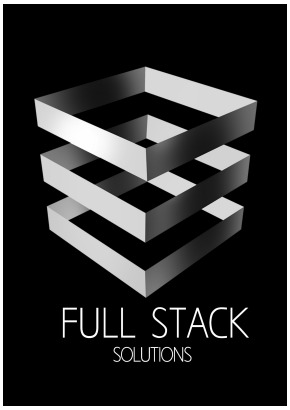
System Requirements Specifications

PidgeySync

Full Stack Solutions



Team members:

David Behr, Ammarah Cassim, Richard Lubbe, Tari Mautsa, Khondwani Sikasote

Contents

1 System Overview

1.1 Purpose

A lot of cartoon animators have trouble syncing their character speech with their character's mouth movements, and as a result a lot of amateur animators lose motivation and get discouraged.

Our System, PidgeySync is a solution to this challenge as it's an simple open source program which automatically syncs a character dialogue with the appropriate mouth shapes to make the animation look more realistic.

1.2 Project Scope

PidgeySync has the following main functions that builds up most of the project:

- Generating text from audio files.
  This functionality gets all spoken words in the audio file and translates it into text, which is therefor used in the following functionality.
- Generating phonemes.
  From the text produced, Phonemes are generated
- Exporting the generated .dat file
  This is achieved by downloading it to your download directory. The .dat file has all the timings related to the audio and phonemes related to the specific sounds of words which will be used for mapping mouth movements in an external animation software.

