Bilkent University

Department of Computer Engineering

CS 491 Senior Design Project

High-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

December 29, 2019

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 Purpose of the system

Our main aim in developing sumOfSound is to solve the inherent problem in voicing out technical documents that contain formulas. SumOfSound is a web application that can be accessed from the browser using the appropriate URL. It will be able to voice out the formulas which will otherwise be skipped by the available text-to-speech applications in the market. This will be useful for the members of academia who may solely rely on text-to-speech technologies for reading texts. SumOfSound is also for those who want to have smooth reading experience devoid of lapses and also for those who want to obtain lateX strings of formulas in documents or images in supported formats.

Our application provides features to achieve the said tasks and enhance the overall user experience. The user interface is not overwhelmed with excess functionalities but the fundamental ones to enable easy navigation. Users will be greeted with a simple interface which allows them to enter source LaTeX, or upload a file to the system. If the user chooses to enter source LaTeX, and are ready to have it read out, they simply have to press the play button to have the program begin reading. In the case of uploading a document, the user can just click upload and choose the file. There is a pause and stop buttons which allow the user to control the reading. The user can also change the speed of reading to suit their preference and also select between two reading voices available. In the case of an uploaded document being displayed the user can choose a display theme to customize the size, font and other attributes of the text being displayed. Once a piece of text has been read, the user will have the option to download an MP3 file of the reading.

1.2 Design goals

1.2.1 Usability

The application will have a user-friendly interface that will allow users to navigate through it easily without difficulty. The application will not be flooded with a lot of functionalities that may overwhelm the user.

1.2.2 Extensibility

The application will be made such that new features can be added easily. As there is always room for improvement, the application will be updatable. The updates will not be heavy with changes that may overwhelm the users. We will implement the application such that minimal changes can be added. These changes will mostly be about applying new technologies rather than changing the functionality of the app.

1.2.3 Reliability

The application will not alter any files or images uploaded to voice out formulas. The application will not store users' information or track their activities in the browser.

1.2.4 Portability

SumOfSound will be implemented as a web-based application and can be accessed by anyone with access to the URL. It will maintain the same regardless of the browser or operating system.

1.2.5 Efficiency

In order to have optimized run-time performance, our application will have a threshold for the text part of the uploaded file or that will be read with the focus on the formula. The text part could have been eliminated completely but we realize reading some text before or after the formula could give more context to the formula.

1.3 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

1.4 Overview

SumOfSound will be a web app that can be accessed from a modern browser with its URL. Users will be greeted with a simple interface which allows them to enter source LaTeX, or upload a file to the system. If the user chooses to enter source LaTeX, they simply have to press the play button to have the program begin reading. The program will have a character limit according to which the source LateX will be read.

In the case that the text entered exceeds the character limit, the program will read everything up to the end of the character limit. If the user chooses to upload a document, the document will be displayed within the web app. If then the user clicks the play button, once again the program will read everything from the beginning up to the end of the character limit. Otherwise, the user will have the option to select the part of the document they want to be read, before clicking play.

The user will also be able to click on parts of the selected text in a document to have them repeated. Once again the character limit applies to this case as well. In all of these cases, the user will be able to click the pause button to pause the audio or the stop button for the program to stop reading. The user will have to choose between two reading voices and select their reading pace. In the case of an uploaded document being displayed the user can choose a display theme to customize the size, font and other attributes of the text being displayed. Once a piece of text has been read, the user will have the option to download an MP3 file of the reading.

2. Current software architecture

We did some research and realized that there are lots of text-to-speech applications on the market now, such as an Intelligent Speaker and Natural Reader that read plain texts without any problems, but when there are mathematical formulas and equations in the text almost all of such text-to-speech applications have trouble with voicing the formulas. For example, when Natural Reader meets mathematical formulas it just ignores them and continues reading the next plain text. Apart from this deficiency in reading formulas, the existing systems have user-friendly interfaces, give users the option to change reading voices and more. However, most have limitations and charge a premium to use the majority of their features [2].

3. Proposed software architecture

3.1 Overview

This section discusses the proposed software architecture of SumOfSounds in detail. The classes and decomposition of the system explained in the Subsystem Decomposition, while the Hardware/Software Mappings section explains which hardware each of the components is going to use. We are not using any database system in our program, and the reason is explained in the Persistent data management section. Access control and security section define the access and exit boundaries of our system. In the end, the Global software control section discusses how our server acts as the main controller, as well as boundary conditions such as initialization, termination, and failure conditions are explained.

3.2 Subsystem decomposition

SumOfSounds follows a Client/Server software architecture style. In Client/Server architecture, the server delivers the data to be displayed by the client [3]. So, by following this architecture, our program can support concurrent requests of many clients without having any conflicts.

