MOVIE SCHEDULING IN A MULTIPLEX MALL

POWER 3

- **Adithya Muthukumar**
- **Alamelu Kannan**
- **C B Ananya**

ABSTRACT

Timetabling is something that all single-screen theatres and multiplexes have to do once a week, but unfortunately, there is no standard method or formula that is used to do this. Most theatre managers in India use their own subjective thinking to gauge the audience interest in a film and use that to roughly create a timetable. While this method has worked till today, it may not always provide the best solution that generates the most profits for a theatre, simply because there is no mathematical hypothesis that is applied to this.

Our solution, an automatic timetabling system for multiplexes, aims to change that.

INTRODUCTION

This project initially started off as a simple end-semester project that was being done in the academic sense. However, the team, being lovers of the movies, ended up conversing with actual theatre owners, and recognised that the problem existed. However, there is no demand for such a solution, simply because theatre owners are following a ‘if-it-aint-broke-dont-fix-it’ attitude. But, the team is confident enough that if this product is taken to market, the need will be generated, because it cuts-off a significant amount of human labour while promising the best results possible. This solution could also have a universal appeal outside of theatres, in places like theme parks, stand-up shows, concerts et all.

Our timetabling solution is a modular program that is built on Python+Django framework. Python was chosen because of it’s support for garbage collection, automatic memory management, but most importantly, we chose it for it’s way it handles scripts. Our solution is quite intensive, and we want it to be as modular and fast as possible, and Python seems like the best solution to it. Moreover, we have built a clean and easy-to-use user interface in the form of a web application, made possible using the Django framework.

The program does the following:

a) A signup/login page, where theatre owners can either login to access their theatre details, or signup and enter theatre details, which includes the screen names and showtimes and the features the screen has (2D, IMAX, Dolby Atmos, 3D)
b) Generates a set of possible shows, and assigns a score to them, based on the features and the timings
c) Once logged in, movie details (name, language, release date, certification) can be entered
d) Generates a score for every movie based on both the input data and real-time data (the popularity on Google of every movie)
e) Generates set of all possible combinations of the movies and showtimes, through an enhanced bruteforcing algorithm
f) From the set of all combinations, returns the timetable with the highest profit, through a maximisation algorithm

EXISTING WORK

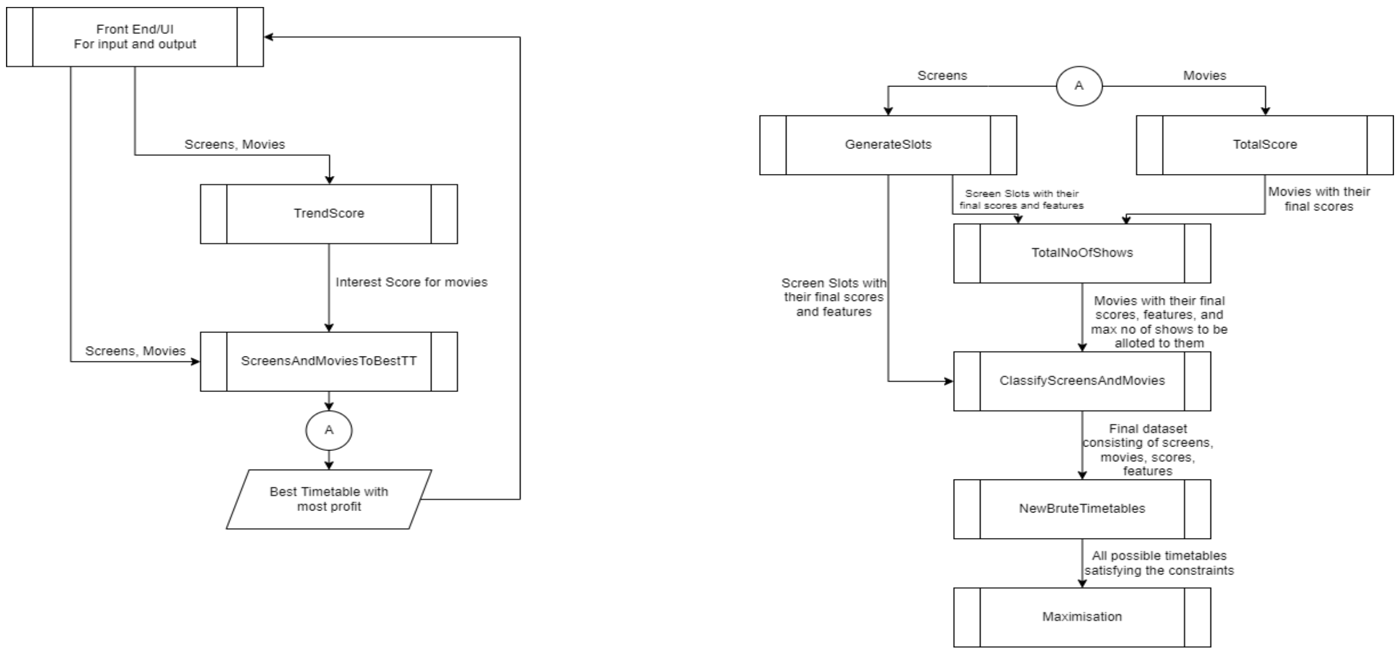
Till date, theater screening systems are not automated and owners have preferred to do it manually as they feel that they haven't found an algorithm that best captures the pulse of the audience and their demands. However, human understanding of certain data may lead to errors and inefficiency. This also results in intensive human labor for the creation of timetables that match all constraints.