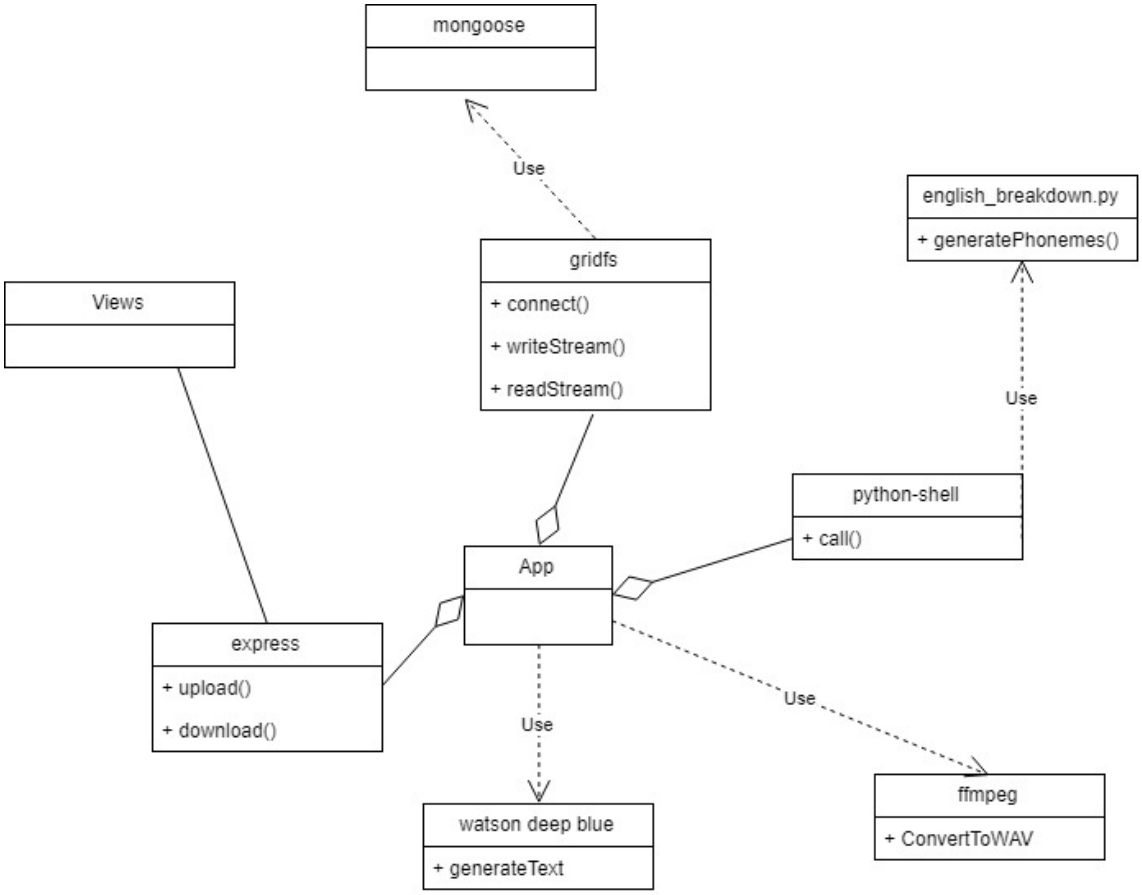
1.3 PidgeySync's Wow Factors

- Generating of sign language from text

1.4 Definitions, acronyms and abbreviations

- WAV - Waveform Audio File, a file format for audio files created by Microsoft and IBM.
- DAT - Digital Audiotape that is Moho compatible.
- DOM - Document Object Model
- AJAX - Asynchronous javascript
- UML- Unified Modeling Language
- Admin- Administrator user
- UI-User Interface
- API-Application Programming Interface
- PC-Personal Computer
- Server - a computer or computer program which manages access to a centralized resource or service in a network.
- Python Shell - Is an interpreter that can be used for the programming language like Python
- Phoneme - any of the abstract units of the phonetic system of a language that correspond to a set of similar speech sounds which are perceived to be a single distinctive sound in the language.

1.5 UML Domain model



2 Functional Requirements

2.1 Users

The types of users for this system include:

- The Administrator user
  The admin user has the privileges of maintaining the system i.e. server, database and User Interface. The admin also provides updates and support for any queries for regular user.

- Regular User
  The user for which the system is designed for i.e. animators/cartoon makers. Users for whom the system was designed for include animators, musicians, people who are computer literate, people with basic experience with animation and it's software. This type of user has the following privileges ;Interacting with the database by uploading an audio file, generating text from the audio file uploaded, editing the text produced. The user is able to download the generated .dat file from the database This user has the ability to correct any wrong action i.e.Uploading the wrong file can be reversed by re-uploading an audio file or by restarting the entire process.

- Non Regular Users This includes deaf musicians who would like to have sing language generated from an audio file. Deaf students who would like to have audio translated to sign language. Regular musicians who would like to have subtitles of their favorite songs generated. People who are into Karaoke

