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 the low-level design of our system. We will discuss the design trade-offs of our project and the engineering standard we used. Necessary diagrams for the client and server-side will be provided, design decisions and subsystem services will be discussed.

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 the sound conversion will be handled in the background to cater to the balance between 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 results.

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. The 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

I/O: Input/Output

UI: User Interface

UML: Unified Modeling Language

2. Packages

2.1. Architecture Style

SumOfSounds follows a Client/Server software architecture style, in which the server delivers the data to be displayed by the client [2]. So, by following this architecture, our program can support concurrent requests of many clients without having any conflicts.

There are two packages Client, which contains classes required for the frontend, and Server, which contains classes required for the backend.

2.2. Client

The Client package contains six classes. These classes are responsible for the graphical user interface and user experience. Since the front-end will be implemented by using the React library, all classes will be React components. A React component is a JavaScript class that optionally accepts inputs and returns HTML via a render function that describes how a section of the UI should appear [3].

App class will be the main class, thereby it will contain all other client classes and will communicate with the server package.

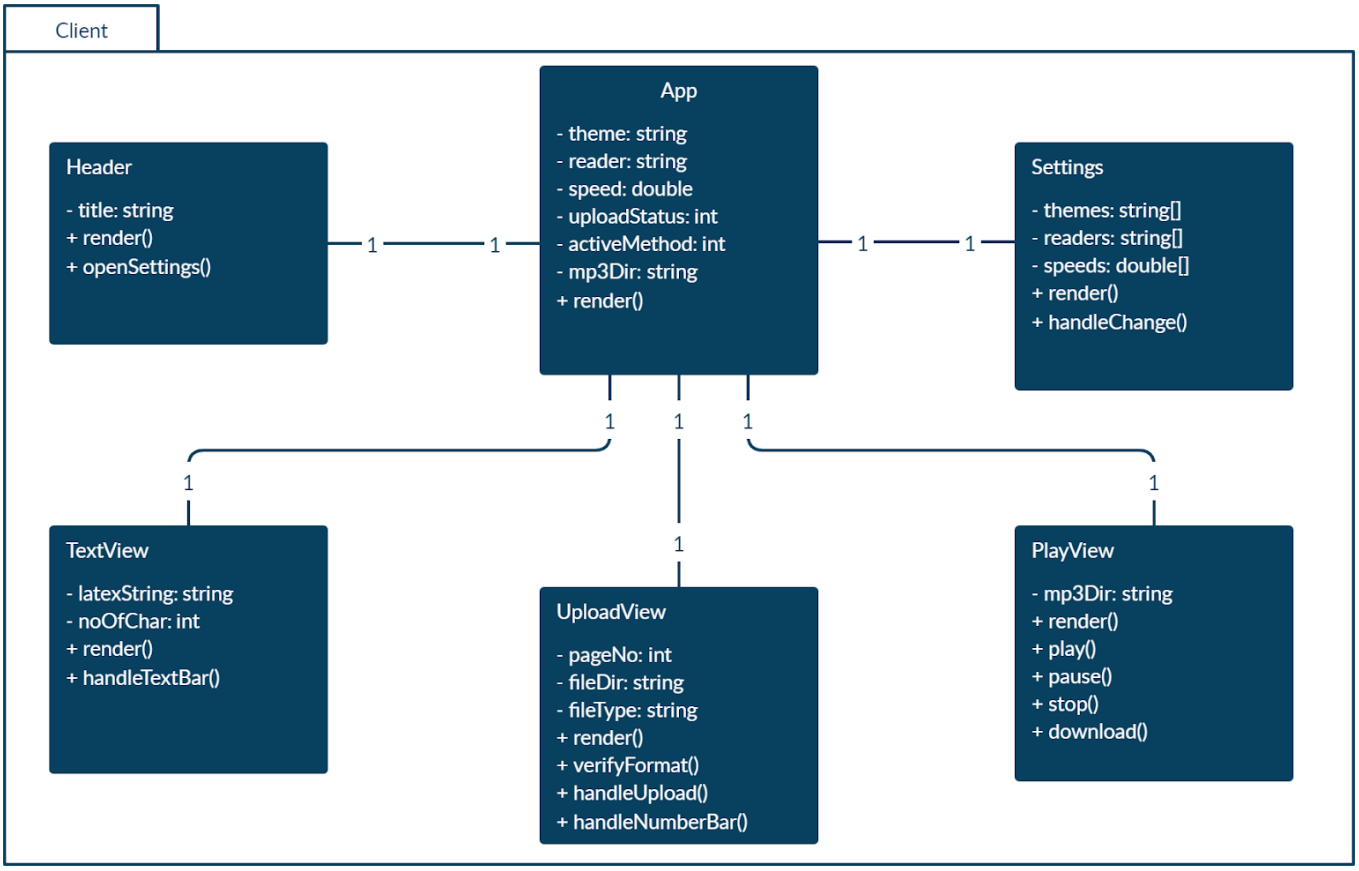

Figure 1. Class diagram of the Web Client

PlayView: Provides the UI elements and functions that allow users to play, pause, stop and download the sound file.

