EE/SE/CPRE 491 - Spring 2019

Student Suggested Project

Sheet Vision

Design Document

Team Number

sddec19-13

Faculty Advisor

Alexander Stoytchev

Team Members

Bryan Fung
Garrett Greenfield
Ricardo Faure
Trevin Nance
Walter Svenddal

Team Website

http://sddec19-13.sd.ece.iastate.edu/

Version: May 2/Version 2

Table of Contents

List of Abbreviations & Symbols **2**

List of definitions **2**

1. Introduction

1.1. Acknowledgement 2

1.2. Project Statement 2

1.3. Operating Environment 3

1.4. Use Case Diagram 3

1.5. Intended Users 4

1.6. Assumptions and Limitations 4

1.7. Expected End Product and Deliverables 4

2. Specifications and Analysis 5

2.1. Project goals, Deliverables 5

2.2. Design Specifications 5

2.3. Proposed Design/Method 5

2.3.1. Block Diagram System 6

2.4. Design Analysis 6

2.5. Process Details 7

3. Testing and Implementation **7**

3.1. Hardware/Software 7

3.2. Graphics 8

3.3. Functional Testing 9

3.4. Non-functional Testing 9

3.5. Modeling and Simulation 9

3.6. Standards 10

3.7. Implementation Issues and Challenges 10

3.8. Design Testing / Implementation 11

4. Closing Materials **12**

4.1. Conclusion 12

4.2. References 12

List of Abbreviations & Symbols

1. AWS Amazon Web Services
2. API Application programming interface
3. MIDI Musical Instrument Digital Interface

List of Definitions

1. Sheet Music - Music in its written or printed form. [1]
2. Musical Notes - A sign or character used to represent a tone, its position and form indicating the pitch and duration of the tone. [2]
3. Tabs - A form of written music, but instead of being represented in the traditional sense (what tone it makes), notes are represented by the specific position they are supposed to be played in.

1 Introduction

1.1 Acknowledgment

We would like to express our gratitude to our advisor Dr. Stoytchev for taking the time to help us map out our project, as well as providing technical assistance. We would also like to thank Dr. Daniels for providing us with course resources and guidelines to follow, to better ensure our project success.

1.2 Project Statement

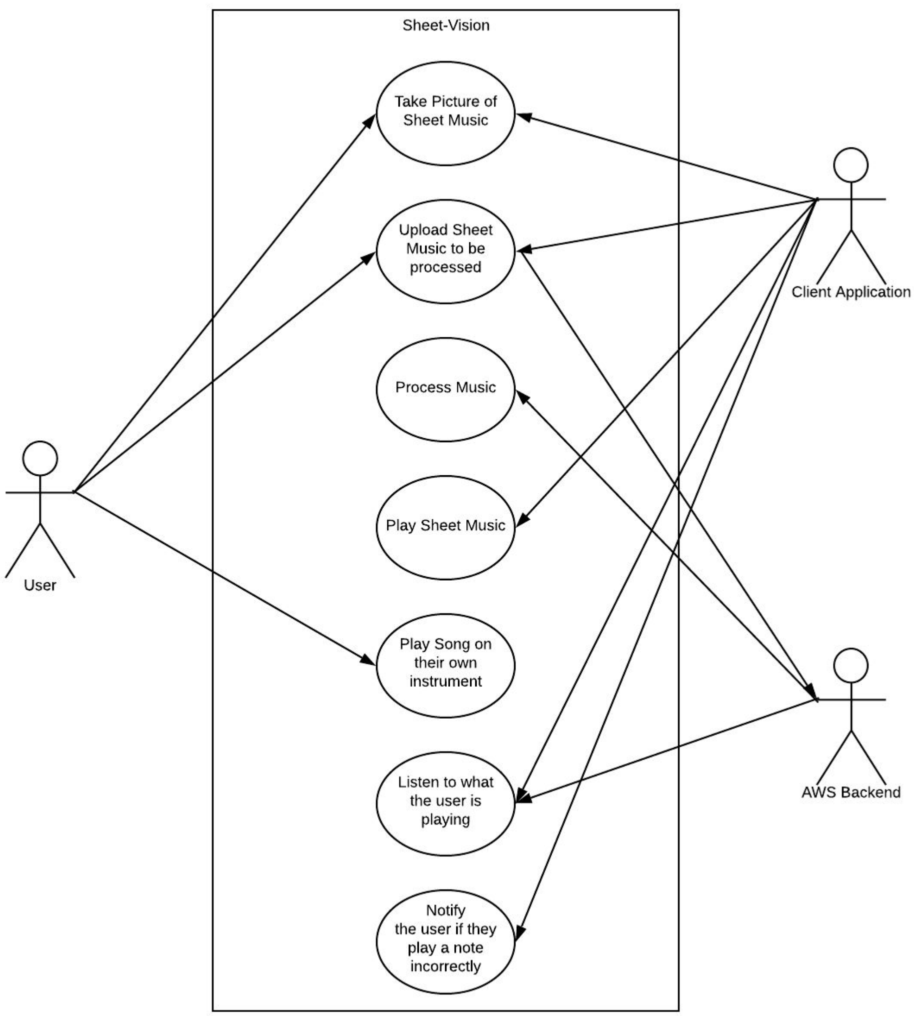
Reading sheet music is no easy task. With the creation of alternate ways to learn how to play music, such as tabs and youtube tutorials, there has been a decline in the amount of people who can properly read sheet music. The problem with tabs and other kinds of methods of reading music is that they lack the complexity to be able to convey all of the specific nuances that a specific piece may have. The best option that captures all of the nuances that most musicians wish to convey when writing music, is sheet music. The problem with sheet music is that it can be very difficult at first, and since there is a decline in the amount of people that can read it, it can be difficult to find a proper way to read it.