In India, solutions such as the Sony STM-100L (bundled with Sony's own projectors) and Cinicloud exist, but are limited to just creating playlists and managing KDMs (Key Delivery Messages) and DCPs (Digital Cinema Packages). A timetabling utility does not exist for this purpose.

In countries such as Canada, USA, UK and Ireland, there exists a solution called the Comscore Cinema Auditorium Control Engine. Comscore is a data collection company, that tracks the box office through a closed-source metric called Comscore International Box Office Essentials. Based on this data, this engine computes timetables. However, this closed-sourced entity could be prone to manipulation and is also a very expensive solution.

DESIGN

Architecture Diagram


Front End/UI
For input and output
Screens, Movies
TrendScore
Interest Score for movies
Screens, Movies
ScreensAndMoviesToBestTT
A
Best Timetable with most profit
Screens
A
Movies
GenerateSlots
TotalScore
Screen Slots with their final scores and features
Movies with their final scores
TotalNoOfShows
Screen Slots with their final scores and features
Movies with their final scores, features, and max no of shows to be alloted to them
ClassifyScreensAndMovies
Final dataset consisting of screens, movies, scores, features
NewBruteTimetables
All possible timetables satisfying the constraints
Maximisation

Python built-in modules used for implementation of our program:
datetime, time, sympy.utilities.iterables (for permutations), numpy, math, itertools, pytrends.

The backend of the project consists of 2 main modules:

1) **ScreensAndMoviesToBestTT** takes all the movie and screen inputs and finally returns the most profitable timetable after numerous computations.
2) This uses real-time (normalized) interest score from google trends using the module **TrendsScore**.

ScreensAndMoviesToBestTT has 6 submodules.

- GenerateSlots
- TotalScore
- ClassifyScreensAndMovies
- TotalNoOfShows
- NewBruteTimeTables
- Maximisation

The algorithms of the submodules are as follows

ALGORITHM GenerateSlots(Screens)

Input: Screens – dictionary (Key – Screen Name(string), Value : List of 8 integers (Each element←1/0) First 4 elements indicating presence of 2D/Atmos/IMAX/3D features in the above order; Next 4 elements indicating presence of showtimes 8am/12pm/4pm/9pm.)

Output: FullSlot_Score – dictionary (key: slotname ex: 'Scr01_08:00am', value : integer score for the slot based on features of screen and timings)

```
1.  Screens_Name_Score←{}
2.  for Names in Screens:
            if Screens[Names][0:4]==[1,0,0,1]:
                    ScoreForName←10
            elif Screens[Names][0:4]==[1,0,1,0]:
                    ScoreForName←8
            elif Screens[Names][0:4]==[1,1,0,0]:
                    ScoreForName←6
            else:
                    ScoreForName←3
            END IF
    END FOR
3.  Screens_Name_Score[Names]←ScoreForName
```

```
4.  FullSlot_Score←{}
5.  for i in Screens:
            if Screens[i][4]==1:
                FullSlot_Score[i+'_'+'08:00am']←Screens_Name_Score[i] + 3
            END IF
            if Screens[i][5]==1:
                FullSlot_Score[i+'_'+'12:00pm']←Screens_Name_Score[i] + 7
            END IF
            if Screens[i][6]==1:
                FullSlot_Score[i+'_'+'04:00pm']←Screens_Name_Score[i] + 10
            END IF
            if Screens[i][7]==1:
                FullSlot_Score[i+'_'+'09:00pm']←Screens_Name_Score[i] + 7
            END IF
     END FOR
```

ALGORITHM TotalScore(NameOfMovie,RelDate,Lng,Bdg,Aud,IntScore,HighestBdg)

Input: NameOfMovie (string), RelDate (Release date-date), Lng (Language-string), Bdg (Budget - int), Aud (Audience-U or U/A or A -string), IntScore (Interest Score -float), HighestBdg (Highest budget observed - float)

Output: TotalScore (per movie – float)

```
1. NormBdg←(Bdg /HighestBdg)*30
2. Today←date.today()
3. Interest←0
4. if Lng == 'Tamil':
           Interest←Interest+10
   else if Lng == 'English':
           Interest←Interest+9
   else if Lng == 'Hindi':
           Interest←Interest+8
   else if Lng == 'Telugu':
           Interest←Interest+7
   else if Lng == 'Malayalam':
           Interest←Interest+6.5
   else:
           Interest←Interest+5
   END IF

5. RelDate←date(int(RelDate[0:4]), int(RelDate[5:7]), int(RelDate[8:10]))
```

```
6.  Difference←Today - RelDate

7.  if (Difference.days) < 7:
            Interest←Interest+10
    else if (Difference.days) < 14:
            Interest←Interest+7
    else if (Difference.days) < 21:
            Interest←Interest+5
    else:
            Interest←Interest+2
    END IF

8.  if (Aud=='U' or Aud=='U/A'):
            Interest←Interest+20
    else:
            Interest←Interest+10
    END IF

9.  Interest←Interest+IntScore
10. TotalScore←(Interest+NormBdg)/10
```

ALGORITHM ClassifyScreensAndMovies (Screens, FullSlot_Score, MovieVals, MovieNames)

Input: Screens – dictionary (key: name of Screen, Value – list of 8 integers – first 4 elements denoting features 2D/Atmos/IMAX/3D , next 4 elements denoting showtimes 8am/12pm/4pm/9pm),

MovieVals – dictionary (key : moviename, value : list of 4 integers denoting features required for the movie in the order 2D/Atmos/IMAX/3D)

