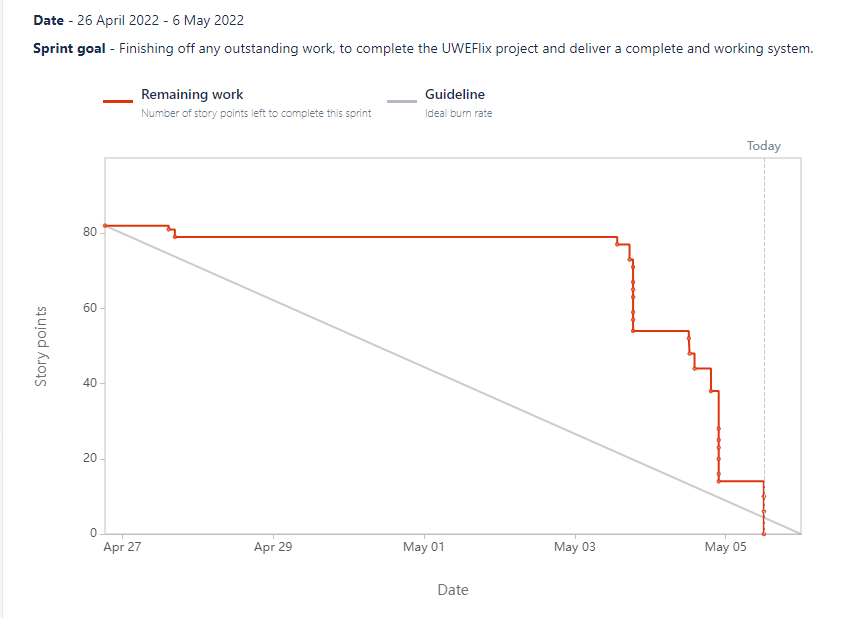
# **UFCF85-30-3 Enterprise System Development**

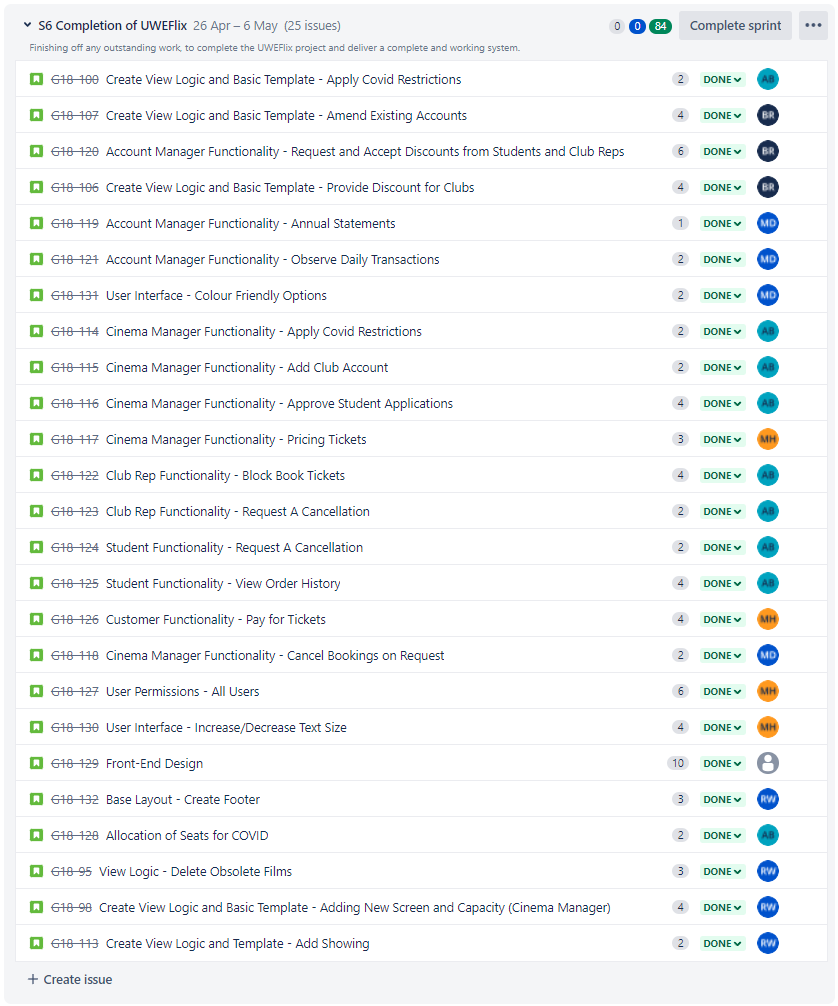
# **Sprint Review Form**

| Group: | 18 |
| --- | --- |
| Sprint: | 6 |
| Members: | Benedict Ramage-Mangles, Ross Williams, Arjun Binning, Michael Duncan, Matthew Hill |

<<Burn-down Chart>>



<<Backlog list>>



<<Communication Issues>>

No communication issues to report. All members have been communicating well and attending meetings regularly.

<<Reflections>>

Please refer to the Evaluation sections of each task, for detailed reflections.

<<Relevant Links>>

Github - [www.github.com/bean64/Group-18-ESD-2022](http://www.github.com/bean64/Group-18-ESD-2022)

Discord - <https://discord.gg/zwjTNUzEhM>

Testing Document - <https://docs.google.com/spreadsheets/d/1OfF0PlnA7NxUuSMhXcDRh7qDHArg3XSYatILbvpZlVk/edit?usp=sharing>

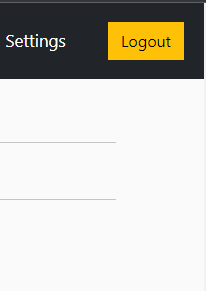
**Michael Duncan - Tasks**

1. **Front-End Design: Front Page and Logo**
   1. **Overview**

This task is about improving the aesthetic of the website, namely the home page. As this is the first page that users see, it is essential that it looks outstanding and encourages users to use the system. I will also be creating a logo to use as the “home” button on the navigation bar.

* 1. **Requirements**
* Appealing and intuitive design, with a consistent colour palette
* Logo design that incorporates UWE’s existing identity, but also ties into the website.
  1. **Design**

The colour palette will be inspired from the website at present. Therefore, will make use of the dark and light greys, white and orange that can be seen in the image below.



As for the Logo, here is the proposed design:



I propose that the home page is made up primarily of large cards for each film, similar to the design before, but with larger images to match the poster size of most film advertisements. Here is an example in a wireframe:



* 1. **Implementation**

Here is a code snippet detailing the design of each of the film cards:

{% for film in films %}

<div class="col-lg-4">

<img class="thumbnail" src="{{ film.image.url }}">

<div class="box-element **bg-dark** film" style="color:darkslategrey">

<h6 style="**height: 30px; color: white**;"><strong>{{film.title}}</strong></h6>

<hr style="**color: white**">

<a class="btn **btn-outline-warning** add-btn" href="{% url 'showings\_by\_film' film.pk %}">Book</a>

</div>

</div>

{% endfor %}

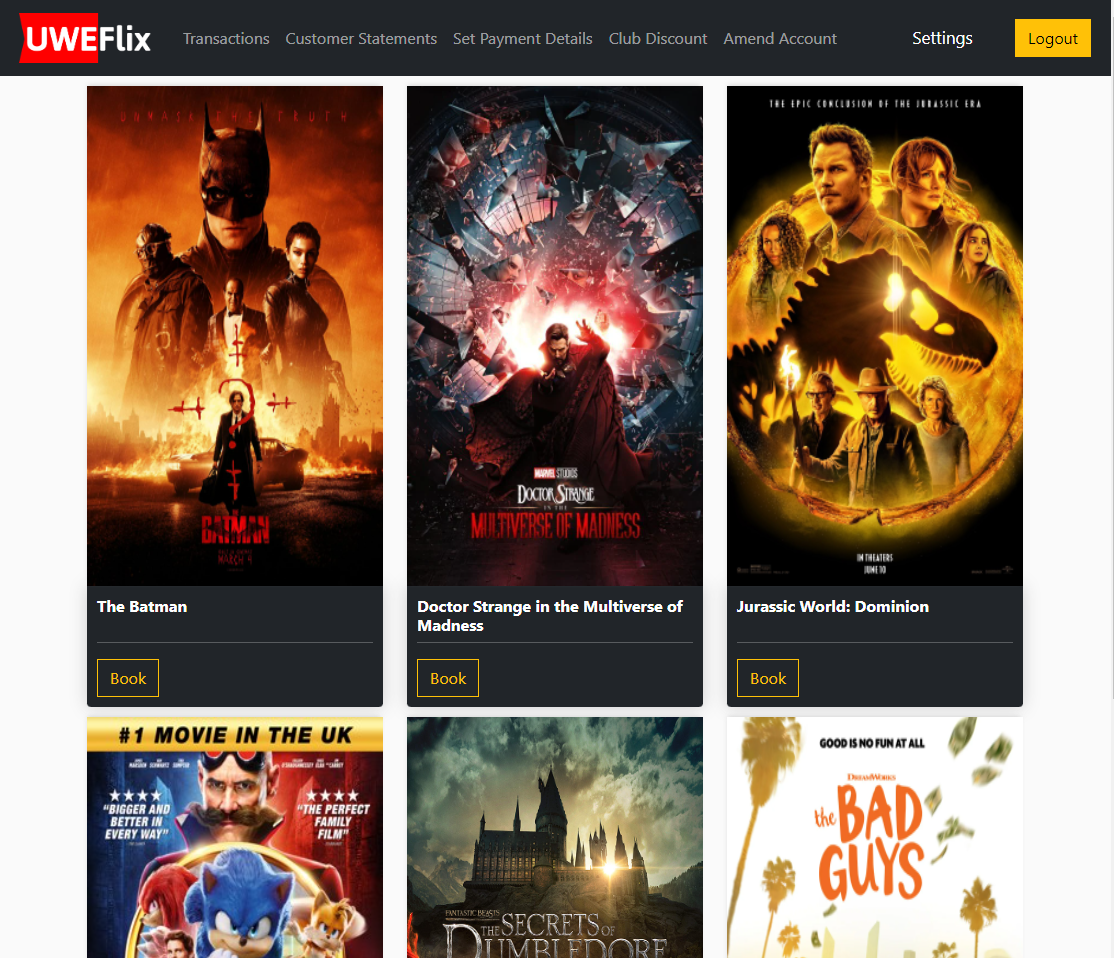
Here is a code snippet showing how the logo is implemented as the home button:

<a class="navbar-brand" href="{% url 'viewings' %}">

<img src="{% static '**uweflix/images/logo.png**' %}" height="50" width="140" style="margin-left: 20px;">

</a>

**Here is the finished result:**



* 1. **Testing**

Please see Tests [89] in the Testing Document.

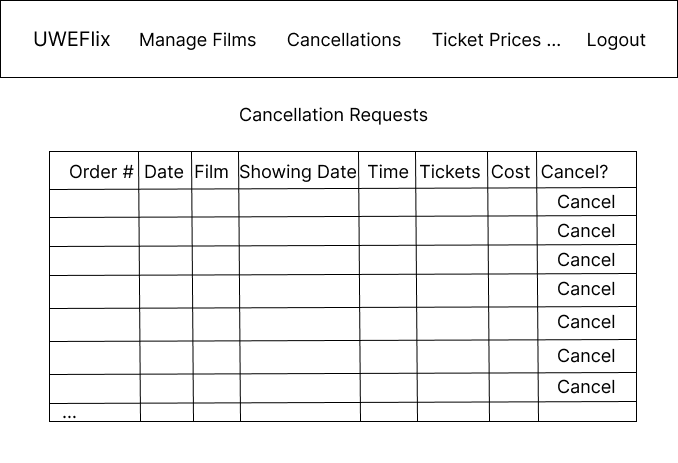
* 1. **Evaluation**

I am very pleased overall with the result. I think the design is very appealing and will give users a good first impression of the cinema system. Therefore, I am confident that this task was delivered as expected.

1. **Cinema Manager - Cancellation of Bookings on Request (View Logic and Template)**
   1. **Overview**

This task will implement the functionality to allow the Cinema Manager to be able to cancel bookings, provided that the customer has requested to do so first. Because of this prerequisite, this task cannot be completed until the functionality to be able to request a cancellation as a customer is completed first (Being completed by Arjun).

