ACKNOWLEDGEMENT

We would like to thank our Lab Teachers, Mrs. Rashma B. M. and Mrs. Suchitra Charles for their aid and evaluation over the course of completion of this project. We would also like to thank our University, Visvesvaraya Technological University for the opportunity to make this project which has given us the chance to learn about these topics. Additionally, we would like to thank our college, PES Institute of Technology for allowing us to use the college resources. Finally, I would like to thank our friends and family for their ceaseless and tireless support throughout this project.

INDEX

ABSTRACT

To define, design, and implement a Database to aid in comprehensively managing all aspects of a theatre. This system deals with a single branch of a theatre. It keeps track of the various halls, currently showing movies, show timings, booked tickets, and price listings in the theatre as well as the associated attributes. It is possible for the management of the theatre to use the database to fully understand all parts of the system without use of additional software. The system allows a cashier to add tickets for a given show, and for a manager to add movies and shows to the database. The system automatically checks the validity of the new entries and disallows incorrect data from being entered. This is done automatically without user intervention.

LIST OF DIAGRAMS

Chapter 1: Introduction

1.1. Introduction

This project is a Theatre Management System. It is a website made in HTML connected to a MySQL Database using Python with JavaScript and CSS enhancements. The database has tables dealing with all aspects of a theatre: halls, movies available, current showing, price lists, and booked tickets. These are accessed by the webpage to show relevant details for a user. Thus, by use of this project, the core aspects of a theatre are easily maintained and manipulated as needed by the manager.

1.1.1. Purpose of the project

This project is aimed at small theatres. It is a tool to be used to simplify the process of adding, storing, viewing and manipulating the movies available at the theatre, the various showings of the movie, the tickets booked for the shows and the pricing of the tickets. It is a robust and comprehensive system that is minimalistic and simple for use by cashiers and managers who have no prior training.

1.1.2. Scope

The scope of this project is at the ticket booking kiosk of the cashier and at the manager's desk. It is only concerned with active use of resources of the theatre and does not have long term storage of information. It only deals with movies, movie showings and booked tickets, and does not handle other aspects, such as online booking, concessions, etc.

1.1.3. Definitions, acronyms and abbreviations

HTML - HyperText Markup Language
CSS - Cascading Style Sheets
AJAX - Asynchronous JavaScript And XML
XML - Extensible Markup Language
SQL - Structured Query Language
HTTP - HyperText Transfer Protocol
PIP - Python Package Manager
Distro - Distribution

1.2. Existing System

The currently used system in theatre management is highly primitive and/or inefficient and/or slow. It is, in some cases, worked on in ledgers and with a manual booking system. Even in cases where it is on computers instead of a manual system, it is not new or often upgraded/maintained as required to make it work properly. Thus, our system is brought in to replace it.

1.3. Proposed system

The proposed system is a fully functional system that takes care of all aspects related to movies on-site. It allows cashiers to book tickets and the manager to add movies and shows. The system is designed so as to be used by anyone with a basic understanding of english and no other training at all.

1.4. Statement of the Problem

Small theatres generally have old or primitive management systems which they do not replace or upgrade as it is perceived to be not worth the cost compared to revenues. However, old systems can be very slow and inefficient and as such, slow down the process. Furthermore, they are difficult to maintain and the cost of doing so can build up over time and eventually exceed costs of not replacing the system. This Project is designed for use by such cinemas, who can use this highly minimalistic design to replace current systems at low cost and with no additional training requirements. The code behind the project is designed to put maximum functionality within it so as reduce cost and difficulty of maintenance and future upgrades.

Chapter 2: Software Requirements Specification

2.1: Software Requirements Specification

The cinema management software is to operate with a client-server architecture, and as such, must have minimum hardware and software to run the server/browser along with all its dependencies. The software is used by cashiers who handle the ticket booking at a ticket counter using a computer with a JavaScript compatible browser.

The software is also used by the manager of the cinema who manipulates entries in the centralised database using a computer at his/her desk, with a JavaScript compatible browser.

The server software runs in a dedicated centralised server hosting center for the cinema franchise. The scripts and http server run on the server, and require a Python interpreter, along with the dependencies for the scripts, as well as the MySQL server.

2.2: Operating Environment

The production ready software is meant to run on a variety of verified hardware and software. As such, many of the required dependencies are available cross platform, both for the front end as well as the backend.

Some of the verified software and hardware are specified below, along with software and hardware that are supposed to be compatible.

2.2.1: Hardware Requirements

The computer can be any x86, x86/64, ARMv7 or higher system that is compatible with either a Linux distribution, Windows, or MacOS for both the front end and the back end. The computer should have a suitable networking interface, sufficient RAM to run the JavaScript enabled browser and/or the Python interpreter.

A network must be set up between the client machines and the host server. The network must be able to handle AJAX and HTTP requests, along with MySQL server connection requests.

The MySQL server may be run on the same machine as the Python interpreter, or a separate machine if required.

2.2.2: Software Requirements

Any operating system with a JavaScript enabled browser / Python Interpreter along with Python's package manager, PIP / MySQL server.

PIP is used to install MySQL Connector for Python, as well as Flask, both of which are dependencies.

pip install mysql-connector
pip install flask

MySQL server is available for for most Linux based distros, as well as Windows and MacOS.
Any JavaScript enabled browser available for your platform will suffice for the interface. The system must have Python 3.6 along with Flask and mysql-connector installed via PIP. A database named 'db_theatre' must be created in the MySQL server. The software comes bundled with an 'initialise.sql' file, which must be executed within MySQL server.

source /path/to/CineManagerDB/initialise.sql

MySQL server must have an account with username 'root' and password 'root123' (changeable within the runQuery function in app.py). This account needs read/write access to all tables in the db_theatre database (or root privileges).

2.3. Functional Requirements

The functional requirements of this project are:

- The project allows login of two entities, cashier and manager:
- The manager has a set of functionalities described as:
    - The ability to insert new Movies. Movie name, language, length, showing start date and end date are inputted. In case of incorrect or invalid input, entry is nullified. If correctly entered, then a movie ID is randomly generated and the data is passed to MySQL via JavaScript, Flask and Python.
    - Similarly, Shows can be entered by entering movie, hall, time and date. If correctly entered, a show ID is randomly generated and ask data is put into database. Then, the price ID is fetched and stored after. SHows cannot be added if an invalid type is chosen or if the hall in which it is to be shown is already occupied.
    - The manager can view tickets that have been booked by the cashier, such as tickets numbers and seat numbers for a given show that have been booked.
    - The manager can alter prices. The list of prices based on day and your is brought up. Then, a price is then selected and altered.