Our solution for this is Sheet Vision, an application that can read and show a user how the sheet music is played, and how it is supposed to sound. This will lead to the user being able to draw parallels between what is on the sheet, and the music being played, supplementing the learning process of reading sheet music. Not only will our application play the music on the sheet, but it will also listen to the user playing it, and will give proper feedback to the user, to fix mistakes they may be making.

1.3 Operating Environment

Our product is expected to be used quiet indoor environment, with bright and uniform lighting. Our ideal environment is quiet to allow our application to more accurately pick up sound from the user when they're playing along with our on-screen piano prompt, showing which keys should be played. Lighting is also important to allow the image taken to be clear and evenly colored. This is necessary so our computer vision algorithm can accurately detect the notes and rests, and where they lie on the staffs.

1.4 Use-Case Diagram



This is our use case diagram for our application. It demonstrates the use case ideas, services and actors for the application.

1.5 Intended Users

This product is intended for beginner musicians readers alike. This application should provide instructions simple and clear enough for even first-time musicians can keep up with using our product, yet powerful enough to ease some of the struggle of reading more complex songs for veteran musicians.

1.6 Assumptions and Limitations

Assumptions:

- Sheet music, though varying in format, will have the same characteristics and symbols you'd expect in a modern piece of sheet music, such as the symbols used to represent notes and rests. An example of the assumed form of the sheet music can be found in Figure 0.1.
- User will have a strong internet connection to be able to take full advantage of note corrections.
- User will have access to a computer or mobile device.
- If the user is using a mobile device, that device must have a camera.

Limitations:

- Some features will be limited based on the users possession of sheet music and a musical instrument.
  - The user needs to provide their own sheet music to scan, since our application will not provide it for them.
  - The user will need an instrument to make use of the play-along feature.
- The quality of our output will rely on the quality of both a users camera and microphone, depending on which feature they are making use of.

1.7 Expected End Product and Deliverables

- Sheet Vision Multi Platform Application - Expected Delivery Date : December 2019
  - System to read in images of sheet music.
  - Computer vision system used to decode sheet music into information useful for the application.
  - System that uses information provided by the computer vision system to select what notes should be played and when.
  - Audio processing system that can detect and decode audio into information useful for application.
  - System that uses information provided by audio processing system to notify the user if they played the song correctly or not.

    - Can run on a multitude of systems:
        - Desktop Environment.
        - Mobile Environment.