* 1. **Requirements**
* Cinema Manager should have a page where they can view all outstanding cancellation requests
* For each request there should be a button which approves this request, which handles and carries out the cancellation process:
  + The customer is given a refund for their purchase; for the exact amount they were charged **if** they had already paid\*. Refunds will be given as credit on their account.
  + *\*In the event of club reps that are settling their accounts monthly, they will not be given a refund if they have not already paid.*
  + The tickets reserved from the showing for this transaction are released, to allow other customers to purchase them
  1. **Design**

Here is a wireframe of the proposed design:

And Pseudocode for the Implementation:

BEGIN

transactionList AS List of all transactions WHERE requestToCancel is True

DISPLAY transactionList in Table

IF Cancel button is pressed:

transaction AS Transaction object corresponding to button pressed

numberOfTickets AS Int

totalCost AS Float

customer AS Customer object

IF customer is registered AND has paid:

customer is given refund

ENDIF

Increase ticketsRemaining for Showing by numberOfTickets

DELETE transaction

DISPLAY confirmation

ENDIF

END

* 1. **Implementation**

**HTML code:**

**{% extends "uweflix/layout.html" %}**

**{% load static %}**

**{% block title %}**

**Request Cancellation**

**{% endblock %}**

**{% block content %}**

**<style>**

**.transaction\_list {**

**margin-left: auto;**

**margin-right: auto;**

**}**

**.transaction\_list th, td {**

**border: 1px solid;**

**text-align: center;**

**padding-right: 15px;**

**height: 7.5ex;**

**overflow:hidden;**

**}**

**.transaction\_list th {**

**background-color: lightgrey;**

**}**

**h1 {**

**text-align: center;**

**}**

**</style>**

**<center>**

**<h1>Cancellation Requests</h1>**

**<hr/>**

**{% if zipped %}**

**<h3>All Requests:</h3>**

**<br/>**

**<table class="transaction\_list">**

**<thead>**

**<tr>**

**<th>Transaction #</th>**

**<th>Date Purchased</th>**

**<th>Film Name</th>**

**<th>Date of Showing</th>**

**<th>Showing Time</th>**

**<th>Tickets</th>**

**<th>Cost</th>**

**<th>Request Cancellation</th>**

**</tr>**

**</thead>**

**<tbody>**

**{% for ticket, adult, child, student in zipped %}**

**<tr>**

**<td>{{ ticket.transaction.id }}</td>**

**<td>{{ ticket.transaction.date | date:'d M Y' }}</td>**

**<td>{{ ticket.showing.film.title }}</td>**

**<td>{{ ticket.showing.time | date:'d M Y'}}</td>**

**<td>{{ ticket.showing.time | date:'h:m'}}</td>**

**<td>**

**{% if adult > 0 %}**

**{{ adult }}x Adult Tickets<br>**

**{% endif %}**

**{% if child > 0 %}**

**{{child}}x Child Tickets<br>**

**{% endif %}**

**{% if student > 0 %}**

**{{student}}x Student Tickets**

**{% endif %}**

**</td>**

**<td>£{{ ticket.transaction.cost | floatformat:2 }}</td>**

**<td>**

**<form action="{% url 'approve\_cancellations' %}" method="post">**

**{% csrf\_token %}**

**<button name="transaction\_number" value={{ticket.transaction.id}} type="submit" class="btn btn-warning">Approve</button>**

**</form>**

**</td>**

**</tr>**

**{% endfor %}**

**</tbody>**

**</table>**

**{% else %}**

**<p>No cancellations at the moment.</p>**

**{% endif %}**

**<hr/>**

**</center>**

**{% endblock %}**

**View.py Code:**

**def approve\_cancellation(request):**

**now = timezone.now()**

**user = None**

**showingsToCancel = []**

**adultTickets = []**

**childTickets = []**

**studentTickets = []**

**transaction\_list = Transaction.objects.filter(request\_to\_cancel=True)**

**for purchase in transaction\_list:**

**allTickets = Ticket.objects.filter(transaction=purchase)**

**ticket = allTickets.first()**

**if ticket is not None:**

**showingTime = ticket.showing.time**

**if showingTime > now:**

**showingsToCancel.append(ticket)**

**adultTickets.append(len(allTickets.filter(ticket\_type="adult")))**

**childTickets.append(len(allTickets.filter(ticket\_type="child")))**

**studentTickets.append(len(allTickets.filter(ticket\_type="student")))**

**zippedList = zip(showingsToCancel, adultTickets, childTickets, studentTickets)**

**context = {**

**'zipped': zippedList**

**}**

**if request.method == "POST":**

**requestedTransaction = Transaction.getTransaction(request.POST.get('transaction\_number'))**

**ticketList = Ticket.objects.filter(transaction=requestedTransaction.id)**

**totalTickets = ticketList.count()**

**transactionCost = requestedTransaction.cost**

**if requestedTransaction.customer is not None:**

**customer = Customer.objects.get(id=requestedTransaction.customer.id)**

**customer.credit += transactionCost**

**customer.save()**

**showing = ticketList.first().showing**

**showing.remaining\_tickets += totalTickets**

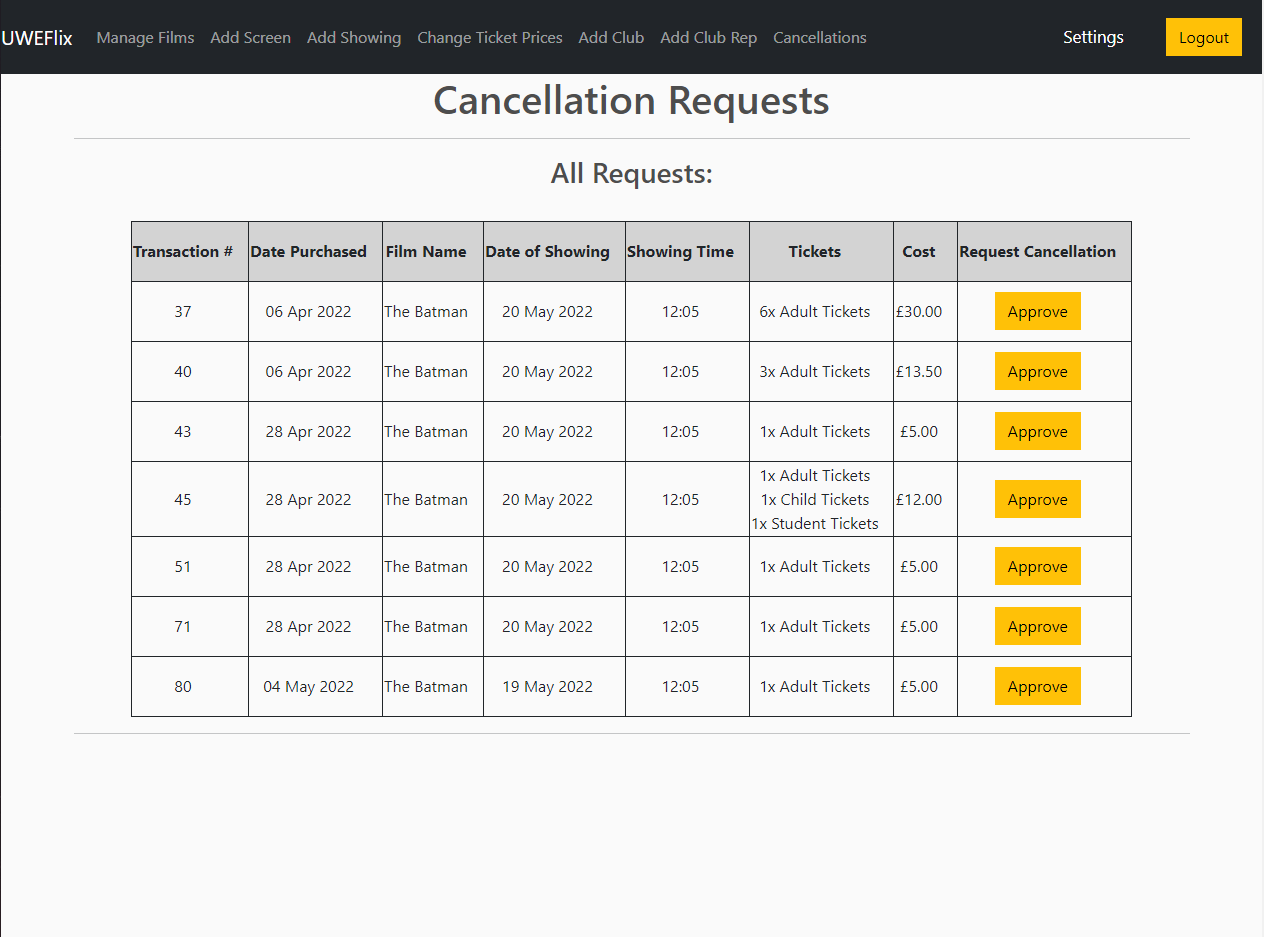
**showing.save()**

**Transaction.deleteTransaction(requestedTransaction.id)**

**return redirect('approve\_cancellations')**

**return render(request, "uweflix/approve\_cancellations.html", context)**

**Result:**

****

* 1. **Testing**

Please see tests [82-84] in the Testing Document.

* 1. **Evaluation**

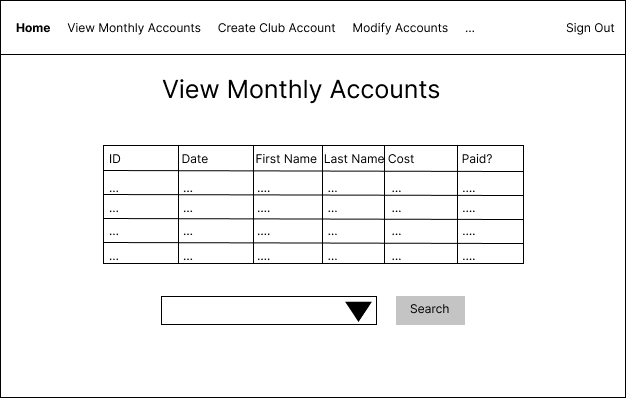
I think that this functionality was implemented well. It covers what was expected from it, and presents all of the relevant information clearly to the Cinema Manager to be able to make their decisions. One thing that was thought about being added was the customer who made the purchase, however in the event of confidentiality, this was omitted.

1. **Account Manager - Annual Transaction Statements**
   1. **Overview**

This task is an extension of a previous implementation (Account Manager viewing End of Month statements). Therefore, this is a small addition to allow the Account Manager to also be able to view Annual Statements.

* 1. **Requirements**
* The Account Manager should be able to select a Club Representative, and select the Annual Statement option, for which they are presented with a list of all statements from the last calendar year.
  1. **Design**

Wireframe is the same as that of the similar task from Sprint 5:

****

Pseudocode for additional functionality:

…

IF transactionType is Annual

transactionList AS List of all transactions FOR customer in the current calendar year

DISPLAY ALL transactions from transactionList AS Table

ENDIF

END

* 1. **Implementation**

**Additional code:**

…

elif trange == "Year":

transaction\_list = Transaction.objects.filter(

customer=clubrep,

date\_\_year = dt.now().year

)

context = {

'club\_rep\_num': clubrep.club\_rep\_num,

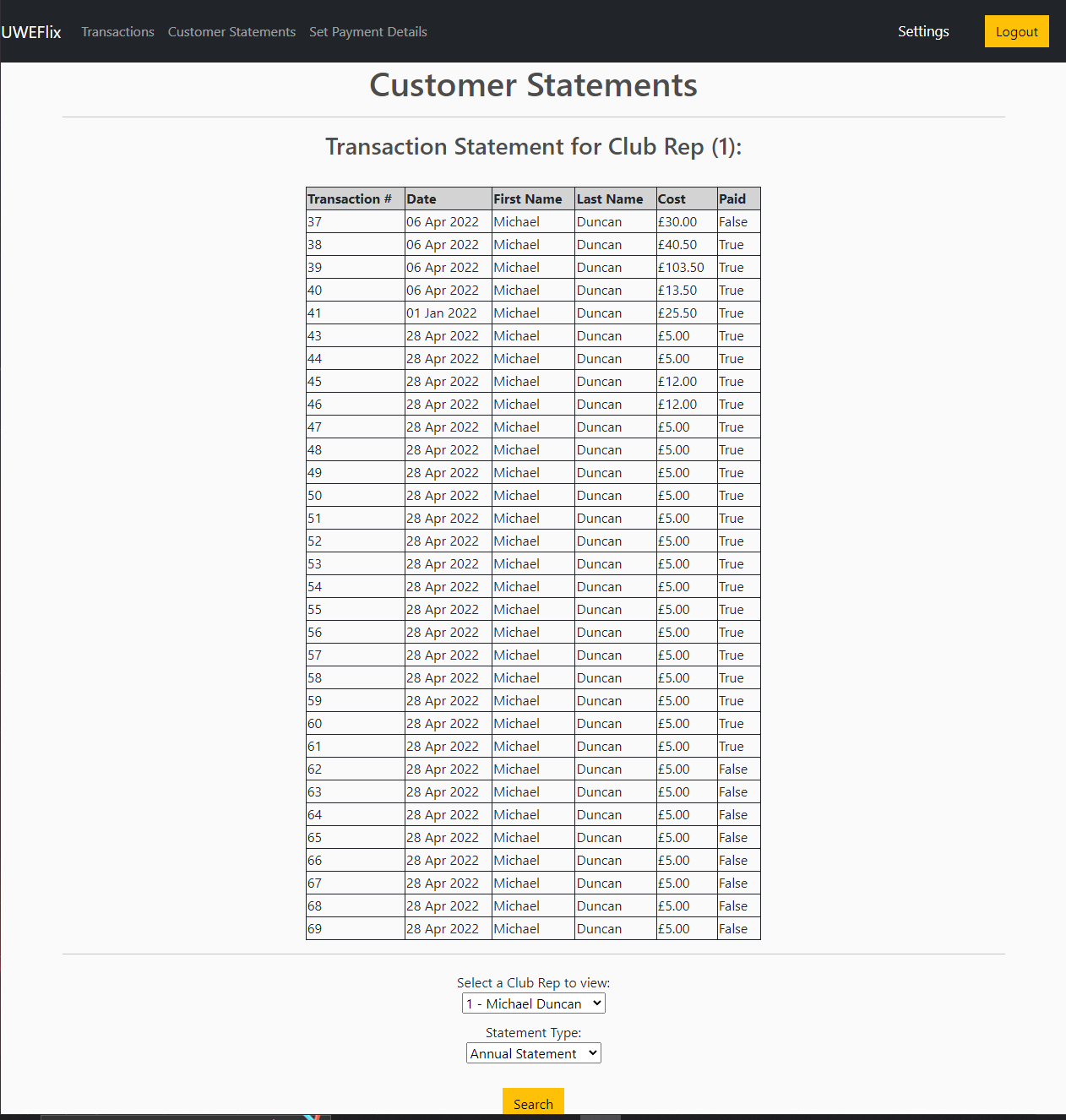
'transaction\_list': transaction\_list,

'form': form

}

return render(request, 'uweflix/customer\_statements.html', context)

**Result:**

****

* 1. **Testing**

Please see tests [85 & 86] in the Testing Document.

