Architecture Document

Client: Zeehondencentrum Pieterburen

Andrei Mihalachi *s3491862*
Annika Möller *s3585832*
Brian de Jager *s4104579*
Mike Lucas *s3519120*
Roan Rosema *s4109449*

Lecturer: Dr. Andrea Capiluppi
Teaching Assistant: C.M. Rosiu
March 30, 2021

Contents

1 Introduction

The vets who work with the rescued seals at the Zeehondencentrum Pieterburen use several methods to assess the health of the seals once they have been brought to the sanctuary. The goal of this application is to create a tool that will make use of artificial intelligence technology to aid them with this process. Application XYZ is a Windows application that uses a convolutional neural network (CNN) to analyze the health level of a seal based on the sounds it makes. The user can upload a sound file of a seal making a noise, which is analysed by the CNN. The CNN then outputs a score certain value indicating the seal's health level (elaborate once CNN details are known).

This information is output in a table format, with the rows indicating the name of each sound file and its corresponding health score. Each row will also have a space for other relevant annotations to be entered by the vets (e.g temperature, bodyweight, general behaviour, etc). This additional information consists out of columns, which are added by the user. The user will also have the option to add more sound files once the table has already been generated.

The data generated by the CNN should be portable, so the application will allow the user to download the generated table as a .csv file. Sessions are autosaved by the program, such that whenever the application is opened again, the previous session of data is opened. Furthermore, the application will also allow users to upload a previously generated .csv file, view it in the application window, and continue editing the table.

2 Stakeholders and their concerns

- Employee at Pieterburen
  - The employee wants a simple application to generate the likelihood of a seal surviving, provide an overview of this data and to add additional information of the seal.

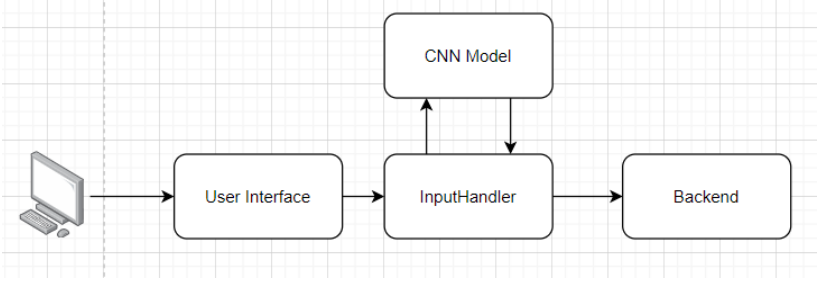
3 Architectural Overview

The application contains the following components:

- The frontend, a GUI which functions as the frontend of the application
- The Input handler, which is responsible to handle the input gotten from the GUI
- The CNN model, which is made by the TA that is working on it in parallel. We did not personally make any architectural decisions to this CNN model
- The backend, which stores the data of the current session

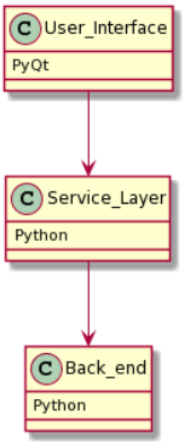
Conceptual model

To gain an understanding in how these different components interact with each other, we have designed a conceptual model. The user can interact with the system using the User Interface. They can either upload a .csv file to the InputHandler which directly updates the Backend or they can upload X number of .wav audio files to InputHandler. In case of a .wav file, the InputHandler parses each uploaded audio file to the correct format. For each audio file, the CNN model generates a result that is sent back to the InputHandler. Together with the file name of the uploaded audio file, the InputHandler updates the Backend accordingly. It is also possible for a user to input data values directly through the user interface. This way, the backend also gets updated with these values.



Module View

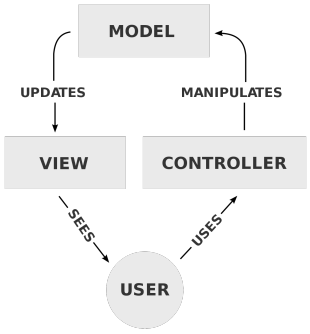
Additionally, to see which language/module is used in our application, we have designed a module view. The conceptual model consists of three layers: The user interface that takes care of user input, a service layer that contains the logic relating to the parsing of data and the back end.



As shown in the graph above, our application is based on Python. The service layer and the back end are fully made with python. The user interface is made with PyQt, which is a python module to create a GUI.