MovieNames -List of string movienames

Output: Featurewise_Slot_Dict – dictionary (Keys – '2D', 'Atmos', 'IMAX', '3D'; Corresponding Values are: list of 2D slots, list of Atmos slots, List of IMAX slots, List of 3D slots respectively)

Featurewise_Movies_Dict – dictionary (Keys - '2D', 'Atmos', 'IMAX', '3D', Corresponding values are List of 2D movies, List of Atmos movies, List of IMAX movies, List of 3D movies respectively)

```
1. Featurewise_Slots_Dict←{'2D':[], 'Atmos':[], 'IMAX': [], '3D':[]
2. for SlotName in FullSlot_Score:
            FeatureOfScreen←Screens[SlotName[0:5]][0:4]
            if FeatureOfScreen==[1,0,0,1]:
                    Featurewise_Slots_Dict['3D'].append(SlotName)
            else if FeatureOfScreen==[1,0,1,0]:
                  Featurewise_Slots_Dict['IMAX'].append(SlotName)
            else if FeatureOfScreen==[1,1,0,0]:
                  Featurewise_Slots_Dict['Atmos'].append(SlotName)
            else:
                  Featurewise_Slots_Dict['2D'].append(SlotName)
            END IF
      END FOR

3.  Featurewise_Movies_Dict←{'2D': [], 'Atmos': [], 'IMAX': [], '3D': []}
4.  for MovieName in MovieVals:
            FeatureOfMovie←MovieVals[MovieName]
            if FeatureOfMovie==[1,0,0,1]:
                  Featurewise_Movies_Dict['3D'].append(MovieName)
            else if FeatureOfMovie==[1,0,1,0]:
                  Featurewise_Movies_Dict['IMAX'].append(MovieName)
            else if FeatureOfMovie==[1,1,0,0]:
                  Featurewise_Movies_Dict['Atmos'].append(MovieName)
            else:
                  Featurewise_Movies_Dict['2D'].append(MovieName)
            END IF
      END FOR
```

ALGORITHM TotalNoOfShows(MovieNames, Scores, TotalScreens)

Input: MovieNames (List of string movienames of a particular feature)
Scores (List of corresponding scores of movies)
TotalScreens (int – total number of screenings of a particular feature)

Output: MaxNoOfShows (List of the corresponding maximum number of shows allowed for each movie according to order of MovieNames)

```
1.  sum_of_scores←sum(Scores)
2.  ratio_factor←TotalScreens/sum_of_scores
3.  MaxNoOfShows←[]
4.  for j in [0, len(MovieNames)):
            MaxNoOfShows.append(round(Scores[j]*ratio_factor))
    END FOR
5.  if sum(MaxNoOfShows) > TotalScreens:
          Scores_sorted←sorted(Scores)
          for i in Scores_sorted:
            minimum←Scores.index(i)
            if MaxNoOfShows[minimum]!=0:
                MaxNoOfShows[minimum]←MaxNoOfShows[minimum]-1
                Break
            END FOR

    elif sum(MaxNoOfShows) < TotalScreens:
            maximum←Scores.index(max(Scores))
            MaxNoOfShows[maximum]←MaxNoOfShows[maximum]+1
    END IF
```

ALGORITHM NewBruteTimeTables(Slots, Movies, MaxNoOfShows)

Input : Slots (List of Slots of a particular feature-type), Movies (List of Movienames of a particular feature-type), MaxNoOfShows (List of the maximum number of shows for corresponding movies)
Output: TimeTables (List of Dictionaries-Each element of the list is a possible TimeTable;
For each dictionary - Keys: Slotnames, Values: MovieNames)

```
1. start = time.process_time()
2. List_freq = []
3. for i in range(len(Movies)):
           for j in range(MaxNoOfShows[i]):
                   List_freq.append(Movies[i])
           END FOR
     END FOR
4. List_freq_arr=np.array(List_freq)
5. ListOfLists = list(multiset_permutations(List_freq_arr))
6. TimeTables=[]
7. for OneTT in ListOfLists:
           d={}
           for j in range(len(Slots)):
                   d[Slots[j]] = OneTT[j]
           END FOR
           TimeTables.append(d)
     END FOR
```

ALGORITHM- Maximisation(TimeTables, ScreenScore, MovieScore):

Input: TimeTables (List of Dictionaries with each dictionary representing a timetable), ScreenScore (Dictionary: Key–slotname i.e. ScrName_Timing, Value: corresp Score of the slot), MovieScore (Dictionary: Key – Movie Name, Value – MovieScore)
Output: BestTimetable (Dictionary : Key – Slotname, Value: MovieName)

```
1. BestTimeTable←dict()
```

```
2.  BestScore←1
3.  for i in TimeTables:
            s←0
            for screening,movie in i.items():
                s←s+MovieScore[movie]*ScreenScore[screening]
            END FOR
            if s>BestScore:
                BestScore←s
                BestTimeTable←i
            END IF
    END FOR
```

Retrieving real-time data

Our project aims at providing an ethical solution to a problem faced by theaters and multiplexes. In order to keep the aim intact we do not use web-scraping which is considered an unethical practice, instead we have opted for using websites which have officially released their API for extracting demand scores.The usage of APIs ensure the safety and privacy of data for all parties involved.These practices make our project a completely legal solution to the above mentioned problem. This thus adds to the legitimacy of our program.

We generate (normalized) interest for each movie by taking real-time data from google trends using our **TrendsScore** module.