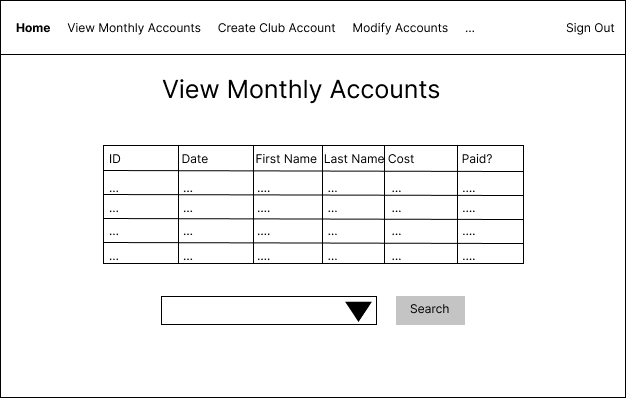
* 1. **Evaluation**

This functionality works as promised and expected. You are able to select a club rep from the list, and then select either monthly or annual statements.

1. **Account Manager - Daily Transactions**
   1. **Overview**

This task is very similar to the previous task, however it requires the viewing of every transaction for a given day, from every customer. The table format will be the same, and therefore reused.

* 1. **Requirements**
* Account Manager should be able to view all transactions by day
* For convenience, there should also be a button to view today’s transactions
  1. **Design**

The design for the page is virtually the same as the design for the end of month account statements, as it is a table full of Transactions. Therefore, the design from that functionality will be reused. ****

**Pseudocode:**

BEGIN

IF todayButton is pressed

requiredDate is today

ELSE

GET requiredDate from user entry

ENDIF

transactionList AS List of all Transaction objects WHERE date is requiredDate

DISPLAY transactionList as a Table

END

* 1. **Implementation**

**HTML Template code:**

**{% extends "uweflix/layout.html" %}**

**{% load static %}**

**{% block title %}**

**Customer Statements**

**{% endblock %}**

**{% block content %}**

**<style>**

**.transaction\_list {**

**margin-left: auto;**

**margin-right: auto;**

**}**

**.transaction\_list th, td {**

**border: 1px solid;**

**text-align: left;**

**padding-right: 15px;**

**}**

**.transaction\_list th {**

**background-color: lightgrey;**

**}**

**h1 {**

**text-align: center;**

**}**

**</style>**

**<center>**

**<h1>Customer Statements</h1>**

**<hr/>**

**{% if transaction\_list %}**

**<h3>All transactions for {{ selected\_date | date:'d/m/Y'}}:</h3>**

**<br/>**

**<table class="transaction\_list">**

**<thead>**

**<tr>**

**<th>Transaction #</th>**

**<th>Date</th>**

**<th>First Name</th>**

**<th>Last Name</th>**

**<th>Cost</th>**

**<th>Paid</th>**

**</tr>**

**</thead>**

**<tbody>**

**{% for transaction in transaction\_list %}**

**<tr>**

**<td>{{ transaction.id }}</td>**

**<td>{{ transaction.date | date:'d M Y' }}</td>**

**<td>{{ transaction.customer.user.first\_name }}</td>**

**<td>{{ transaction.customer.user.last\_name }}</td>**

**<td>£{{ transaction.cost | floatformat:2 }}</td>**

**<td>{{ transaction.is\_settled }}</td>**

**</tr>**

**{% endfor %}**

**</tbody>**

**</table>**

**{% else %}**

**<p>{{ title\_text }}</p>**

**{% endif %}**

**<hr/>**

**<script>**

**$( function() {**

**$( "#id\_date" ).datepicker({**

**dateFormat: "yy-mm-dd"**

**});**

**} );**

**</script>**

**<form action="{% url 'daily\_transactions' %}" method="post" autocomplete="off">**

**{% csrf\_token %}**

**<div class="form-area">**

**<div class="form-field">**

**Date: {{ form.date }}**

**</div>**

**<button name="search" type="submit" class="btn btn-warning">Search</button>**

**</div>**

**<br/>**

**<b>or:</b>**

**<br/>**

**<br/>**

**<button name="today" type="submit" class="btn btn-warning">Show Today's Transactions</button>**

**</form>**

**</center>**

**{% endblock %}**

**Views.py code:**

**def daily\_transactions(request):**

**form = DatePickerForm()**

**titleText = "Please select a day to view transactions for:"**

**context = {**

**'form': form,**

**'title\_text': titleText}**

**if request.method == "POST":**

**form = DatePickerForm(request.POST)**

**if form.is\_valid():**

**selectedDate = dt.today()**

**if "search" in request.POST:**

**selectedDate = form.cleaned\_data['date']**

**elif "today" in request.POST:**

**selectedDate = dt.today()**

**transaction\_list = Transaction.objects.filter(date = selectedDate)**

**if not transaction\_list:**

**titleText = f"There were no transactions made on {str(selectedDate)}"**

**context = {**

**'form': form,**

**'title\_text': titleText**

**}**

**else:**

**context = {**

**'selected\_date': selectedDate,**

**'transaction\_list': transaction\_list,**

**'form': form**

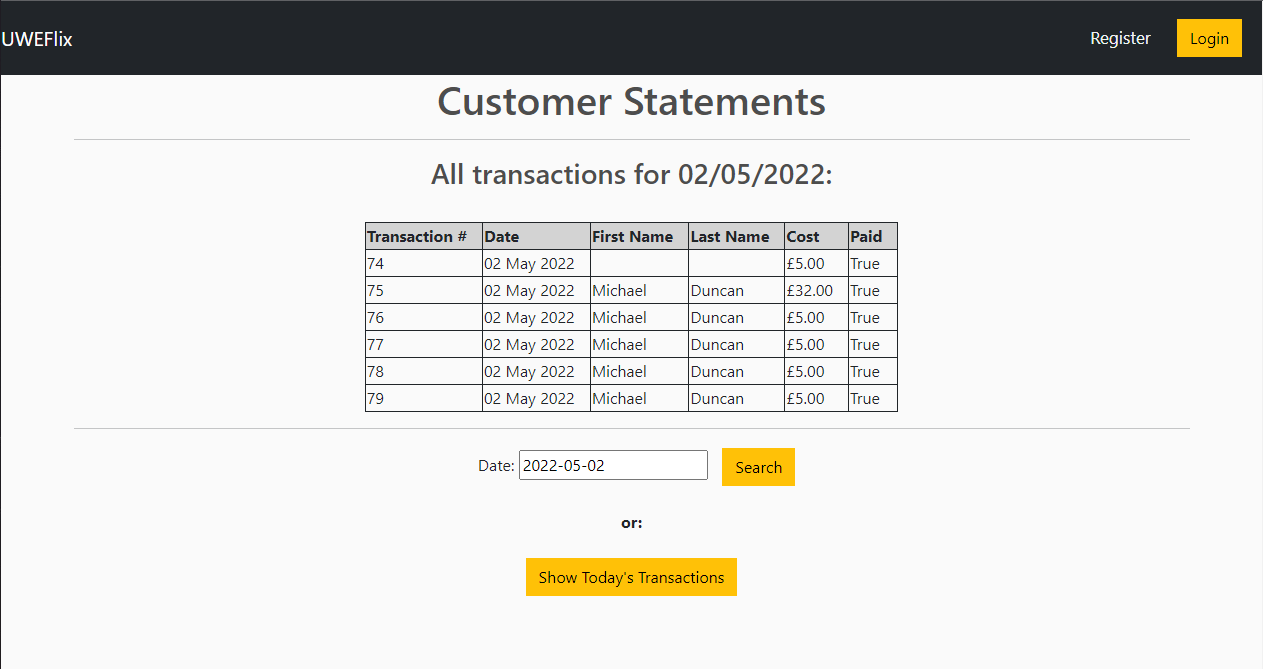
**}**

**return render(request, "uweflix/daily\_transactions.html", context)**

**else:**

**return render(request, "uweflix/daily\_transactions.html", context)**

**Result:**

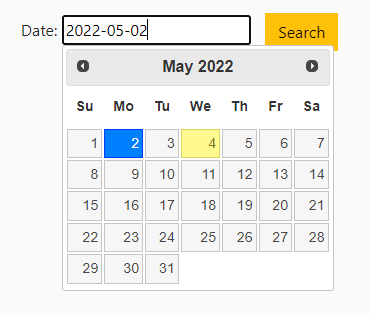
****

* 1. **Testing**

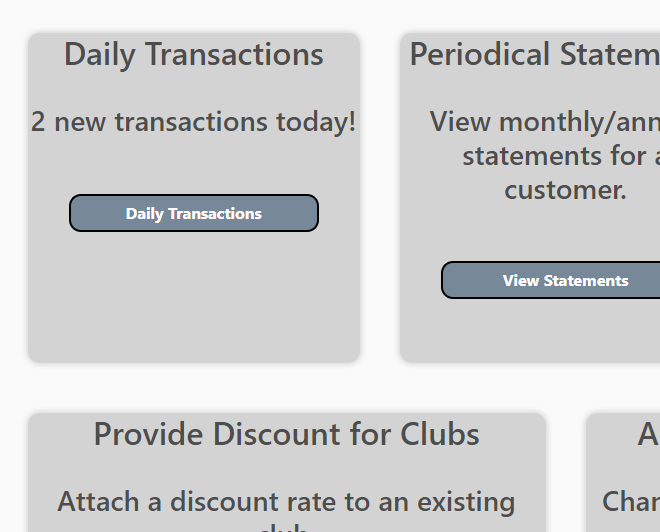
Please see tests [78-81] in the Testing Document.

* 1. **Evaluation**

I feel that this functionality was delivered as promised, and therefore is a successful task completion. This allows the Account Manager to search for all transactions that occur on a given day (a Date Picker has also been implemented for convenience, pictured below), or a shortcut to view today’s transactions.



On the Account Manager dashboard, the statistic for the number of new transactions today is also shown (pictured below) for convenience:



1. **User Interface - Colour-Blind Friendly Options**
   1. **Overview**

As required by the UWE Student Experience Team, we are required to make adjustments to the system to cater for those who suffer from colour-blindness. Therefore, this task will implement a toggle in the footer of the layout page which will change the colour scheme of the website to allow those in question to see the content much easier.

* 1. **Requirements**
* There must be a setting on all pages that allows for the colour palette of the website to be changed to one that is more colour-blind friendly.
* The effectiveness of the colour palette used will be judged using a user questionnaire, of which it must score 7/10 or more for the colour-blind friendliness section.
  1. **Design**

To begin this development, research must first be done into which colours would be acceptable for use. As described by Tableau (2022), blue is a common colour to use in this scenario, as it usually views mostly true to its colour for both those who suffer from Colour Vision Deficiency (CVD), and those who do not. Therefore, we will use **blue** as the background colour.

As also described in the same article, orange is also a good colour to pair with blue, as they contrast well for those with CVD. Therefore, this will be our foreground colour. As most of the buttons in the design of the website are already yellow/orange, this will be easy to incorporate with the existing design.

In terms of the actual implementation, this will be added as a slider on the base layout, so that all pages can benefit from this feature.

**Pseudocode for Implementation:**

BEGIN

previousBackgroundColour, previousTextColour AS Colours

IF Slider is ON

Change backgroundColour to blue

Change textColour to Orange

ELSE

Change backgroundColour to previousBackgroundColour

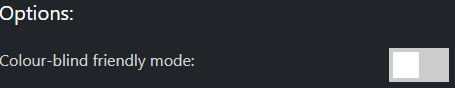
Change textColour to previousTextColour

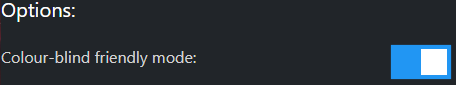
ENDIF

END

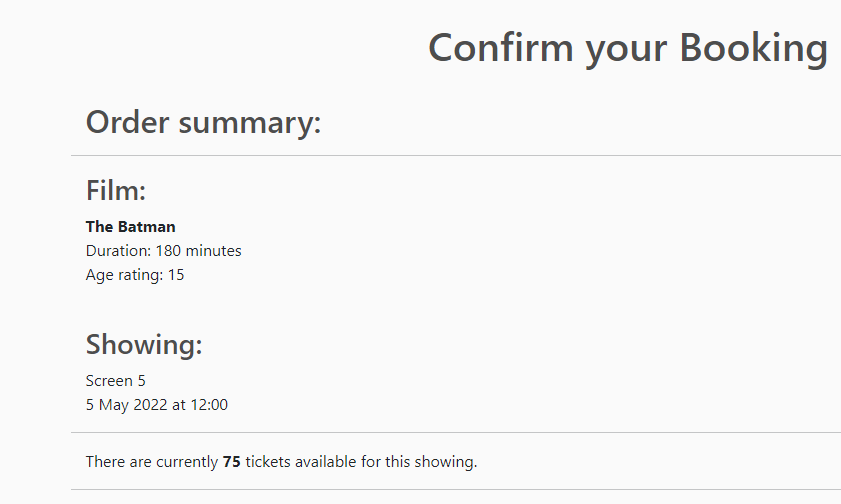
* 1. **Implementation**

Here is the functionality on screen:

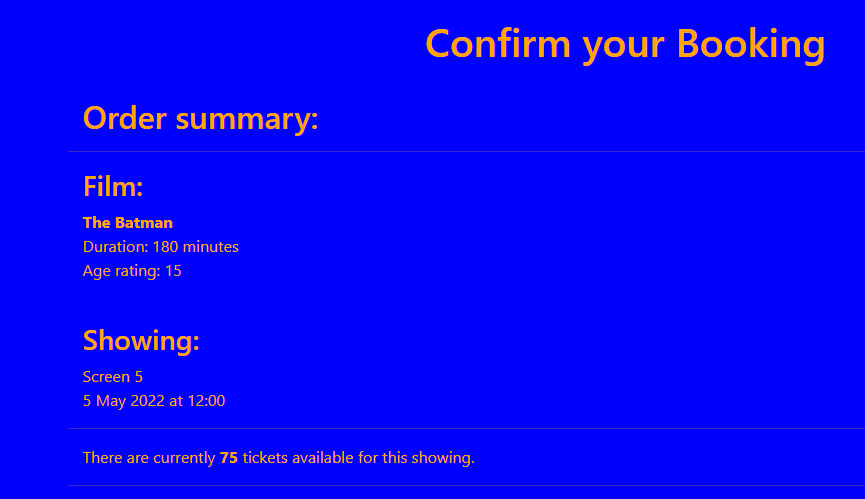




Here is an example page before colour conversion:



And after:



* 1. **Testing**

Please see tests [87 & 88] in the Testing Document.