TextView: Provides a 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 and 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. Moreover, it communicates with the server package by using the API provided by the server.

2.3. Server

The Server package contains four classes, which are responsible for generating mp3 files from the uploaded image, PDF or LaTeX files. Classes will be implemented in JavaScript using the Node.js framework. The Server class will be a Facade class, so it will provide a simple API to the client.

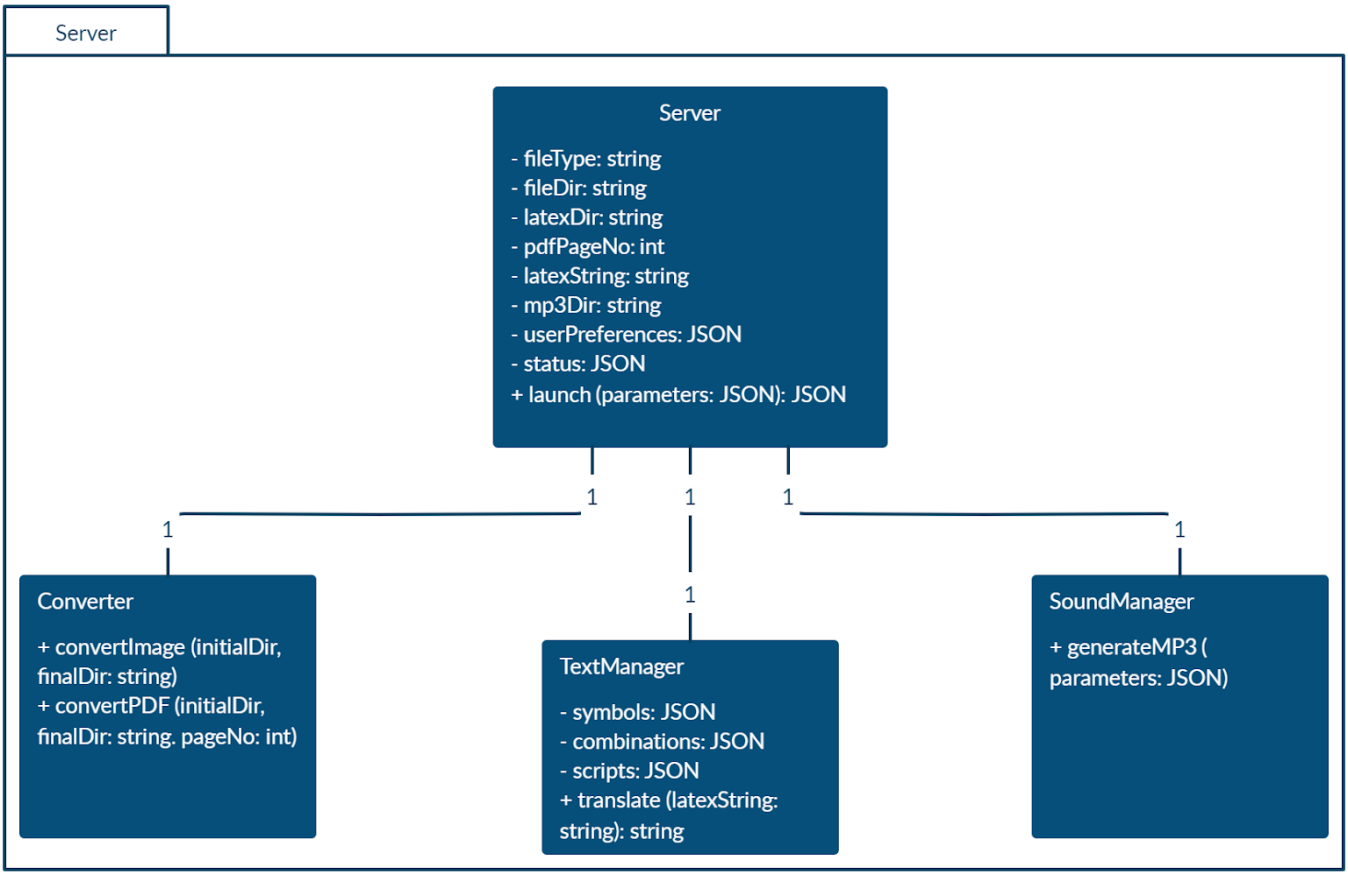

Figure 2. Class diagram of the Web Server

Converter: Mages the conversion of uploaded files to LaTeX. If the uploaded file is PDF, firstly it converts the chosen page to the image, then by using MathPix API converts the image to LaTeX.