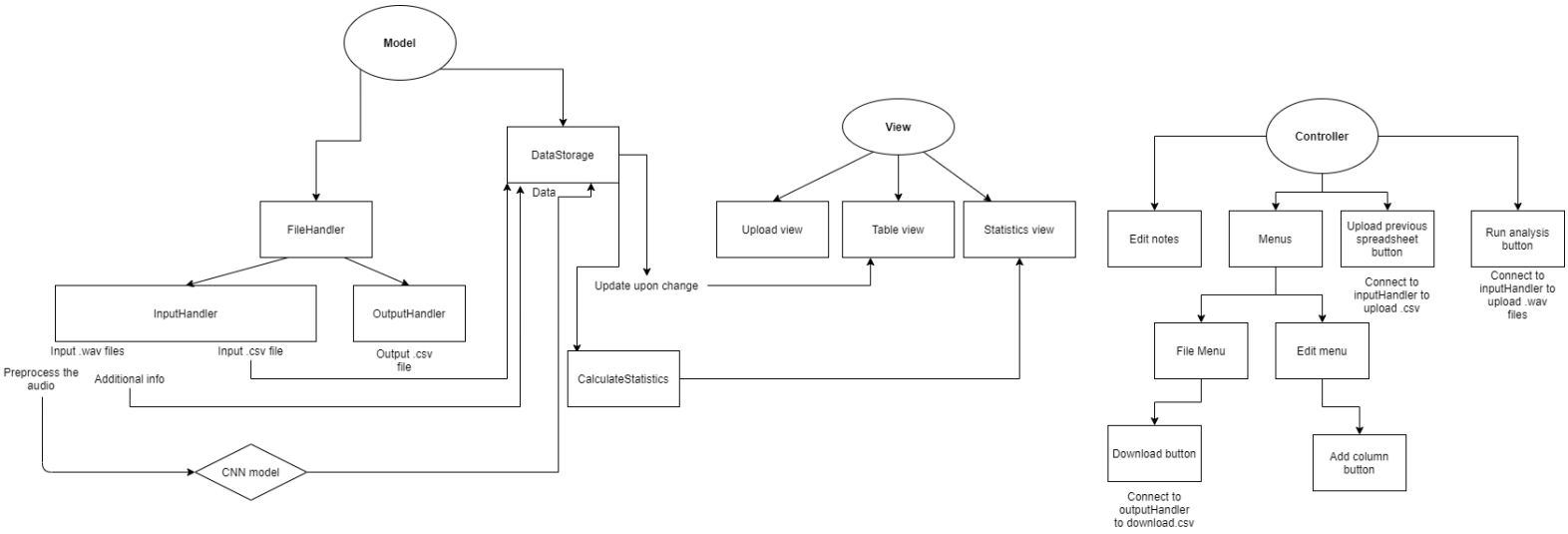
MVC Diagram

We have split the system into three components; the model, the view and the controller. MVC is a design pattern which splits the application into three interconnected parts. To easily explain this concept, we can use the following figure:



Below you can see a diagram we have made which displays the structure of

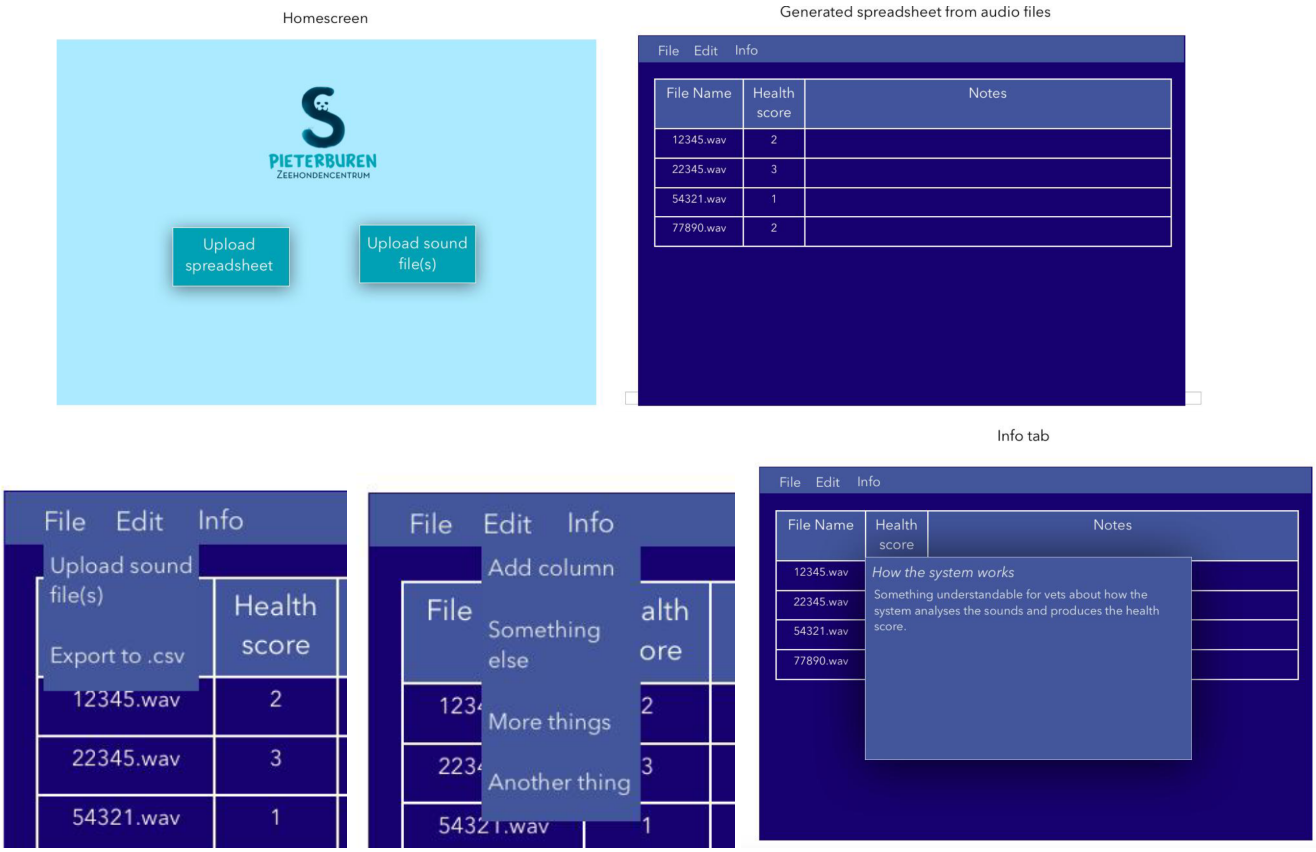
the code of the system.



3.1 Frontend

The frontend is a user friendly Windows application on which the user can upload audio files and automatically analyze these files. The frontend receives the uploaded audio files from the user and passes these files onto the backend where all the analyzing and calculation is done. The model then proceeds to use a calculation model, in our case the cnn model, on the audio files to extract certain types of information. These values are then passed given back to the view, which are put in its table. This table is the view's tool representing data to the user and allowing the user to alter data.

For the design of the graphical user interface, we first created a simple prototype using a graphical design software. The prototype design contains the following aspects: a main screen for the app, acting as a homepage, containing two buttons. The buttons are for either uploading a sound file, or uploading a spreadsheet. The other screen included in the prototype, is the screen displaying all of the information and data. This screen is called the table view. It contains a table which takes up most of the screen, and has some columns which should always be included (in the early stages of development, these are just the file name and the health score given by the cnn model). The other columns are optional and can be added. At all times, there is a menu bar present above the view. This menu bar provides menu's named: file, edit and info. The file menu would produce a submenu with options like exporting and saving. The edit menu covers options like adding columns. The info menu is supposed to offer an overview for the veterinarians on how to use the application, sort of like a digital manual. Below, you can find the prototype with images of the home screen, table view, file menu, edit menu and info menu respectively.