- The cashier had one functionality. The cashier books tickets for the customer. The cashier selects a movie, and based on that, a showing is selected for a time and date. Then a seating chart is brought up and a seat is chosen. Then a unique ticket ID is generated and the data is stored in the database.
- All data is stored on a common database. All systems in the theatre (cashier's terminals and manager's computer) access the same database for their bookings. Multiple branches of the same theatre may share the same database, but different theatres use different database.

2.4. Non-functional requirements

The non-functional requirements of this projects are:

- Security : The website does not allow access to any functionality by directly jumping to any particular link to that function's page. Additionally, anything that is needed to be done can only be done by first logging in. There are multiple accounts, with login by cashiers only allowing cashier functionalities and manager allowing only manager functionalities.
- Data Integrity : The project does not allow entry of data in case data is invalid. This is very important as if invalid data is added, then it can cause large problems, such as getting tickets for shows which are invalid, creating shows for invalid or incorrect movies/halls, or inserting invalid movies, each of which can have cascading effects.
- Automatic data processing : a lot of information is processed by the project instead of relying on the user to add perfect information and perform numerous functions each time. Examples include deleting old shows and movies, retrieving prices, validating inserted information in movies/shows etc. This is an important task as it can be performed much more efficiently and quickly by the system than by a human.

2.5. User characteristics

The user characteristics are:

- The first user is the manager, who holds administrative powers in the context of the Project, which include:
    - The ability to insert new Movies
    - The ability to add new Showings of available Movies
    - The ability to view Booked Tickets of various Showings
    - The ability to alter prices of various Shows
- The second user is the cashier, who works at the Ticket Booking Kiosk, and takes input of the customer to book tickets for them. He has only one ability, which is:
    - The ability to book tickets for a customer.

2.6. Applications of Theatre Management System

Applications of the theatre management system are for the manager and the cashier which means that there are three primary sets of functionalities provided by the system:

- It can be used to comprehensively aid the manager's movie related duties:
    - To add new movies to the System, all details about the movies must be added, such as name, length, language, etc. A new unique movie ID is randomly generated and assigned to the movie for future reference.
    - To add new Showings for the available movies. Details about the movie showing must be added, such as which movie is playing, which hall it is playing in, etc. A show ID is randomly generated and assigned to the show, and a price ID referring to the pricing for tickets for the show is also assigned to the show.
    - Tickets booked by the cashier are available for review
    - Price Lists for made available for viewing and editing by the manager. Previously booked tickets are not affected, but any tickets booked in the future will have new ticket prices.
- It is used by the cashier to book tickets for the customer.
    - Date and time must be selected. Then a list of available movies is shown. Once a movie is selected, then available halls are shown. Once all these details are filled, ticket type (gold or standard) is selected and price is shown.
- Additionally, in the background,
    - If any details are incorrectly entered, then system rejects entry of item.
    - Whenever login is done successfully, old shows and movies whose show period has ended are automatically deleted.
    - IDs and assigning prices are done in the background.
- All data is stored on a common database. All systems in the theatre (cashier's terminals and manager's computer) access the same database for their bookings. Multiple branches of the same theatre may share the same database, but different theatres use different database. Thus, superiors in a chain cinema may access all data of their various theatres.

2.7. Advantages of Theatre Management System

The advantages of the Theatre Management System are:

- It is based on a Client-Server System, meaning multiple cashiers and managers can be supported.
- It operates on a minimalistic User Interface so that any user of the system can do what they need to do with almost no training and extreme ease.
- Erroneous data is not entered into the system and rendered invalid

- Old shows and movies are removed automatically without user intervention, meaning old data does not clog up the system.
- All IDs are attached to items automatically after being randomly generated, so as to prevent mismatches and they operate solely on the back-end, meaning that users only need to be concerned with human-useable data.

2.8. Summary

We have developed a highly comprehensive and easy to use system for any small Theatre. It is easy to implement and requires no training to use. It provides options for managers and cashiers. It is error-proof and does large amount of work in the background. Thus, the system aids to simplify the processes used by cashier and manager as well as reduce operational costs, the primary concerns of all businesses.