* 1. **Evaluation**

The implementation of this feature is somewhat functional, however not entirely practical in day to day use. For example, we used a colour blind simulator to indicate what it would look like to someone who suffers from CVD, and in some cases the original colour scheme is easier to view than this representation.  
Secondly, the colour change only applies to text, titles and the background. Therefore any additional assets, such as cards, nav bar, footer, buttons, etc. do not have this applied. This can result in either the same viewing experience, or sometimes can result in worse, where some colours change, and others do not.  
Finally, the colours used are not very aesthetic. Whilst this is not the point of the feature, further research could be done to find a good colour scheme that is both functional and easy on the eye. Thankfully this would be a very simple change to the code and would not take a lot of effort.

**Arjun Binning - Tasks**

1. **Cinema Manager - Apply Covid Restrictions**
2. **Overview**

This task revolved around the Cinema Manager and allowed for them to toggle Covid Restrictions. Per the UWEFlix Case Study, the Cinema Manager was required to have the ability to apply Covid Restrictions, which decreased the screen capacity and allowed for social distanced seating. This was achieved by toggling the ‘apply\_covid\_restrictions’ attribute in every screen model every time the button was pressed.

1. **Requirements**

For this task:

* A button should be placed that can easily be accessed to the Cinema Manager, such as the home page, and an area displaying the status of covid restrictions.
* When the button is pressed, the restrictions should be toggled within the Screen Model and displayed on the home screen.
* When the button is pressed, the screen capacity should change to suit the amount of people the screen can hold.

1. **Design**

This would occur in the view.py file and the pseudocode is as follows:

BEGIN

screens AS Screen Model

IF request.POST:

restrictions = screens.objects.first().apply\_covid\_restrictions

restrictions = NOT restrictions

FOREACH screen in screens:

screens.updateScreen(screen.id, restrictions)

IF restrictions:

screens.updateScreen(screen.id, screen.capacity/2)

ELSE:

screens.updateScreen(screen.id, screen.capacity \* 2)

ENDIF

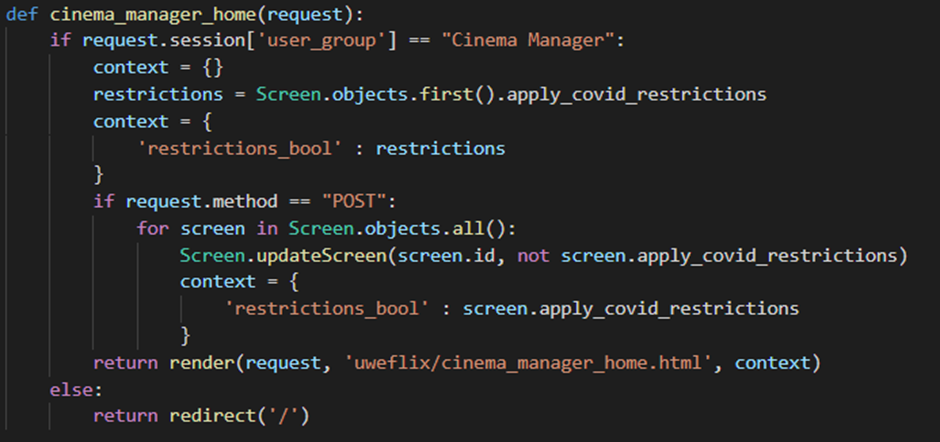
ENDFOR

ENDIF

END

1. **Implementation**

This was successfully implemented into the Cinema Manager’s Home Screen. The full implementation can be seen below in the views.py file:



1. **Demonstration**

The button appeared on the Home Page exactly as we needed:



1. **Evaluation**

The Button appeared as intended on the Cinema Managers home page, and when the button was pressed, the page would redirect into itself and the new status of social distancing was shown - working as intended.

1. **Cinema Manager - Register Club Account**
2. **Overview**

This task revolved around the Cinema Manager and allowed for them to register clubs for the Cinema Site. Per the UWEFlix Case Study, Student Clubs must be registered by a Cinema Manager, and the purpose of this task is to allow them to do that.

1. **Requirements**

For this task:

* A form should be created that takes in the necessary constructor arguments for creating a Club Model Instance.
* The form should be presented in a template that allows the Cinema Manager to enter the necessary details in the form.
* When the form is submitted, the club should be created.

1. **Design**

Processing the form would occur in the view.py file and the pseudocode is as follows:

BEGIN

form AS addClub Form

IF form IS valid:

clubName = form[clubName]

clubStreetNum = form[clubStreetNum]

clubStreet = form[clubStreet]

clubCity = form[clubCity]

clubPostcode = form[clubPostcode]

clubLandline = form[clubLandline]

clubMobile = form[clubMobile]

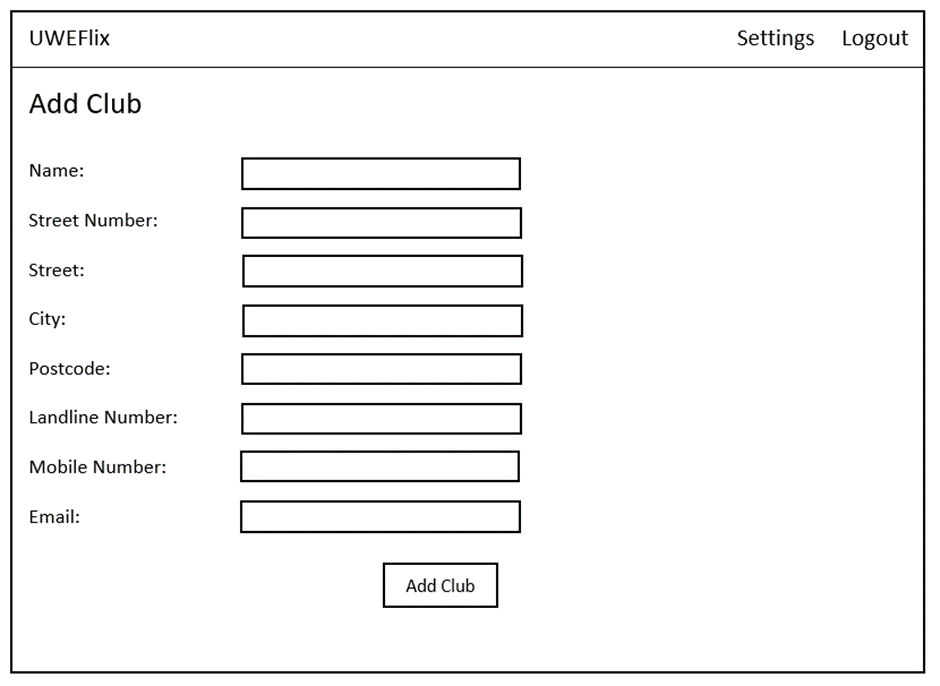
clubEmail = form[clubEmail]

Club.newClub(clubName, clubStreetNum, clubStreet, clubCity, clubPostcode, clubLandline, clubMobile, clubEmail)

ENDIF

END

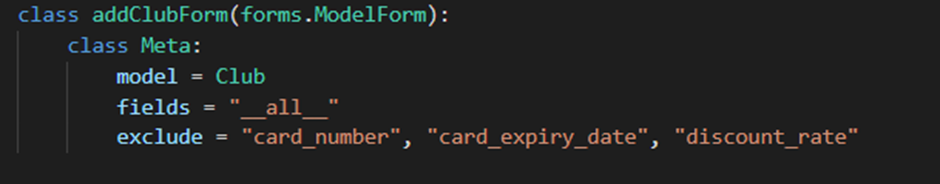
The Wireframe for the form would be:



1. **Implementation**

This was successfully implemented as such:

The form:



The views.py file:

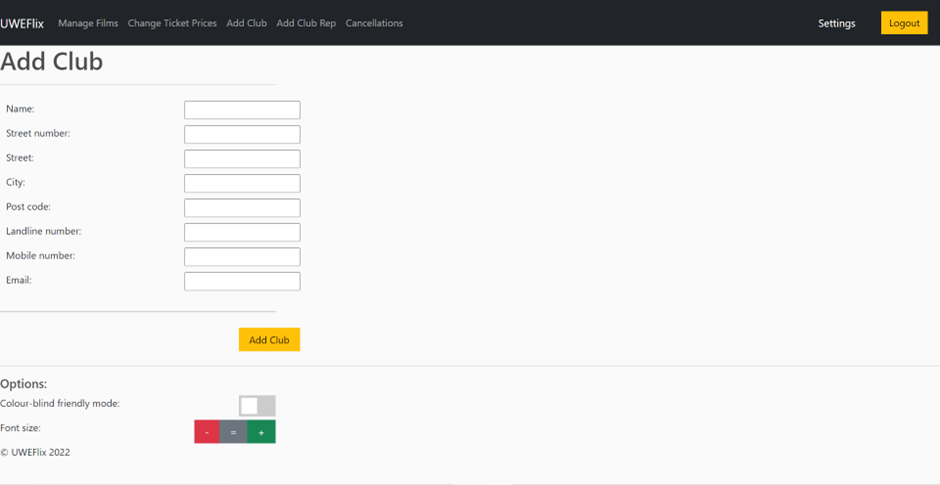


And finally, the HTML code:



1. **Demonstration**

The page worked and ended up looking like this:



1. **Evaluation**

Clubs were successfully registered when the form was filled out. All inputs of the form were required to be filled out, and once done so, the Club Object was successfully created with the correct information.

1. **Cinema Manager - Approve Student Applications**
2. **Overview**

This task revolved around the Cinema Manager being able to view a student’s applications, and then choosing whether to accept or deny them. Per the UWEFlix Case Study, students were required to apply to get access to their accounts, with the applications being reviewed by the Cinema Manager.

1. **Requirements**

For this task:

* Student Accounts should NOT be valid upon registration.
* The Cinema Manager should be able to view information of each invalid student account and choose to accept or deny them.
* The Account ID that the Manager is viewing should be fed into the views file and URL so the necessary account can be altered.
* If they are accepted, their account becomes valid, however if they are denied, their account is deleted.

1. **Design**

The Pseudocode for accepting the student would be as follows:

BEGIN

student AS User Object

IF accept student:

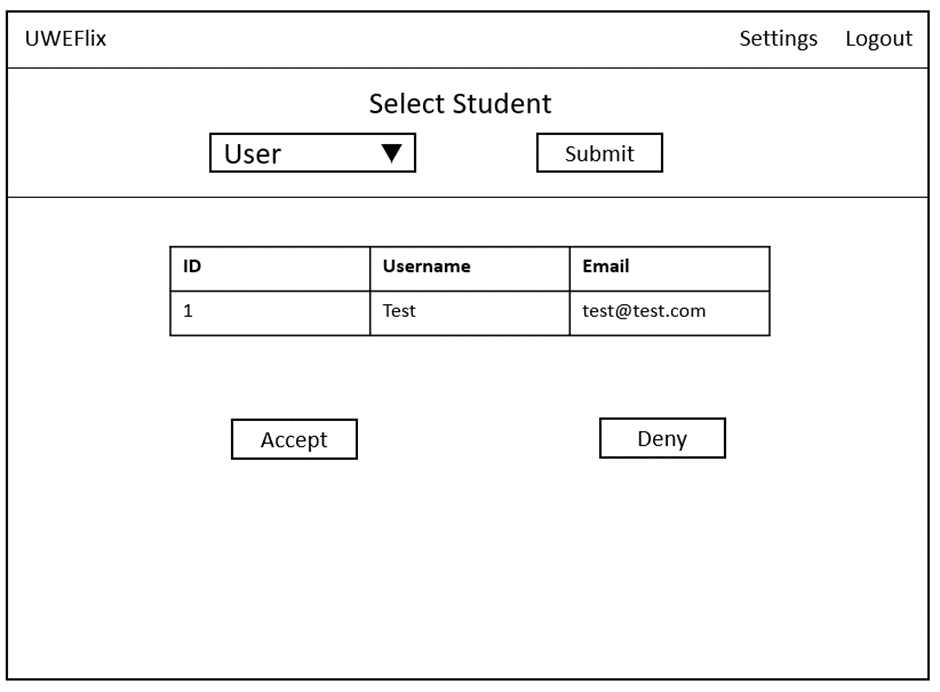
student.update(is\_valid = true)

ELSE:

student.delete()

END

The Wireframe for the page would be:



1. **Implementation**

This was successfully implemented as such:

The views.py file:



The HTML code:

{% extends "uweflix/layout.html" %}

{% load static %}

{% block title %}

Review Students

{% endblock %}

{% block content %}

<style>

.student\_list {

margin-left: auto;

margin-right: auto;