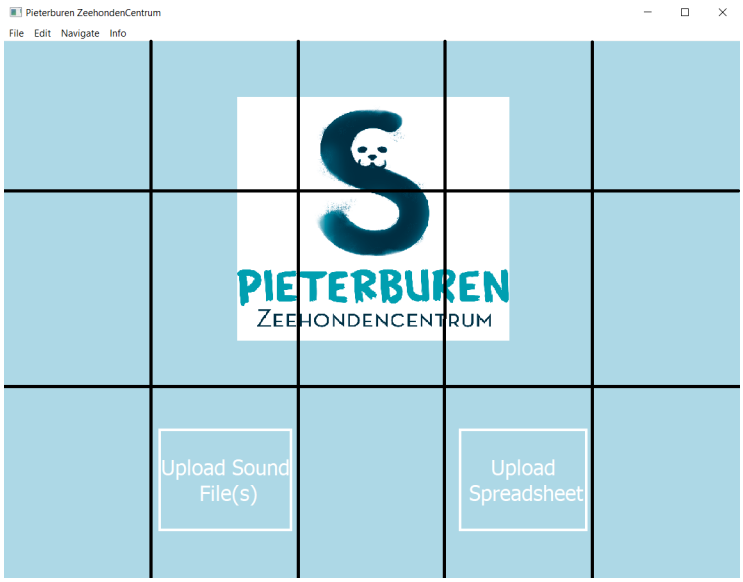


When contemplating the design of the user interface for this particular project, multiple frameworks have come to mind. Since most, if not all, of this project is coded in python, it seemed like the most obvious choice to choose a python framework. The first option considered is a package that is standard included with python. This framework is called Tkinter. The Tkinter framework seems to be a very basic and simple framework to build graphical user interfaces with. It requires very few lines of code to set up very basic apps. However, that particular feature also seems to be its downside and weakness. Whereas other frameworks have a vast selection of 'widgets' (the elements of a graphical interface, like a scroll bar or a button), Tkinter only has very little choice. We started building the prototype with Tkinter until we encountered a roadblock. Tkinter does not support something resembling a spreadsheet or a table, as is desired in the application. To work around this, we had two options: use an unofficial package or attempt to

create this component ourselves with the means provided by Tkinter. After being disappointed by the available alternatives for the first options and finding out that building such a component is rather complex, we opted for a different framework.

After some thorough research, we found a new framework which offers everything we needed for the project. This framework is called PyQt5, and has a lot of documentation available. The framework also comes with a table widget and spreadsheet widget, which is exactly what is needed. Since the framework offers plenty documentation and a lot of representation online, it is a suitable option for this project.

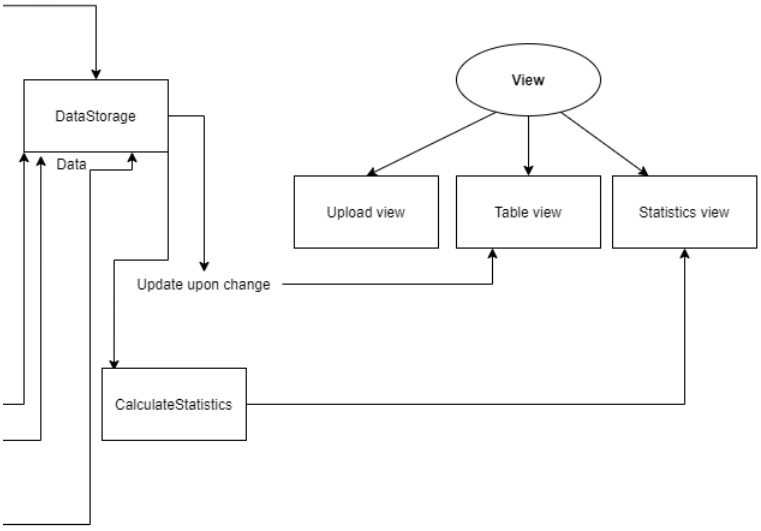
Using the PyQt5 framework, we have created a graphical user interface the following way. One class is designed to act as the 'main' window of the application, and has frames stacked on top of it. These frames represent the various views which the app should provide. Users are able to traverse the interface by clicking buttons, which in turn call functions that put the desired frame/view on top of the stack. The views are designed using a grid layout manager. This allows us to divide the views into rows and columns and place items at desired locations. An example of this is given below.



