Department of Computer Engineering

CS 492 Senior Design Project

Low-Level Design Report

Sum of Sounds

Team members

Aliyu Vandana Saifullah

Bilal Siraj

Kasymbek Tashbaev

Supervisor

Prof. Dr. Uğur Doğrusöz

Jury Members

Prof. Dr. Varol Akman

Prof. Dr. Özcan Öztürk

February 15, 2020

Table of Contents

1. Introduction

SumOfSound is a web application that voices out mathematical formulas and equations. In recent times speech-to-text applications became popular in the technological market. Most of these applications read books, articles, documents, etc. in the almost human voice. The only aspect they fall short is in reading out mathematical formulas and equations. As such, most of these applications skip the formula parts when they encounter them in documents making the reading less coherent. This tends to largely affect people with visual impairments who do research in mathematics or physics and need to read scientific articles and books, which usually contain lots of formulas and cannot be read by these applications [1].

Having observed this situation and identified a possible target market, we as a team decided to develop an application that could voice the mathematical formulas and equations. Such a project can be useful for both developers of text-to-speech applications and people with vision problems.

In this report, we provide a detailed description of the architecture and system design of our project. We will discuss the purpose of the system and the design goals of our project. Existing architectures in the market will be discussed in relation to what makes our proposed architecture similar in some regard and different in terms of what we are bringing to the table. Necessary diagrams for the subsystem decomposition will be provided, design decisions and subsystem services will be discussed. With respect to the progress we have made so far in this project, we will discuss the new knowledge acquired and learning strategies we have used in the last section.

1.1 Object design trade-off

1.1.1 Complexity vs Usability

More features add more complexity which in turn reduces the user-friendliness of the application. The application, however, should be able to handle the interaction between the server and the client in the background. So while keeping the user interface simple with minimal

features such as: upload document and enter lateX string, functions like text and sound conversion will be handled in the background to cater for the balance between the usability and complexity.

1.1.2 Performance vs Cost

SumofSound is a web application that will be used by different devices on different operating systems. As such, performance is very important to us. While keeping the performance high, we should keep the cost as low as possible.

1.1.3 Compatibility vs Extensibility

SumofSound uses the mathpix api to convert image files to lateX string. These image files can either be in jpg or png format. Due to this, we have to compromise between using different api that may accept additional file formats but may not necessarily be as reliable as mathpix and sticking with mathpix's limited formats but reliable result.

1.2 Interface documentation guidelines

Our report follows the standard convention where all class names are named with standard 'ClassName()' form, methods are named 'methodName()' and variables as 'variableName'. A complete description of the class follows the convention where class names come first, then attributes and methods in that respective order. Brief class description is provided below the class name.

1.3 Engineering standards

UML is the most commonly used software engineering standard when it comes to describing the components of a system. As such, in this report we follow the UML design principles in each class description. Each package section also contains the UML diagram of that package.

1.4 Definitions, acronyms, and abbreviations

Client: The part of the system the user's inputs will be handled and delivered to the server

Server: The part of the system that will maintain our backend which consists of song scheduling phases and managing data.

API: Application Programming Interface

UI: User Interface

URL: Uniform Resource Locator

HTTP: Hypertext Transfer Protocol

UML: Unified Modeling Language

2. Packages

This section discusses the software architecture of SumOfSounds in detail. The system is composed of two packages: Web Client and Web Server.

