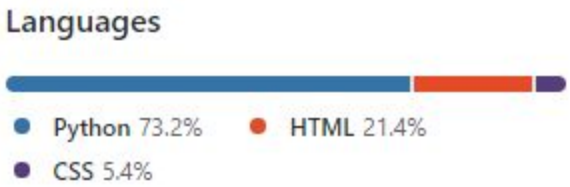
# TheOscarProject

Jonathan Cross, Marcus Huston, Jordan OConnell, Gary Young, Jason Fares, Justin Moua

# Introduction

- We used python for functionality- HTML/CSS for website
- We used Flask for the API
- We used heroku to host the website



# Product Vision

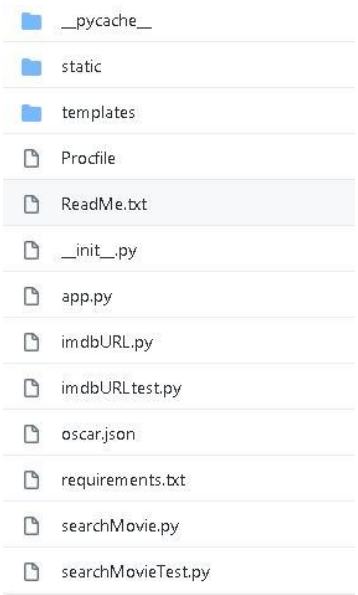
Our product is for people who want an easy way to find Oscar winning movies. The product is a website that utilizes the internet to search for an Oscar winning or Oscar nominated movie by award categories. Unlike IMDB, our product provides the user with more Oscar award categories to search from. Our product will provide users information about a particular Oscar winning or Oscar nominated movie, including reviews and where the movie can be watched. It will also allow for other developers to use an API to extend the functionality of our product.

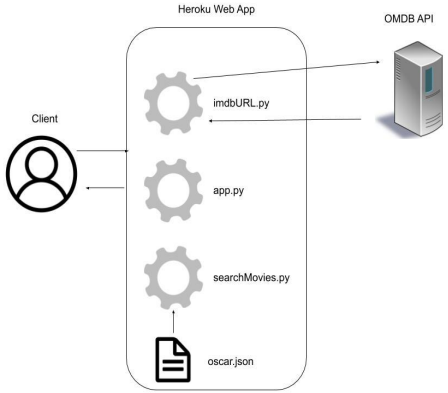
# Product Demo

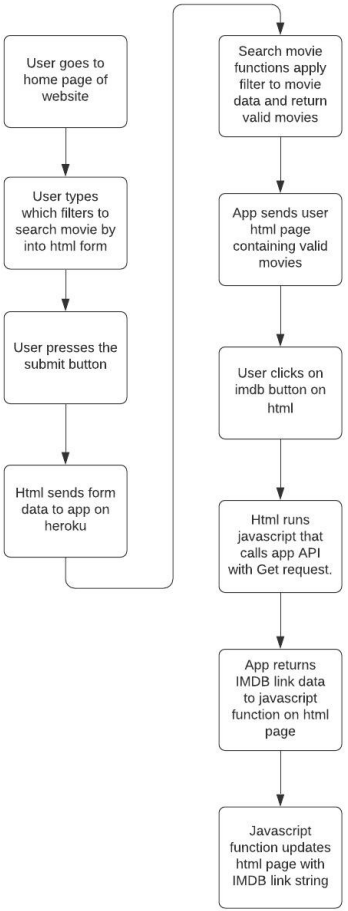
- Show the different website tabs
- Demonstrate API functionality
  - Type in URLs and show json display
- Demonstrate search functionality
  - Do Name, Category, Year, winner

# How Our Application Works

- Picture of our app directory
  - Show project repository
  - Brief description of each file



# Platform, technologies, tools used

- Flask
  - What is flask?
    - Flask is a framework that allows people to build web applications with python
    - It abstracts most of the complicated logic and details regarding web applications and client requests
    - It is essentially a library used to make our API
- Jinja
  - What is Jinja?
    - It's a web templating language for python that allows people to have more control over html pages, such as helping display data from python onto the html page.
- Heroku
  - What is Heroku?
    - Heroku is a free cloud platform that allow users to put their apps on the web.
    - It works in combination with Git to push files to the heroku server.