The server in our program is responsible for the following three main operations:

1. Converting non-LaTeX files to LaTeX
2. Translating LaTeX code to plain text
3. Creating an mp3 file, in which given text is voiced

While, the client has to provide user-friendly UI, which allows users to upload supported files, customize preferences and play mp3, got from the server application.

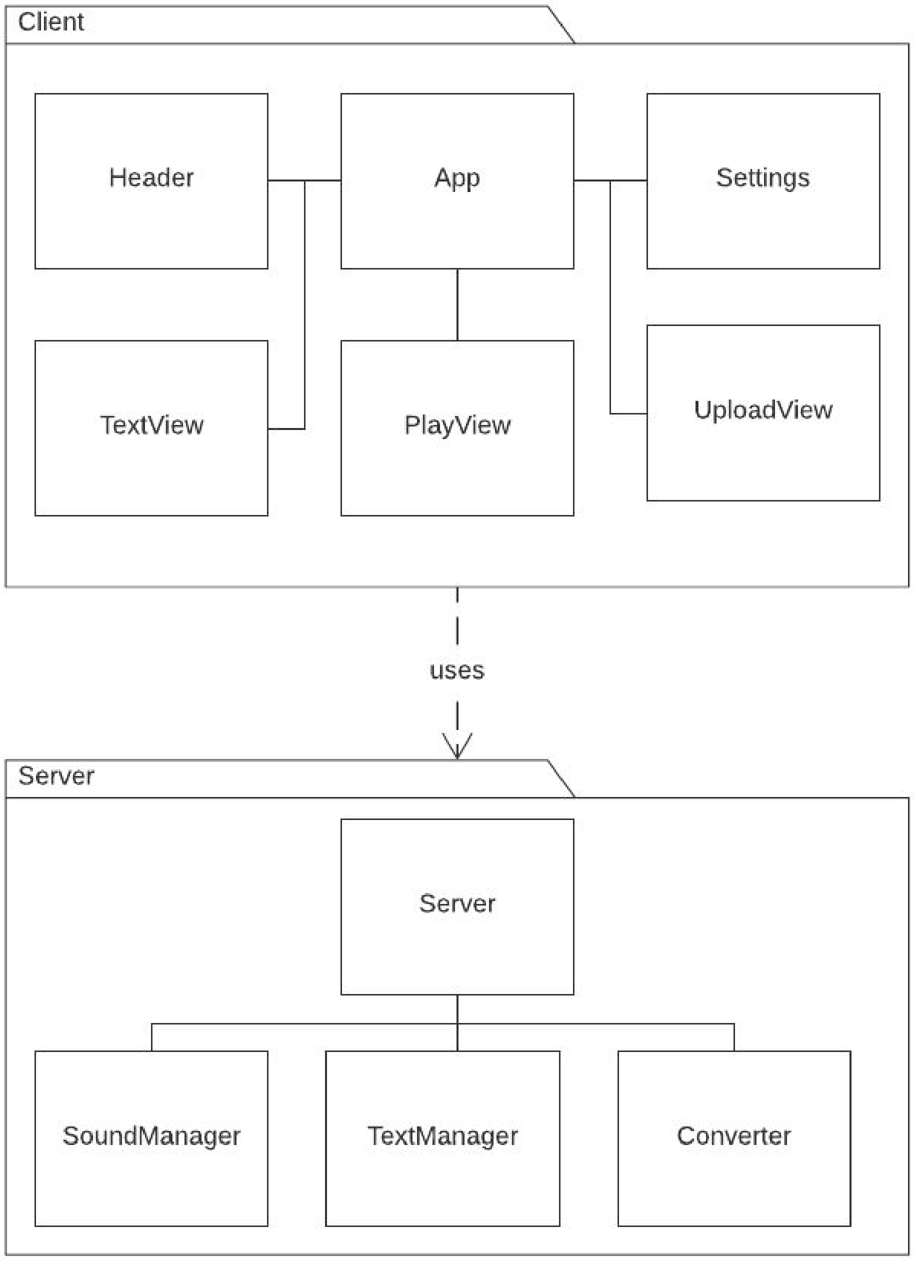

Figure 1. System Decomposition Diagram showing Client-Server architecture

3.3 Hardware/software mapping

The server application will run on the webserver, while the client application will run on the user's computer and it will require an internet connection to retrieve data from the webserver. The client will use the HTTP protocol to connect with the webserver.