2 Specification and Analysis

2.1 Project goals, Deliverables

- ReactJS desktop application for windows.
- Application can read images from camera/file directory.
- Algorithm can recognize music notes in sheet music.
- Application can play the correct notes that come from the processed sheet music.
- Application can listen to user audio using the microphone.
- Algorithm can recognize what music notes are being played by the user based on audio input.
- Application can compare output from audio processing and image processing to determine if user is playing the correct notes.

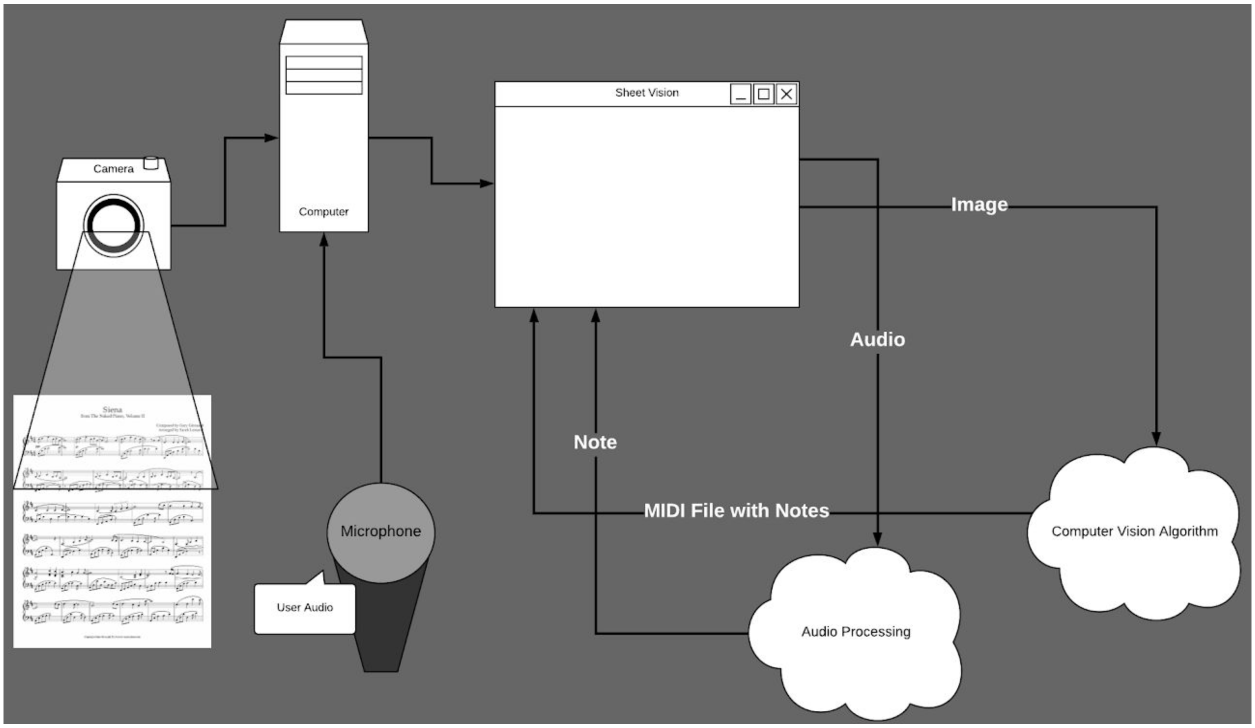
2.2 Design Specifications

The mobile phone application will be a standard mobile application, the Windows application will be .exe and the Mac application will be .app. The application will be able to accept both .png and .jpg images to analyze. Then the a front end application will be able to play the MIDI files that are received from AWS, and then as an additional feature, it will be able to listen to an audio feed from the microphone in order to detect errors in the music it hears from.

2.3 Proposed Design/Method

The design calls for the front end code to be written in ReactJS, wrapping it in Electron and React Native to provide multi platform usability. The machine vision will written in Python using the OpenCV library along with Numpy on an Amazon Web Services (AWS) server to take the load off the users device, and to allow the machine vision code to be reused without having to be re written for each platform.

