Fall 2021 Report

actoraesthetic.com

Taylor Bera

Professor John Jenq || CSIT697

department of computer science || montclair state university

Table of Contents

Introduction

Within the theatre community, there is always someone looking to learn. From inexperienced to professional, performers are constantly in class, teaching class, and studying. However, taking classes is inaccessible to those who live outside of New York City. Even in a place that encourages the arts, it can be impossible to find appropriate teachers. This mobile resource includes information on NYC auditions, the actors union, budgeting as a performer, and many more tips and stories from working actors. Additionally, it will be a place for networking, discussions, and safe space for any performer who needs a community.

The system will allow the client to publish their content, so it is imperative that the administrative tasks are easy to perform in-browser by a non-technical user of this system. As a result, the system includes an in-browser CMS(Content Management System) accessible only to users in administrative roles. Within this administrative system, additional to the capability to create content, will be analytics of website traffic, to ensure the most resourceful user experience. Here, the administrator can see which content is getting the most views and which aspects of this website need attention.

In future developments, an account management system will be added to allow users to post, share, and interact with each other to build a community of actors.

ActorAesthetic.com is formatted using python, mysql, html, the flask framework and Bootstrap templates. It is still currently under development.

Similar Web Resources

BroadwayWorld.com

BroadwayWorld is the most similar resource to ActorAesthetic. This website provides information on all aspects of theater, which includes Equity casting calls, exclusive interviews, and articles about the industry:old and new. Additionally, there is a message board for the community to post questions and discussion topics. As of now it is the main trusted outlet for news within the Broadway community, however, the interface is difficult to navigate. Users often complain of sloppy and overcrowded pages.

ActorsAccess.com/BackStage.com

ActorsAccess and Backstage are audition hubs, whose primary purpose is to connect casting directors to actors across the country. Users sign in and browse all auditions happening within their area, both amature and professional. A few blogs are posted to help guide actors to a successful career.

ActorsEquity.com

This is only a resource for those in the Actors Equity Union. Also an audition hub, the Equity website is very informational for anyone looking for business or financial questions, however there is a lack of colloquialism.

Use Case Model

Actors

- Users
- Network Administrator (ActorAesthetic)

Use Cases

ACCOUNTS

- Create user account
    - Provide information <include>
- Edit user profile information
    - Provide information <include>
    - Add/change a profile picture <generalization>
    - Delete user account
- View all user accounts
- Lock account

BLOGS & PODCASTS

- Post posts/podcasts
    - Post special announcements <extends>
- Edit Posts/podcasts
    - Edit announcements <generalization>
    - Moderate image content <generalization>
    - Delete posts

