code base level when designing the framework, especially when designing the API. Most of the challenges were overcome with the help of the team members' knowledge and expertise in different aspects. The scope of the project was achieved fully by providing all the functional and non-functional requirements of the project. The great commitment of the team members from the beginning of the project lead to the success of the project.

Appendix M

Contribution of G.D.D Kanchana

Initially, we identified the requirement of a framework for linguistic applications development. Then we studied the background of linguistic components, frameworks, architectural patterns, design patterns and etc. I did a background study on speech to text libraries mainly Kaldi, CMU Sphinx, and HTK. I configured all three libraries on my computer and tested them by considering about performance, accuracy, integration cost and resource utilization. Then I decided to use CMU Sphinx as the Speech To Text endpoint of FLAD. Finding the text corpus for Kaldi was a challenge so I used a freely available text corpus to train it. I used VLC Player to convert audio files to required format and frequency to use as inputs for STT libraries.

Then I studied the plugging architecture. How it is used in different systems and how to implement it. The selection of most suitable Java web server for a RESTful web service was a difficult task. I read server documentations, numerous blog posts, Quora questions and Stack Overflow questions to find the relevant server for our task. I configured Tomcat[1] and Jetty[2] to get an experience about each server. By considering configuration cost, ease of use, community support, resource consumption, suitability for project development phrase and portability I decided to use Jetty server for our system.

To develop a REST API we have to use Java JAX-RS[3]. Using JAX-RS without any supporting wrapper library was difficult and time-consuming. So I researched different libraries and frameworks which are the implementations of JAX-RS. I tested DropWizard [4], Play framework [5], and Jersey[6] to select the most suitable framework for our system. I decided to use Jersey as it is more flexible than the

[1]tomcat.apache.org
[2]https://www.eclipse.org/jetty/
[3]https://docs.oracle.com/javaee/6/tutorial/doc/giepu.html
[4]www.dropwizard.io/
[5]https://www.playframework.com/
[6]https://jersey.github.io/

other two frameworks.

Handling dependencies in a Java-based group project is challenging as group members add different libraries and plugins to the project. As a solution, I added Maven which is a dependency management tool to manage dependencies of the project. Conflicts between Java versions in different member's computers were resolved using Maven. In the development environment, the server should be restarted when the source code changed. So I added automatic restart functionality to the server using Maven.

I implemented a generic file uploading functionality using Java multipart forms. The files are categorized by examining the file format. JPG and PNG files are saved in OCR directory and Wave files are stored in STT directory on the server. Other file types are rejected by the server for security purposes. All the files uploaded to the server renamed with a unique ID.

I configured CMU Sphinx as the STT endpoint. This component was the first component plugged into the system. I faced the challenges of configuring a standalone library to a RESTful web service. I used Jackson library to generate JSON messages from as STT responses.

I implemented the main route controller in the back end to expose the REST API from one place. All the requests come into route controller and the annotations like @GET, @POST, @PUT are used to implement the route.

Handling exceptions and providing meaningful error messages is a quality attribute of a REST API. To handle exceptions I implemented the exception mapper component using Jersey Exception Mapper.

Loggin all the activities of the back end is required for recovery purposes. I researched about various Loggin libraries and decided to use Log4J[7] in the system. All the request to the back end is logged in a log file with the IP address, request endpoint, request time and request type.

App secret key (token) generation function was implemented using JWT as discussed in Chapter 5 under JSON web token generation process in FLAD. By this time all the STT, TTS, OCR and translation endpoints were configured to FLAD back end. So we started developing front end FLAD console.

I was assigned to develop all the functionalities of the clients. The initial challenge was connecting front end with back end because the back end system does not allow cross-origin access. I added a script to back end server to allow the cross-origin requests. As the front end is fully separated from the back end, the front end development was a considerable task. In the front end FLAD console, I implemented google login facility. Integrating Google login with our system was

[7]https://logging.apache.org/log4j/

achieved using an NPM package React Google login [8].

I found difficulties when using Redux JS initially because it has its own data flow. I read the documentation several times to understand Redux JS. I also implemented client registration functionality as discussed in Chapter 5. The implementation of authentication component was complex. I used higher order components of React JS to implement it as discussed in Chapter 5 under Authentication component.

The request and responses are handled using JavaScript promises. Persisting Redux State in browser's local storage and deleting it when the user logs out was a challenging task. I implemented notification components using Material UI Snack Bars. I used Redux State to show different notification popup messages using the same notification component.