}

.student\_list th, td {

border: 1px solid;

text-align: left;

padding-right: 15px;

}

.student\_list th {

background-color: lightgrey;

}

h1 {

text-align: center;

}

.dropdown {

width: 100px;

padding: 5px;

margin: 10px 0;

border-radius: 12px;

font-weight: bold;

box-shadow: 0 0 5px 4px rgba(211,211,211,0.8);

}

.selectButton {

width: 100px;

padding: 5px;

margin: 10px 0;

border-radius: 12px;

background-color: lightslategrey;

color: white;

font-weight: bold;

box-shadow: 0 0 5px 4px rgba(211,211,211,0.8);

}

.acceptDenyButton {

display: inline-block;

width: 100px;

padding: 5px;

margin: 5px 0;

border-radius: 12px;

background-color: lightslategrey;

color: white;

font-weight: bold;

margin-left: auto;

margin-right: auto;

box-shadow: 0 0 5px 4px rgba(211,211,211,0.8);

}

.flex-container {

display: flex;

justify-content: center;

}

.flex-container > div {

margin: 5px;

padding: 20px;

font-size: 20px;

}

</style>

<center>

<h1>Approve Students</h1>

{% if students %}

<form method="POST" name="changeStudent">

{% csrf\_token %}

<select name="ReviewStudentForm" class="dropdown">

{% for student in students %}

{% if student.id == urlID%}

<option value="{{student.id}}" selected>{{student.username}}</option>

{%else %}

<option value="{{student.id}}">{{student.username}}</option>

{%endif%}

{%endfor%}

<input type="hidden" name="name" value="changeStudent">

<input type="submit" value="Search" name="changeStudent" class="selectButton"/>

</select>

</form>

<hr/>

<table class="student\_list">

<thead>

<tr>

<th>ID</th>

<th>Username</th>

<th>Email</th>

</tr>

</thead>

<tbody>

<tr>

<td>{{ chosenStudent.id }}</td>

<td>{{ chosenStudent.username }}</td>

<td>{{ chosenStudent.email }}</td>

</tr>

</tbody>

</table>

<div class="flex-container">

<div><form method="POST" name="acceptStudent">

{% csrf\_token %}

<input type="hidden" name="name" value="acceptStudent">

<input type="submit" name="acceptStudent" value="Accept" class="acceptDenyButton" />

</form></div>

<div><form method="POST" name="denyStudent">

{% csrf\_token %}

<input type="hidden" name="name" value="denyStudent">

<input type="submit" value="Deny" name="denyStudents" class="acceptDenyButton"/>

</form></div>

</div>

{%else%}

<hr/>

<h3>No Student Applications!</h3>

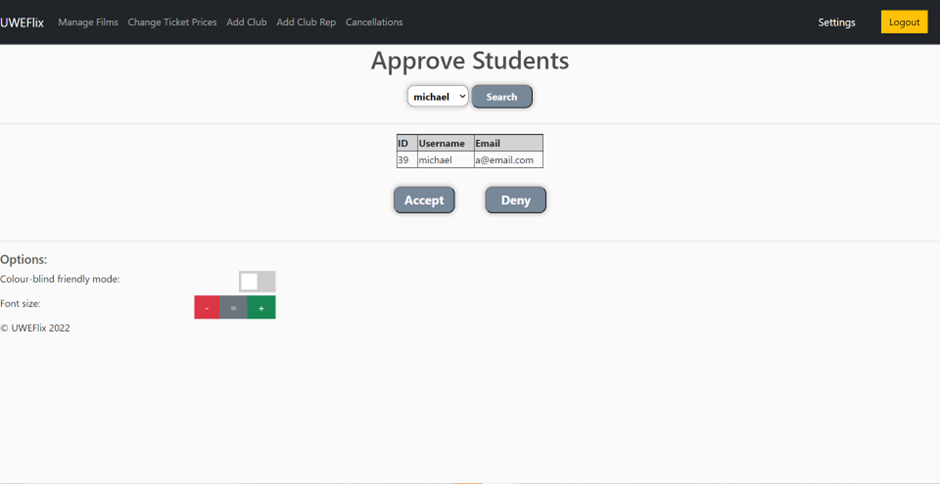
{% endif %}

</center>

{% endblock %}

1. **Demonstration**

The page worked and ended up looking like this:



1. **Evaluation**

The screen showed students who had registered and they could be selected from a dropdown box at the top if the Cinema Manager was searching for a specific student. If they accepted the student, the account was validated and able to log in, however if it was denied, the account was deleted.

1. **Club Rep - Block Book Tickets**
2. **Overview**

This task revolved around the Club Rep block booking tickets to a cinema showing. Per the UWEFlix Case Study, Club Representatives are able to block book tickets for Club Social Events, as long as they purchase a minimum of 10 tickets.

1. **Requirements**

For this task:

* The Club Rep should NOT be able to purchase less than 10 tickets.
* A form should be created allowing the Club Rep to enter the number of tickets required, and to show a total that uses JQuery to update the value live.
* The discount rate of the Club should be automatically added to the total value.
* The Club Rep can either pay with their credit, or the Clubs monthly statement. However, if there is not enough credit on the account, the payment process fails.
* If the Club Rep asks for more tickets than there are available, the process should also fail.
* A transaction object and ticket objects should be created for the showing if the payment succeeds.

1. **Design**

The main functionality would occur in the view.py file and the pseudocode is as follows:

BEGIN

showing AS Showing Object

rep as ClubRep Object

form AS RepPaymentForm

total\_cost = form[total\_cost]

tickets = form[tickets]

payment\_option = form[payment\_option]

IF showing.tickets < tickets:

Return Redirect(‘Error – Insufficient Tickets’)

ELSE:

paying AS Boolean

IF payment\_option == “Credit” AND rep.credit > total\_cost:

rep.credit -= total\_cost

paying = True

ELIF payment\_option == “Statement”:

paying = False

ELSE:

Return Redirect(‘Error – Insufficient Credit’)

ENDIF

transaction = Transaction.newTransaction(rep, total\_cost, paying)

FOR i in range tickets:

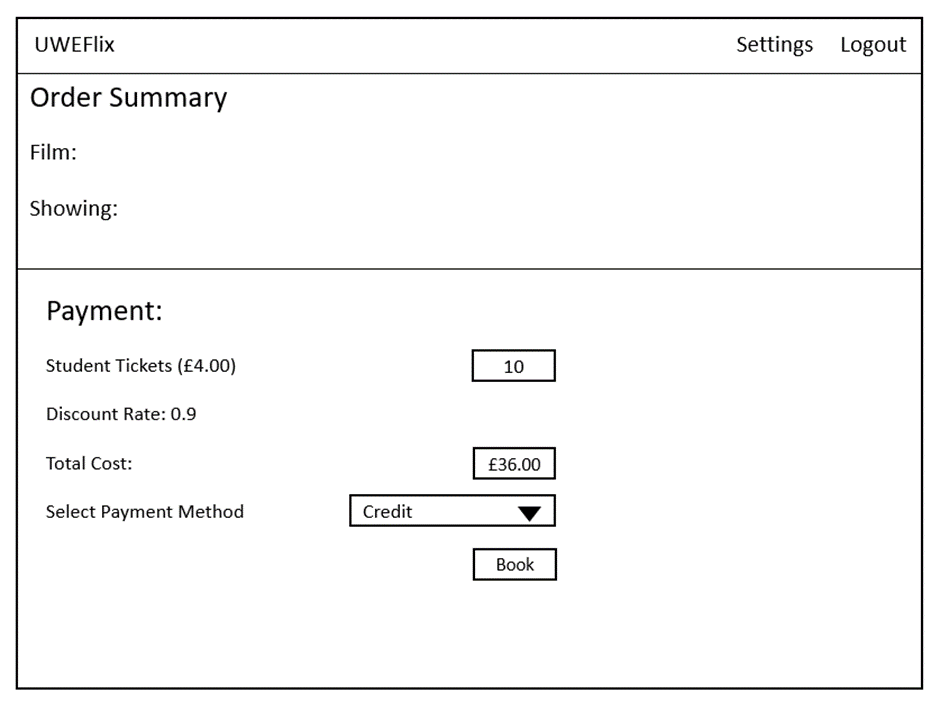
Ticket.newTicket(transaction, showing, “student”)

ENDFOR

ENDIF

END

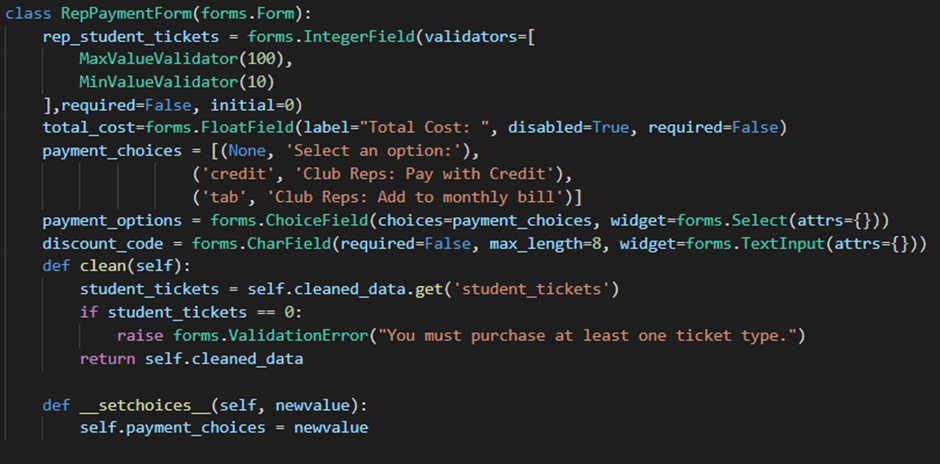
The Wireframe for the payment page would be:



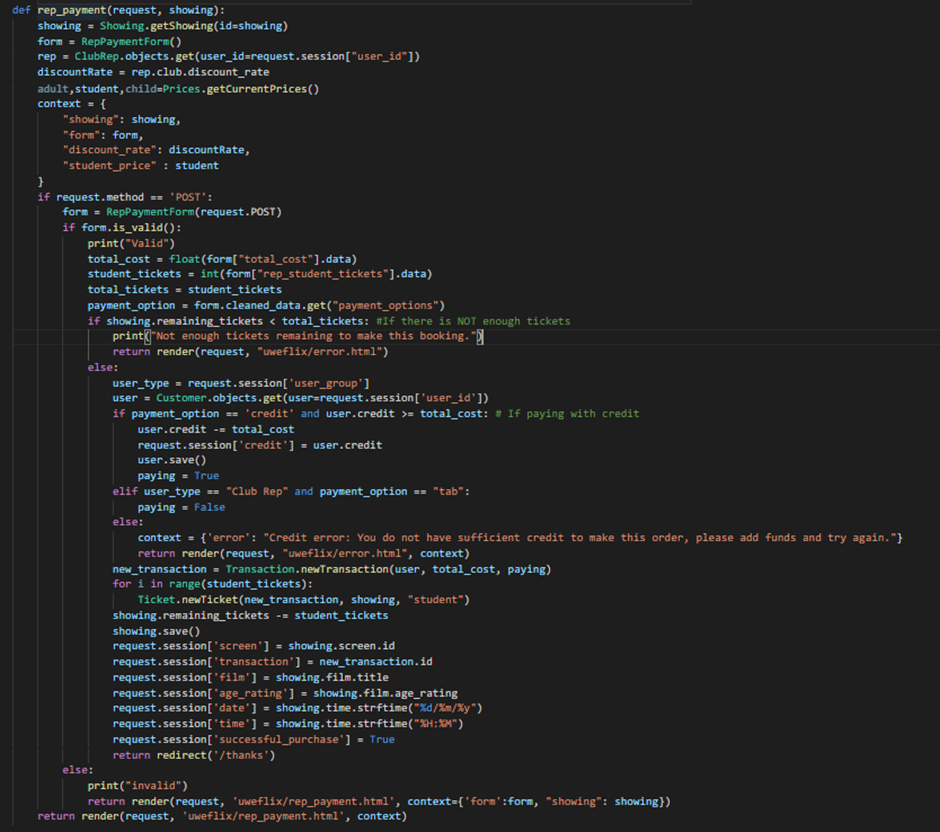
1. **Implementation**

This was successfully implemented as such:

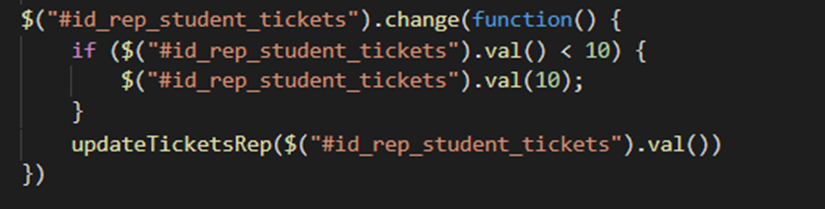
The form:

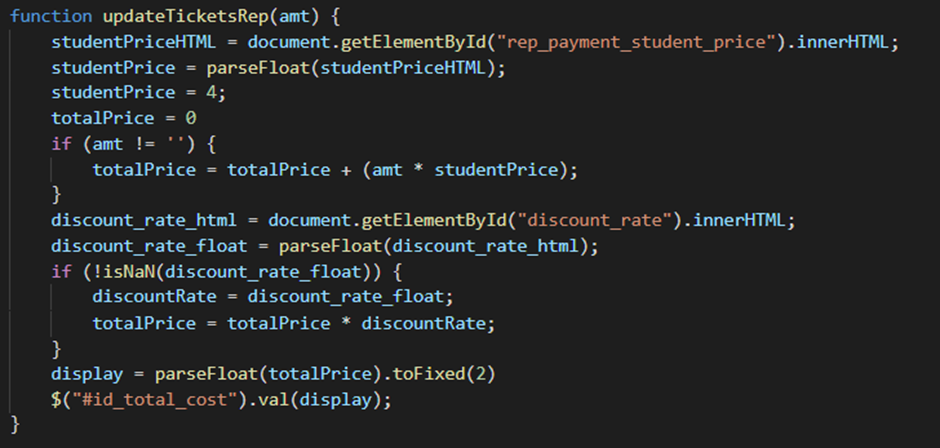


The views.py file:



The JQuery code:





And finally the HTML Code:

{% extends "uweflix/layout.html" %}

{% load static %}

{% block title %}

Rep Payment

{% endblock %}

{% block content %}

<style>

form {

display: inline-block;

}

h1 {

text-align: center;

}

h2, h3, h5, p, input {

padding-left: 15px;

}

</style>

<br/>

<h1>Confirm your Booking</h1>

<br/>

<body>

<h2>Order summary:</h2>

<hr/>

<h3>Film:</h3>

<p>

<b>{{ showing.film.title }}</b><br/>

Duration: {{ showing.film.duration }} minutes<br/>

Age rating: {{ showing.film.age\_rating }}

{% if showing.film.age\_rating == '18' %}

(Please bring ID with you to verify your age!)