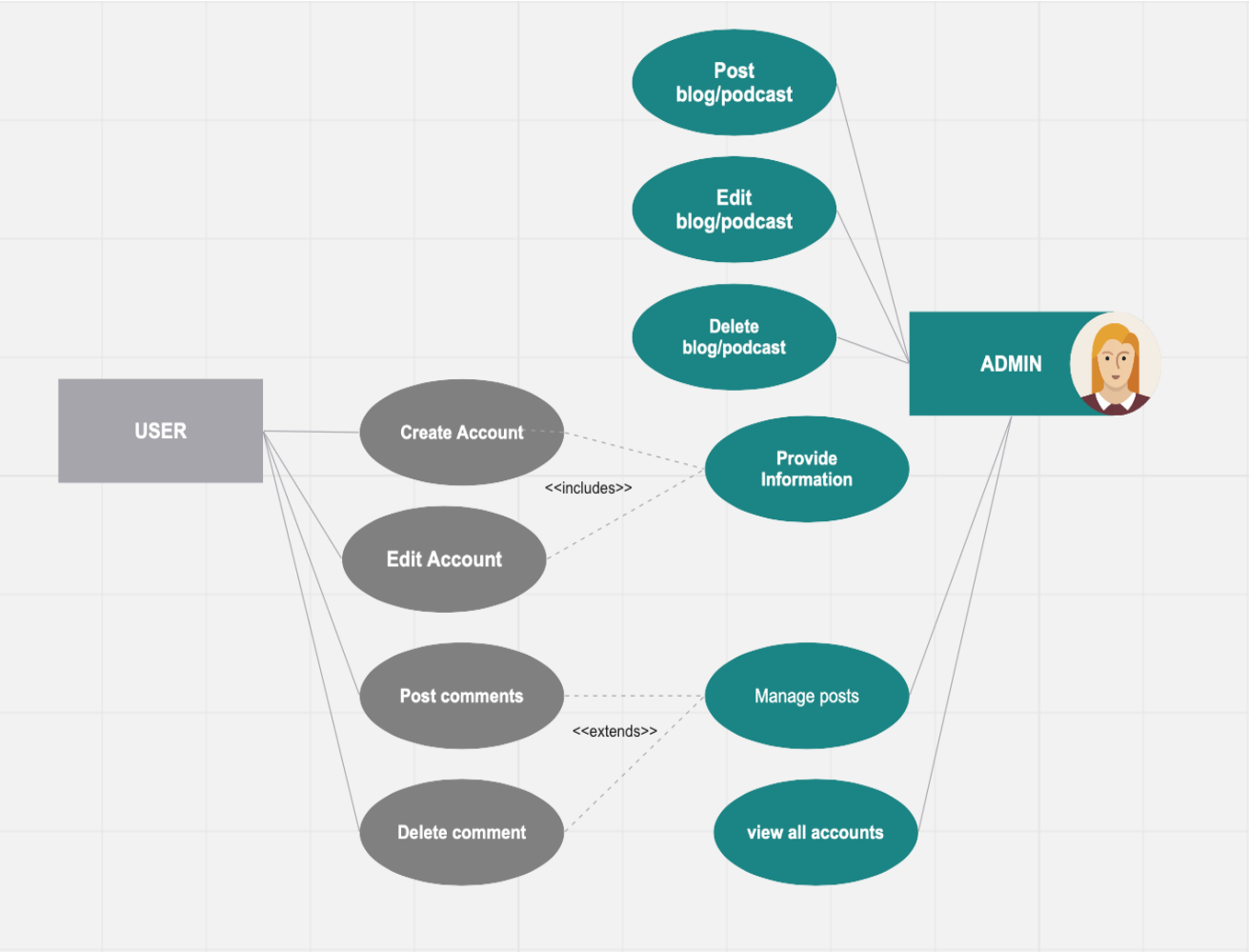
AUDIO

- Transcribe podcasts
    - Edit
    - Delete
    - Redo

OTHER

- Log into system
    - Forgot password <extends>
- Log out of system

Use Case Diagram



System User Interaction

Login Interface

Registered users may login with the email address and password used when their account was created. New users may create an account by providing a name, email address, and password.

User Permissions

ActorAesthetic users will be categorized into three different roles or permission levels.

Users created as ‘SuperAdmin’ accounts will have all the permissions of an Admin account, but will also be able to create Admin accounts as necessary. There will be a maximum of two SuperAdmin accounts, one of which will be handled by the website development team. The other will be handled by the owner of the website.

Users created as ‘Admin’ accounts will be able to post and edit content within the database through an in-browser CMS, as well as create lower level ‘User’ account. These include other coaches and journalists on the actor aesthetic team.

The majority of users will have basic ‘User’ permissions. By default, new accounts will be ‘User’ accounts. Users created at this permission level will be able to maintain their own user profile, comment on other posts, but will not have the ability to edit or delete any other user’s posts or comments.

Data Storage

Blogs, Podcasts, User accounts, permission levels, and profile information will all be stored in a backend database on a remote server. Because ActorAesthteic is an online, web-based platform, no data will be stored on any user’s computer.

Content Management System

A sleek and easy to use in-browser interface will allow for all of the CRUD (Create, Read, Update, Delete) operations needed to manage content and user profiles in the system. The Admin will see all database entries on one page.

Account Management

Users will use their email address to create an account, or an account can be created for them by an Admin level user if necessary. Posted content will consist of text and images. In the interest of maintaining an appropriate environment, selected words will be censored automatically in user posts.

Inappropriate content will be moderated by Admin user accounts. If necessary, an Admin may lock a user's account and they will no longer be able to post, or create a new account with the same email address.

Voice to Text Integration

A sleek and easy to use in-browser interface will allow for all podcast mp3s to be transcribed. The podcast will be transcribed into html format to be displayed on the individual podcast page.

Feasibility

The ActorAesthetic system consists of several features that must be prioritized in order for the system to be ready for distribution by the end of the Fall 2021 semester. In the case of time constraints, lower-priority features will be left to be completed at a later date.

High-priority features include the content management system used by Admins. Without this feature, Actor Aesthetic can not serve its fundamental purpose. This included setting up a database for the website, which can now be easily manipulated for faster turn around on important tasks and updates. Additionally, speech to text recognition was a high priority, since many users rely on this accessibility feature. The speech to text feature still remains as an outside program. In future updates, a web console will be created to allow admin users to transcribe, not just the web developer.