Using this technique, components can be placed with sufficient precision. In this example, by splitting the frame up into three rows and five columns,

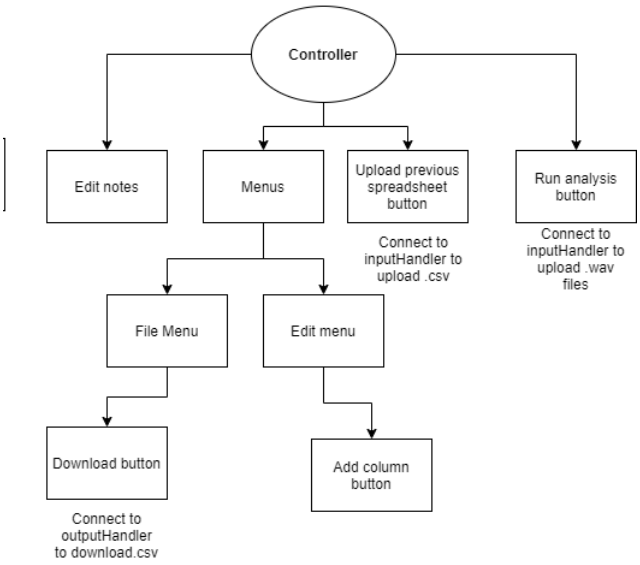
we can allow the logo (a placeholder image in this example) to span three columns and two rows, and place it in the middle. We then allow the buttons to span a single row and column, and place these to the left and right of the logo without touching the borders of the frame.
The frontend relates to the view- and the controller part of the MVC model. The following to paragraphs talk about the position of these parts in the overall application.

View



The view part is the front end, which contains different views that are displayed to the user. The user can change the view by pressing the displayed buttons. There are numerous kinds of views: An upload view, a table view and a statistics view. All views show some kind of data. The table view is directly linked to the DataStorage from the model. Once the DataStorage is updated, the table view will know that it has to update. All the data within the DataStorage can also be viewed in various different statistical ways. This statistics view is made in the second block, and will be further explained next iteration.

