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: December 10/Final Version

Table of Contents

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.
4. Amazon S3 Bucket - Storage location for our Amazon Web Services.

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

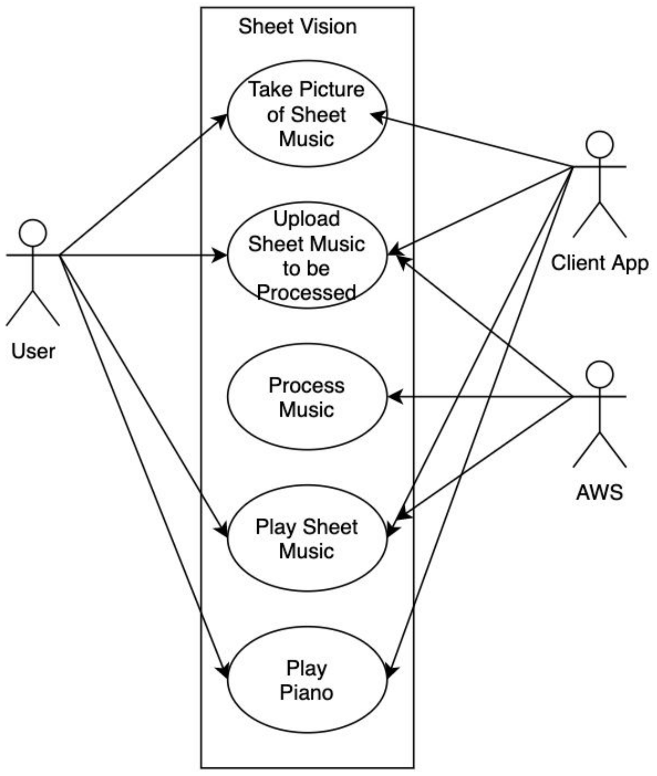
Reading sheet music is no easy task. With the creation of alternate ways to learn how to play music, such as tabs and youtube tutorials, there has been a decline in the amount of people who can properly read sheet music. The problem with tabs and other kinds of methods of reading music is that they lack 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.

1.3 Operating Environment

Our product is expected to be used with bright and uniform lighting. Lighting is 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 user's 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.
    - Can run on a mobile multi platform environment.
        - Android.
        - iOS.

2 Specification and Analysis

2.1 Project goals, Deliverables

- React Native/Expo application for mobile.
- 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.

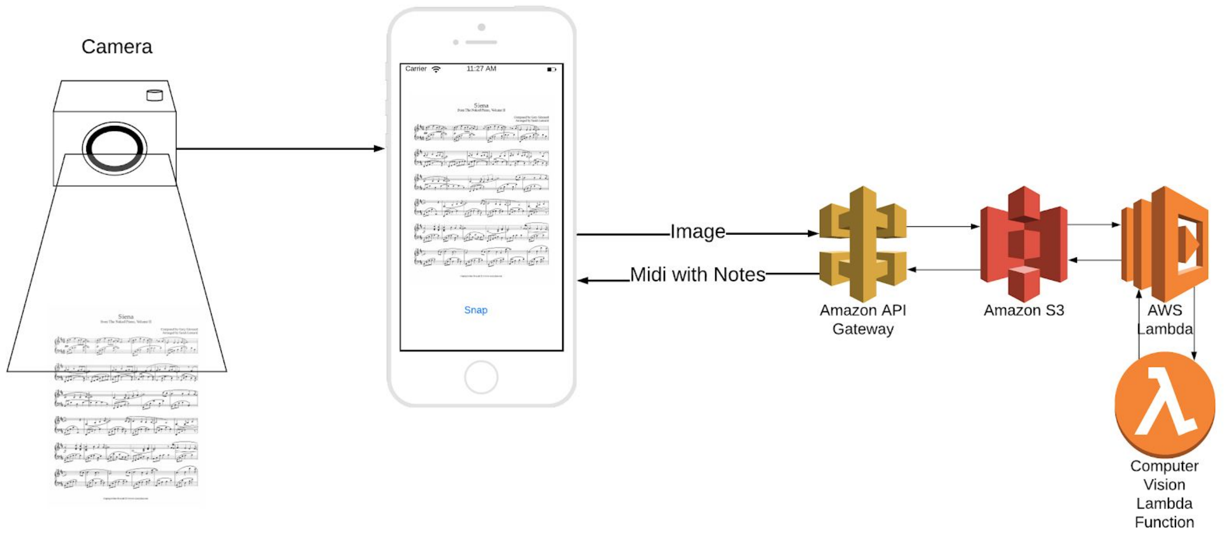
2.2 Design Specifications

The mobile client will be a standard mobile application. The application will be able to take images from the users camera and take pictures from their gallery and upload them to our server. Then the front end application will be able to play the JSON files that are received from AWS, and then as an additional feature.