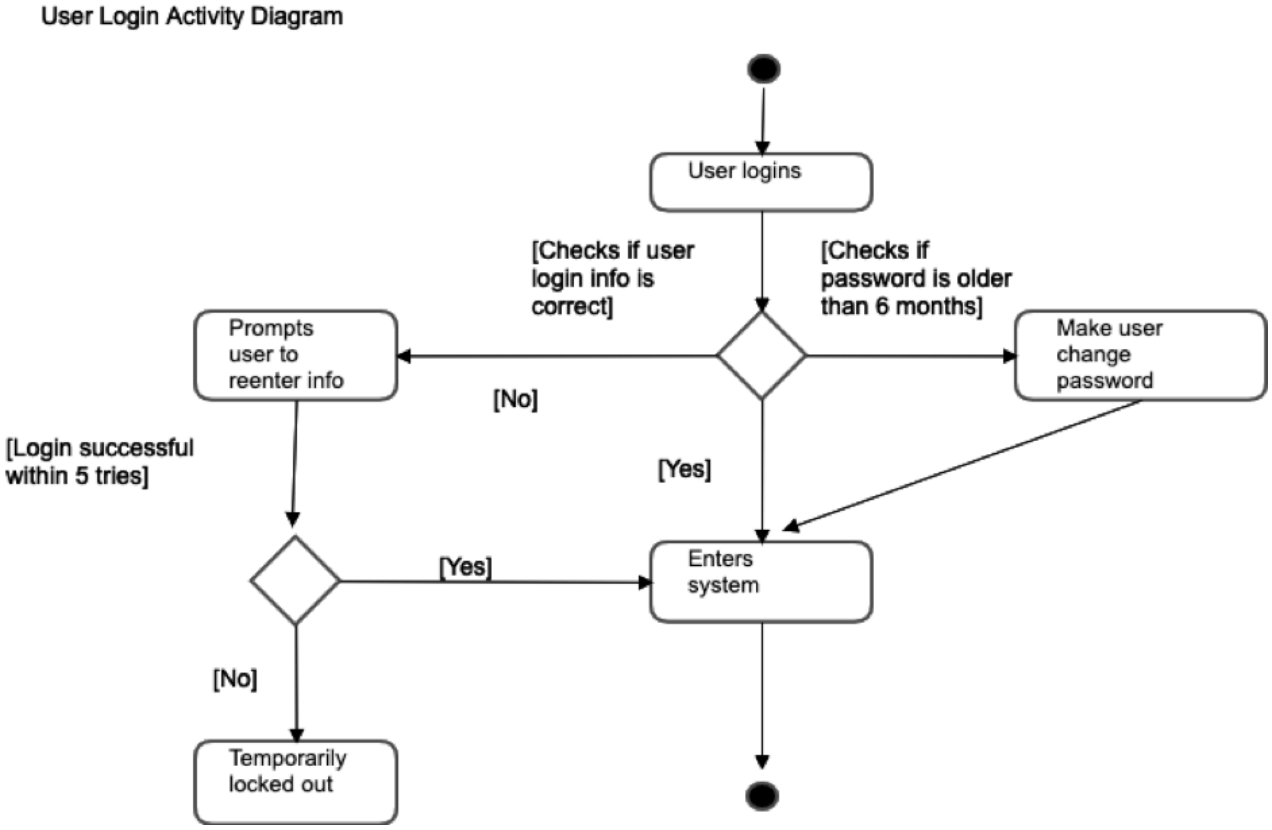
Lower-priority features include the eCommerce area, where users can buy courses, or merchandise. Also, user permissions, accounts, and registration have not been specified yet.