ALGORITHM TrendsScore(MovieNames)

Input: MovieNames (List of string movienames of a particular feature)

Output: Scores (Dictionary of movies and their scores)

```
1.  pytrend←TrendReq()
2.  Scores←{}
3.  for i in MovieNames:
            kw_list←[i]
            pytrend.build_payload(kw_list,timeframe,geolocation)
            df←pytrend.interest_over_time()
            for i in kw_list:
                Scores[i]←df[i].mean()
   END FOR
5.  return Scores
```

ALGORITHM OF THE PARENT MODULE ScreensAndMoviesToBestTT is as follows:

ALGORITHM ScreensAndMoviesToBestTT (Screens, Movies, IntScore)

Input:
Screens – dictionary (Key: name of Screen, Value – list of 8 integers – first 4 elements denoting features 2D/Atmos/IMAX/3D , next 4 elements denoting showtimes 8am/12pm/4pm/9pm)

Movies – dictionary (Key: moviename, Value – List whose elements are as follows:
List of 4 integers ex:[1,1,0,0] indicating features required for the movie in the order 2D/Atmos/IMAX/3D, String Release Date 'YYYY-MM-DD', String Language, Int Budget, String Audience Type ('U'/'U/A', 'A'))

IntScore – List (each element with integer interest score out of 30 corresponding to the order of Movies entered)

Output: BestTT (Dictionary – Key: SlotName, Value: MovieName)

```
1.  FullSlot_Score←{}
2.  FullSlot_Score ← GenerateSlots(Screens)
3.  MovieVals←{}
4.  for i in Movies:
            MovieVals[i] ← Movies[i][0]
        END FOR

5.  MovieNames ← list(Movies.keys())
6.  RelDate ← [Movies[i][1] for i in Movies]
7.  Lng ← [Movies[i][2] for i in Movies]
8.  Bdg ← [Movies[i][3] for i in Movies]
9.  Aud ← [Movies[i][4] for i in Movies]
10. HighestBdg ← max(Bdg)
        11. MovieScore←{}
12. for i in interval [0, len(MovieNames)):
                MovieScore[MovieNames[i]]←TotalScore(MovieNames[i],
                RelDate[i], Lng[i], Bdg[i], Aud[i], IntScore[i],
                HighestBdg)
        END FOR
13. Featurewise_Slots_Dict, Featurewise_Movies_Dict ← ClassifyScreensAndMovies(Screens,
    FullSlot_Score, MovieVals, MovieNames)

14. Featurewise_Moviewise_MovieScore_Dict←{'2D':[], 'Atmos': [], 'IMAX': [], '3D': []}

15. for i in MovieNames:
```