{% endif %}

</p>

<br/>

<h3>Showing:</h3>

<p>Screen {{ showing.screen.id }}<br/>{{ showing.time | date:'j M Y' }} at {{ showing.time | time:'G:i'}}</p>

<hr/>

<p>There are currently <b>{{ showing.remaining\_tickets }}</b> tickets available for this showing.</p>

<hr/>

<div class="form">

<form id="payment-form" action="{% url 'rep\_payment' showing.pk %}" method="POST" autocomplete="off">

{% csrf\_token %}

<h3>Payment:</h3>

<div class="alert-error">

{{ form.non\_field\_errors }}

</div>

<div class="form-area">

<br/>

<div class="form-group">

Student Tickets (£<span id="rep\_payment\_student\_price">{{ student\_price | floatformat:2 }}</span>): {{ form.rep\_student\_tickets }}

</div>

<br/>

<p>The following discount rate will be applied:</p>

<p id="discount\_rate">{{ discount\_rate }}</p>

<hr/>

<div class="form-group">

Discount Code: {{ form.discount\_code }}<button id="cancel-discount-btn" type="button" style="display: none;">X</button><button id="discount-btn" type="button"> Apply </button>

</div>

<div class="form-group">

Total Cost: {{ form.total\_cost }}

</div>

<hr/>

<div class="form-group">

Select Payment Method: {{ form.payment\_options }}

</div>

</div>

<hr/>

<div class="form-group">

<input id="confirm-btn" class="btn btn-warning" type="submit" value="Book">

</div>

</form>

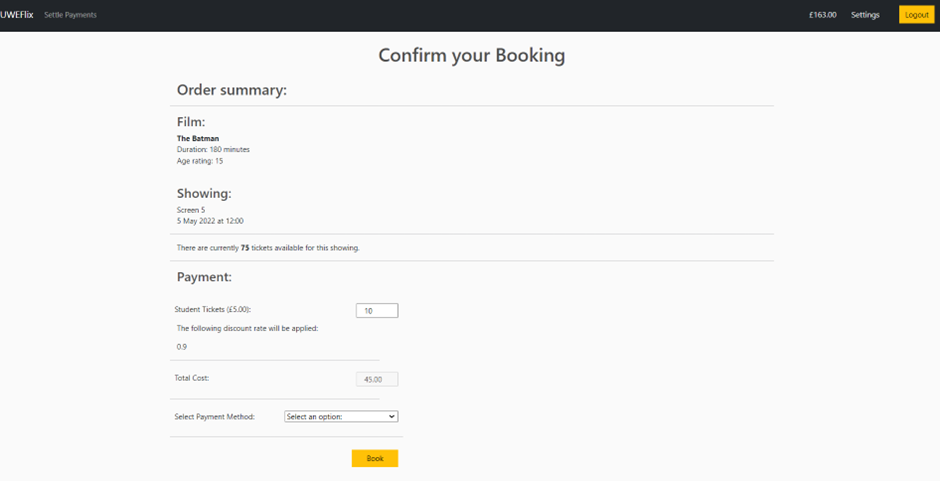
</div>

</body>

{% endblock %}

1. **Demonstration**

The page worked and ended up looking like this:



1. **Evaluation**

The Club Representatives were successfully able to purchase tickets. They had to purchase a minimum of 10 and were able to pay with either account credit, or put it on the monthly bill. Both of these worked successfully. The correct discount rate for the club was also applied to the total.

1. **Club Rep & Student - Request a Cancellation**
2. **Overview**

This task revolved around both Students and Club Reps requesting cancellations for any showings. Per the UWEFlix Case Study, both Students and Club Representatives should be able to request a film cancellation which can then be reviewed by the Cinema Manager who will either accept or deny it.

1. **Requirements**

For this task:

* The users should only be able to request cancellations for showings that are set in the future.
* The users should be able to see the details of their upcoming bookings, such as film name, showing time, what was purchased, and the amount so they are able to ensure they are cancelling the correct booking.
* IF multiple bookings appear, they should be presented in a table, and each booking should have its own cancel button linked to the booking ID which can be processed to request the booking.
* When a booking is cancelled, the option ‘request\_cancel’ in the transaction model should be set to true.

1. **Design**

The main functionality would occur in the view.py file and the pseudocode is as follows:

BEGIN

now AS DateTime Object

transaction\_list = Transaction.GetObject(request\_to\_cancel = False)

showingsToCancel AS Empty List

FOR purchase IN transaction\_list:

ticket = Ticket.GetFirstObject(transaction = purchase)

IF ticket.showing.time > now:

showingsToCancel.append(ticket)

ENDIF

ENDFOR

Template\_Context {showings\_to\_cancel: showingsToCancel}

IF request.POST:

transaction = Transaction.GetObject(id=request.POST[transaction\_number])

transaction.update(request\_to\_cancel = True)

ENDIF

END

The Wireframe for the cancellation page would be:



1. **Implementation**

This was successfully implemented as such:

Views.py file:



HTML Code:

{% extends "uweflix/layout.html" %}

{% load static %}

{% block title %}

Request Cancellation

{% endblock %}

{% block content %}

<style>

.transaction\_list {

margin-left: auto;

margin-right: auto;

}

.transaction\_list th, td {

border: 1px solid;

text-align: center;

padding-right: 15px;

height: 7.5ex;

overflow:hidden;

}

.transaction\_list th {

background-color: lightgrey;

}

h1 {

text-align: center;

}

</style>

<center>

<h1>Request a Cancellation</h1>

<hr/>

{% if zipped %}

<h3>Upcoming Showings:</h3>

<br/>

<table class="transaction\_list">

<thead>

<tr>

<th>Transaction #</th>

<th>Date Purchased</th>

<th>Film Name</th>

<th>Date of Showing</th>

<th>Showing Time</th>

<th>Tickets</th>

<th>Cost</th>

<th>Request Cancellation</th>

</tr>

</thead>

<tbody>

{% for ticket, adult, child, student in zipped %}

<tr>

<td>{{ ticket.transaction.id }}</td>

<td>{{ ticket.transaction.date | date:'d M Y' }}</td>

<td>{{ ticket.showing.film.title }}</td>

<td>{{ ticket.showing.time | date:'d M Y'}}</td>

<td>{{ ticket.showing.time | date:'h:m'}}</td>

<td>

{% if adult > 0 %}

{{ adult }}x Adult Tickets<br>

{% endif %}

{% if child > 0 %}

{{child}}x Child Tickets<br>

{% endif %}

{% if student > 0 %}

{{student}}x Student Tickets

{% endif %}

</td>

<td>£{{ ticket.transaction.cost | floatformat:2 }}</td>

<td>

<form action="{% url 'request\_cancellation' %}" method="post">

{% csrf\_token %}

<button name="transaction\_number" value={{ticket.transaction.id}} type="submit" class="btn btn-warning">Cancel</button>

</form>

</td>

</tr>

{% endfor %}

</tbody>

</table>

{% else %}

<p>No Upcoming Showings</p>

{% endif %}

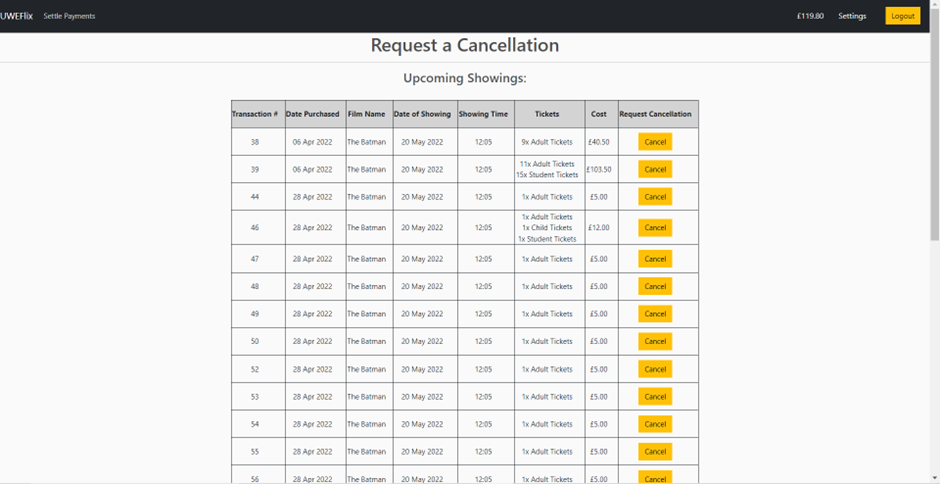
<hr/>

</center>

{% endblock %}

1. **Demonstration**

The page worked and ended up looking like this:



1. **Evaluation**

The Student and Club Representatives were able to see their order history for any UPCOMING showings and had the ability to request a cancellation once the button was pressed. The table showed necessary details of the showings, and if the button was pressed, the showing would be removed and sent to the Cinema Manager to review whether or not the cancellation should be accepted.

1. **Club Rep & Student - View Order History**
2. **Overview**

Students should be capable of viewing their overall account history. This can be achieved by two date pickers that allow Students and Club Representatives to view their purchase history between a selected timeframe.

1. **Requirements**

For this task:

* 2 DatePickers should be implemented as a start and end date, allowing the user to choose dates they would like to see their history between.
* The DatePickers should be implemented with a form that can be read into views.py
* The Date Pickers should be validated. This means that the Start Date should be unable to go after the End Date and vice versa. The Dates should also be unable to be in the future.
* The transactions of a user should be cycled and only those in the certain dates should be fed into the template and viewed.
* The data should be viewed in a table.

1. **Design**

The main functionality would occur in the view.py file and the pseudocode is as follows:

BEGIN

form AS DateIntervalForm

startDate = form[startDate]

endDate = form[endDate]

transaction\_list = Transaction.GetObjects(date\_\_range = (startDate, endDate))

IF NOT transaction\_list:

Template\_Context = {title\_text: “no transactions made in date range}

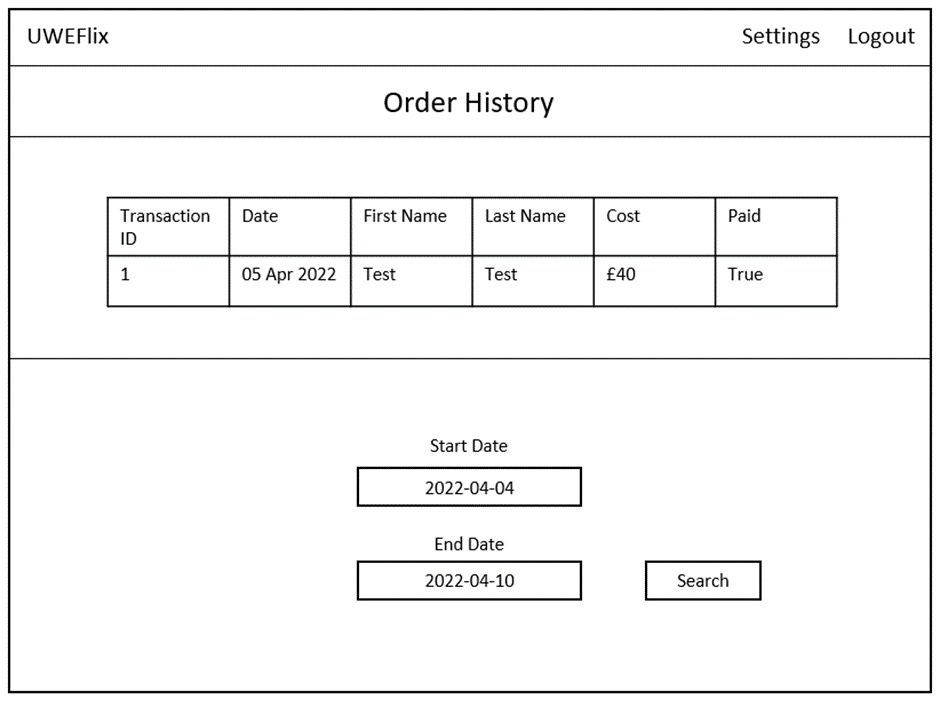
ELSE:

Template\_Context = {start\_date: startDate, end\_date: endDate, transaction\_list: transaction\_list}