# Platform, technologies, tools continued...

- OMDb
  - What is OMDb?
    - OMDb(Open Movie Database) is a third party API that searches data from IMDb
    - We used OMDb retrieve links for the movies on IMDb’s website

# Coding Challenges / Obstacles

- How we filter movies from the data file
  - The movie data
    - The movie data was in a CSV format, so we converted it into a JSON to make things easier.

```
year_film,year_ceremony,ceremony,category,name,film,winner
1927,1928,1,ACTOR,Richard Barthelmess,The Noose,False
1927,1928,1,ACTOR,Emil Jannings,The Last Command,True
1927,1928,1,ACTRESS,Louise Dresser,A Ship Comes In,False
1927,1928,1,ACTRESS,Janet Gaynor,7th Heaven,True
1927,1928,1,ACTRESS,Gloria Swanson,Sadie Thompson,False
1927,1928,1,ART DIRECTION,Rochus Gliese,Sunrise,False
1927,1928,1,ART DIRECTION,William Cameron Menzies,The Dove;,True
1927,1928,1,ART DIRECTION,Harry Oliver,7th Heaven,False
1927,1928,1,CINEMATOGRAPHY,George Barnes,The Devil Dancer;,False
1927,1928,1,CINEMATOGRAPHY,Charles Rosher,Sunrise,True
1927,1928,1,CINEMATOGRAPHY,Karl Struss,Sunrise,True
1927,1928,1,DIRECTING (Comedy Picture),Lewis Milestone,Two Arabian Knights,True
1927,1928,1,DIRECTING (Comedy Picture),Ted Wilde,Speedy,False
1927,1928,1,DIRECTING (Dramatic Picture),Frank Borzage,7th Heaven,True
1927,1928,1,DIRECTING (Dramatic Picture),Herbert Brenon,Sorrell and Son,False
1927,1928,1,DIRECTING (Dramatic Picture),King Vidor,The Crowd,False
1927,1928,1,ENGINEERING EFFECTS,Ralph Hammeras,,False
1927,1928,1,ENGINEERING EFFECTS,Roy Pomeroy,Wings,True
1927,1928,1,ENGINEERING EFFECTS,Nugent Slaughter,,False
1927,1928,1,OUTSTANDING PICTURE,The Caddo Company,The Racket,False
```


```
[
  {
    "year":1928,
    "category":"ACTOR",
    "name":"Richard Barthelmess",
    "film":"The Noose",
    "winner":false
  },
  {
    "year":1928,
    "category":"ACTOR",
    "name":"Emil Jannings",
    "film":"The Last Command",
    "winner":true
  }
```

# Coding Challenges / Obstacles

- How we filter movies from the data file continued...
  - How to deal with different combinations of filters
    - Created four functions, each function takes in a data set, and returns a data set; now the functions have a sort of pipe and filter characteristic.
    - Used a dictionary to keep track of repeated movies

```
def searchByName(par, d):
    resultsDict={}
    arr=[]
    for i in range(0,len(d)):
        val = d[i]['film']
        if par.upper() in val.upper():
            if not val in resultsDict:
                if type(d[i]['category']) is list:
                    resultsDict[val]={
                        "year":d[i]['year'],
                        "category":d[i]['category'],
                        "name":d[i]['name'],
                        "film":d[i]['film'],
                        "winner":d[i]['winner']
                    }
                    arr.append(resultsDict[val])
                else:
                    resultsDict[val]={
                        "year":d[i]['year'],
                        "category":[d[i]['category']],
                        "name":d[i]['name'],
                        "film":d[i]['film'],
                        "winner":d[i]['winner']
                    }
                    arr.append(resultsDict[val])
            else:
                if not type(d[i]['category']) is list:
                    resultsDict[val]['category'].append(d[i]['category'])
    return arr
```

```
def searchByCategory(par, d):
    resultsDict={}
    arr=[]
    par=par.upper()
    for i in range(0,len(d)):
        val = d[i]['category']
        nVal = d[i]['film']

        if not nVal in resultsDict:
            if type(val) is list:
                for j in range(0,len(val)):
                    if par in val[j]:
                        resultsDict[nVal]={
                            "year":d[i]['year'],
                            "category":d[i]['category'],
                            "name":d[i]['name'],
                            "film":d[i]['film'],
                            "winner":d[i]['winner']
                        }
                        arr.append(resultsDict[nVal])
                        break
            else:
                if par in val:
                    resultsDict[nVal]={
                        "year":d[i]['year'],
                        "category":[d[i]['category']],
                        "name":d[i]['name'],
                        "film":d[i]['film'],
                        "winner":d[i]['winner']
                    }
                    arr.append(resultsDict[nVal])
        elif nVal in resultsDict:
            #add other categories to movie
            if not type(val) is list:
                resultsDict[nVal]['category'].append(val)
    return arr
```

```
def searchByYear(par, d):
    resultsDict={}
    arr=[]
    for i in range(0,len(d)):
        val = d[i]['year']
        nVal = d[i]['film']
        if par == val:
            if not nVal in resultsDict:
                if type(d[i]['category']) is list:
                    resultsDict[nVal]={
                        "year":d[i]['year'],
                        "category":d[i]['category'],
                        "name":d[i]['name'],
                        "film":d[i]['film'],
                        "winner":d[i]['winner']
                    }
                    arr.append(resultsDict[nVal])

                else:
                    resultsDict[nVal]={
                        "year":d[i]['year'],
                        "category":[d[i]['category']],
                        "name":d[i]['name'],
                        "film":d[i]['film'],
                        "winner":d[i]['winner']
                    }
                    arr.append(resultsDict[nVal])
            elif nVal in resultsDict:
                if not type(d[i]['category']) is list:
                    resultsDict[nVal]['category'].append(d[i]['category'])
    return arr
```

```
def searchByWinner(par, d):
    if type(par) != bool:
        print("error, value passed to searchByWinner must be a bool")
    resultsDict={}
    arr=[]
    for i in range(0,len(d)):
        val = d[i]['winner']
        nVal = d[i]['film']
        if par == val:
            if not nVal in resultsDict:
                if type(d[i]['category']) is list:
                    resultsDict[nVal]={
                        "year":d[i]['year'],
                        "category":d[i]['category'],
                        "name":d[i]['name'],
                        "film":d[i]['film'],
                        "winner":d[i]['winner']
                    }
                    arr.append(resultsDict[nVal])

                else:
                    resultsDict[nVal]={
                        "year":d[i]['year'],
                        "category":[d[i]['category']],
                        "name":d[i]['name'],
                        "film":d[i]['film'],
                        "winner":d[i]['winner']
                    }
                    arr.append(resultsDict[nVal])
            elif nVal in resultsDict:
                if not type(d[i]['category']) is list:
                    resultsDict[nVal]['category'].append(d[i]['category'])
    return arr
```