2.3.1 Block Diagram of System



2.4 Design Analysis

<u>Architecture:</u>

The computer vision algorithms will mainly be run on a machine on an AWS web machine and be accessed with the use of API requests sent from the client application to this machine, which we will refer to as the server. The client application will be available in the form of both mobile and desktop application, both multiplatform, thanks to the frameworks that we are using to create them. A stack of ElectronJS and ReactJS are used on the desktop client to permit it to be available on MacOS, Windows and Linux, with little to no changes to our code, making it simple to maintain and solves the problem of scalability of the project. Likewise, for the mobile client, we will be using React-Native, which allows our application to be available to both android, iOS, and desktop devices. With so many clients, how is it that we are maintaining the main component of the application intact and consistent through all operating systems and devices?

The way we are maintaining this consistency is by only having this component stored in one place, in which any kind of client can access, therefore, we decided it would be best to keep the computer vision and processing all separate from the client application and have a clear API in which we simply send data to the AWS machine, the machine processes the data and simply returns a data structure with data we can use to play sounds and create UI updates that act on the data returned. The data will be consistent no matter what device it gets sent to, making it easy to make multiple versions of the same app, without risking functionality and avoiding data inconsistencies when processing data on different systems.

2.5 Process Details

When the user opens the application they will be greeted with the option to upload an image of sheet music or take a picture themselves. Once the image has been selected/taken, the user will then be able to upload the images. Once uploaded a POST request will be sent to the AWS machine containing the machine. The AWS machine will accept the POST which will contain the image. Upon receiving the image the main Python program will be ran. Inside of the main program the image will be groomed for the NoteFinder object, this grooming includes removing image corrections, such as removing noise or rotating the image and converting the video to binary. When it receives the black and white image the NoteFinder will find all of the regions of interest and send them to different methods to have the notes extracted from them and put into arrays. Then these arrays are combined in the NoteMapper, this output is sent to the MidiConverter. Once the MIDI converter has finished the MIDI file is returned to the front end. From here the MIDI file can be played while animating the notes which are being played on an animated piano. The user can also play the sheet music themselves while the application listens to their playing and alerts them when they make mistakes.

3 Testing and Implementation

3.1 Hardware/Software

AWS Machine:

- Software:
- OpenCV
- Python
- NumPY
- Apache

Client (Desktop)

- Hardware:
- High-Precision Microphone
- High-Resolution Camera/Document Scanner
- Software:
- ReactJS
- ElectronJS
- HTML, JS, CSS]

Client (Mobile)

- Hardware:
  - Mobile Camera
  - Mobile Microphone
- Software:
  - React Native
  - JS