ENDIF

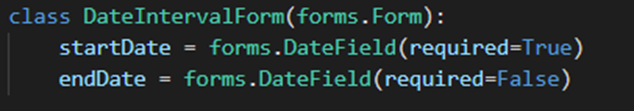
END

The Wireframe for the order history page would be:

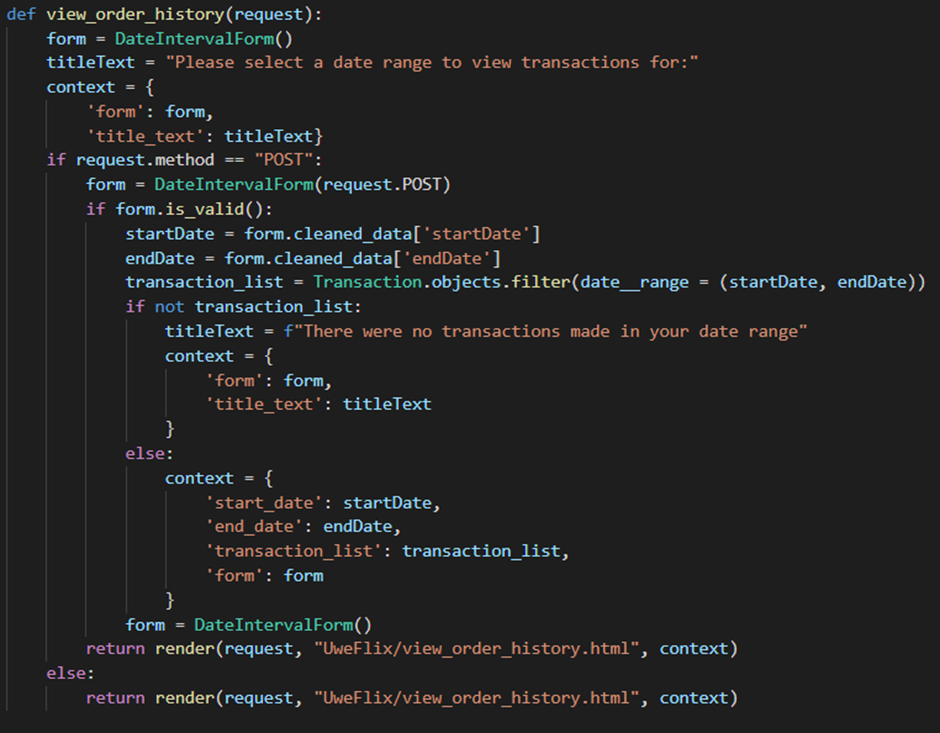


1. **Implementation**

This was successfully implemented with the Form:



The views.py file:



And the HTML Code:

{% extends "uweflix/layout.html" %}

{% load static %}

{% block title %}

Order History

{% endblock %}

{% block content %}

<style>

.transaction\_list {

margin-left: auto;

margin-right: auto;

}

.transaction\_list th, td {

border: 1px solid;

text-align: left;

padding-right: 15px;

}

.transaction\_list th {

background-color: lightgrey;

}

h1 {

text-align: center;

}

</style>

<center>

<h1>Order History</h1>

<hr/>

{% if transaction\_list %}

<h3>All transactions between {{ start\_date | date:'d/m/Y'}} - {{ end\_date | date:'d/m/Y'}}:</h3>

<br/>

<table class="transaction\_list">

<thead>

<tr>

<th>Transaction #</th>

<th>Date</th>

<th>First Name</th>

<th>Last Name</th>

<th>Cost</th>

<th>Paid</th>

</tr>

</thead>

<tbody>

{% for transaction in transaction\_list %}

<tr>

<td>{{ transaction.id }}</td>

<td>{{ transaction.date | date:'d M Y' }}</td>

<td>{{ transaction.customer.user.first\_name }}</td>

<td>{{ transaction.customer.user.last\_name }}</td>

<td>£{{ transaction.cost | floatformat:2 }}</td>

<td>{{ transaction.is\_settled }}</td>

</tr>

{% endfor %}

</tbody>

</table>

{% else %}

<p>{{ title\_text }}</p>

{% endif %}

<hr/>

<script>

$( function() {

$( "#id\_startDate" ).datepicker({

dateFormat: "yy-mm-dd",

maxDate: 0,

onSelect: function(){

$("#id\_endDate").datepicker("option","minDate",

$("#id\_startDate").datepicker("getDate"));

}

});

$( "#id\_endDate" ).datepicker({

dateFormat: "yy-mm-dd",

maxDate: 0,

onSelect: function(){

$("#id\_startDate").datepicker("option","maxDate",

$("#id\_endDate").datepicker("getDate"));

}

});

} );

</script>

<form action="{% url 'view\_order\_history' %}" method="post" autocomplete="off">

{% csrf\_token %}

<div class="form-area">

<div class="form-field">

Start Date: {{ form.startDate }}

<br/>

End Date: {{form.endDate}}

<br/>

</div>

<button type="submit" class="btn btn-warning">Search</button>

</div>

<br/>

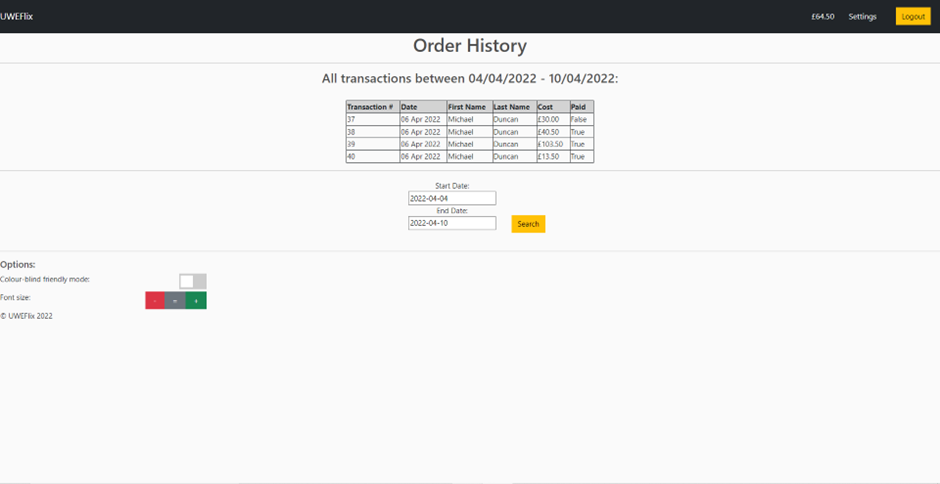
</form>

</center>

{% endblock %}

1. **Demonstration**

The page worked and ended up looking like this:



1. **Evaluation**

Students and Club Representatives were able to select 2 dates with the date picker forms. These date pickers were successfully validated to ensure the Start Date did not cross the End Date and vice versa. Once the ‘search’ button was pressed, the respective transactions were placed.

**Matt Hill - Tasks**

**1 – Customer Functionality – Pay for Tickets – Status DONE**

**a. Overview of task**

This task involves creating the logic and a basic template for creating a payment page for card details, for a customer to make payment for tickets, this page will be used by customers looking to make a payment by card, it will have the relevant fields to be able to add in card details such as, the 16-digit card number, along with expiry month and year.

**b. Requirements**

- Must be able to take card payments

- Must be able to take in 16-digit card number

- Must be able to take in card expiry month and year

- Must be able to take payment

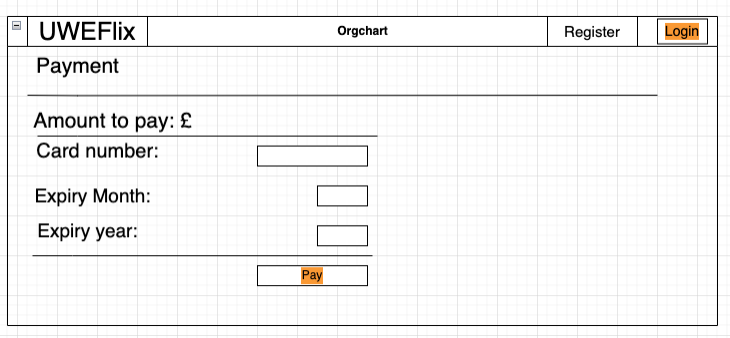
- The form must include all relevant fields.

- Once created the system should tell you that payment has been successful

- The form must alert the user if invalided details have been implemented.

**c. Design**

**Wireframe for the Pay for Tickets:**

****

**d. Implement**

**Code Snippet of the Forms.py Logic:**

****

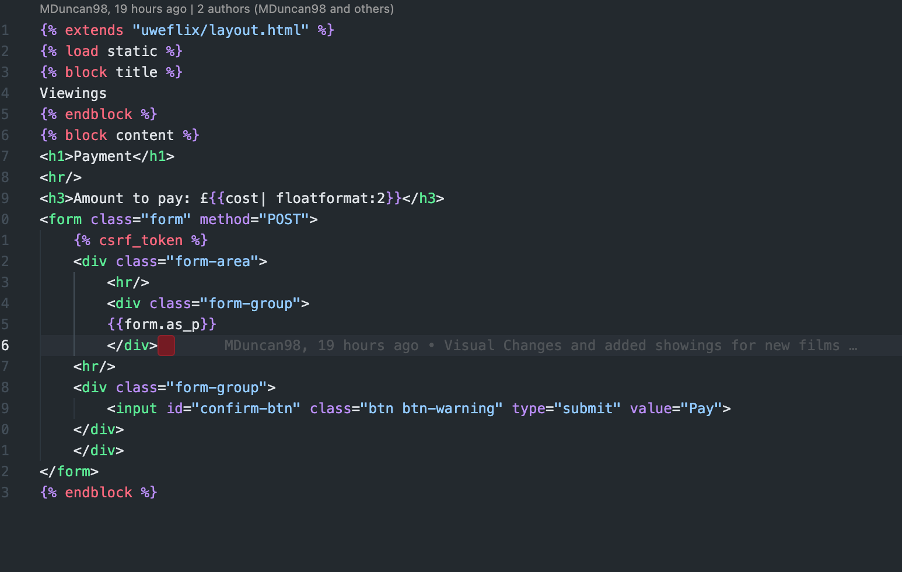
**Code Snippet of the Views.py Logic:**

****

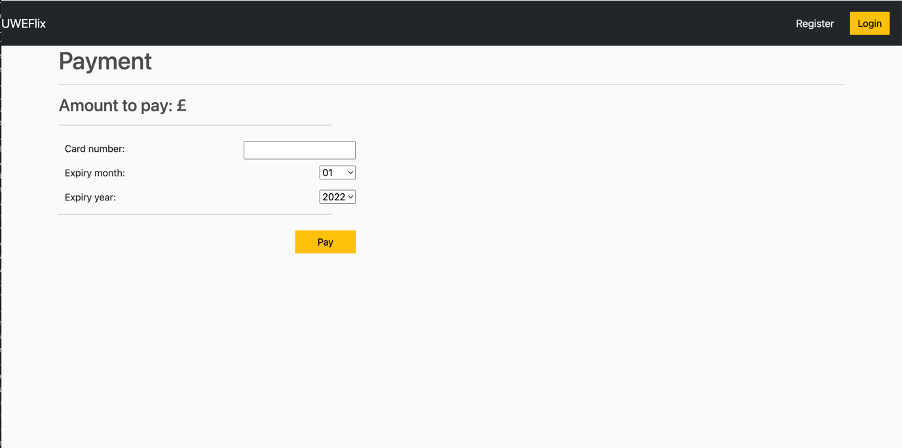
**Code Snippet of the Urls.py Logic:**

****

**The Template was implemented through HTML and was created using this code:**

****

**Picture of the end result below:**

****

**e. Testing**

**Please refer to tests in the Testing Document (Link in ‘Relevant Links’).**

**f. Reflections**

The overall implementation went well with no issues, It has been tested thoroughly and works as is expected. Therefore, I can confidently say that this task has been delivered and met the requested design specification. The end product is still a little simple, the UI experience could be worked on and improved as well as adding further functionality.

**2 - User Interface – Increase/Decrease Text Size – Status DONE**

**a. Overview of task**

As explained in the specification, the user interface must have some sort of accessibility features to allow and cater for all user groups. This task involves creating the ability to increase and decrease the text size to help users with visual impairment. Therefore, this task will implement a add and minus feature in the footer of the layout page which will change the font size of the website to allow those in question to see the content much easier.

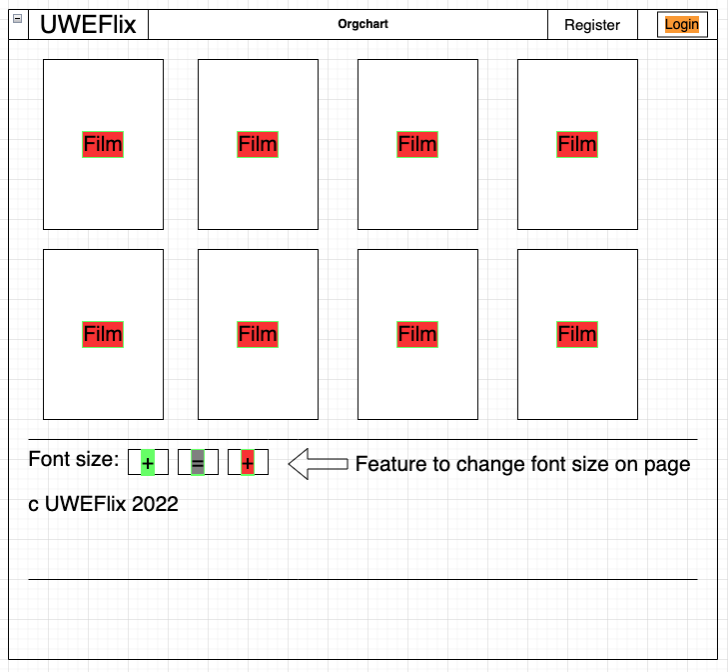
**b. Requirements**

There must be a setting on all pages that allows for the the visually impaired to be able to increase or decrease the font size to allow the website to be more visually friendly to users and easier to see

The effectiveness of the Font size will be dependent on the user and will have no upper or lower limit, but it will have a = button to allow users to revert to the original start size with ease.