Activity Diagram

List of Activities

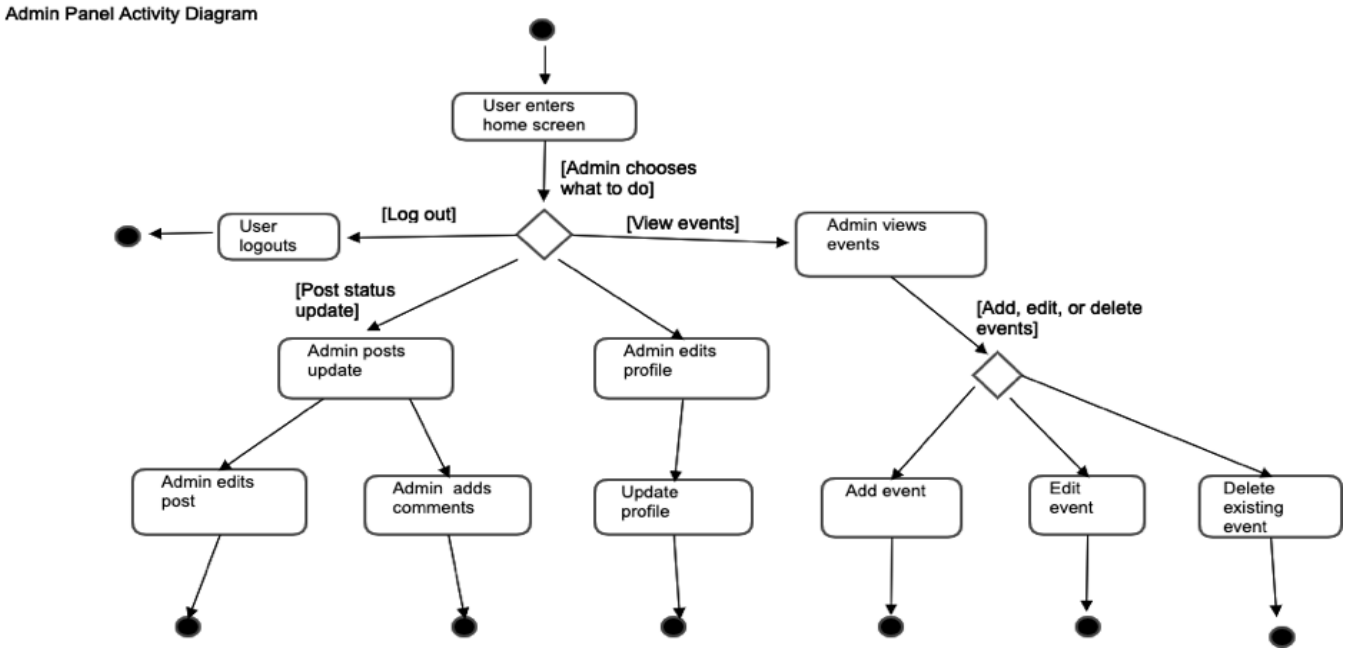
- Admin Logs In
- Network Admin Post Creation

Login Activity



The user login activity diagram shows the flow of the user login system. The user must go through various checks. If they pass all these checks, then the user is allowed to enter the user panel.

Admin Dashboard



The dashboard activity diagram shows the flow for the privileges the admin is given once logged in. The admin is able to log in, post and edit updates, as well as view/add/edit/delete content.

Top Level Design

Each page can be slightly modified or duplicated to add more sections as the client requests.

AAserver.py

I. @app.route('/')

Included is a brief overview of what the company is and any special announcements. This page will establish the theme to enable the user to familiarize themselves with the flow of the site. The menu/ page items are clearly displayed in the header. Additionally, shown is a favorite blog/podcast section, as well as a meet the team. The main purpose of the homepage is to make the user feel included in the community. The menu button is fixed in the top right corner of each page, and the 'actor aesthetic' logo in the top left corner will always be clickable to return the user to the homepage.

START *(index.html)*
Header Content is displayed
SORT BLOG by the most recent date
SELECT from BLOG where postCat is 'top'
Display image/ title
FOR selected
POST *postTitle* to /blogpost
redirect to blogpost
SELECT from BLOG where postCat is 'lifestyle'
Display image/ title
FOR selected
POST *postTitle* to /blogpost
redirect to blogpost
SELECT from BLOG where postCat is 'auditioning'
Display image/ title
FOR selected
POST *postTitle* to /blogpost
redirect to blogpost
SELECT from BLOG where postCat is 'equity"
Display image/ title
FOR selected
POST *postTitle* to /blogpost
redirect to blogpost

II. @app.route('/about')

Here is the page for the history of the website and a biography of the founder. Also included are testimonials from students, and outside articles that deem 'Actor Aesthetic' a professional and reputable

source. This will help parents and students feel comfortable trusting this brand and their knowledge. More information about the coaches and their lessons can be found either through the top navigation or scrolling further down the page.

III. @app.route('/register')

Allow users to sign up to an email list by entering their name and email address.

START ***(login.html)***
Prompt user to **INPUT** name(*fullName)* and email(*userEmail)*
IF userSubmit is posted
fullUser, userEmail **INSERT INTO** USER

IV. @app.route('/login')

The login screen allows the system administrator to view their admin dashboard. It also holds the register modal.

START(*login.html)*
Prompt user to **INPUT** name(*fullName)* and email(*userEmail)*

V. @app.route('/verify)

This function processes the username and password and if inputted correctly, will return the dashboard. Otherwise, the user will be brought back to the main page.

START ***(dashVerify())***
check *newUser* form or *user and password*
IF user and password do not match the set administrative login
RETURN to homepage
ELSE render dashboard template

VI. @app.route('/dashboard')

The dashboard section is only viewable by the owner of ActorAesthetic, and anyone they give permission to. The user of the page can create blogs and podcasts by uploading images and writing their own content. Blogs are categorized in order to be searched internally through the server, as well as tagged to strengthen user engagement on the world wide web. The podcast form allows the admin to upload a url to play their podcast through PODBEAN(a free outside resource).