2.3 Proposed Design/Method

The design calls for the front end code to be written in 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

Architecture:

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 a multi-platform mobile application. The mobile client will be written using React-Native, which allows our application to be available to both Android, iOS. With a multi-platform client, 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 the Computer Vision 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 stack to be processed and get back 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. A PUT request will be sent to our AWS Gateway, which will send the image to an Amazon S3 Bucket. The AWS machine will accept the POST which will contain the image. Upon receiving the image Amazon S3 will run Lambda trigger, which processes our image and posts the JSON result back into our S3 Bucket, ready to be returned to our client.

Inside the main image processing, the image goes through multiple stages to turn the sheet music into a JSON list of notes and timings, similar to a MIDI file. The JSON string is then sent from AWS to the client application. From here the JSON string can be read by the application, which will play the music while animating the notes which are being played on an animated piano.

3 Testing and Implementation

3.1 Hardware/Software

AWS Machine:

- Software:
    - OpenCV
    - Python
    - NumPY

Client (Mobile)

- Hardware:
    - Mobile Camera
- Software:

- React Native/Expo
- JavaScript

3.2 Graphics

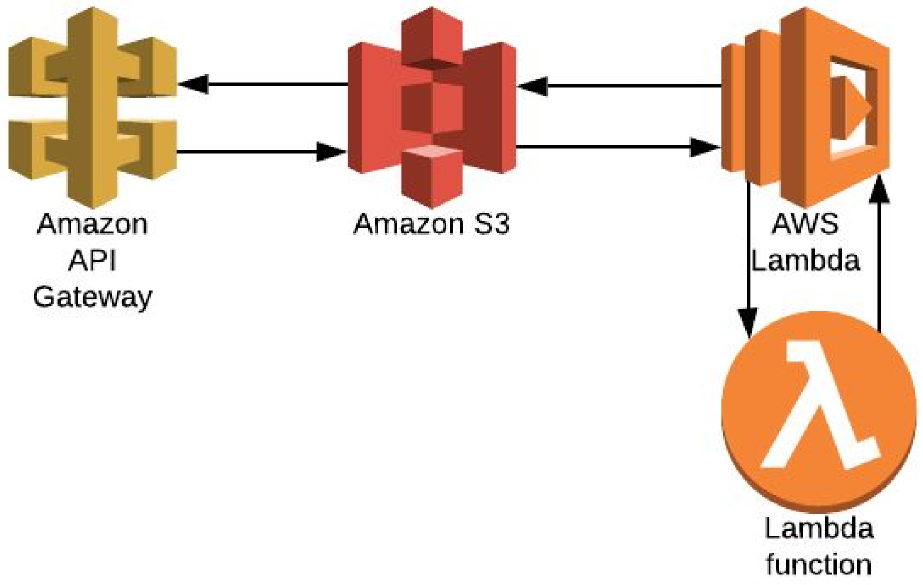


Figure 3.2

Figure 3.2 shows the different sections AWS backend stack and the inner modules within it. The modules detailed in the graphic are the AWS API Gateway, S3 and Lambda. The AWS API Gateway receives the request to upload the image to the S3 bucket, then sends it to S3. S3 then saves the image and calls the AWS Lambda trigger function which processes our image with our Music Note Reader Algorithm. The resulting JSON is then saved to the S3 box and is ready for the client to get it back from the server.

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 JSON
IV. The application shall be able to accurately find and play simple music such as *Mary Had a Little Lamb*.

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. 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.

At the beginning of this project, we started up an ubuntu box and installed Apache and OpenSSL on it to host our python code. This design was revisited on multiple occasions but later removed due incompatibilities with React Native after updates that happened to React Native over the Summer 2019, which makes React-Native deny all requests that aren't received from servers with CA's that are on the phone's trusted CA's list, in which we would have to manually approve on every single phone that uses our app, which will not be possible.

After scrapping EC2, we tried moving to API Gateway and AWS Lambda to host our serverless compute, this would let us host our OpenCV code without having to worry about a server, and since Amazon has a certificate that is authorized on most devices by default already, this removes this hurdle for us as well. The main problem that we had with AWS Lambda is that it would limit the sizes of requests we would be able to send to it, to 10 MB's. The images that we were sending were constantly going over this threshold. We could compress these images before sending them, but for our Computer Vision algorithms to work properly, we need to give it the best images we possibly can, therefore, compression was not an option. To remediate this, we went with our current implementation of our serverless compute, which is API Gateway + AWS S3 + AWS Lambda.

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 JSON file to determine success. We will take pictures of the sheet music using our phones, then supply them to the algorithm either through the application or directly. So long as the algorithm produces a correct JSON 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 JSON files. The piano animation must also be tested, and for that we will create a JSON 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 an 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.

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/.