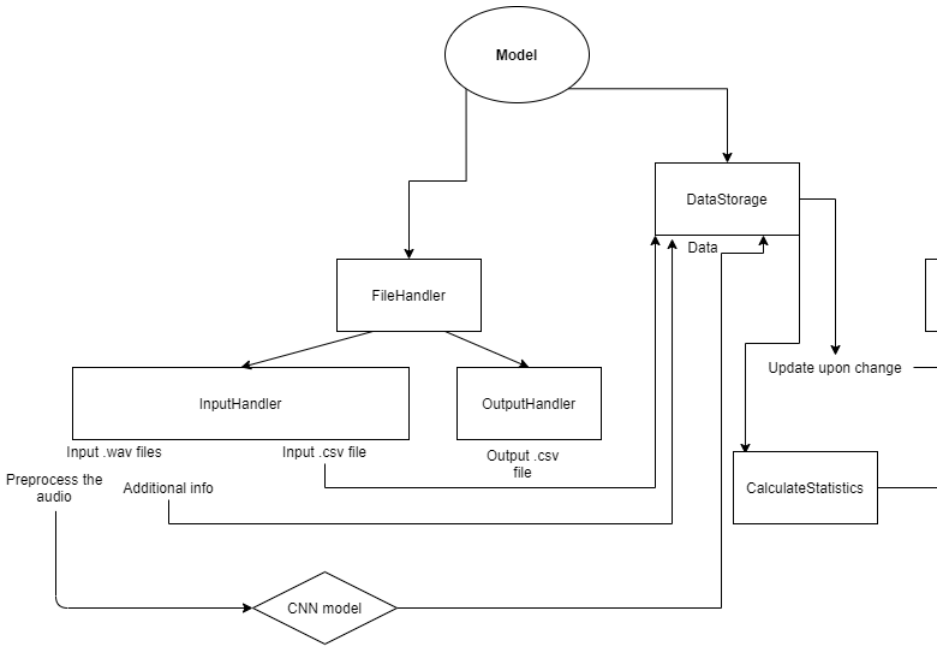
Controller



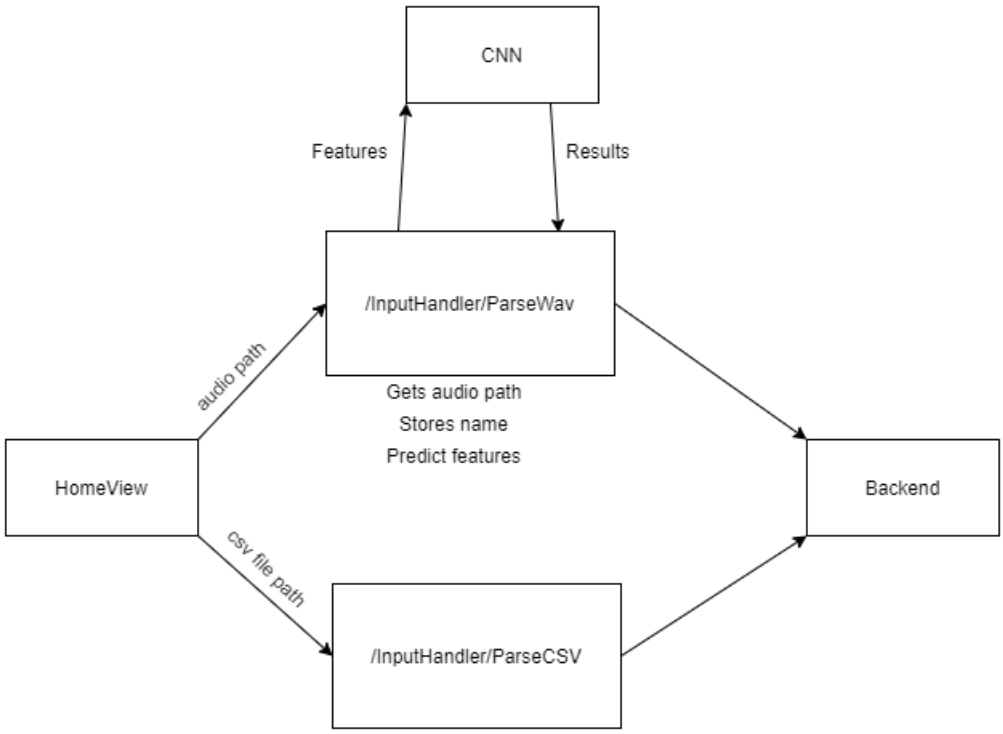
The user can use the controller to interact with the system. This can be done by pressing various buttons and menus integrated in the graphical user interface. The figure above shows different buttons and to which methods it is linked to in the model.

3.2 Input Handler

The Input Handler is an important part of our application. It can be found within the box ´FileHandler´ in the model part of our MVC diagram:



The Input Handler is used to ensure that the audio files are processed correctly and that the correct files are sent with the corresponding CNN-output to the data storage. We made the following graph to get an idea on how we were going to be structuring the Input Handler:



On the main page of the program, a user can decide whether they want to upload a sound file/multiple sound files, or to upload a .csv file.

When selecting to upload a sound file, the user can select a file or multiple files, after which the files (file paths) are sent to the `ParseWav` of `InputHandler`. Here, the audio files are sent to the CNN model, which returns a value for each file separately. The files and its corresponding value are then sent to the backend.

When selecting to upload a .csv file, the user can select a single .csv file. This file is then sent to the `ParseCSV` of the `InputHandler`, which parses the data to a dictionary that is used to update the backend.

3.3 Audio pre-processing and CNN

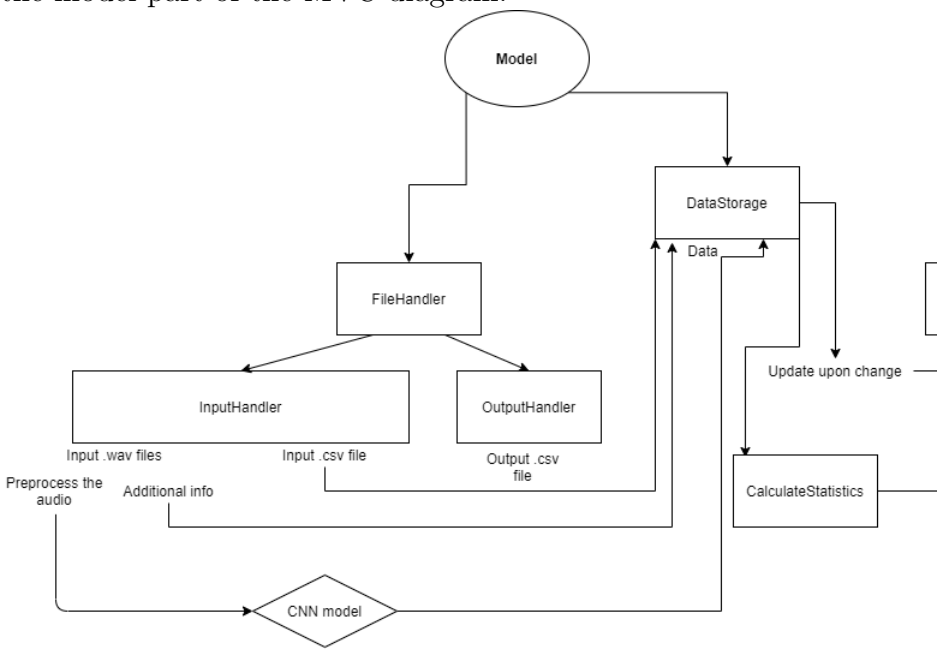
When an audio file has been uploaded, the `InputHandler` will send this file (or the file path rather) to a method where the pre-processing of the audio is done.