2.1 Web Client

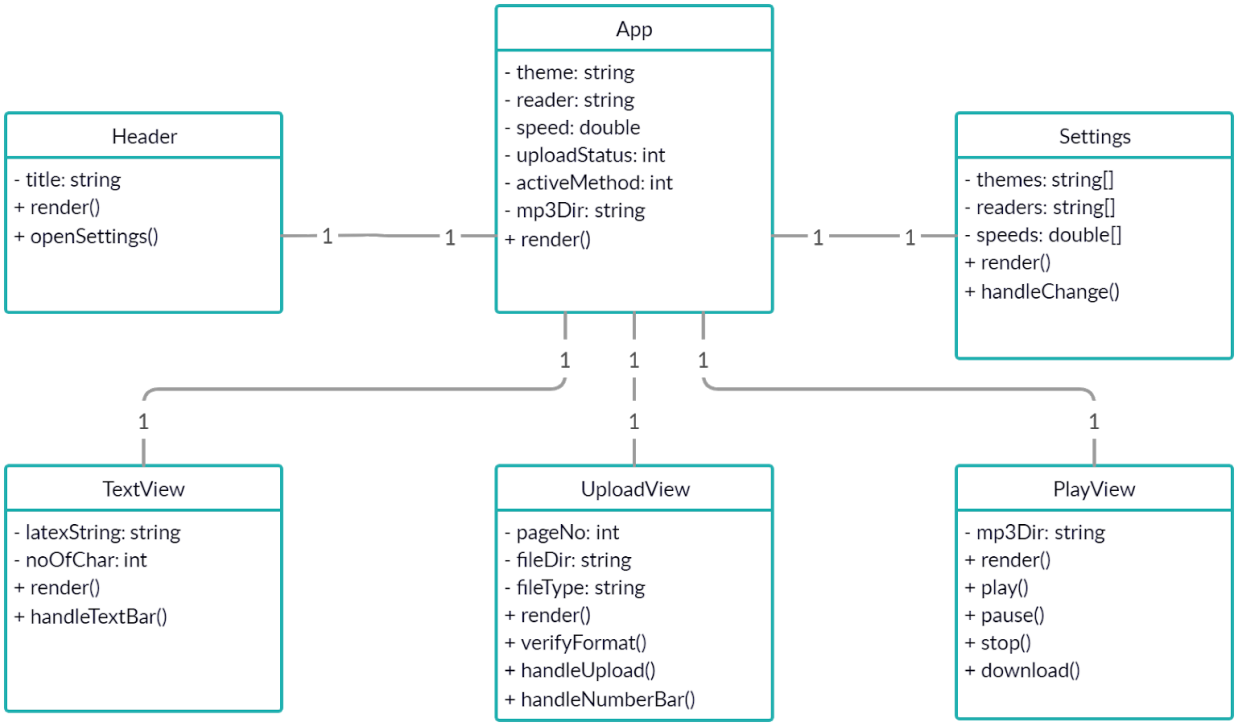

Figure 1. Class diagram of the Web Client

PlayView: Provides the functions to play, pause, stop and download the sound file.

TextView: Provides text bar, where users can enter a LaTeX string, and the function to handle the text bar.

UploadView: Provides an upload screen, where the user can upload the file, and the functions to handle the upload.

Header: Displays the title provides a button, which opens the Settings page, and functions to handle this operation.

Settings: Provides a new screen, where the user can change the theme and set the speed of mp3 play.

App: Contains all other frontend components and provides communication between them.