START ***(dashboard.html)***
Display BlogDB and PodDB

SORT BLOG by the most recent date
SORT PODCAST by the most recent date

IF 'blogSubmit' is posted
    postTitle, postSub, postContent, postFile, postCat, postDate **INSERT INTO** BLOG

IF 'podSubmit' is posted
    podTitle, podSub, podAudio, podContent, podURL, podGuest, podDate
    INSERT INTO PODCAST
SELECT BLOG
    Display image/ title
    FOR selected
        POST *postTitle* to /blogpost
        redirect to blogpost
SELECT PODCAST
    Display image/ title
    FOR selected
        POST *podTitle* to /podPost
        redirect to podcastPost

VII. @app.route('/blog')

Page where users can search articles based on categories and tags.

START ***(blog.html)***
    Header Content is displayed
    SORT BLOG by the most recent date

    SELECT from BLOG
        Display image/ title
        FOR selected
            POST *postTitle* to /blogpost
            redirect to blogpost
    if SELECT from BLOG where postCat is 'college'
        Display image/ title
        FOR selected
            POST *postTitle* to /blogpost
            redirect to blogpost
    elif SELECT from BLOG where postCat is 'covid'
        Display image/ title
        FOR selected
            POST *postTitle* to /blogpost
            redirect to blogpost
    elif SELECT from BLOG where postCat is 'equity'
        Display image/ title
        FOR selected
            POST *postTitle* to /blogpost

redirect to blogpost
elif SELECT from BLOG where postCat is 'lifestyle'
Display image/ title
FOR selected
POST *postTitle* to /blogpost
redirect to blogpost
elif SELECT from BLOG where postCat is 'auditioning'
Display image/ title
FOR selected
POST *postTitle* to /blogpost
redirect to blogpost

VIII. @app.route('/podcast')

Page where users can view all podcasts and testimonials from listeners. Podcasts are grouped by guests and sorted by most recent.

START ***(podcast.html)***
Header Content is displayed
SORT PODCAST by the most recent date
SELECT from PODCAST *LIMIT 8*
Display image/ title
FOR selected
POST *podTitle* to /podcastPost
redirect to podcastPost

SELECT guests from PODCAST
if 'Maggie'
Display image/ title
FOR selected
POST *podTitle* to /podcastPost
redirect to podcastPost
elif
Display image/ title
FOR selected
POST *postTitle* to /blogpost
redirect to blogpost

IX. @app.route('/blogPost')

The template for every blog post to follow. This template gathers the information from the database, and uploads the appropriate row the user would like to read.

START ***(blogpost.html)***
IF POST is requested
GET *postTitle* from selected post
FOR selected post in BLOG

Display *postTitle, postSub, postContent, postFile, postDate*

X. @app.route('/podcastPost')

The template for every podcast post to follow. This template gathers the information from the database, and uploads the appropriate row the user would like to read. Each podcast includes a written transcript for listeners who may prefer to read our content.

START ***(podcastPost.html)***
IF POST is requested
GET *podTitle* from selected post
FOR selected post in PODCAST
Display *podTitle, podSub, podAudio, podContent, podURL, podGuest, podDate*

XI. @app.route(/transcript)

START ***({***.mp3}.html)***
IF POST is requested
GET *podText* from selected post
FOR selected post render html template

XII. ActorAesthetic.sql

The database is hosted locally. It is managed through phpmyAdmin by a root user on the development team, but can be updated by the ActorAesthetic Admin as they add blogs.

Main.py (VoicetoText)

Many users requested transcriptions of the Actor Aesthetic Podcasts, due to disabilities and comfort. Before uploading to the main webpage, the Admin is able to input their downloaded mp3 audio of the podcast into main.py to produce a template page that can be displayed through *actoraesthetic.com/transcript*.

I. audio_transcription()

START
User **INPUTS** name of the mp3 they want to transcribe
Audio is processed and **EXPORTS to WAV** file
WAV file is **cut into chunks** based on silence
Each of those chunks gets audio processed through **speechRecognition**
Texts is **WRITTEN** to html file