3.2 Graphics

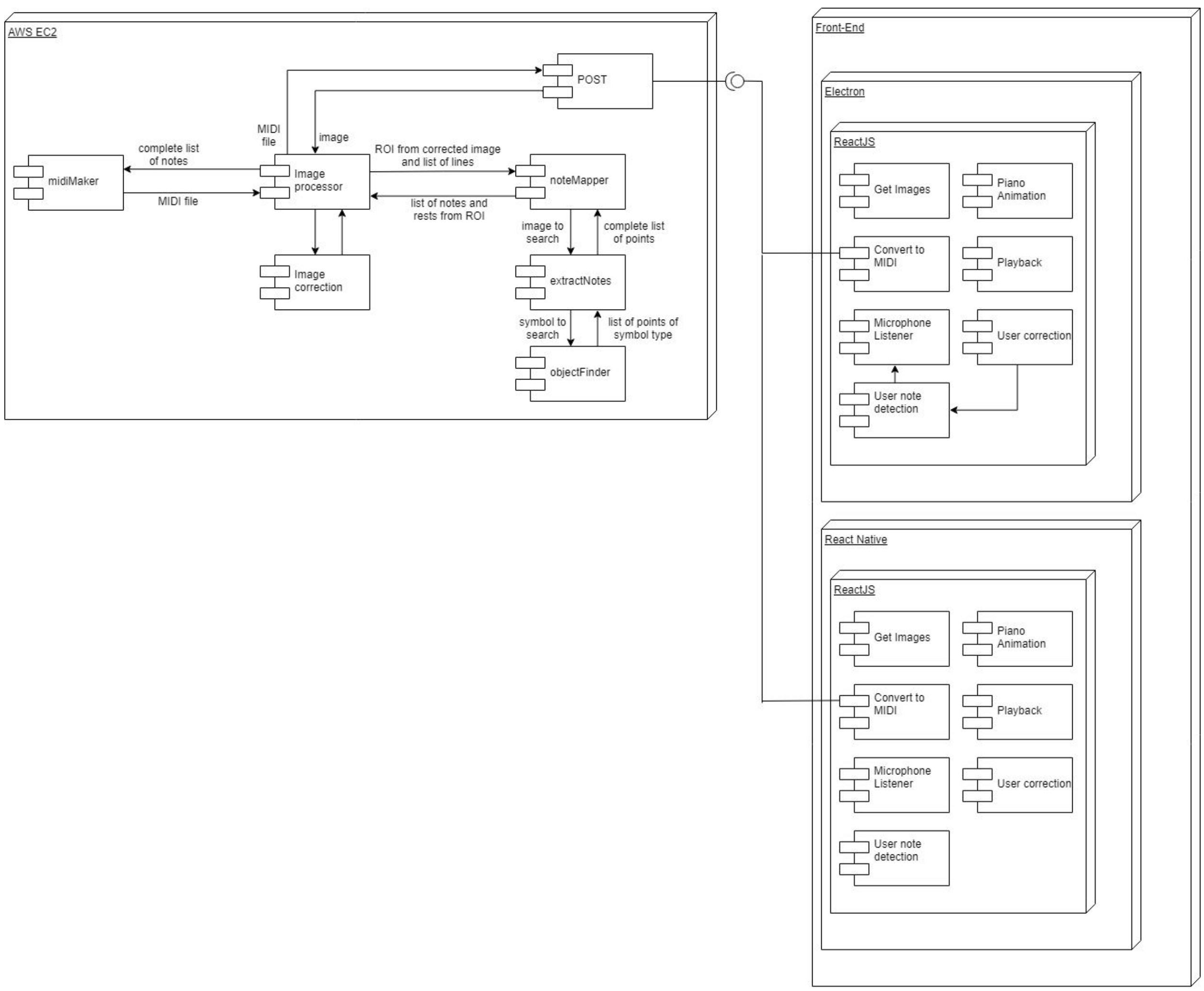


Figure 3.2

The above figure shows the different sections of the the project and the inner modules within it. The modules detailed in the graphic are the Desktop Client, Mobile Client and Backend AWS machine. The desktop client on the top right consists of a stack of ReactJS on top of ElectronJS, as well as the individual components that are part of the application. The Mobile application shown in the bottom right, consists of react-native as well as the individual components needed in the application. Finally the AWS machine detailed in the top left is the final component needed to make up our platform. Inside this section is the architecture we will be using for our image processing algorithm.

3.3 Functional Testing

The following list includes testing for functional requirements

I. The application shall be able to access the devices camera
II. The application shall be able to access the devices saved images
III. The application shall accurately display piano animations which correctly for a given MIDI file
IV. The application shall be able to accurately find and play simple music such as *Mary Had a Little Lamb*.
V. The application shall be able to listen to sound and notify the user if the notes are incorrect.

3.4 Non-functional Testing

The following list includes testing for non-functional requirements

I. Performance: Test that the machine vision algorithm should be able to analyze the music sheet within several seconds of the initial query.
    A. Because of the finite datasets you can stress test the runtime of requests on different datasets manually.
II. Scalability: At least 50 devices should be able to make post requests simultaneously without affecting performance.
    A. Because AWS will be handling the communication, we can use their available resources to test scalability.
III. Extensibility: Test that the algorithm should be able to run on AWS/APP.
    A. This can be completed through just running the application.

3.5 Modeling and Simulation

The model for our application has three major components (shown in figure 3.2). The computer vision component which is hosted on AWS, the desktop client side application which is ReactJS wrapped in Electron, and ReactJS wrapped in React Native for the client side mobile application. Both of the client side components will use ReactJS modules for the different requirements for our client side app. These client side components will access the computer vision module via an HTTP post request, sending an image of the sheet music. The sheet music will then be processed and a MIDI file will be returned to the client side application where the MIDI file will be played and parsed for visualization.

Each of the client side components will have the ability to get images either from the camera, or the memory of their respective platform. They will also have modified algorithms for displaying piano animations and for detecting notes being played by users for note feedback. The application will be able to read the notes from and play *Mary Had a Little Lamb* by using either the mobile or desktop client side application to take, or retrieve an image of the sheet music for the song, and send it to the AWS server to be processed. The server will process the image and return a MIDI file which the respective client side application will play.