3. Chapter 3: Design

3.1. ER diagram

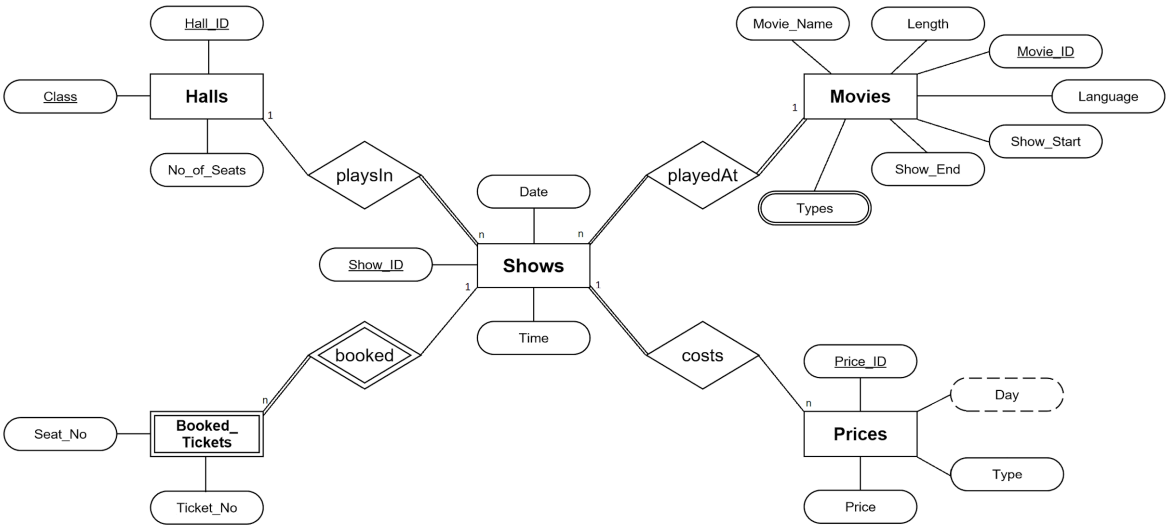

Diagram 3.1.D1: Entity Relation Diagram

3.2. Schema diagram

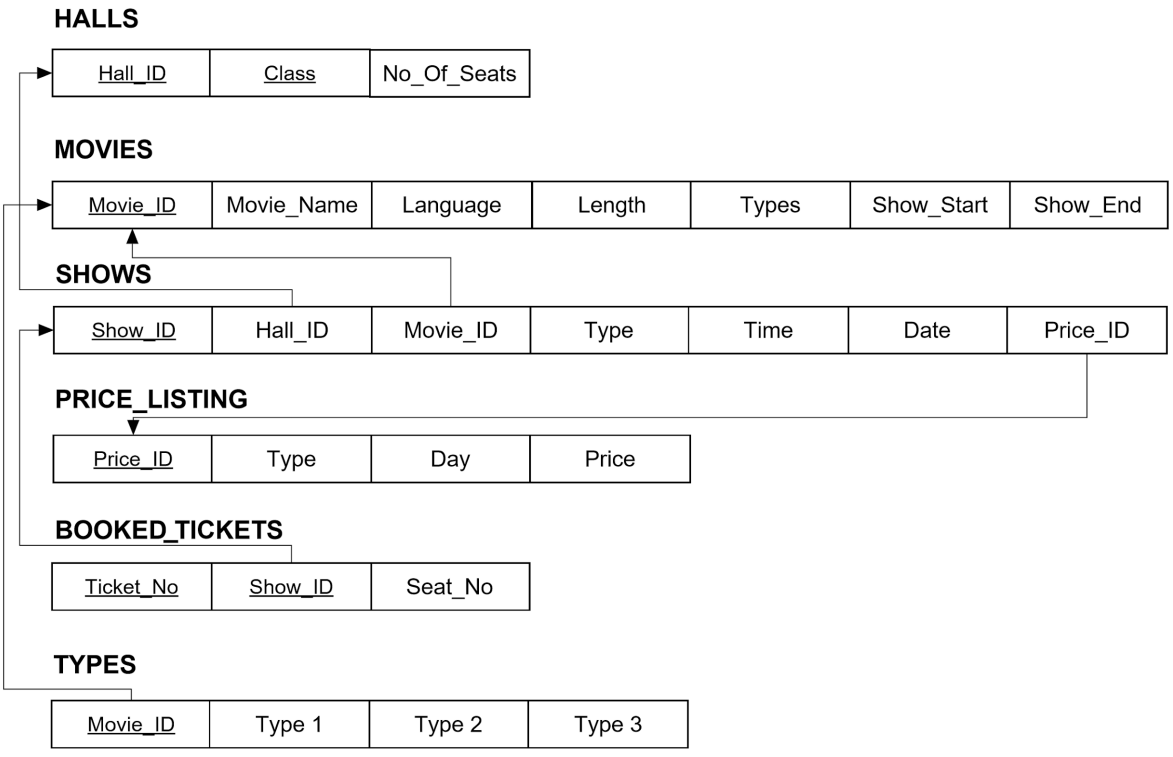

Diagram 3.2.D1: Schema Diagram

3.3. Relations in Third Normal Form (3NF)

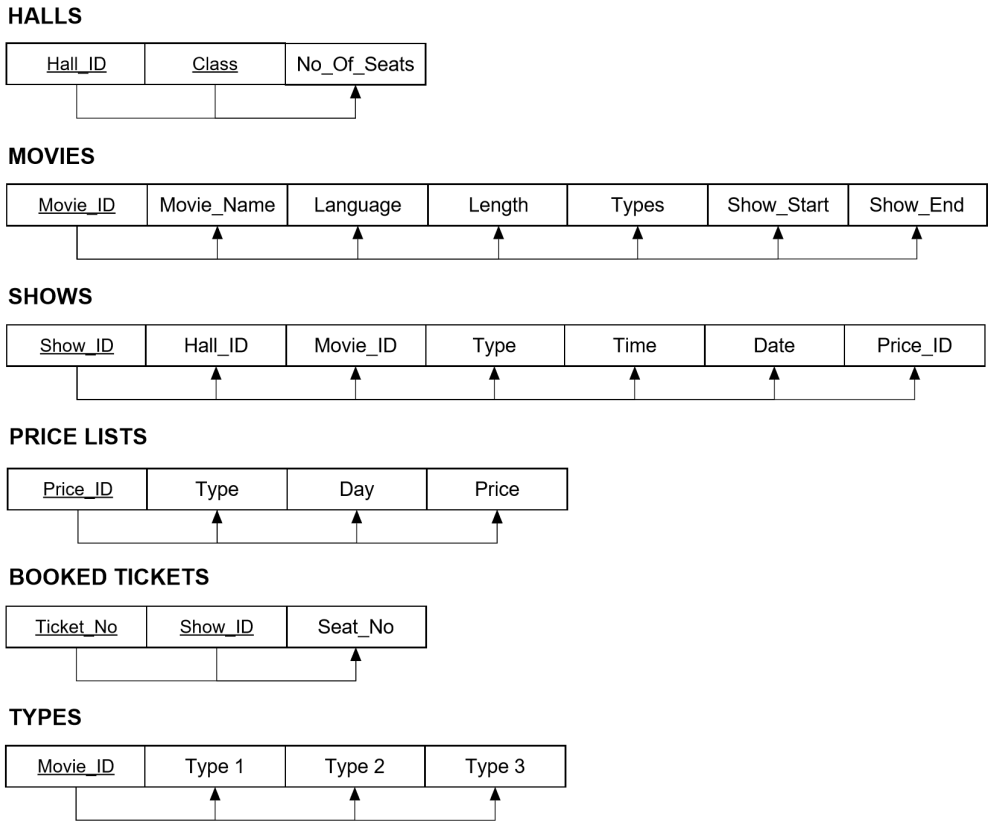

Diagram 3.3.D1: Functional Dependencies showing 3NF

HALLS

- FD: Hall_ID,Class → No_Of_Seats

MOVIES

- FD: Movie_ID → Movie_Name,Language,Length,Types,Show_Start,Show_End

SHOWS

- FD: Show_ID → Hall_ID,Movie_ID,Type,Time,Date,Price_ID

PRICE LISTS

- FD: Price_ID → Type,Day,Price
- FD: Type,Day → Price

  (Type,Day forms a Super Key, hence it does not violate Third Normal Form)