Detailed Design

AAserver.py

```
import os
import speech_recognition as sr
import pymysql

os.environ.setdefault('PATH', '')
from flask import Flask, render_template, request, redirect, url_for

con = pymysql.connect(host='localhost',
                      user='root',
                      password='',
                      db='ActorAesthetic')
cursor = con.cursor()
app = Flask(__name__)
r = sr.Recognizer()


# home
@app.route('/')
def index():
  cursor.execute("SELECT * from Blog ORDER BY `postDate` DESC LIMIT 4")
  favorites = cursor.fetchall()

  cursor.execute("SELECT * from Blog where postCat = 'lifestyle' ORDER BY `postDate` DESC LIMIT 3")
  lifestyle = cursor.fetchall()

  cursor.execute("SELECT * from Blog where postCat = 'auditioning' ORDER BY `postDate` DESC LIMIT 3")
  audition = cursor.fetchall()

  cursor.execute("SELECT * from Blog where postCat = 'equity' ORDER BY `postDate` DESC LIMIT 3")
  equity = cursor.fetchall()
  return render_template('index.html', favorites=favorites, lifestyle=lifestyle, audition=audition, equity=equity)


# about
@app.route('/about')
def about():
  return render_template('about.html')


# dashboard
@app.route('/dashboard', methods=['GET', 'POST'])
def dash():
  cursor.execute("SELECT * FROM `blog` ORDER BY `postDate` DESC")
  blogdata = cursor.fetchall()

  cursor.execute("SELECT * FROM `podcast` ORDER BY `podDate` DESC")
  podata = cursor.fetchall()

  if request.method == 'POST':
```

```
    if request.form.get('blogSubmit'):
        title = request.form['postTitle']
        sub = request.form['postSub']
        content = request.form['postContent']
        file = request.form['file']
        cat = request.form['category']
        date = request.form['postDate']

        newBlog = "INSERT INTO ActorAesthetic.Blog(postTitle, postSub, postContent, postFile, postCat, " \
                  "postDate) VALUES(%s, %s, %s, %s, %s, %s); "
        info = (title, sub, content, file, cat, date)
        cursor.execute(newBlog, info)
        con.commit()
        return redirect(url_for('dash'))

    if request.form.get('podSubmit'):
        title = request.form['podTitle']
        sub = request.form['podSub']
        audio = request.form['podAudio']
        content = request.form['podContent']
        url = request.form['podURL']
        guest = request.form['podGuest']
        date = request.form['podDate']
        mp3Text = request.form['podText']

        newPod = "INSERT INTO ActorAesthetic.Podcast(podTitle, podSub, podAudio, podContent, podURL, podGuest,
" \
                 "podDate, podText) VALUES(%s, %s, %s, %s, %s, %s, %s, %s); "
        info = (title, sub, audio, content, url, guest, date, mp3Text)
        cursor.execute(newPod, info)
        con.commit()
        return redirect(url_for('dash'))

  return render_template('dash.html', blogdata=blogdata, podata=podata)


@app.route('/blog', methods=['GET', 'POST'])
def blog():
  cursor.execute("SELECT * FROM `blog` ORDER BY `postDate` DESC")
  blogdata = cursor.fetchall()

  cursor.execute("SELECT * from Blog where postCat = 'top' ORDER BY `postDate` DESC")
  top = cursor.fetchall()

  cursor.execute("SELECT * from Blog where postCat = 'college' ORDER BY `postDate` DESC")
  college = cursor.fetchall()

  cursor.execute("SELECT * from Blog where postCat = 'equity' ORDER BY `postDate` DESC")
  equity = cursor.fetchall()

  cursor.execute("SELECT * from Blog where postCat = 'lifestyle' ORDER BY `postDate` DESC")
  lifestyle = cursor.fetchall()

  cursor.execute("SELECT * from Blog where postCat = 'auditioning' ORDER BY `postDate` DESC")
```

```
  auditioning = cursor.fetchall()
  return render_template('blog.html', blogdata=blogdata, top=top, college=college, equity=equity, lifestyle=lifestyle,
                         auditioning=auditioning)


# podcast
@app.route('/podcast', methods=['GET', 'POST'])
def podcast():
  cursor.execute("SELECT * FROM `podcast` ORDER BY `podDate` DESC LIMIT 10")
  recent = cursor.fetchall()

  cursor.execute("SELECT * FROM `podcast` where `podGuest` != 'Maggie' ORDER BY `podDate` DESC")
  guest = cursor.fetchall()

  cursor.execute("SELECT * FROM `podcast` where `podGuest` = 'Maggie' ORDER BY `podDate` DESC")
  maggie = cursor.fetchall()

  return render_template('podcast.html', recent=recent, guest=guest, maggie=maggie)


@app.route('/blogPost', methods=['GET', 'POST'])
def blogPost():
  if request.method == 'POST':
    if request.form.get('readButton'):
      selectedPost = request.form['readButton']
      cursor.execute("SELECT * from Blog")
      blogdata = cursor.fetchall()
      html = str(selectedPost)
      return render_template('blogPost.html', blogdata=blogdata, html=html)


@app.route('/podcastPost', methods=['GET', 'POST'])
def podcastPost():
  if request.method == 'POST' and request.form.get('listenButton'):
    selectedPod = request.form['listenButton']
    cursor.execute("SELECT * from Podcast")
    podata = cursor.fetchall()
    id = str(selectedPod)
    return render_template('podcastPost.html', podata=podata, id=id)


@app.route('/transcript', methods=['GET', 'POST'])
def transcript():
  if request.method == 'POST' and request.form.get('textButton'):
    selectedPod = request.form['textButton']
    selectedPod = f"speech-files/{selectedPod}"
    return render_template(selectedPod)


if __name__ == "__main__":
  app.run(host='127.0.0.1', debug=True)
```