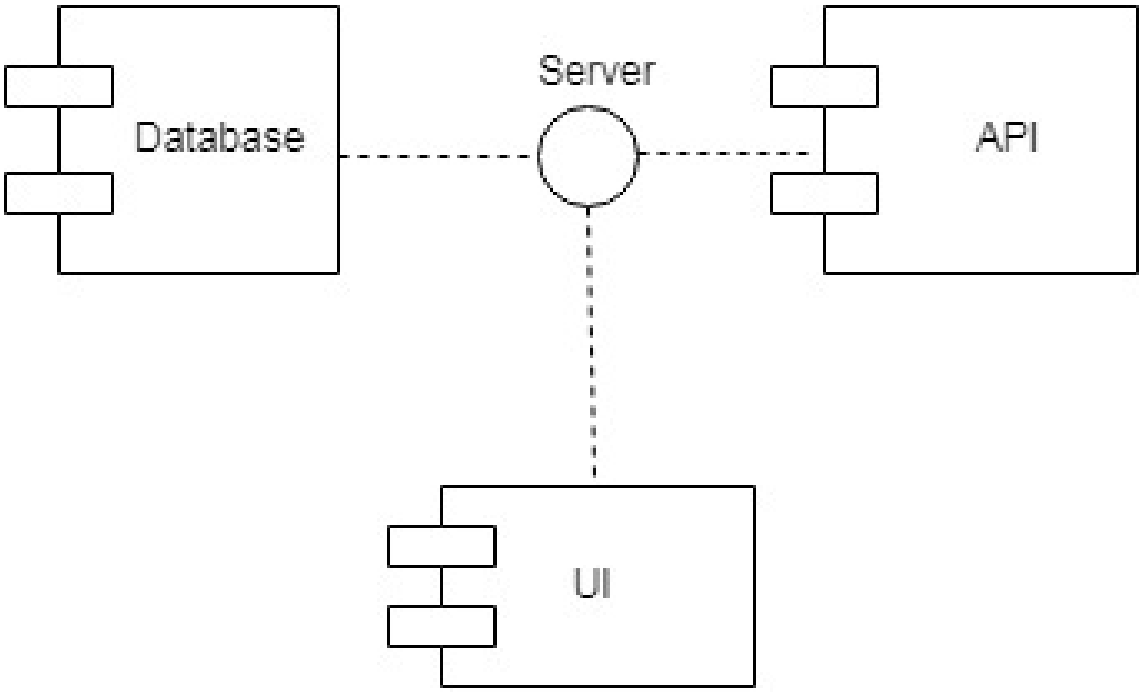
2.2 Subsystems

- Server(Nodejs) and Database (MongoDB)
  Our server acts as the middle-ware between the client and the database by sending through the uploaded audio file from the client for storage in the database and it retrieves a copy of the generated .dat file from the database to the client.Database is loosely coupled to the server because the database doesn't need to know much about the server and vice versa.

- Server(Nodejs) and Speech to Text API (Deep Blue)
  The Server also interacts with the Speech to Text API (Deep Blue), for the generation of text from the audio, as well as the generation of phonemes from the text to mapping the phonemes to each word in the text produced by Deep Blue. Deep Blue is highly coupled to the Server as the Server depends on Deep Blue to provide the text and phonemes used by the Server to map the phonemes to the text.

- UI(Client) and Server (Nodejs)
  Our Front end/UI is what the user interacts with. This is where the user uploads an audio file which is used in the generating of text. The Server returns the generated .dat file to the UI. The UI is loosely coupled to the server as the user/UI doesn't need to know much about the server. As for the Server it is highly coupled to the User Interface for it needs to have much knowledge on what the User interface needs or Uploads.

UML Component Diagram



2.3 Specific Requirements

- Server (Nodejs) and Speech to Text API (Deep Blue)

    1. The server will be able to convert the speech in the audio file to text by use of Deep Blue(a speech to text API).
    2. The server will be able to make use of Deep Blue to map phonemes to the text generated from the audio file.

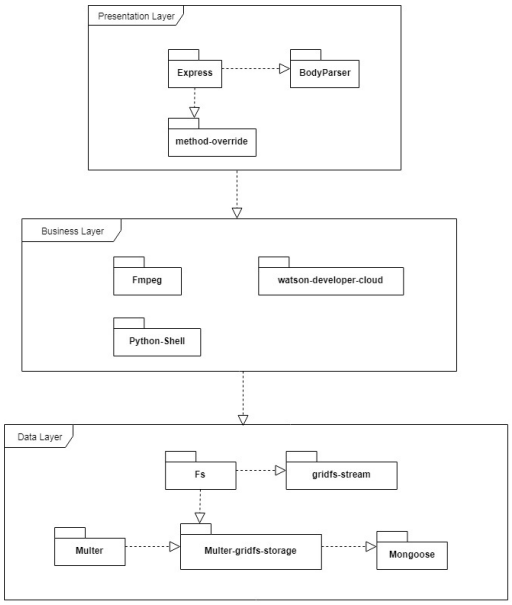
- Database and Server (Nodejs)

    1. The database shall store imported audio file from the server.
    2. The database shall allow the users to retrieve the generated .dat file.
    3. The database shall store the text generated from the Deep Blue API.
    4. The database shall store the phonemes generated.
    5. The server shall be able retrieve the uploaded audio file into the database.