BOOKED TICKETS

- FD: Ticket_No,Show_ID → Seat_No

TYPES

- FD: Movie_ID → Type_1, Type_2, Type_3

3.4. Data Flow Diagrams

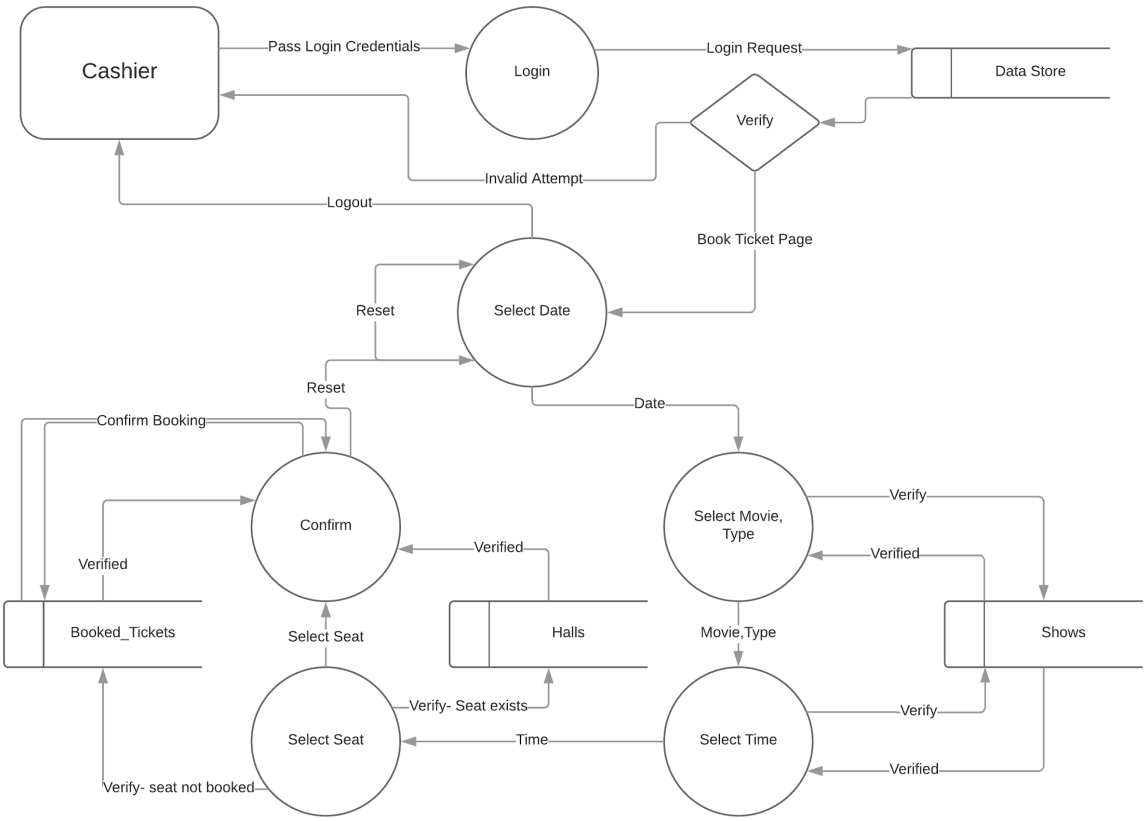

Diagram 3.4.D1: Data flow Diagram for Cashier

4 Chapter 4: Implementation

4.1. Implementation

The project is implemented in HTML, JavaScript and CSS for Front-End (website) and Python, Flask and MySQL for Back-End (database). It has a three-tier architecture with Front-End forming Application Layer and Back-End forming Middle Layer and Database. Input validation is done in JavaScript. JavaScript communicates with the Python server using the Flask framework over AJAX requests. The server responds with HTML code that is then displayed on the Front-End. The server communicates with MySQL using mysql-connector.

The external libraries used include Bootstrap for styling, and jQuery for AJAX requests and general JavaScript functionality.

4.2. Programming Language Selection

Python 3.6 was used to communicate with MySQL and to act as the server for the various clients. This decision was made in part due to Python's simplicity, dynamic typing and various robust data structures available which are built in. Python also has the ability to integrate with Flask, which acts as a framework and a router for the application. This method was used as opposed to using PHP as use of PHP in the industry has slowly been declining in favor of competing web servers such as Node and Django.

HTML 5 and CSS 3 were used to create the front end UI. JavaScript is used to fetch responses from the server as well as for validation of inputs.

4.3. Query for Creation of Tables

- create table halls (hall_id int, class varchar(10), no_of_seats int, primary key(hall_id,class));

- create table movies (movie_id int primary key, movie_name varchar(40), length int, language varchar(10), show_start date, show_end date);

- create table price_listing (price_id int primary key, type varchar(3), day varchar(10), price int);

- create table shows (show_id int primary key, movie_id int, hall_id int, type varchar(3), time int, Date date, price_id int, foreign key(movie_id) references movies(movie_id), foreign key(hall_id) references halls(hall_id), foreign key(price_id) references price_listing(price_id) on update cascade);

- create table booked_tickets (ticket_no int, show_id int, seat_no int, primary key(ticket_no,show_id), foreign key(show_id) references shows(show_id) on delete cascade);

- create table types(movie_id int primary key, type1 varchar(3), type2 varchar(3), type3 varchar(3), foreign key(movie_id) references movies(movie_id) on delete cascade);

4.4. Description of tables

The tables used in this project are:

- **Halls**: This table is used to hold the information about the halls in the theatre. The entry of data to this table can only be done by admin, and is not done by manager or cashier.
- **Movies**: This table holds any movies currently playing in the theatre. It also holds the length of the movie, which is used for scheduling, and the showings start and end dates, out of whose range movies cannot be shown.
- **Shows**: This table holds the individual showings of the movie. Price_ID is used to refer to the price listings, which holds the price of the show considering type and day. Each show is also of a specific type.
- **Price_Listing**: This table holds a list of prices, which are defined by Day of the showing and Type of the movie, such that 4DX costs more than 3D which costs more than 2D and weekdays cost less than weekends including Friday.
- **Booked_Tickets**: This table holds the tickets booked by cashier on behalf of customer. It has ID of the showing as well the seat number, which starts from 1 in case of Standard seats and from 1001 in case of Gold seats.
- **Types**: This table holds the showing types available for any movie. These types are 2D, 3D, and 4DX.

4.5. Triggers used

The trigger used in this project is:

```
delimiter //

create trigger get_price`
after insert on halls
for each row
begin

UPDATE shows s, price_listing p
        SET s.price_id=p.price_id
        WHERE p.price_id IN
```

```
        (SELECT price_id
        FROM price_listing p
        WHERE dayname(s.Date)=p.day AND s.type=p.type);

end; //

delimiter ;
```

The purpose of this trigger is to dynamically assign the appropriate price listing's ID to the newly inserted show. As trigger on a particular table cannot affect that table, instead the front-end automatically adds and deletes a dummy entry from the halls table, which invokes the trigger.

4.6. Procedure used

The procedure used in this project is:

```
delimiter //

create procedure delete_old()
begin

        declare curdate date;
        set curdate=curdate();

        DELETE FROM shows
                WHERE datediff(Date,curdate)<0;

        DELETE FROM shows
                WHERE movie_id IN
                (SELECT movie_id
                        FROM movies
                        WHERE datediff(show_end,curdate)<0);

        DELETE FROM movies
                WHERE datediff(show_end,curdate)<0;

end; //

delimiter ;
```

This procedure is used to delete old shows and movies whose show date is before the current date. This done using the inbuilt datediff(date1,date2) procedure which returns a negative

value if the second date is after the first, and curdate() which returns the current date. This procedure is called every time a login happens.