I implemented project creation, view, edit and delete functionalities in the front end. All the components are developed separately to improve usability. All the projects of the user is loaded to User Dashboard when user login to the system. He can View all the details of that project by clicking on it. I have used Material UI Card component to render projects of the user. Validation of input fields of create and edit project forms was done using regular expressions. In the create project form the project ID is auto generated in the front end when user types the project name.

The project scope was fully covered with all the functional and non-functional requirements and additionally, we could provide a Mobile Application as a proof of concept. The great commitment of all the group members lead the project to a success.

[8]https://www.npmjs.com/package/react-google-login

Appendix N

Contribution of P.A.D. Chandana

FLAD can be described simply as a framework. As explained on the Chapter 01 Background section, The key idea of this project integrates all the linguistic components Text To Speech (TTS), Optical Character Recognition (OCR), Speech To Text (STT) and language translation in a framework and expose their functionalities as a RESTful web service. The process of integrating this component also has a major functionality which is support for the plug and play architecture. A part of this architectural functionality should be handled in a code-level.

The designing of this type of architecture was a new thing which combines some design patterns together to achieve the pluggable structure and modularity of the system. In designing of this architecture I have studies all 3 types of design patterns creational, behavioural and structural to understand the functioning of each design pattern in a deep level which can map the system flow and behaviours in the most suitable manner.

The Structure phase of FLAD System Design designed based on Component Architecture. In Figure 4.5 shows the class diagram architecture of core system. This diagram consists of Abstract Factory Pattern, Builder Pattern, Command Pattern. These patterns selected based on its behaviour as explained previously. Each of these patterns acts a role of the chaining process of plugging components and expose these components features to the upper layer.

After that this architecture translated into code level by adding several components to test the intended behaviour of this model. The further details of this architecture are described in chapter 04 design consideration under Core Component architecture of the System section.

FLAD consists of front end console back end core system as explained in chapter 04. I have done a background study of front end implementation frameworks. At the end of the study, ReactJS and Redux selected as the most suitable front end framework for the FLAD. The behaviours of these technologies and features they are providing described further in Chapter 02 background study under the sections

of ReactJS and Redux. These notable features mapped with our requirement and learning curve.

The configuration of the React and Redux was the first challenge in the front end implementation phase. Redux has provided a detailed tutorial of understanding the behaviour of the code flow within React redux application.

The Front end implementation has a part of consuming REST API and take the data to the front end UI components. Managing the data flow within the application was a tough activity because these consumed data are required to dispatch to the service request function of redux based on the action type. The understanding of this flow made easier to develop the rest of the application.

In the front end designing phase, next challenge was the selection of UI framework. I have studied several frameworks such as Bootstrap [1], Semantic UI[2], Ant Design[3], Grommet[4] and Matetial UI[5]. The study of the Google Cloud console; which is described in Chapter 2 under Google Cloud Console showed the use of material UI as the most suitable UI framework to go with a React Project.

The front end of the FLAD console is designed based on Google's design documents. The usability of the application is based on the human computer interaction principles. This detailed document provided the arrangement of UI component of a system in the best way to achieve a higher user experience within the system. I have designed the System UI theme of the FLAD based on the material UI theme and I also have designed the several parts of the UI of the FLAD which is explained later in this section.

A React application should have a component structure within the root container. I have designed this component architecture for the front end application by considering the usability and effectiveness of the system. FLAD consists of several containers which contains react components in different formats. After the studying of the components and containers of FLAD I was able to provide a simple and manageable component structure for the FLAD console.

In FLAD we have used several built-in components such as Google Sign-In. These components supposed to redesign the rendering view in order to align with the composed theme of the FLAD front end console. This redesigning was a bit harder activity since styles should be injected into the component separately. Material UI uses a method named CSS in JS to create the style object to be injected. This method provides object-oriented way of writing style objects. A sample code of style object is below.

[1]https://github.com/react-bootstrap/react-bootstrap
[2]https://github.com/Semantic-Org/Semantic-UI-React
[3]https://github.com/ant-design/ant-design
[4]https://github.com/grommet/grommet
[5]https://github.com/mui-org/material-ui

```
  const styles = {
    authLogo: {
        paddingRight: '10px',
        fontSize: '1.2em',
    },
    authBtn: {
        '&:hover': {
            backgroundColor: '#0D47A1 !important',
        }
    },
    links: {
        textDecoration: 'none',
        paddingLeft: '15px'
    }
};
```

The usage of CSS in JS has a small learning curve to understand the mapping of normal CSS into object way. After the learning, this method provided a considerable speed to the front end implementation process.