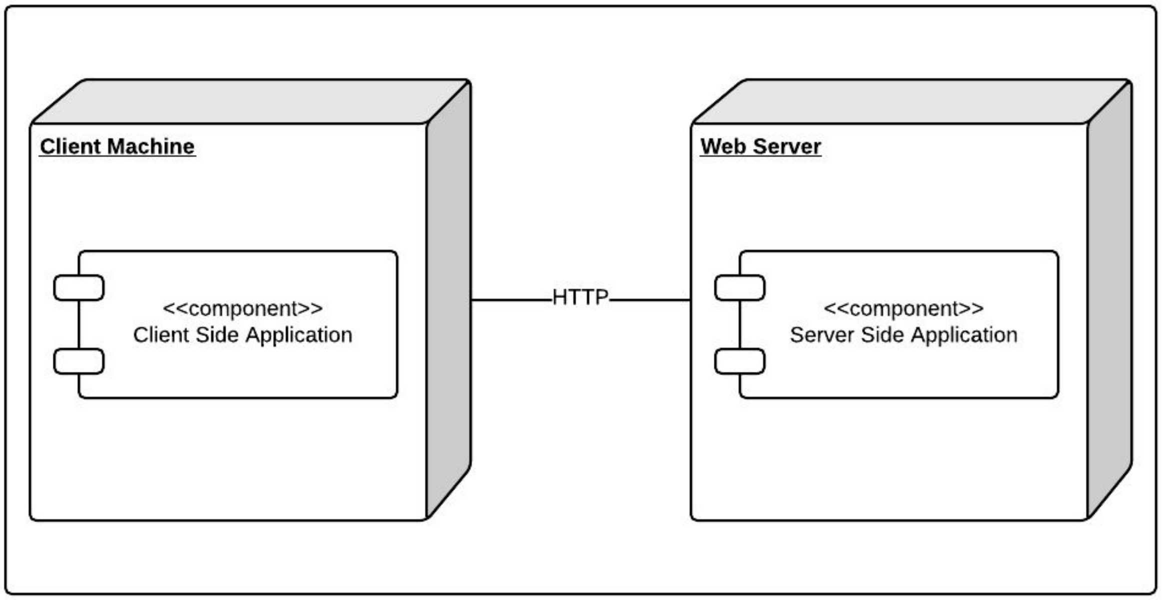

Figure 2. Hardware/Software Mapping

3.4 Persistent data management

In the current system design, there is no data that needs to be stored beyond any instance of the program execution that could be necessary in the future. All data that is needed for the proper execution of the program will only be maintained during program runtime. This ensures that the program's speed will not be limited by database accesses or writes. This also guarantees any data taken from the user is strictly for that session and is discarded when the session ends. For this reason, the software will not have any persistent data management system.

3.5 Access control and security

Since the program is a browser-based web application, it will be open for use by anyone with access to the web. There are no accounts and therefore no log on the system since there is no need to differentiate access privileges between users. The system is also not reliant on user-specific data, using only what is necessary for its legitimate purpose. There is no sensitive information that needs to be encrypted or to which access needs to be controlled. All users have access to the services offered through the interface of the web application. As of now, all code will be hosted on a public GitHub repository, this is mainly to make the code available to view as per the course requirements. This is not ideal for source code protection as the core of our program will be embedded within the code. The smallest step we could take towards source code protection would be to host our code off of a private repository or our own servers.

3.6 Global software control

Our system will follow a centralized control flow model, specifically a call-return model. The App subsystem on the client-side and the Server subsystem on the Server side will communicate with each other and pass on control down to other subsystems. It is important to note that this is not a structural model but rather a model of program dynamics. The subsystems at the top will call and await the response of lower subsystems until control is returned back to the top again.

3.7 Boundary conditions

The main boundary condition that will be described is how the whole system can be started up and what services need to be initialized first. It is necessary that all server components and the Server subsystem itself are started first because the Client (App service) depends on it and the data provided by the server. Once both the Client-side App service and the Server-side Server service have been started, the Server will test for communication with the client-side to ensure there is a connection. It will also test the communication with the Converter and SoundManager subsystems to detect any issues connecting to the dependent API services.

In the case that the user input, either text or the file uploaded is not valid, it will be detected and stopped at the top of the control hierarchy. The user will be warned with an error message showing the types of valid input. If the processing of a valid input fails due to loss of connection to the dependent services, the user will be prompted to try again in some time so that connection can be reestablished before receiving another request, which would otherwise fail again. The input of the failed request will be discarded. In the case that connection to the client is lost after a valid request is made, any output that cannot be delivered to the client will be discarded and the user will have to try again. The user will be notified of the loss of connection and failure of the request when the connection is reestablished.