4.7 Important Code Snippets

app.py app.js seating.html

```python
@app.route('/getHallsAvailable', methods = ['POST'])
def getHalls():
    movieID = request.form['movieID']
    showDate = request.form['showDate']
    showTime = request.form['showTime']

    res = runQuery("SELECT length FROM movies WHERE movie_id = "+movieID)

    movieLen = res[0][0]
    showTime = int(showTime)
    showTime = int(showTime / 100)*60 + (showTime % 100)
    endTime = showTime + movieLen

    res = runQuery("SELECT hall_id, length, time FROM shows NATURAL JOIN movies WHERE Date = '"+showDate+"'")
    unavailableHalls = set()

    for i in res:
        x = int(i[2] / 100)*60 + (i[2] % 100)

        y = x + i[1]

        if x >= showTime and x <= endTime:
            unavailableHalls = unavailableHalls.union({i[0]})

        if y >= showTime and y <= endTime:
            unavailableHalls = unavailableHalls.union({i[0]})

    res = runQuery("SELECT DISTINCT hall_id FROM halls")
    availableHalls = set()

    for i in res:
        availableHalls = availableHalls.union({i[0]})

    availableHalls = availableHalls.difference(unavailableHalls)

    if availableHalls == set():
        return '<h5>No Halls Available On Given Date And Time</h5>'

    return render_template('availablehalls.html', halls = availableHalls)
```

Diagram 4.4.D1: Python routed function to retrieve available halls for show scheduling

app.py | app.js | seating.html

```python
def runQuery(query):
    try:
        db = mysql.connector.connect(
            host='localhost',
            database='db_theatre',
            user='root',
            password='root123')

        if db.is_connected():
            cursor = db.cursor(buffered = True)
            cursor.execute(query)
            db.commit()
            return cursor.fetchall()

    except Error as e:
        #Some error occured
        return e.args[1]

    finally:
        db.close()

    #Couldn't connect to MySQL
    return None


if __name__ == "__main__":
    app.run(host='0.0.0.0')
```

Diagram 4.4.D2: Python function for connecting to MySQL and running a query

app.js | seating.html

```javascript
function filledMovieForm(){
    availTypes = $('[name="movieTypes"]')[0].value.toUpperCase().trim();
    movieName = $('[name="movieName"]')[0].value;
    movieLang = $('[name="movieLang"]')[0].value;
    movieLen = $('[name="movieLen"]')[0].value;
    types = ($('[name="movieTypes"]')[0].value.toUpperCase().trim()).split(' ');
    atleastTypes = ['2D', '3D', '4DX'];
    allTypes = [undefined].concat(atleastTypes);
    if($('#datepicker-manager-2')[0].value == '' || $('#datepicker-manager-3')[0].value == '' ||
    movieName == '' || movieLang == '' || movieLen == '' || $('[name="movieTypes"]')[0].value == '')
        $('#manager-dynamic-2').html('<h5>Please Fill In All Fields</h5>');
    else if(!( atleastTypes.includes(types[0]) && allTypes.includes(types[1]) && allTypes.includes(types[2])) )
        $('#manager-dynamic-2').html('<h5>Invalid Format For Movie Types</h5>');
    else if(! $.isNumeric(movieLen))
        $('#manager-dynamic-2').html('<h5>Movie Length Needs To Be A Number</h5>');
    else if(Date.parse(startShowing) > Date.parse(endShowing))
        $('#manager-dynamic-2').html("<h5>Premiere Date Must Be Before/On Last Date In Theatres</h5>");
    else{
        movieLen = parseInt(movieLen, 10);
        $.ajax({
            type: 'POST',
            url: '/insertMovie',
            data: {
                'movieName' : movieName,
                'movieLen' : movieLen,
                'movieLang' : movieLang,
                'types' : availTypes,
                'startShowing' : startShowing,
                'endShowing' : endShowing
            },
            success: function(response){
                $('#manager-dynamic-2').html(response);
            }
        });
    }
}
```

Diagram 4.4.D3: JavaScript function for form checking of movie insertion input

app.js

```
function createShow(){
    $('#options button').prop('disabled', true);
    $('#manager-dynamic-1').html('<input id="datepicker-manager-3" placeholder="Pick a date">+'
        '<input id="timepicker-manager-1" placeholder="Pick a time"><button onclick="getValidMovies()">Submit</button>');
    $('#datepicker-manager-3').pickadate({
                formatSubmit: 'yyyy/mm/dd',
                hiddenName: true,
                min: new Date(),
                onSet: function( event ) {
                    if ( event.select ) {
                        showDate = this.get('select', 'yyyy/mm/dd' );
                    }
                }
    });
    $('#timepicker-manager-1').pickatime({
                formatSubmit: 'HHi',
                hiddenName: true,
                interval: 15,
                min: new Date(2000,1,1,8),
                max: new Date(2000,1,1,22),
                onSet: function( event ) {
                    if ( event.select ) {
                        showTime = parseInt(this.get('select', 'HHi' ), 10);
                    }
                }
    });
}
```

Diagram 4.4.D4: JavaScript: initialising date and time pickers

login.html

```
<!DOCTYPE html>
<html>
<head>
    <title>CineManager</title>
    <link rel="icon" href="{{ url_for('static', filename='favicon-film.ico') }}">
    <link href="{{ url_for('static', filename='bootstrap.min.css') }}" rel="stylesheet">
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='icons/css/all.css') }}">
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='lib/themes/default.css') }}">
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='lib/themes/default.date.css') }}">
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='lib/themes/default.time.css') }}">
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
    <div class="container-fluid">
        <div class="jumbotron login-header">
            <i class="fas fa-film"></i>
            <h1>CineManager</h1>
            <h4>A theatre management and ticket booking system</h4>
        </div>
        <div class="jumbotron module">
            <input type="text" name="username" placeholder="Username" autocomplete="off">
            <input type="password" name="password" placeholder="Password">
            <button type="button" class="btn btn-warning" onclick="login()">Login</button>
        </div>
    </div>
    <script src="{{ url_for('static', filename='jquery.min.js') }}"></script>
    <script src="{{ url_for('static', filename='bootstrap.bundle.min.js') }}"></script>
    <script src="{{ url_for('static', filename='lib/picker.js') }}"></script>
    <script src="{{ url_for('static', filename='lib/picker.date.js') }}"></script>
    <script src="{{ url_for('static', filename='lib/picker.time.js') }}"></script>
    <script type="text/javascript" src="{{ url_for('static',filename='app.js') }}"></script>
</body>
</html>
```