As explained in Chapter 04 Design consideration under the section of Front end design consideration, FLAD uses ESLint coding style to improve the code base. This method helped to keep the code base clean and easy to understand manner. It took some time to get in touch with this approach during the front end development phase.

FLAD provides REST API to the end users in order to consume the services they are expected use as explained in Chapter 01 under Introduction section. I have studied the process of designing API and best practices. In Chapter X Overall process describes the usage of REST API and Chapter X describes the API design approach I have followed during the designing of the API. There were several approaches available on the internet. After that by considering these approaches, features of these approaches used to design the API of FLAD. During the designing stability, usability, versioning, robustness and developer-friendliness were the main concerned areas in order to achieve the effective API design for the FLAD. Next activity related to the API designing was designing of JSON objects which consumes and provides by FLAD. This also provided the considerable amount of easiness of understanding the API.

FLAD consists of Admin panel for the FLAD administrators to manage the system as explained in Chapter 5. I have designed this admin panel and implemented both front end and back end services. Admin panel home includes summaryBox

components and usageTable component. These components behaviour is described in the later of Chapter 05 under front end implementation. The implementation of summaryBox component was a bit harder activity. It should behave as a fully dynamic component because this component repeats four times within the same page in order to reduce the complexity of the page rendering. SummaryBox uses the higher-order component created by withStyles to inject an array of styles into the DOM as CSS.

FLAD Admin panel home page contains the usageChart component which describes the requests handled by the server within a selected time frame in each category named TTS, STT, OCR and Translation. In designing this component I have used recharts which is free and open source react chart design tool. I have studied the documentation provided by recharts developers to enhance the chart functionalities and view of the chart to map with the FLAD functionalities.

FLAD Admin panel contains container for users, projects, earnings and requests. Each of these containers contains breadcrumbs component and EnhancedTable component. Both of these components are designed from the scratch. The behaviour and implementation of these components explained in Chapter 05 under front end implementation.

When Implementing breadcrumbs component, It is required to write custom CSS for the component to get the look and feel of the final design as shown in Chapter 05 under front end implementation. Breadcrumb component designed using Chip component from Material UI and right arrow component from Material icons.

The enhancedtable component was the most time consumed UI component I have implemented in FLAD. This table consists of a set of features. As described in Chapter 05 under front end implementation, parent component state changes depending on the user action on the child component. This issue overcome with the passing function as a component prop. This table actions depend on the data set it is representing. Due to this reason actions has to be passed with descriptive JSON array to the component for the dynamic rendering purpose.

The next part I have worked in this project is Admin panel back end services implementation. Admin panel requires endpoints for users, projects and requests components. These APIs contains different features for each component such as get user details, get all users, get projects authored by given user. The response JSON for each endpoint is constructed using org.json.JSONObject package. This package provides a large set of features to construct JSON object using Java objects. In addition to that, Additional endpoint added for the data displayed in FLAD admin panel such as daily counts of registered users, created projects and handled requests. Chart data is also passed in this above response in the required format

in the usageChart component.

Appendix O

Contribution of D.A.B.P. Dodangoda

My part of the project includes two main sections of the project. I contributed to the core framework of the project. I also built a mobile application as a proof of concept to the framework we built. Our framework is focused on linguistic application developers. Most of the linguistic application developers are mobile application developers. Therefore building a mobile application as the proof of concept is more suitable than building a web application. In the following context, first I have mentioned how I contributed to the core framework. Then I have discussed about mobile application which is the proof of concept.

From the work I have done in core framework I can mention three main contributions. They are, integrating Tesseract-OCR with the framework, integrating Google Cloud Vision API (Google OCR) with the framework and integrating Google Cloud Speech API (Google STT) with the framework. Our framework is composed of services of four linguistic components - Text to Speech (TTS), Speech to Text (STT), Optical Character Recognition (OCR) and Language Translation. From these four components I contributed to 2 components - OCR component and STT component.

Integrating Tesseract-OCR to the framework was my first task. Tesseract library which is know as 'libtesseract' is originally written in C/C++. [1]. As our framework is built using Java this 'libtesseract' library cannot be used. But there are bindings to 'libtesseract' for other languages like Java, .NET. Therefore I used the Java binding, which is a JNA wrapper known as Tess4J. [2]. Tess4J is released and distributed under the Apache License, v2.0. Tess4J is available from Maven Central Repository. [3]. As we have used Maven as the build tool in our project, I could integrate Tess4J with the help of Maven. Then I used the services of Tess4J library in the application by calling its methods.