# Coding Challenges / Obstacles

- How we generate correct IMBD link for movies efficiently
  - Check if the movie returned by OMDB is close enough to the ceremony year of the target movie
  - Have javascript run on the html that calls our API to get the IMDB link for the movie

```python
def giveURL(movie, year):
    #use string replace to format movie title for url
    strpmvie = movie.replace(' ', '+')
    strpmvie = movie.replace("'", "'")

    #query omdb api for the imdb id of movie
    respString = 'http://www.omdbapi.com/?s='+ strpmvie + '&type=movie' +'&apikey=[redacted]'
    r = requests.get(respString)
    #put api response into json format
    dictionary = r.json()

    upperBound=5
    imdbURL="NA"
    if ('Search' in dictionary) and len(dictionary['Search']) >= 1:
        for i in range(0, len(dictionary['Search'])):
            movieYear=dictionary['Search'][i]['Year']
            if  movieYear <= year and (int(year)-int(movieYear) <= upperBound):
                imdbURL = "https://www.imdb.com/title/" + dictionary['Search'][i]['imdbID']
                break

    #put imdb id into the imdb movie url to get their webpage for the given movie and return the result
    return(imdbURL)
```

```javascript
<script>
    // Does a api call to the server to get the IMDB link
    function myFunction(par, par2) {

        var elem = document.getElementById(par);
        var s = "http://127.0.0.1:5000/imbdbURL?mname=" + par + "&year=" + par2;
        fetch(s).then(function(response) {
            response.text().then(function(text) {
                elem.href = text;
                elem.innerHTML = text;
            });
        });
    }
</script>
```

# Coding Challenges / Obstacles

- Getting the code to work in the online and offline versions and difficult to do testing on online version
  - Solution was to do testing on offline and then change the links and run it online
  - The code between the online and offline versions are mostly the same, the only difference are the URLS in the html pages.

offline version

```
<body>
    <ul>
        <img src="{{ url_for('static', filename='Images/TheOscarProjectLogo.png') }}" class= "logo" alt="The Oscar Project" width="180" height="50">
        <li><a class="active" href="http://127.0.0.1:5000/">Home</a></li>
        <li><a href="http://127.0.0.1:5000/about">About</a></li>
        <li><a href="http://127.0.0.1:5000/apiHelp">API</a></li>
    </ul>
```

online version

```
<body>
    <ul>
        <img src="{{ url_for('static', filename='Images/TheOscarProjectLogo.png') }}" class= "logo" alt="The Oscar Project" width="180" height="50">
        <li><a class="active" href="https://oscarproj1.herokuapp.com/">Home</a></li>
        <li><a href="https://oscarproj1.herokuapp.com/about">About</a></li>
        <li><a href="https://oscarproj1.herokuapp.com/apiHelp">API</a></li>
    </ul>
```

# Lessons Learned

- Learned that planning before coding makes better code
- Things take longer to complete than estimated