Diagram 4.4.D5: HTML: login page

seating.html

```
<h4>Available Seats</h4>
<h5>Gold</h5>
{% for seat in goldSeats %}
<button onClick="selectSeat({{ seat[0] }},'gold')" {{ seat[1] }}>{{ seat[0] }}</button>
{% endfor %}
<h5>Standard</h5>
{% for seat in standardSeats %}
<button onClick="selectSeat({{ seat[0] }},'standard')" {{ seat[1] }}>{{ seat[0] }}</button>
{% endfor %}
```

Diagram 4.4.D6: HTML: Flask embedded code for seating arrangement

5 Chapter 5: Results snaps

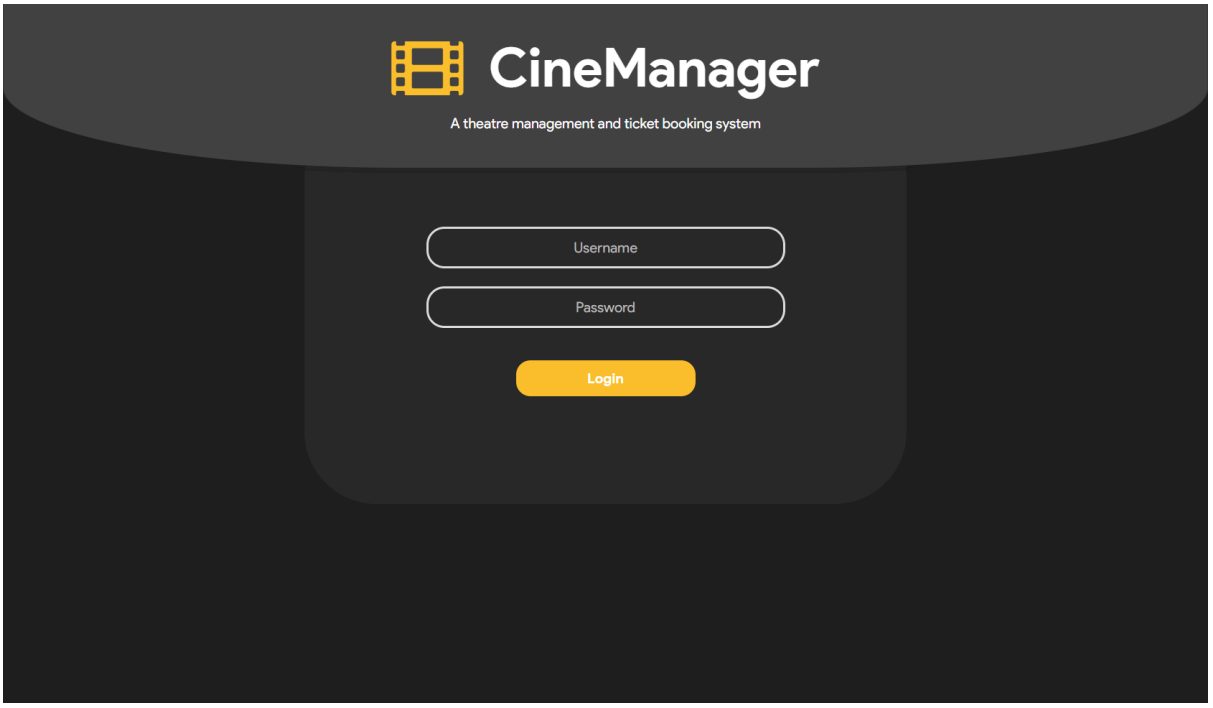

Diagram 5.D1: Login Page

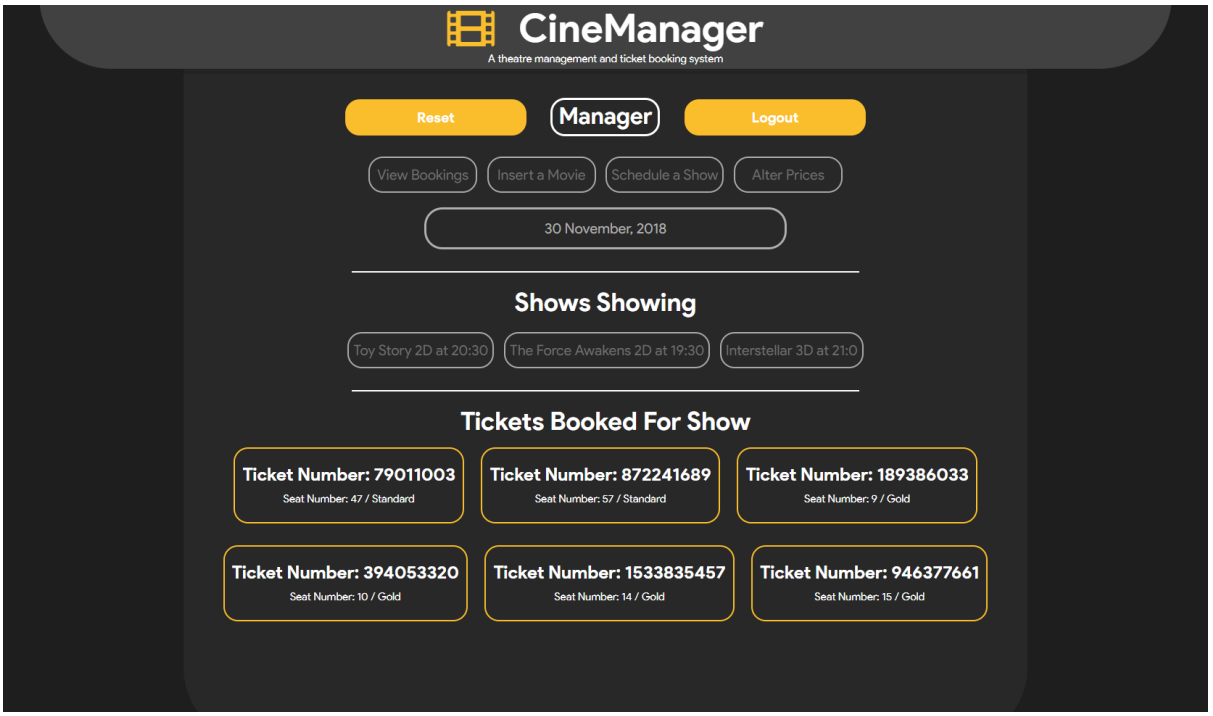

Diagram 5.D2: View Booked Tickets

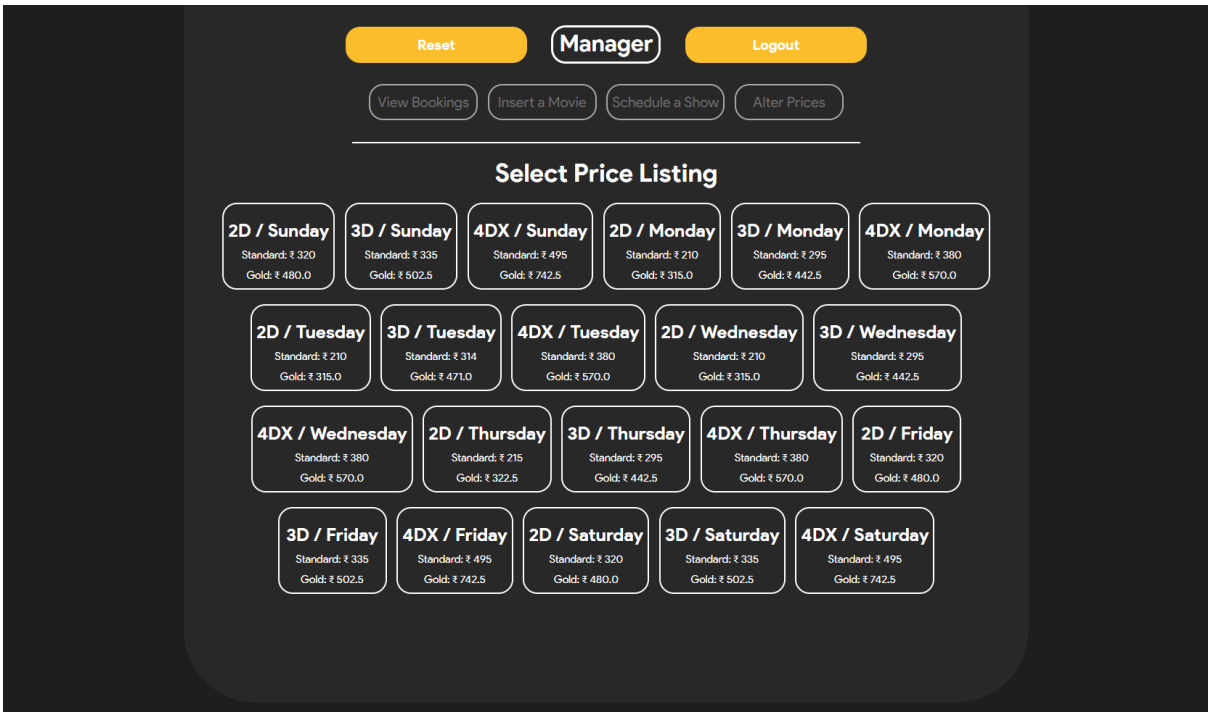

Diagram 5.D3: Alter Price Lists

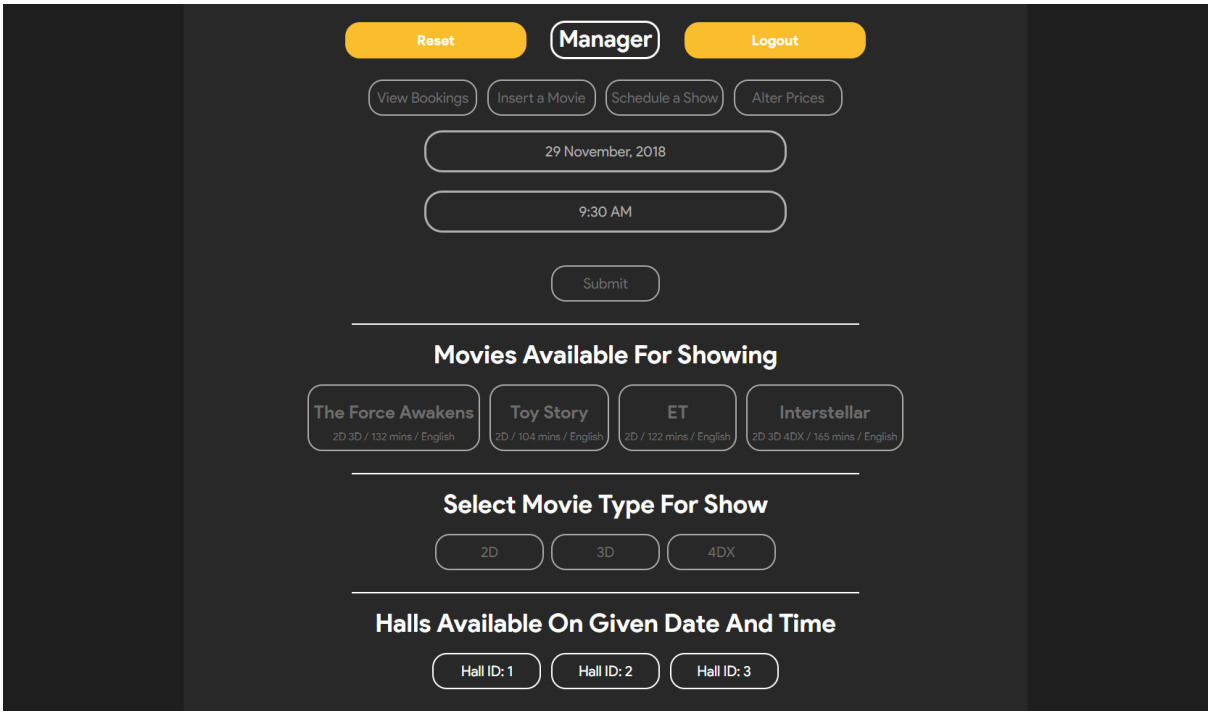

Diagram 5.D4: Schedule A Show

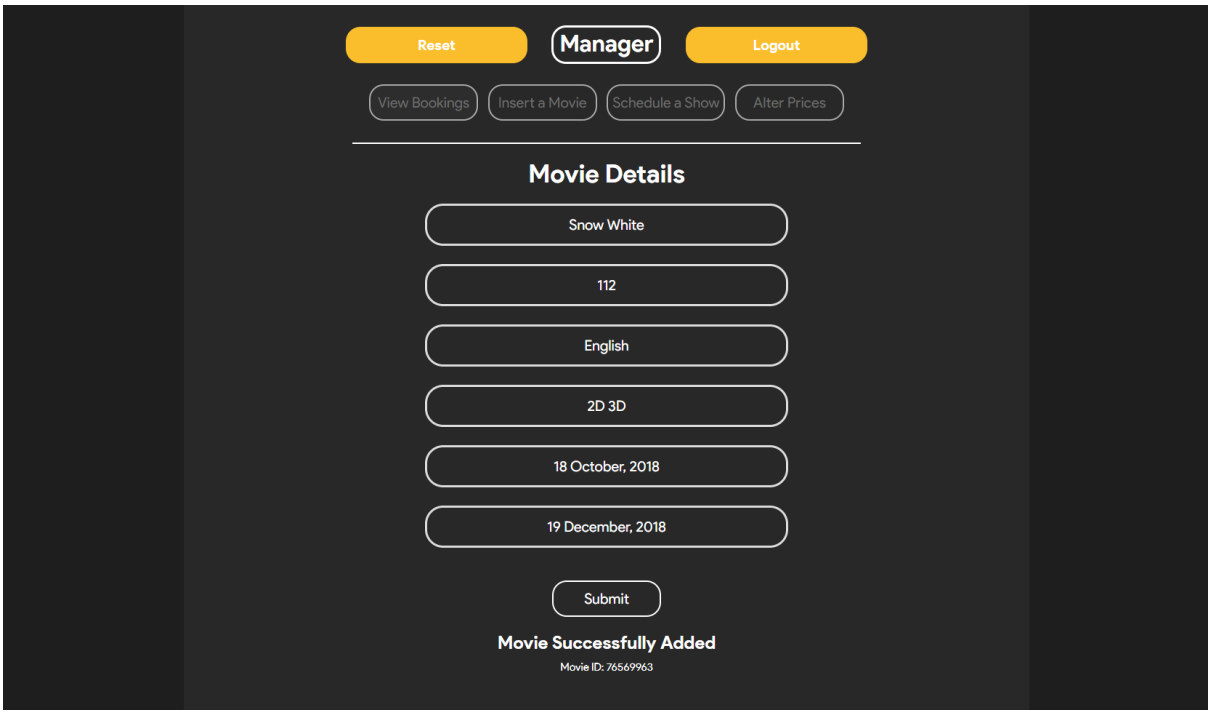

Diagram 5.D5: Insert A Movie

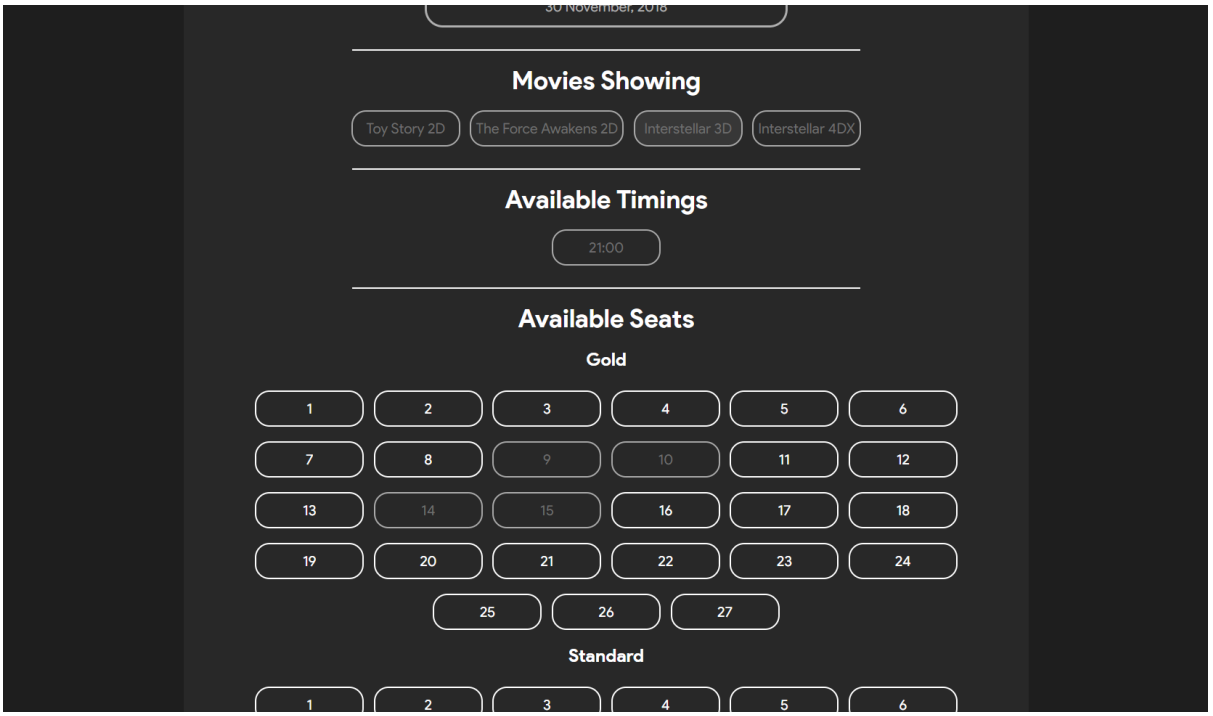

Diagram 5.D6: Book A Ticket

6 Chapter 6: Conclusion

6.1. Conclusion

As per the Requirements of the project, here is implemented a System that enables Theatre staff to fully minimize all aspects of a theatre movie management to an efficient and fast computer