[1]https://github.com/tesseract-ocr/tesseract

[2]http://tess4j.sourceforge.net/

[3]http://mvnrepository.com/artifact/net.sourceforge.tess4j/tess4j

Integrating Google Cloud Vision API was my next task. Providing services of different vendors of same linguistic component is a main functionality of our system. So we decided to use Tesseract-OCR and Google Cloud Vision API as two vendors for the OCR linguistic component. To use the service Google Cloud Vision API, I had to create a project in Google Cloud Platform which is accessed using the FLAD's Google account. Then I created an API key from the project in order to authenticate requests. The service of Cloud Vision API is available as a REST service that uses HTTP POST operation. The requests should be directed to the REST API `https://vision.googleapis.com/v1/images:annotate`. The request has to be authenticated by appending the API key to the end of the above mentioned Google Cloud Vision API endpoint as `https://vision.googleapis.com/v1/images:annotate?key=API_KEY`. I implemented a method to send request to the Google Cloud Vision API with the JSON body containing image content in base64-encoded format and handle the response sent by the Google Cloud Vision API.

My next task was to integrate Google Cloud Speech API to the framework. We can send an audio and receive a text transcription from the Google Cloud Speech API service. Google Cloud Speech API provides two types speech recognition techniques as synchronous and asynchronous. Synchronous technique returns the recognized text for short audio (less than 1 minute) in the response as soon as it processed. I used this synchronous speech recognition technique when integrating Google Cloud Speech API to the framework. The service of Google Cloud Speech API is available as a REST service from the `https://speech.googleapis.com/v1/speechendpoint`. So I used this endpoint to get the speech recognition service of Google Cloud platform. I had to authenticate the request by appending the API key to the end of the above mentioned endpoint as `https://speech.googleapis.com/v1/speech:recognize?key=API_KEY`. I implemented a method to send the request to the Google Cloud Speech API with a JSON body containing the content of the sound file as base64-encoded and handle the response sent by Google Cloud Speech API.

When integrating the above mentioned components I had to face several challenges. Google Cloud Vision API requires image as a base64-encoded image string in the request. Therefore I had to find a way to convert the image to a base64-encoded image string using Java and implement it in the framework. The same problem occurred when integrating Cloud Speech API to the framework. The audio content supplied in the request body should be base64-encoded. Therefore I had to find a mechanism to convert an audio content to a base64-encoded audio content using Java and implement it in the framework.

After contributing to the framework core, I shifted to mobile application de-

velopment. This mobile application is built on top of the framework, using all the services of the framework which are provided through the REST APIs. When building the mobile application I had to play the role of a user of the framework. The mobile application is built using Xamarin which is used to build mobile applications in C#. Mobile application has three main dashboards including the home screen. In the home screen shown in I.1 users can select the type of service they need. It can be either single services or complex services. Single services dashboard shown in Figure I.2 offers the services of single linguistic components such as TTS, STT. Complex services dashboard shown in Figure I.3 offers the services of combined linguistic components. Before getting the OCR service and the STT service I had to upload the file to the FLAD server using the /upload endpoint and get the name of the file in the server. Then this name is used to get the OCR or STT service. When getting the TTS service, I had to download the generated audio file using the downloadable link provided by the server. After that file being saved in the device I had to make it play automatically. When getting the Translation service I didn't have to do extra activities like in OCR, STT and TTS. I only had to send the text and the language in to which the text should be translated and handle the response received by the server. When requesting all services I had to pass the API key with the header section of the POST request to get authenticated. The API key is received when I created the project using the FLAD Console.

When building the mobile application I had to face lot of challenges. I used Xamarin Forms to develop the application. Xamarin Forms is a cross-platform natively backed UI toolkit abstraction that allows developers to easily create user interfaces that can be shared across Android, iOS, Windows, and Windows Phone. When building this application I only focused on Android. Xamarin Forms solution comes with two or more projects. One project is a portable project to write the shared code which is shared between the Android, iOS and Windows platforms. Other projects are to write the native code for Android, iOS and Windows platforms. So when building this mobile application, some functionalities like downloading the audio file that is rendered in TTS service and playing that file cannot be done from the portable project where the shared code is written. Therefore I had to find solutions to perform those actions from the native android project. In future if I'm going to expand this application to iOS and Windows platforms I need to write native codes in iOS and Windows platforms to perform those actions.

Finally the project Framework for Linguistic Applications Development ended covering a very big scope with a back end (core framework) done in Java, front end (FLAD Console and Admin Panel) done in React/Redux and Proof of Concept (mobile application) done in Xamarin. As a team we were able to achieve such a

big milestone because of hard working nature and commitment of every member of the team who worked at same capacity and sharing of knowledge with the team.