4. Subsystem services

4.1. Client

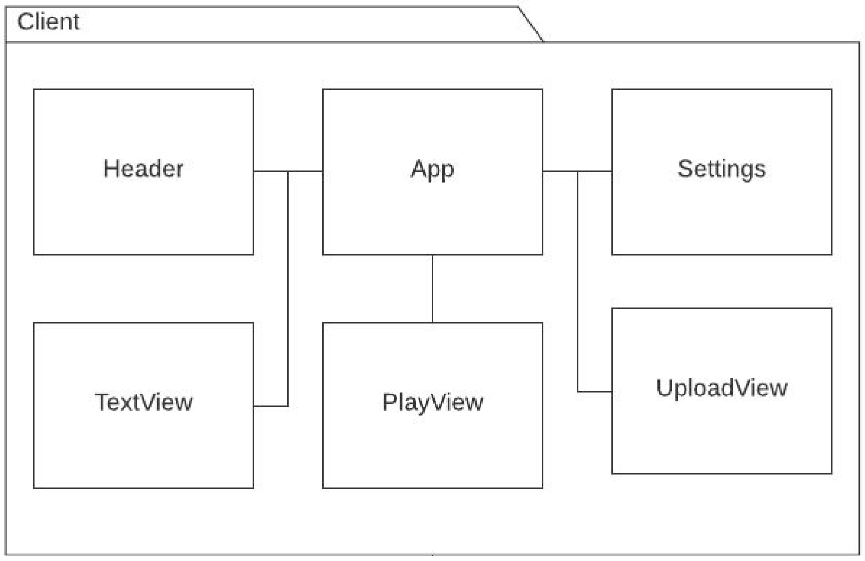

Figure 3. Client Subsystem

PlayView: provides buttons, which allow users to play/pause or download the sound file, and a progress bar, which displays how much sound file is played.

TextView: provides text bar, where users can enter LaTeX code, and buttons to upload text and switch between upload methods.

UploadView: provides an upload screen, where the user can upload the file, and a button to switch between upload methods.

Header: displays the name of the program and provides buttons to switch between the settings and main pages.

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

App: provides communication between other client components, as well as communication between them and server application.

4.2. Server

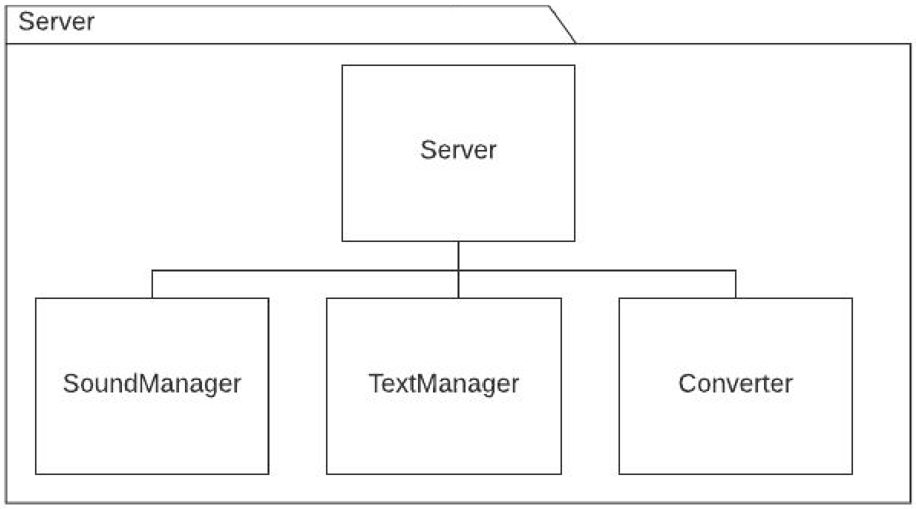

Figure 4. Server Subsystem

Converter: manages conversion of non-LaTeX files to LaTeX.

TextManager: manages translation of LaTeX code to plain text.

SoundManager: manages the creation of an mp3 file from plain text.

Server: provides communication between server components and provides APIs to the client application.

5. New Knowledge Acquired and Learning Strategies Used

To categorize our system design, we learned about the various software control models doing internet research.

6. References

[1] V. Vertogradov, “ Looking for volunteers,” *VKontakte*. [Online]. Available: https://vk.com/id11510315?w=wall11510315_11361/all. [Accessed: 13-Oct-2019].
[2] “Natural Reader” *Free Text to Speech: Online, App, Software & Commercial license with Natural Sounding Voices.* [Online]. Available: https://www.naturalreaders.com/. [Accessed: 10-Nov-2019].
[3] “What is Client/Server Architecture? - Definition from Techopedia,” *Techopedia.com*. [Online]. Available: https://www.techopedia.com/definition/438/clientserver-architecture. [Accessed: 29-Dec-2019].