system. It allows movies, and shows to be added to the system, and tickets booked for shows. As such, need for bulky ledgers drops sharply, and all relevant is stored and can be printed as needed. It automatically determines prices and deletes old shows, movies and tickets reducing the amount of data to traverse for manager.

Thus, we have implemented a fully comprehensive and minimalistic efficient system for use by managers and cashiers without any additional training.

6.2. Limitations of the Project

The limitations of this project are:

- This project only considers theatre functions relating to movies and showings. Other functions such as employees, concessions, cleaning schedules, etc. have not been considered in this release.
- Halls cannot be inserted or updated from the front end in any way.

6.3. Future Enhancements

The future enhancements that may be brought about in the project may pertain to the limitations of the project. This includes:

- Factoring in additional functionalities relating to the theatre not covered under movies:
    - Concession Stands: Stalls selling food and drinks for movie-goers. This includes items sold, pre-orders etc.
    - Cleaning schedules: Halls need to be cleaned between movies, and theatre needs to be cleaned before opening and after closing.
    - Employees: Keeping track of employees hired, salaries, bonuses, duties, remarks etc.
- Account for expansion: Allow a high-ranking executive (with a new login ID) to add new halls to a location, or add new locations, with manager able to add shows only for their location, etc.

References

1. http://amsul.ca/pickadate.js/date/
2. https://teamtreehouse.com/community/nested-loops-in-flask-how-to-iterate-and-make-nested-lists
3. https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html
4. https://dev.mysql.com/doc/connector-python/en/connector-python-example-connecting.html
5. https://getbootstrap.com/docs/4.1/getting-started/introduction/
6. https://www.w3resource.com/mysql/date-and-time-functions/date-and-time-functions.php
7. https://w3resource.com/mysql/date-and-time-functions/mysql-dayname-function.php
8. https://w3resource.com/mysql/date-and-time-functions/mysql-curdate-function.php
9. https://w3resource.com/mysql/date-and-time-functions/mysql-datediff-function.php