2.2 Web Server

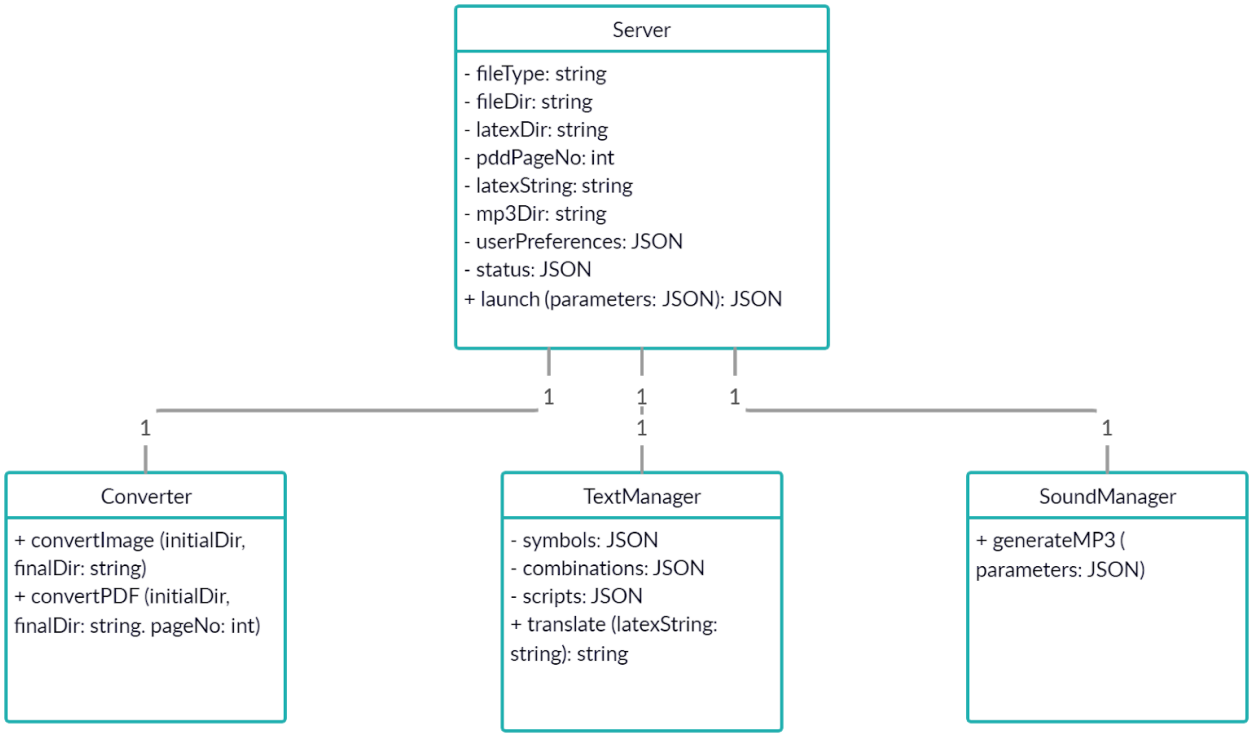

Figure 2. Class diagram of the Web Server

Converter: Converts image and pdf files to the LaTeX file. Uses MathPix API.

TextManager: Manages translation of LaTeX code to plain text.

SoundManager: Manages the creation of an mp3 file from plain text. Uses Google text-to-speech API.

Server: Contains all other server components and provides communication between them, as well as provides APIs to the client application.

3. Class Interfaces

3.1 Web Client

class App
Handles communication between client components, and with the server application. Holds certain client preferences, for example the display theme.
Attributes
var theme
var reader
var speed
var uploadStatus
var activeMethod
var mp3Dir
Methods
render()

Class Header
Attributes
var title
Methods
render()
openSettings()

Class Settings
Provides an interface of options to change the theme of the web app, the reader and the speed of the MP3 file.

Attributes
var themes
var readers
var speeds
Methods
render()
handleChange()

Class TextView
Provides interface to enter Latex input.
Attributes
var latexString
var noOfChar
Methods
render()
handleTextBar()

Class UploadView
Provides interface to upload input files to the program, and allowing to switch between upload methods.
Attributes
var pageNo
var fileDir
var fileType

Methods
render()
verifyFormat()
handleUpload()
handleNumberBar()

Class PlayView
Provides media control interface.
Attributes
var mp3Dir
Methods
render()
play()
pause()
stop()

3.2 Web Server

Class Server
Handles communication between server components and to the client side.
Attributes
var fileType
var fileDir

var latexDir
var pddPageNo
var latexString
var mp3Dir
JSON userPreferences
JSON status
Methods
JSON launch(parameters: JSON)

Class SoundManager
Requests and returns audio of plain text input from Google Text-to-Speech API.
Methods
generateMP3(parameters: JSON)

Class Converter
Handles conversion of non-LaTeX input into LaTeX.
Methods
convertImage(var initialDir, var finalDir)
convertPDF(var initialDir, var finalDir, var pageNo)

Class TextManager
Handles conversion of LaTeX into plain text that is readable.
Attributes

JSON symbols
JSON combinations
JSON scripts
Methods
var translate(var latexString)

4. Glossary

5. References