In order to correctly retrieve results from the CNN model, we need the uploaded audio signal to be resampled at a sampling rate of 8000Hz. After this we can get the Mel Sceptral Coefficients of the audio file. These are a set of features that concisely describe the overall shape of a spectral envelope. In our case, we get the first 20 coefficients. All of this is quite simply done with the `librosa` package in Python.

The Mel Sceptral Coefficients array is then reformatted, such that it follows the Convolutional Neural Network input format. This array is then put into the CNN model, where it will currently either return a 0 or 1 depending on the severity of the audio signal. This label is returned to the `InputHandler`, where it will connect this label to the inputted audio file.

3.4 Backend

The back end of our application is a single class which functions as a sort of mini database. This back end is placed as the 'DataStorage' block within the model part of the MVC diagram:



The back end class holds a simple python dictionary, which represents all the data to be available in the application. The idea behind this is that data which is in the back end, is shown to the user via the front end (graphical user interface). The way that the data representation is designed, makes it that the keys of the python dictionary represent the column headers, and the values of each key contains a list of values (strings). This list of values represents the values of each row under a certain column with the column header corresponding to the key of that list.

When the application is loaded, the back end attempts to recreate the latest session by a user, by looking for a .json file in a particular location. If this .json file is available, it should contain the data that is formatted in such a way that the application can process it. This file will have the data from the last session stored inside of it. This way, when a user closes the

application without saving, the data is not lost.

Whenever a user uploads a sound file, the model actually handles the updating of the back end. Uploading a sound file is the equivalent of adding a row in this application, because one row represents information that corresponds to a specific audio file. Since the back end is a python dictionary, when adding a new row, the following steps are taken. Per column header, which is a key, the value is obtained. To this list is then added the value, which represents the new row.

If a column is added, a new key is added to the python dictionary. Because this column does not have any values under it yet, the value to this key is simply a list with empty strings.

The backend is constantly updated because information is obtained from the table. The table is the thing which is dynamically updated by the user.

4 Technology Stack

The software was asked to be made for Windows, so is currently only available as a Windows application. The software is based on a single user profile, so no login is currently required.

Languages:

- Python: Used to create the front- and back end

Python libraries:

- PyQt5: Qt GUI framework
- librosa: Audio analysis
- numpy: Complex computations
- math: Mathematical functions
- versioned-hdf5: Weights of the CNN model
- keras: Used to load the CNN model
- tensorflow: To run the CNN model

5 Team Organization

We've split up our team of 5 in two groups: group 1 (Mike, Brian and Roan), and group 2 (Andrei and Annika).

Group 1: In charge of creating the front end, back end and handling of data, merging components together, and creating the architectural document.
Group 2: Assigned to incorporate the CNN model of our TA in our application, together with the responsibility of the requirements document.

We've met up every Friday to discuss our constructed parts. Each week after receiving feedback, we equally divided the different components that had to be made.

6 Change Log

Who	When	Which section	What
Annika	05-03-21	The document	Created the document
Brian & Roan	09-03-21	Start of models	First draft
Brian	09-03-21	Conceptual model	Addition of InputHandler and additional description
Brian & Roan	16-03-21	Conceptual model	Change of flow
Roan & Brian	16-03-21	Module View	Updated the module view
Roan & Brian	16-03-21	MVC Diagram	Added MVC Diagram with explanations
Brian & Roan	29-03-21	The document	Restructured document and added additional information to View and Input Handler
Mike	29-03-21	Architectural Overview	Added Frontend and Backend
Roan	30-03-21	Architectural Overview	Added Audio pre-processing and CNN
Brian	30-03-21	Architectural Overview	Updated Frontend, Backend and InputHandler
Brian & Roan	30-03-21	Technology Stack & Team Organization	Added info to the sections
Mike	30-03-21	The document	Making adjustments and adding info