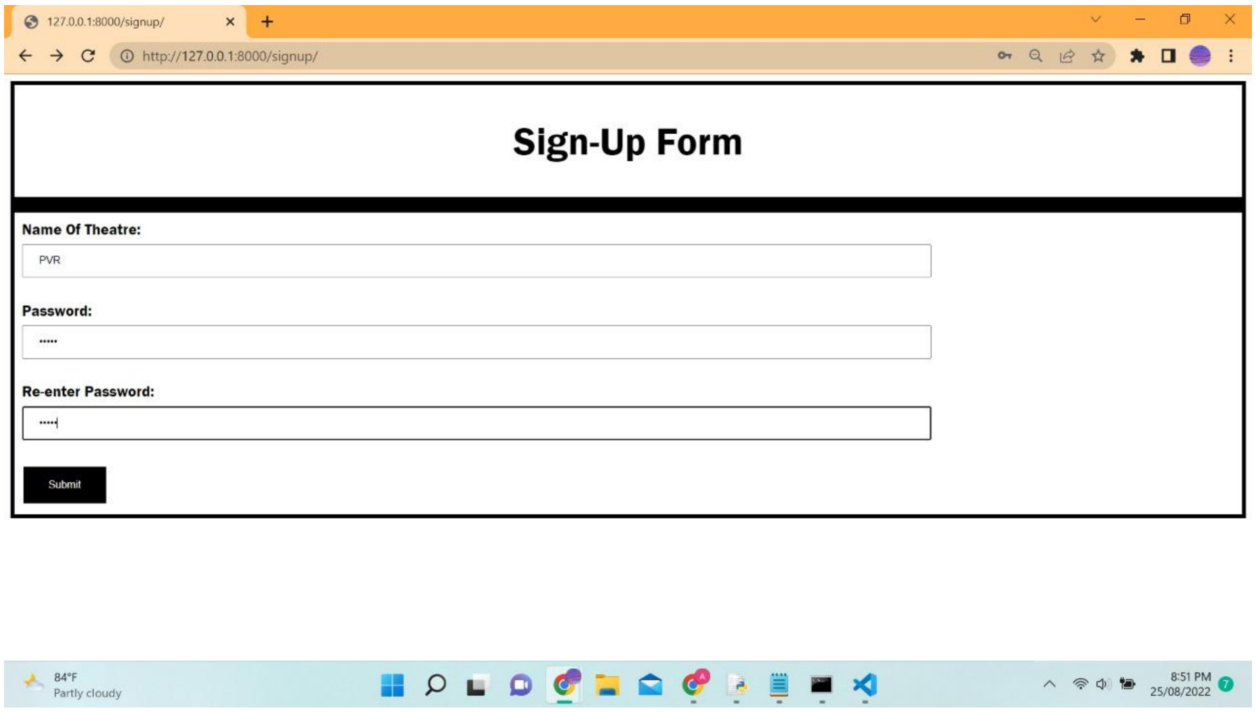
```
                if i in Featurewise_Movies_Dict['3D']:
                Featurewise_Moviewise_MovieScore_Dict['3D'].append(MovieScore[i])
                else if i in Featurewise_Movies_Dict['IMAX']:
                Featurewise_Moviewise_MovieScore_Dict['IMAX'].append(MovieScore[i])
                else if i in Featurewise_Movies_Dict['Atmos']:
                Featurewise_Moviewise_MovieScore_Dict['Atmos'].append(MovieScore[i])
                else:
                Featurewise_Moviewise_MovieScore_Dict['2D'].append(MovieScore[i])
                END IF
        END FOR
16. TotalScreens_2D ← len(Featurewise_Slots_Dict['2D'])
17.  TotalScreens_Atmos ← len(Featurewise_Slots_Dict['Atmos'])
18. TotalScreens_IMAX ← len(Featurewise_Slots_Dict['IMAX'])
19. TotalScreens_3D ← len(Featurewise_Slots_Dict['3D'])
20.  Featurewise_Moviewise_MaxNoOfShows ← {'2D':[], 'Atmos':[], 'IMAX':[], '3D': []}

21. if len(Featurewise_Movies_Dict['2D'])!←0:
                Featurewise_Moviewise_MaxNoOfShows['2D'] ←
                TotalNoOfShows(Featurewise_Movies_Dict['2D'],
                                        Featurewise_Moviewise_MovieScore_Dict['2D'],
                                        TotalScreens_2D)
        END IF

22. if len(Featurewise_Movies_Dict['Atmos'])!←0:
                Featurewise_Moviewise_MaxNoOfShows['Atmos'] ←
                TotalNoOfShows(Featurewise_Movies_Dict['Atmos'],
                                        Featurewise_Moviewise_MovieScore_Dict['Atmos'],
                                        TotalScreens_Atmos)
        END IF

23. if len(Featurewise_Movies_Dict['IMAX'])!←0:
                Featurewise_Moviewise_MaxNoOfShows['IMAX'] ←
                TotalNoOfShows(Featurewise_Movies_Dict['IMAX'],
                                        Featurewise_Moviewise_MovieScore_Dict['IMAX'],
                                        TotalScreens_IMAX)
        END IF

24. if len(Featurewise_Movies_Dict['3D'])!←0:
                Featurewise_Moviewise_MaxNoOfShows['3D'] ←
                TotalNoOfShows(Featurewise_Movies_Dict['3D'],
                                        Featurewise_Moviewise_MovieScore_Dict['3D'],
                                        TotalScreens_3D)
        END IF
```

```
25. TimeTables_2D= TimeTables_Atmos =TimeTables_IMAX =TimeTables_3D ←{}
26. BestTT_2D=BestTT_Atmos = BestTT_IMAX = BestTT_3D ← {}
27. BestTT←{}

28.    if len(Featurewise_Movies_Dict['2D'])!←0:
            TimeTables_2D ←
            NewBruteTimeTables(Featurewise_Slots_Dict['2D'],
                               Featurewise_Movies_Dict['2D'],
                               Featurewise_Moviewise_MaxNoOfShows['2D'])

            parameter2 ← {k:v for(k,v) in zip(
                      [i for i in Featurewise_Slots_Dict['2D']],
                      [FullSlot_Score[i] for i in Featurewise_Slots_Dict['2D'}]
                      )}

            parameter3 ← {k:v for(k,v) in zip(
                      [i for i in Featurewise_Movies_Dict['2D']],
                      [MovieScore[i] for i in Featurewise_Movies_Dict['2D']]
                      )}

            BestTT_2D ← Maximisation(TimeTables_2D, parameter2, parameter3)
        END IF

29. if len(Featurewise_Movies_Dict['Atmos'])!←0:
            TimeTables_Atmos ←
            NewBruteTimeTables(Featurewise_Slots_Dict['Atmos'],
                               Featurewise_Movies_Dict['Atmos'],
                               Featurewise_Moviewise_MaxNoOfShows['Atmos'])

            parameter2 ← {k:v for(k,v) in zip(
                      [i for i in Featurewise_Slots_Dict['Atmos']],
                      [FullSlot_Score[i] for i in Featurewise_Slots_Dict['Atmos']]
                      )}

            parameter3 ← {k:v for(k,v) in zip(
                      [i for i in Featurewise_Movies_Dict['Atmos']],
                      [MovieScore[i] for i in Featurewise_Movies_Dict['Atmos']]
                      )}


            BestTT_Atmos ←Maximisation(TimeTables_Atmos, parameter2, parameter3)
        END IF
```

```
30. if len(Featurewise_Movies_Dict['IMAX'])!←0:
            TimeTables_IMAX ←
            NewBruteTimeTables(Featurewise_Slots_Dict['IMAX'],
                               Featurewise_Movies_Dict['IMAX'],
                               Featurewise_Moviewise_MaxNoOfShows['IMAX'])

            parameter2 ← {k:v for(k,v) in zip(
                      [i for i in Featurewise_Slots_Dict['IMAX']],
                      [FullSlot_Score[i] for i in Featurewise_Slots_Dict['IMAX']]
                      )}
            parameter3 ← {k:v for(k,v) in zip(
                      [i for i in Featurewise_Movies_Dict['IMAX']],
                      [MovieScore[i] for i in Featurewise_Movies_Dict['IMAX']]
                      )}

            BestTT_IMAX ←Maximisation(TimeTables_IMAX, parameter2, parameter3)
     END IF

31. if len(Featurewise_Movies_Dict['3D'])!←0:
            TimeTables_3D ←
            NewBruteTimeTables(Featurewise_Slots_Dict['3D'],
                               Featurewise_Movies_Dict['3D'],
                               Featurewise_Moviewise_MaxNoOfShows['3D'])

             parameter2 ← {k:v for(k,v) in zip(
                      [i for i in Featurewise_Slots_Dict['3D']],
                      [FullSlot_Score[i] for i in Featurewise_Slots_Dict['3D']]
                      )}

            parameter3 ← {k:v for(k,v) in zip(
                      [i for i in Featurewise_Movies_Dict['3D']],
                      [MovieScore[i] for i in Featurewise_Movies_Dict['3D']]
                      )}
            BestTT_3D ← Maximisation(TimeTables_3D, parameter2, parameter3)
     END IF

32. BestTT.update(BestTT_2D)
33. BestTT.update(BestTT_Atmos)
34. BestTT.update(BestTT_IMAX)
35. BestTT.update(BestTT_3D)
```