Main.py (voicetotext)

```
import speech_recognition as sr
import os
from pydub import AudioSegment
from pydub.silence import split_on_silence

r = sr.Recognizer()


def audio_transcription(path):
  sound = AudioSegment.from_wav(path)
  chunks = split_on_silence(sound,
                        min_silence_len=1000,
                        silence_thresh=sound.dBFS - 20,
                        keep_silence=300, )

  folder = "voicetotext/audio-chunks"

  whole_text = ""

  for i, audio_chunk in enumerate(chunks, start=1):
    # export audio chunk and save it in the 'folder_name' directory.
    chunk_filename = os.path.join(folder, f"chunk{i}.wav")
    audio_chunk.export(chunk_filename, format="wav")

    with sr.AudioFile(chunk_filename) as source:
      audio_listened = r.record(source)
      try:
        text = r.recognize_google(audio_listened)
      except sr.UnknownValueError as e:
        print("Error:", str(e))
      else:
        text = f"{text.capitalize()}. \n"
        print(text)
        whole_text += text
  file = open(f"{mp3}.html", "w+")
  file.write("<html><p>" + whole_text + "</p></html>")
  file.close()
  return whole_text


if __name__ == '__main__':
  mp3 = input("Which mp3 do you want to transcribe:\n")
  audio = AudioSegment.from_mp3(f"voicetotext/{mp3}")
  path = audio.export("transcript.wav", format="wav")
  audio_transcription(path)
```