Using AWS provides several desirable properties for our computer vision component. Firstly the AWS machine provides excellent computational power which will allow our computer vision algorithms to run quickly. Secondly using a remote server for our computer vision application allows any front end application to access the endpoint, this allows the application to be easily extended to other platforms without having to rework the computer vision code, while also providing the processing power to handle numerous simultaneous requests.

Our application after creation is very easily mass produced to the public for the final step of revising our functionality through public testing. Because our product can be easily sent to the public we can have people who are excited about our product test the functionality and simulate any edge cases we may run into while creating new use cases for the product.

3.6 Standards

1857.5-2015 - IEEE Standard for Advanced Mobile Speech and Audio

- This standard provides tools for audio processing as well as providing encoding information for both high and low bit rate audio.

P24748-3 - ISO/IEC/IEEE Draft International Standard - Systems and Software Engineering-Life Cycle Management-Part 3: Guidelines for the Application of ISO/IEC/IEEE 12207 (Software Life Cycle Processes)

- This is a standard for how the basic flow of a project should go. It covers many of the technical aspects of creating a project and is useful for any project in our field of study

3.7 Implementation Issues and Challenges

The biggest challenge of the project will be the implementation of the computer vision to accurately read sheet music. There are many problems which can cause errors in the note recognition and detection. These problems range from issues with image quality to the specific stylings of the symbols on the page. In order to deal with these problems we are taking a series of steps. The first step is to use a gaussian blur filter to remove image noise, then realign the image by detecting lines in staves of music and using these to rotate the image. After doing this we use binary thresholding to fix lighting issues. To accommodate for different size images we scale the images up and down when looking for symbols. In order to deal with different stylings of symbols we attempt to gather a large variety of symbols of different styles to use as templates in our search.

3.8 Design Testing / Implementation

In order to test the computer vision algorithm we break it down into two portions. First is the image preprocessing. To test the preprocessing we simply will give the algorithm different pictures of sheet music and compare the number of regions of interest found to the actual amount of regions of interest present in the music. This one test will work for testing the image correction since image correction is a mostly qualitative process, but we will only get the right amount or regions of interest if the preprocessing works. The second portion of the computer vision we can test is the symbol extraction, in order to test this we will give the symbol extraction algorithm different images of sheet music and count the number of symbols extracted correctly versus the number found and the number actually present in the picture.

To test the integration of our sheet-reading algorithm, we have selected 5 different pieces of sheet music which must be translated correctly to a well-formatted MIDI file to determine success. We will take pictures of the sheet music using our test phones, then supply them to the algorithm either through the application or directly. So long as the algorithm produces a correct MIDI file for the input of these pictures of sheet music, we will not need to do integration testing with the application to determine the correctness of the algorithm.

To test that our mobile application functions correctly, we will run through the process of receiving a picture by all accessible means and testing the communication to and from the backend with the MIDI files. The piano animation must also be tested, and for that we will create a MIDI file which contains a large range of different notes, time signatures, and so on to attempt to catch any errors with our piano. Finally, we will also test as many use cases for our app as possible, and ask acquaintance test subjects to attempt to use the app to try to find any errors.

4 Closing Material

4.1 Conclusion

Throughout these two semesters we will work on creating a application that can scan through a sheet of music and create a simple animation of which keys on a piano you should press that correlate to the beats that will be playing. With this you can change features and after hearing what it should sound like and seeing what you should play; you can turn the application into practice mode and it can tell you if you are playing the correct notes.

4.2 References

"OpenCV." *OpenCV*, 20 Nov. 2018, opencv.org/opencv-4-0/.

"React – A JavaScript Library for Building User Interfaces." *– A JavaScript Library for Building User Interfaces*, reactjs.org/.

"The Official MIDI Specifications." *Specs*, www.midi.org/specifications.

"Use React Native." *Use React Native*, Use React Native, 25 Mar. 2019, www.reactnative.com/.

"Getting Started with Amazon EC2." *Amazon*, Amazon, aws.amazon.com/ec2/getting-started/.