IMPLEMENTATION AND RESULTS

Signup page (For New User)



Sign-Up Form

Name Of Theatre:

PVR

Password:

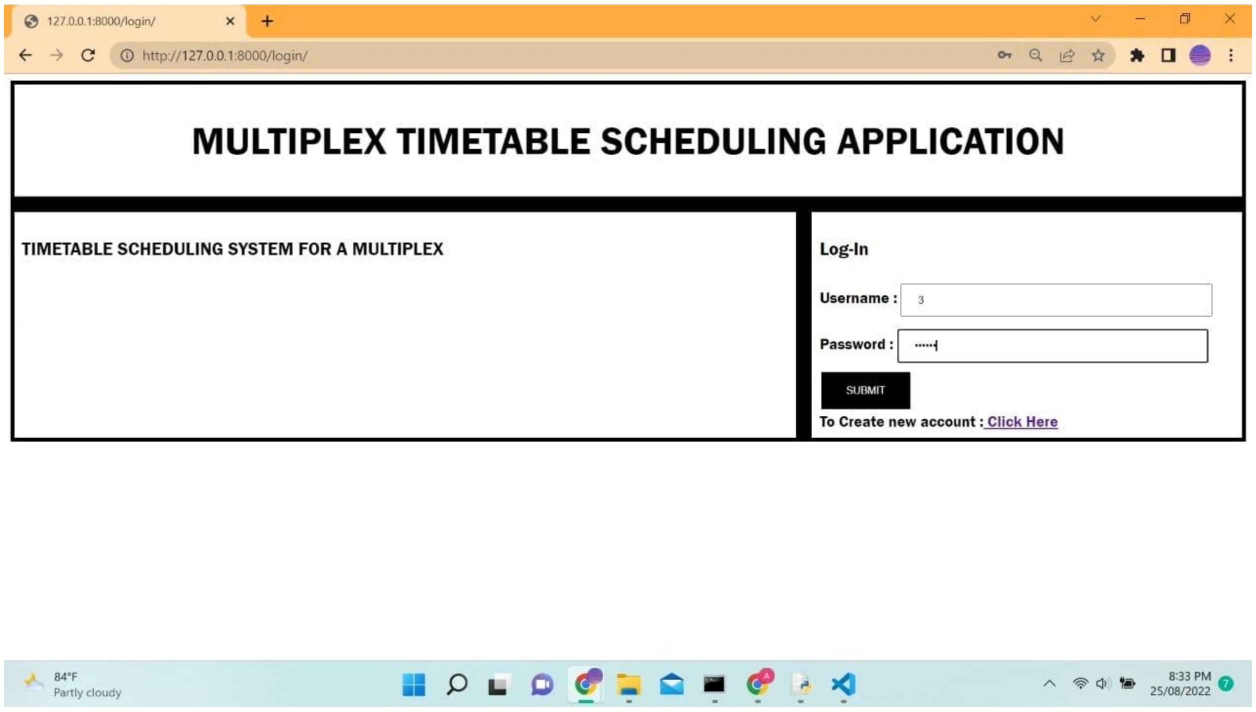
•••••

Re-enter Password:

•••••

Submit

Login page



MULTIPLEX TIMETABLE SCHEDULING APPLICATION

TIMETABLE SCHEDULING SYSTEM FOR A MULTIPLEX

Log-In

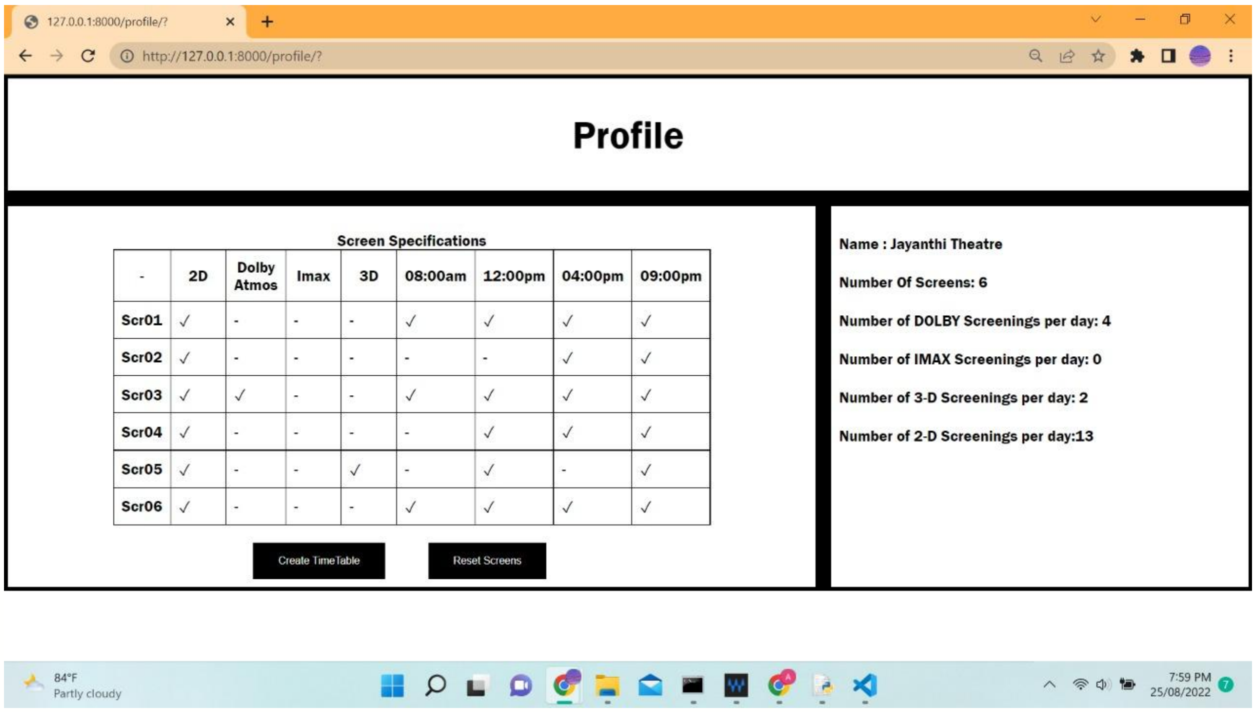
Username : 3

Password : •••••

SUBMIT

To Create new account : Click Here

Screen Profile Page



Profile

Screen Specifications

-	2D	Dolby Atmos	Imax	3D	08:00am	12:00pm	04:00pm	09:00pm
Scr01	✓	-	-	-	✓	✓	✓	✓
Scr02	✓	-	-	-	-	-	✓	✓
Scr03	✓	✓	-	-	✓	✓	✓	✓
Scr04	✓	-	-	-	-	✓	✓	✓
Scr05	✓	-	-	✓	-	✓	-	✓
Scr06	✓	-	-	-	✓	✓	✓	✓

Name : Jayanthi Theatre

Number Of Screens: 6

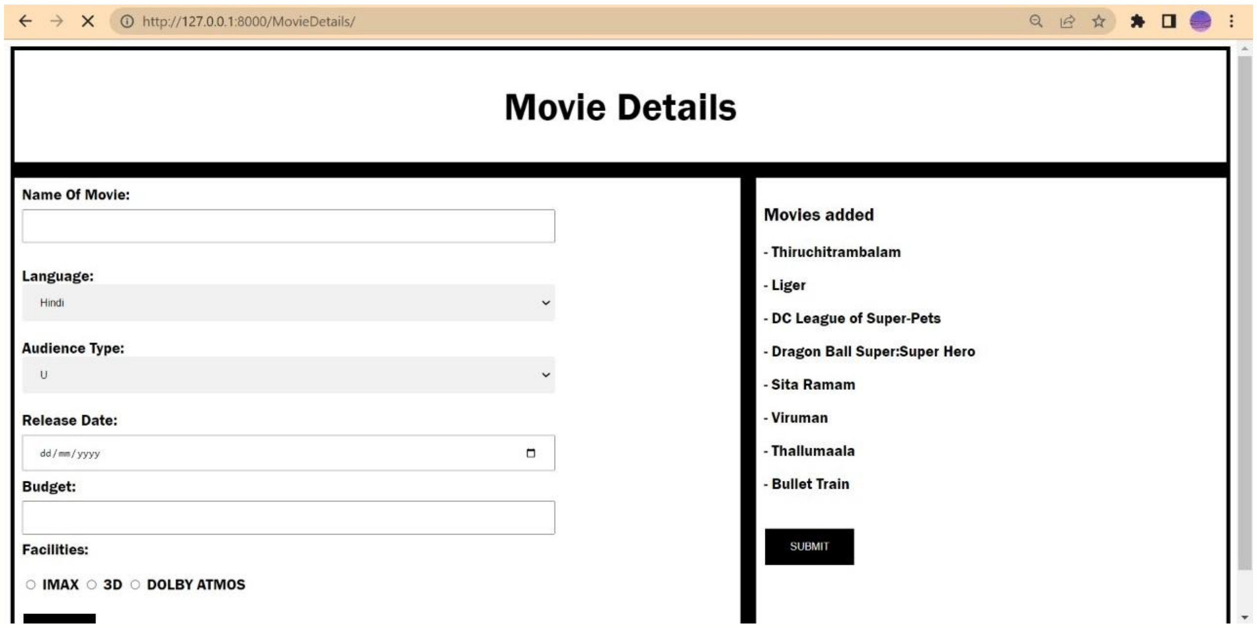
Number of DOLBY Screenings per day: 4

Number of IMAX Screenings per day: 0

Number of 3-D Screenings per day: 2

Number of 2-D Screenings per day:13

Movie input page



Movie Details

Name Of Movie:

Language: Hindi

Audience Type: U

Release Date: dd/mm/yyyy

Budget:

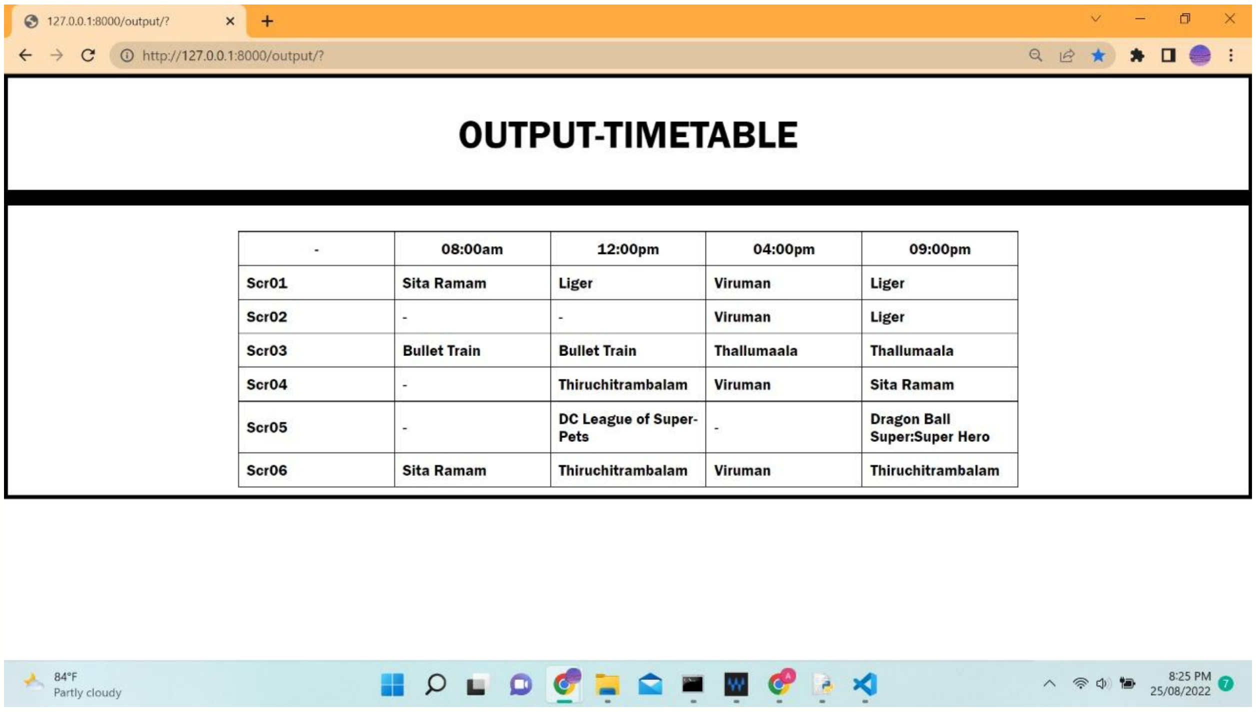
Facilities:

○ IMAX ○ 3D ○ DOLBY ATMOS

Movies added

- Thiruchitrambalam
- Liger
- DC League of Super-Pets
- Dragon Ball Super:Super Hero
- Sita Ramam
- Viruman
- Thallumaala
- Bullet Train

Output TimeTable page



OUTPUT-TIMETABLE

-	08:00am	12:00pm	04:00pm	09:00pm
Scr01	Sita Ramam	Liger	Viruman	Liger
Scr02	-	-	Viruman	Liger
Scr03	Bullet Train	Bullet Train	Thallumaala	Thallumaala
Scr04	-	Thiruchitrambalam	Viruman	Sita Ramam
Scr05	-	DC League of Super-Pets	-	Dragon Ball Super:Super Hero
Scr06	Sita Ramam	Thiruchitrambalam	Viruman	Thiruchitrambalam

CONCLUSION

The team's solution is based on real-life scenarios and constraints, built as a result of interviews conducted with real theater and multiplex managers to actually understand the needs of the exhibitors.This project offers a way to decrease budget invested by multiplex owners for human labor.This will in turn increase profit margins for the owners and managers.

It is highly compatible and can be used for all theaters without much intervention in the algorithm. An effective timetable would greatly reduce wastage of resources and help schedule the exact number of shows required. Expenditures starting from cleaning the theater after a show, to the air condition requirements would be made efficient when the crowd is maximized for a show. It also reduces manual labor involved in formulating the timetable.

This program is efficient in terms of space and time , thus will work on any basic computer system . The theater need not invest in any complex and expensive machines to run our program.

The team had a very insightful learning experience through the course of the project and came across a lot of new functions and modules that could be deployed for our needs. The team is confident that the application provides an efficient solution to the problem at hand.

REFERENCES

http://www.patatconference.org/patat2012/proceedings/3_17.pdf
https://egrove.olemiss.edu/cgi/viewcontent.cgi?article=1442&context=etd
https://www.qubecinema.com/#Products
https://pro.sony/en_TH/products/digital-cinema-software/stm-100l#ProductFeaturesBlock-stm-100l
https://stackoverflow.com/questions/2177836/algorithm-for-creating-a-school-timetable
https://d-nb.info/967478146/34
https://www.comscore.com/Products/Movies/Theater-Management-System
https://www.gdc-tech.com/cinema-solutions/cinema-enterprise-software/theatre-management-system-tms-2000/
https://yourstory.com/2019/03/chennai-startup-cinicloud-zoho-movie-theatre-856loz1q79/amp
https://www.artsalliancemedia.com/products/theatre-management-system
https://www.cinicloud.com/
https://csit.am/2019/proceedings/AAL/AAL1.pdf
Django Tutorials by Corey Schaffer

Special thanks to the theater managers of S2 Theyagraja, Thiruvanmiyur - Mr. Sunderrajan and Ganapathiram Theater, Adyar - Mr. Mathew Sebastian for their valuable inputs on the challenges faced in creating a timetable for their theatres.