ActorAesthetic.sql

```
-- phpMyAdmin SQL Dump
-- version 5.1.1
-- https://www.phpmyadmin.net/
--
-- Host: localhost
-- Generation Time: Dec 16, 2021 at 12:19 AM
-- Server version: 10.4.20-MariaDB
-- PHP Version: 8.0.9

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
START TRANSACTION;
SET time_zone = "+00:00";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

--
-- Database: `ActorAesthetic`
--

-- --------------------------------------------------------

--
-- Table structure for table `blog`
--

CREATE TABLE `blog` (
  `postTitle` text NOT NULL,
  `postSub` text NOT NULL,
  `postFile` varchar(250) NOT NULL,
  `postContent` longtext NOT NULL,
```

```
  `postCat` text NOT NULL,
  `postDate` timestamp NOT NULL DEFAULT current_timestamp() ON UPDATE current_timestamp()
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

--
-- Dumping data for table `blog`
--


-- --------------------------------------------------------

--
-- Table structure for table `podcast`
--

CREATE TABLE `podcast` (
  `podTitle` text NOT NULL,
  `podSub` text NOT NULL,
  `podAudio` varchar(250) NOT NULL,
  `podContent` longtext NOT NULL,
  `podUrl` text DEFAULT NULL,
  `podGuest` text NOT NULL,
  `podDate` datetime NOT NULL,
  `podText` varchar(300) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

--
-- Dumping data for table `podcast`
--


--
-- Indexes for dumped tables
--

--
```

```
-- Indexes for table `blog`
--
ALTER TABLE `blog`
  ADD PRIMARY KEY (`postDate`);


--
-- Indexes for table `podcast`
--
ALTER TABLE `podcast`
  ADD PRIMARY KEY (`podDate`);
COMMIT;

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

Testing

Objectives and Success Criteria

The objective of testing is to make certain that the ActorAesthteic functions as intended per set specifications. System testing executes and tests every component of the system by individual modules. All possible scenarios including uncommon edge cases are considered to assure the system runs smoothly every time.

Testing consists of a repeating cycle of identifying bugs and fixing them. Testing allows for identification of any present defects or bugs in the software system, along with mitigating future problems and defects. Once these problems are identified, they are rectified.

The end goal of testing is to have a dependable fully functional software, that is ready for production and deployment, and satisfactory to the client. Each module must execute individually and holistically with other modules in a stable state.

ActorAesthetic runs with many repeated functionalities, while the core of the website rests in the dashboard. The dashboard has two large components; first is the database dashboard. This is the grid that displays all content sorted by most recent. Second, are the new blog and new podcast forms. Upon submitting, these forms must update on every page, consistently. These components are isolated, and tested in a similar fashion. The individual input of information for each component is validated and verified.

Additionally, testing has been completed for the voice to text feature to ensure not only that it works, but how it works and what to expect from the outcome. Since it is a newly integrated software, the reliability of this component is low.

After each component is tested separately, all components are tested together to ensure the whole system works in a stable and seamless manner.

Test Scripts

Enter dashboard

Software Component: login to dashboard Software Module: Dashboard Module				
Sequence	Source	Script Event	Evaluation	Comments
1	User	User inputs username “maggiebera” and password “admin”	The System should bring the user to to dashboard	
2	User	User inputs username “taylorbera” and password “admin”	The System should bring the user to to the homepage	

Post a New Blog

Software Component: Post a Blog Software Module: Dashboard Module				
Sequence	Source	Script Event	Evaluation	Comments
1	User	User inputs a PostTitle “The 3 Social Media Platforms All Actors Should Try”	The System should add the postTitle to ‘Blog.’ It should appear as the header of the blogPost	
2	User	User inputs a postSub “Instagram, TikTok, Twitter, Facebook, Clubhouse… which social media platforms should actors be on these days?”	The System should add the postSub to ‘Blog.’ It should appear as the subheader of the blogPost	
3	User	User does not upload file	System must generate an error, “Attachment must be in jpeg, png format and less than or equal to 1MB”. Then prompt the user to try a second time.	
4	User	User uploads ‘socialmedia.png’	The System should add the file to ‘Blog.’ It should appear as the image of the blogPost	

5	User	User uploads at '2021-02-17 13:43:00'	The System should add postDate to 'Blog.' It should appear as the date of the blogPost	
6	User	User selects 'top' for postCategory	The System should update the Blog page and homepage to display the new blog in the top category	
7	User	User selects 'equity' for postCategory	The System should update the Blog page and homepage to display the new blog in the equity category	

Post a New Podcast

Software Component: Post a Podcast Software Module: Dashboard Module				
Sequence	Source	Script Event	Evaluation	Comments
1	User	User inputs a PodTitle "College auditions, self tapes, and survival jobs with Ben Biggers (Beautiful: The Carole King Musical)"	The System should add the podTitle to 'Podcast.' It should appear as the header of the podcastPost	
2	User	User inputs a podSub "ep. 151"	The System should add the podSub to 'Podcast.' It should appear as the subheader of the podcastPost	
3	User	User does not upload file	System must generate an error, "Attachment must be in jpeg, png format and less than or equal to 1MB". Then prompt the user to try a second time.	
4	User	User uploads 'ep151.png'	The System should add the file to 'Podcast.' It should appear as the image of the podcastPost	
5	User	User uploads at '2021-02-17 13:43:00'	The System should add postDate to 'Podcast.' It should appear as the date of the podcastPost	System sorted podcasts appropriately. This podcast was not the most recently recorded, but it was the most recently uploaded. It appeared second in the database.

6	User	User uploads ‘2nmw2-112788b’ to podURL	The System should add podURL to ‘Podcast.’ Podcast audio should play.	
7	User	User does not upload to podURL	The System will not connect to ‘podbean’ and no audio will play.	

Transcribe Audio

Software Component: Transcribe Audio Software Module: voicetotext				
Sequence	Source	Script Event	Evaluation	Comments
1	System/ User	System asks user which file they want to work on	The System should take the user input and feed it to audio_transcription()	User input “test.mp3”
2	System	Audio is converted to .wav	The System should export .mp3 to ‘transcript.wav’	
3	System	Audio is split into smaller sections	The System should split chunks based on silence	*This is a short audio*
4	System	Text is written to system console	As the System processes audio chunks, the text is displayed to the user	
5	System	Text is written to new html document	“Test.mp3.html”should be created and modified	Incorrect title for file generated html

Software Component: Transcribe Audio Software Module: voicetotext				
Sequence	Source	Script Event	Evaluation	Comments
1	System/ User	System asks user which file they want to work on	The System should take the user input and feed it to audio_transcription()	User input “134.mp3”
2	System	Audio is converted to .wav	The System should export .mp3 to ‘transcript.wav’	

3	System	Audio is split into sections	The System should split chunks based on silence	*This is a long audio*
4	System	Text is written to system console	As the System processes audio chunks, the text is displayed to the user	
5	System	Text is written to new html document	"134.mp3.html"should be created and modified	Issue fixed by changing variable in python script

Resources

I referred to *https://www.w3schools.com* for clarification on any issues.

ActorAesthetic.com

Broadwayworld.com

ActorsAccess.com

ActorsEquity.org

Acknowledgements

Dr. John Jenq
Associate Professor, Computer Science
Department of Computer Science
jenqj@montclair.edu || 973-655-7237

Dr. Dawei Li
Assistant Professor, Computer Science
Department of Computer Science
lida@montclair.edu || 973-655-3127

Dr. Jiacheng Shang
Assistant Professor, Computer Science
Department of Computer Science
lshangj@montclair.edu