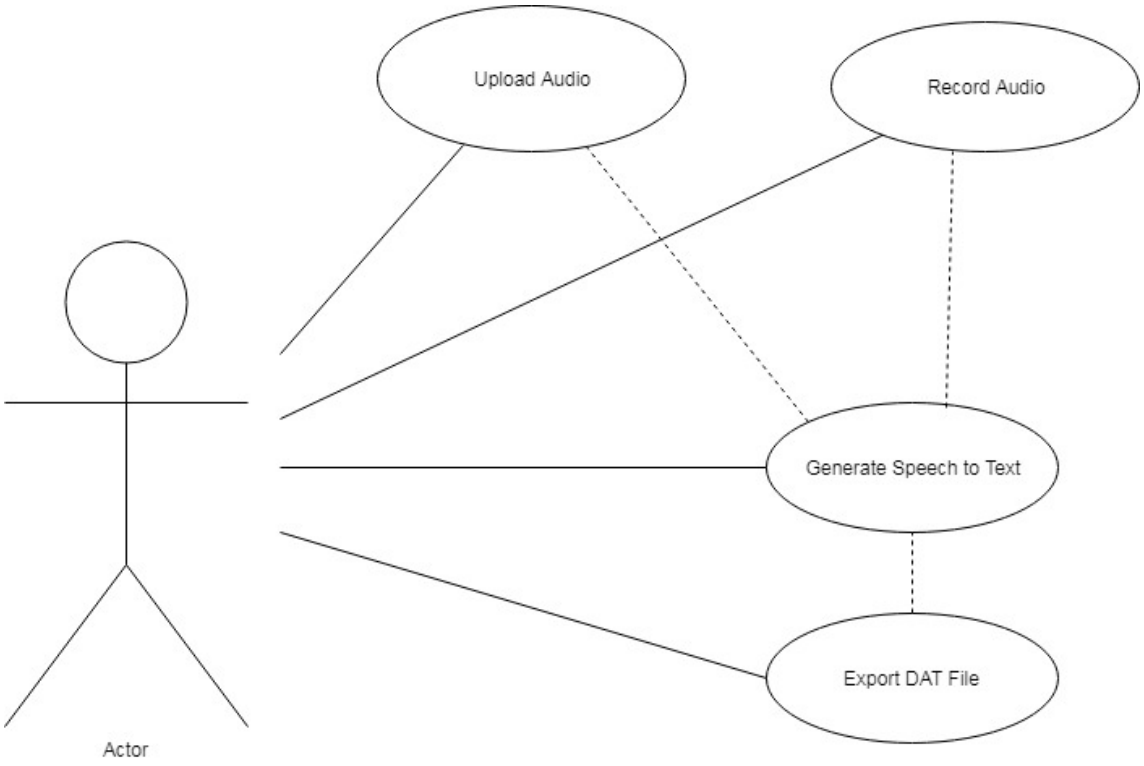
- UI(Client) and Server (Nodejs)

    1. Users shall be able to import audio WAV files.
    2. Users shall be able to edit and save the text generated if there will be any errors
    3. Users shall be able to test the output produced by playing the audio recording and observing the mouth piece generate lip movements.
    4. Users shall be able to export the produced DAT file once conversion is complete and users are satisfied.
    5. Users shall be able to upload another WAV audio file if they upload an incorrect file.
    6. Users shall be able to play, pause and stop the audio file when testing the file in the application.
    7. Users shall be able to move time stamps placed on the waveform indicating the start ad end.
    8. Users shall be able to use the exported DAT file as a plug-in into other animating softwares.
    9. The server will retrieve the audio file from the User interface.
    10. The server will allow for concurrent users of the system.