TextManager: Manages translation of LaTeX script to plain text. The class contains JSON files with symbols, scripts, and their transcription. By using this data, it generates plain text from the LaTeX script. Moreover, our system is context-dependent, so according to context, a symbol can be translated differently.

SoundManager: Manages the creation of an mp3 file from plain text. It gives the user preference settings and the plain text as a parameter to Google text-to-speech API and gets an mp3 file.

Server: Contains a single function that runs other classes' functions in the right sequence. In addition, provides simple APIs to the client application.

2.4. Design Patterns

There will be only one design pattern used in our software: Facade for the Server class. In the Server package, all the complexities of the package are encapsulated in the Server class. The Server class acts as a facade through which internal components of the Server package can be used. Since we have only ten classes divided into two packages, our system is not complex and no other design patterns are applicable.

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
theme: holds user's choice for the display theme
reader: holds the user's choice for the reading voice, the user will be given options such as male or female
speed: holds users choice for the speed of the reading voice
uploadStatus: true if the file is uploaded, otherwise it is false
activeMethod: selected method to upload
mp3Dir: holds the location of the generated MP3 file
Methods
render()

Class Header
The header of the webpage with the Settings button and necessary functions to handle user settings.
Attributes
title
Methods
render()
openSettings(): opens settings page

Class Settings
It provides an interface of options to change the theme of the web app, the reader and the speed of the MP3 file.
Attributes
themes: array holding display themes
readers: array holding reader voices

speeds: array holding the reader voice speed options
Methods
render()
handleChange(): updates the relevant attributes when the user changes a setting

Class TextView
Provides an interface to enter LaTeX input.
Attributes
latexString: holds the user's input LaTeX
noOfChar: holds the number of characters of latexString, used to limit the length of the input
Methods
render()
handleTextBar(): will alert the user if the input is above the limit or accept valid input

Class UploadView
Provides an interface to upload input files to the program, and allows you to switch between upload methods.
Attributes
pageNo: the page number of the page to be read of an uploaded PDF file (only one page at a time)
fileDir: holds the location of the uploaded file
fileType: holds the file type of the uploaded file, used to decide how to handle file
Methods
render()
verifyFormat(): verifies the uploaded file is in a valid format
handleUpload()

handleNumberBar(): takes the user's choice for the page number of the PDF document to be read, if the uploaded file is PDF

Class PlayView
It provides a media control interface.
Attributes
mp3Dir: holds the location of the generated MP3 file
Methods
render()
play()
pause()
stop()

3.2 Web Server

Class Server
Handles communication between server components and to the client-side.
Attributes
fileType: holds the file type of the uploaded file, used to decide how to handle file
fileDir: holds the location of the uploaded file
pdfPageNo: holds a page of the pdf file, which will be voiced out
latexString: holds the LaTeX string the user input
mp3Dir: holds the location of the generated MP3 file
userPreferences (JSON): holds all the user preferences related to the output audio
status (JSON): error or success status that will be returned to the client
Methods

launch(parameters: JSON): runs all other functions

Class SoundManager
Requests and returns audio of plain text input from Google Text-to-Speech API.
Methods
generateMP3(parameters: JSON): generates MP3 audio of plain text using Google Text-to-Speech API

Class Converter
Handles conversion of non-LaTeX input into LaTeX.
Methods
convertImage(var initialDir, var finalDir): converts image to LaTeX using MathPix API
convertPDF(var initialDir, var finalDir, var pageNo): converts the page to be read of the uploaded PDF document to an image, to be later converted to LaTeX

Class TextManager
Handles conversion of LaTeX into plain text that is readable.
Attributes
symbols: mathematical and scientific symbols with their transcription
combinations: the combination of symbols that pronounced differently according to the context
scripts: LaTeX scripts and their transcriptions
Methods
translate(parameters: latexString): converts LaTeX to readable plain text based on predefined mappings

4. Glossary

Frameworks and Libraries

Node.js: JavaScript runtime environment, which provides a set of asynchronous I/O primitives in its standard library

React: JavaScript library for building user interfaces

5. References

[1] V. Vertogradov, “ Looking for volunteers,” *VKontakte*. [Online]. Available: https://vk.com/id11510315?w=wall11510315_11361/all. [Accessed: 13-Oct-2019].

[2] “What is Client/Server Architecture? - Definition from Techopedia,” *Techopedia.com*. [Online]. Available: https://www.techopedia.com/definition/438/clientserver-architecture. [Accessed: 29-Dec-2019].

[3] J. Kagga, “Understanding React Components,” *Medium*, 07-Jul-2019. [Online]. Available: https://medium.com/the-andela-way/understanding-react-components-37f841c1f3bb. [Accessed: 01-Mar-2020].