Package Diagram

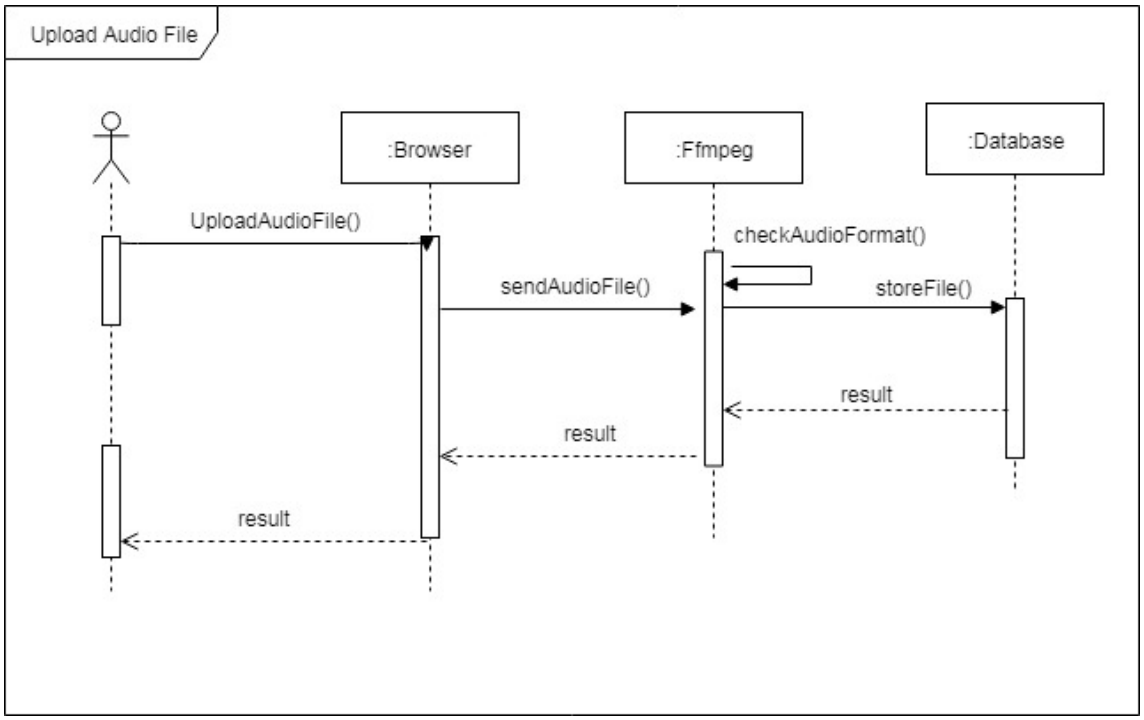


Use Case Diagram



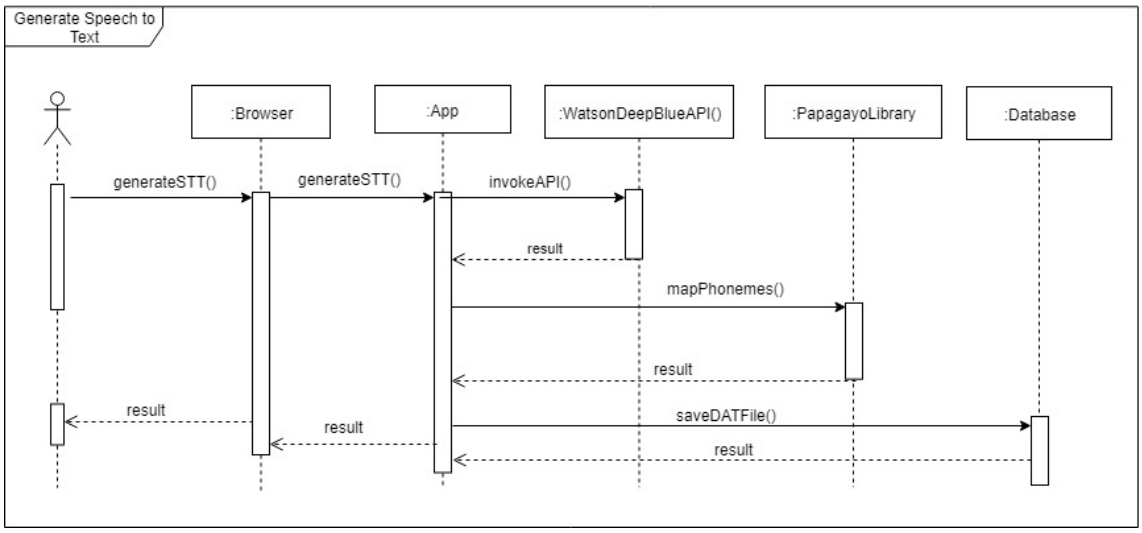
2.4 Sequence Diagrams for PidgeySync

Upload Audio



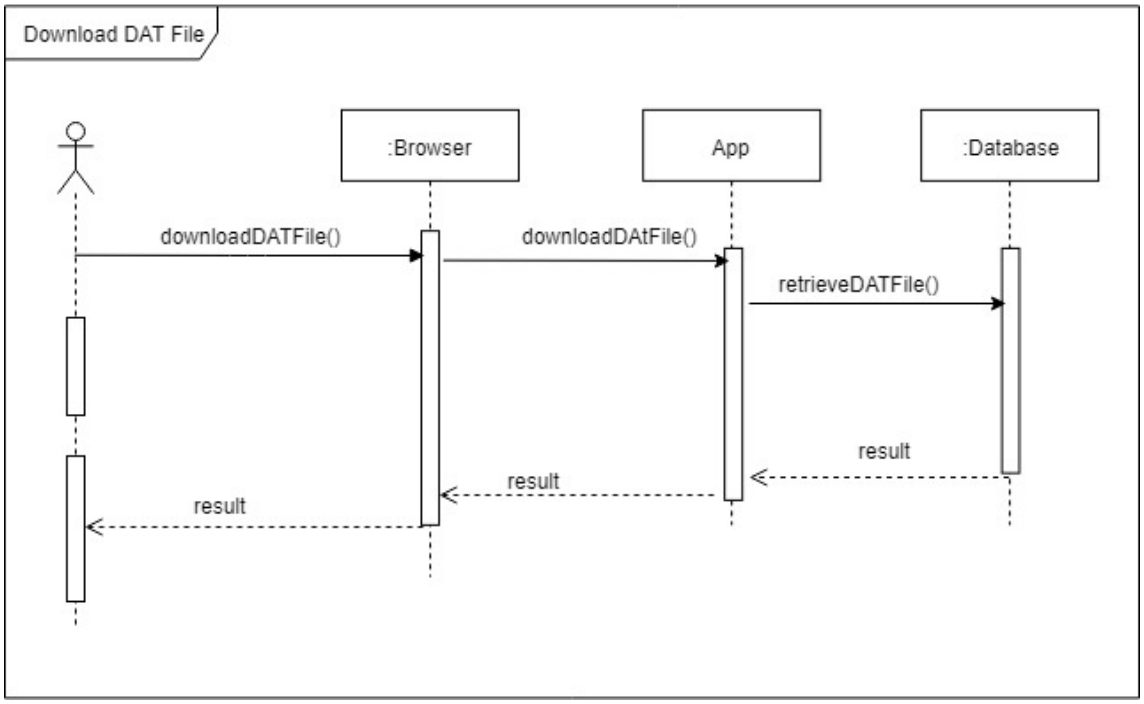
.

Generate Speech To Text



.

Download DAT File



2.5 Non-Functional Requirements

- Effectiveness
  We achieve effectiveness by relying on the speed of the Server. Why this is important is because we want the system to be as effective as possible in terms of returning results to the user and not wasting the users time waiting for results. How we tested for effectiveness was by uploading different audio file sizes and lengths and see how quick the server responded to each.

- Efficiency of use. This is important in the sense of getting an efficient outcome from the speech to text done by the API (Deep blue). How this was measured was by the use of different voices to measure how efficient the API is. We achieve efficiency by creating a simple user friendly and interactive interface. We also provided the user with a tutorial video to observe to explain how to use the system

- Maintainability
  We achieve maintainability by using technologies that are part of a large community where people will be able to contribute to maintaining and improving the system.

- Reliability We achieve reliability of our system when it crashes by informing the user with a error message. Detailed steps on how to recover from the error are in the user manual

- Availability We achieve Availability of our system to users who would have a Internet connection where ever they will be and we rely on our server provider's maintenance and backup of the server.

- Integrity We achieve Integrity of the results produced by running many tests to feed input/data to the speech to text API as its a learning agent/machine so it becomes better and efficient with the output data it produces. This has was proved with time.

- Scalability We achieve scalability by allowing for concurrent users to use the system at the same time causing no conflicts with people's data. How we measured this non functional requirement was by having multiple users using our website at the same time.

- Usability We achieve usability by allowing our system to be tested by many users so we get feedback from them on how we can improve the system and make it more usable and user friendly. How we measured usability was by getting feedback forms from the users that used our website.

3 System Architecture

3.1 User Interfaces

1. Upload Interface
   This is where the user uploads the audio file into the system

2. Speech to Text interface
   This is where the text is generated and from the generated text, the phonemes are formed and mapped to the mouth shapes movements which will be in sync with the audio file

3. Export Interface This is where the user will be able to export/download the .dat file produced

3.2 Hardware Interfaces

- We make use of the hypertext transfer protocol secure (HTTPS) protocol. For secure communication between the server and the user interface.

- We make use of the users computer inbuilt microphones. Thats in the case of the user wanting to record an audio file while using the system.

3.3 Software Interfaces

- Gridfstream is the interface we're using to together with mongoose to write data to the database.

- The python shell is a child process that the server(Nodejs) spawns, it sends its data in the form of JSON and gets a return from python script.

3.4 Technologies

- Node is a javascript library which is used to build servers, because it is written in javascript it greatly simplifies the interaction between the client and the server since both of them use the same programming language. We use Node to listen for user input, to process data and then to send a response to the client. The Node server also makes calls to API servers in addition to listening for user input, and processes data that is received from those servers. Node is inherently a concurrent framework which means that it supports sessions right out of the box.

- Express is a Node library that simplifies and automates many processes that are used repeatedly in a Server. Express provides middle ware which essentially processes data before a request is handled. in the case of handling form data Express has middle-ware called multer that greatly simplifies the task of reading form data from the client. Express is used alongside node on the Server.

- Python is used when a child process is spawned by node to send data to be written to the dat file and subtitle file.

- Javascript is a scripting language that is native to most web browsers, it is used to interact with the DOM to dynamically change content as the user interacts with the system. Javascript is the underlying language of the nodejs library.
  we use Javascript for validation checks, AJAX calls so that there is no page refresh and to retrieve user information to be sent to the server.

- Jquery is a javascript library that greatly simplifies working with the DOM and it is simplifies ajax requests.JQuery also has support for animations and transitions which we use to make our page more user friendly. // JQuery provides methods to load EJS templates dynamically via using the GET protocol

- HTML5 We use HTML5 together with Javascript to build our User interface

- MongoDB is a non-relational database which stores data objects in documents in the JSON format. We use MongoDB for storage for the uploaded audio files, storing of the generated dat file and the subtitle file. We Choose mongo because of its support for large file uploads and that each mongo document is a javascript object that can be processed easily.

3.5 Product Functions

The application shall allow for user(s) to upload/import an audio files into the database and a text file is then generated from the audio. Phonemes are then mapped to the text in the text file and the end product will be a DAT file that can be exported, and plugged into animation softwares. The text file can also be used for the sign language translation from text to sign language Our application will be utilized and accessed on Computers

3.6 Constraints

- Users shall only upload audio files.
- Uploaded audios files shall not be larger than 5MB.
- Uploaded audio files should not be longer than 30 seconds.
- Users shall not be able to upload images or any other format apart from audio file.

3.7 Assumptions and Dependencies

- Client will be dependent on the server to perform the conversion from the audio file to text file and finally to the DAT file.
- The server is dependent on the client to receive the audio file.
- We expect the user to have an Internet connection at all times
- We expect the users to use this system on a desktop computer

3.8 Architectural Styles

- Server(Nodejs) and Database (MongoDB)
  Our server acts as the middle-ware between the client and the database by sending through the uploaded audio file from the client for storage in the database. It retrieves a copy of the generated .dat file from the database to the client. The architectural style utilized is N-tier style because it provides simplified access to data stored in a persistent storage like MongoDB and it also provides a centralized location for all calls into the database and thus makes it easier to port the application to other database systems

- Server(Nodejs) and Speech to Text API (Deep Blue)
  The Server also interacts with the Speech to Text API (Deep Blue), for the generation of text from the audio, as well as the generation of phonemes from the text to mapping the phonemes to each word in the text produced by Deep Blue. The architectural style utilized is the master slave architectural style also known as the main program and subroutines. We chose this architectural style because the Server acts as the master giving instructions and receiving feedback/results from the API. The Server sends the audio file to API for further processing.

- UI(Client) and Server(Nodejs)
  Our Front end is what the UI with. This is where the user uploads an audio file which is used in the generating of the speech to text for mapping of Phonemes to the text and audio. The Server retrieves the generated .dat file which is presented to the UI via an export option. The user can also play the uploaded audio file,thus the server gets the uploaded file from the database to allow the user to play it back. The architectural style utilized is the master-slave architectural style which is also known as the main program and subroutines. We chose this architectural style because the User Interface sends the audio file to the Server(giving it instructions) and the Server takes the audio file for further processing. The Server then sends back the results to the UI. We also implemented the Client and server style because its the UI that requests a service from the server and the server comes back with a response for the UI.

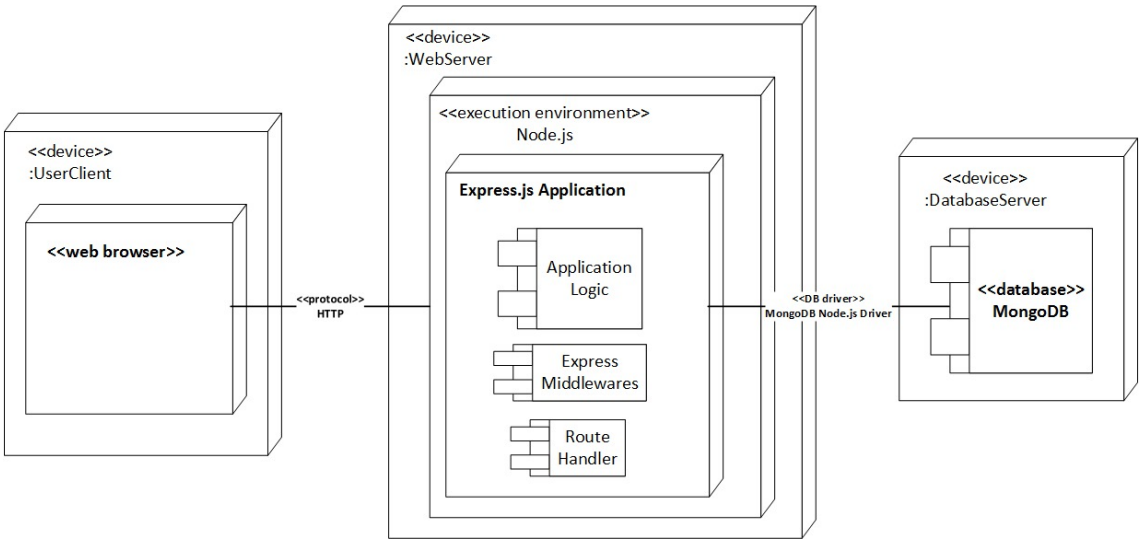
3.9 System Configuration

- Equipment
  Our System can be accessed with Desktop computers or Personal Computers(PC)

- Communications
  Our System utilities the Network card on a computer to communicate with the Internet. Our system communicates with the IMB Servers via Watson Deep Blue API for the speech to text Conversions

- Networks
  Our System is connected to the Internet and the IBM Network.

The type and configuration of computer input our system will take are audio files of a format ranging from .WAV files .MP3 files .MP4 files etc. Our System will also utilize a microphone, if user would prefer to record audio files as opposed to uploading audio files. The output device our system will require is a computer screen and computer speakers.



UML Deployment Diagram