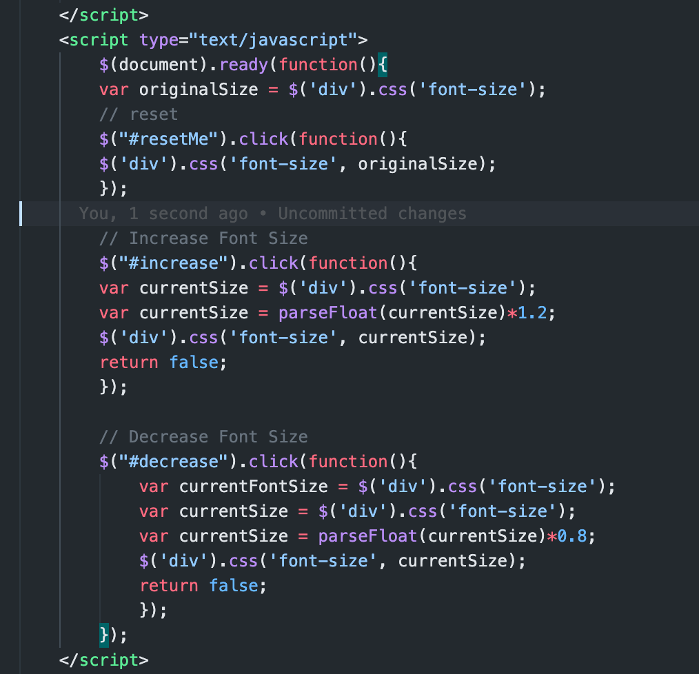
**c. Design**

**Wireframe for font sizing on page:**

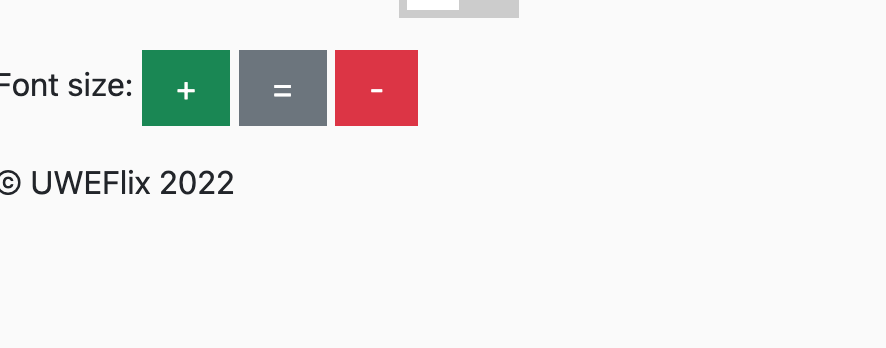
****

**d. Implement**

**Snapshot of the code in the Layout.html file.**

****

**Snapshot of the Font Size buttons result on the main Page:**

****

**e. Testing**

**Please refer to tests** [101-103] **in the Testing Document.**

**f. Reflections**

The overall implementation went well with no issues, It has been tested thoroughly and works as is expected. Therefore, I can confidently say that this task has been delivered and met the requested design specification. The end product is still a little simple, the UI experience could be worked on and improved as well as adding further functionality.

**3 – Cinema Manager Functionality – Pricing Tickets – Status DONE**

a. Overview of task

This task involves creating the logic and the ability for the Cinema Manager to have the functionality to be able to set the price of the tickets and change it if needed, this will be added to the already existing Cinema Manager page as an added feature.

b. Requirements

- Once signed in, the Club Manager (**only**) should be able to change/alter the prices of the tickets.

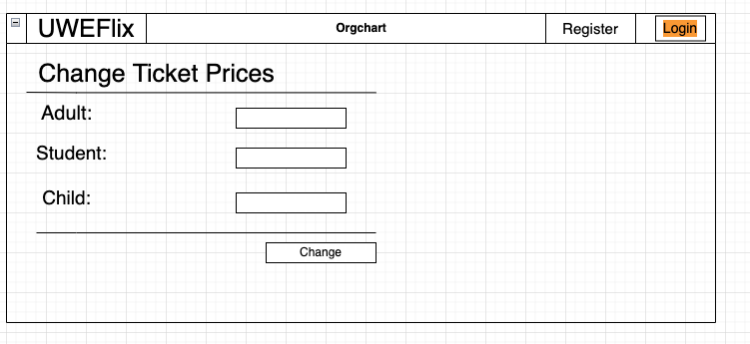
- The ability to change and alter each individual ticket, i.e., Adult, Student and Child.

- Once altered, be able to store the new ticket price.

- The ability to revert to the original price.

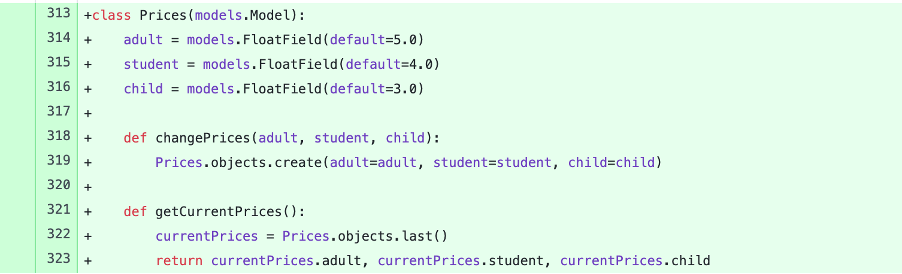
c. Design

Wireframe for the change ticket Prices:

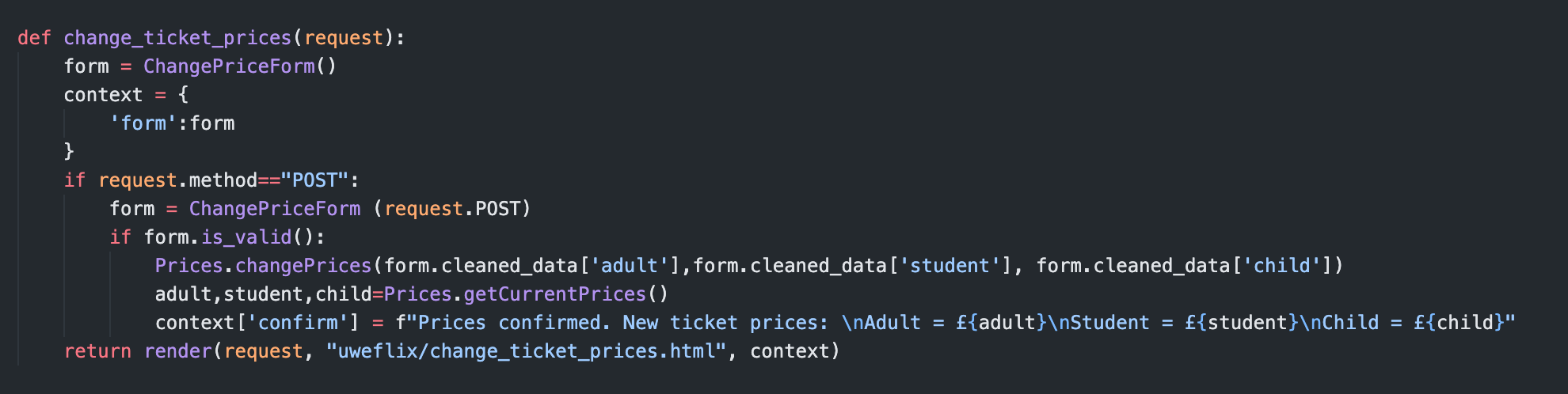


d. Implement

Code snippet of the Models.py logic:



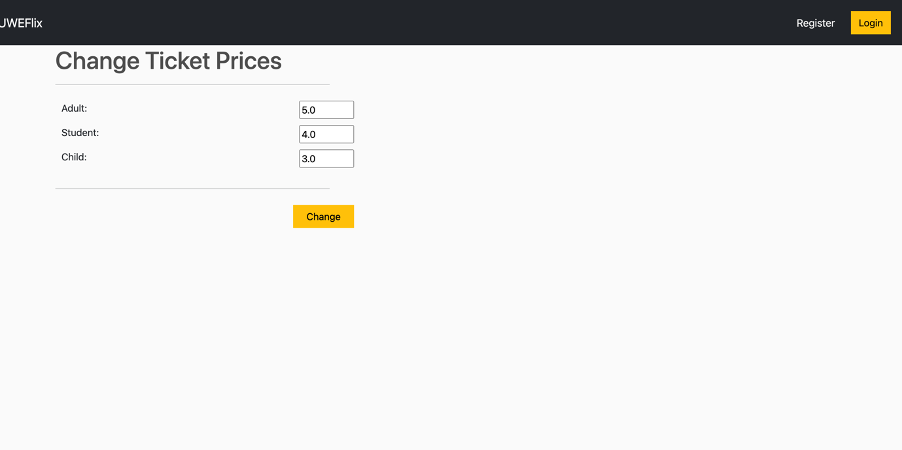
Code snippet of the Views.py logic:



Code snippet of the Urls.py logic:



Picture of the result below:

****

e. Testing

**Please refer to tests** [101-103] **in the Testing Document.**

f. Reflections

The overall implementation went well with no issues, it has been tested thoroughly and works as is expected. Therefore, I can confidently say that this task has been delivered and met the requested design specification. The product is still a little simple, the UI experience could be worked on and improved as well as adding further functionality.

**Ross Williams - Tasks**

**1. Cinema Manager – Deleting Obsolete Films (View Logic and Template)**

1. **Overview**

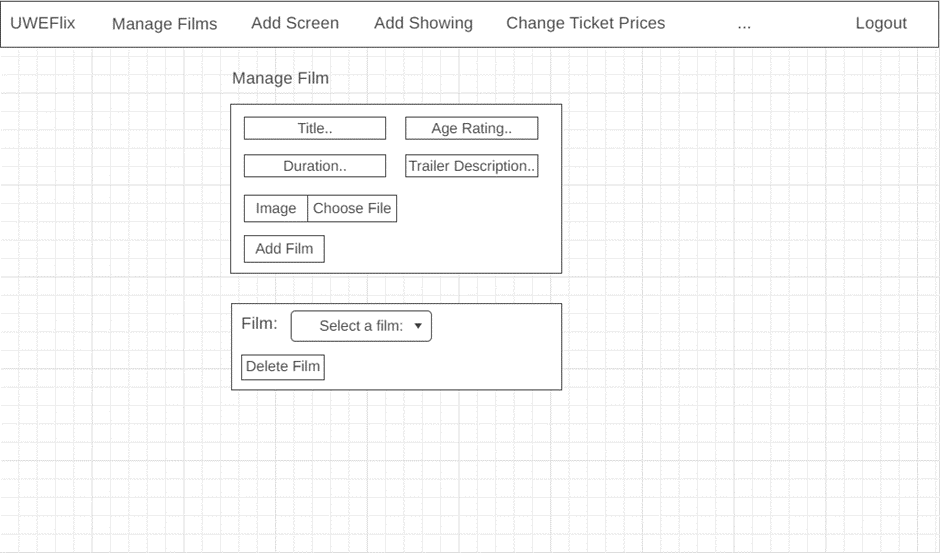
This task will implement the functionality of deleting films, this is added to allow the Cinema Manager control of this action. Upon deletion, any showings for that film will also be deleted.

1. **Requirements**

* The Cinema Manager must be able to delete obsolete films
* A dropdown menu listing all films should be available for selection.
* When the ‘Delete Film’ button is pressed, the selected film must be deleted.

1. **Design**

**A wireframe of the proposed design:**

****

1. **Implementation**

**HTML code:**

{% extends "uweflix/layout.html" %}

{% block title %}

Add a film

{% endblock %}

{% block content %}

<h1>Add Film</h1>

<div class="row">

<div class="col-lg-6">

<div class="box-element" id ="form-wrapper">

<form id="form" method="POST" enctype="multipart/form-data">

{% csrf\_token %}

<div id="film-info">

<div class="form-field">

<input required class="form-control" type="text" name="title" placeholder="Title.." maxlength="100">

</div>

<div class="form-field">

<input required list="ages" class="form-control" name="age\_rating" placeholder="Age Rating..">

</div>

<datalist id="ages">

<option value="U">

<option value="PG">

<option value="12">

<option value="12A">

<option value="15">

<option value="18">

</datalist>

<div class="form-field">

<input required class="form-control" type="text" name="duration" placeholder="Duration.." maxlength="3">

</div>

<div class="form-field">

<input required class="form-control" type="text" name="trailer\_desc" placeholder="Trailer Description.." maxlength="500">

</div>

<div class="form-field">

(Optional) Image: <input class="form-control" type="file" accept="image/\*" name="image">

</div>

</div>

<hr>

<input type="submit" value="Add Film">

</form>

</div>

<div class="box-element" id ="form-wrapper">

<form id="form" method="POST">

{% csrf\_token %}

{{ form.as\_p }}

<hr>

<input type="submit" name="delete\_film" value="Delete Film">

</form>

</div>

</div>

</div>

{% endblock %}

**Views.py code:**

def add\_film(request):

form = deleteFilmForm()

context = {"form":form}

if request.method == "POST":

ages = {"U", "PG", "12", "12A", "15", "18"}

title = request.POST.get('title')

age\_rating = request.POST.get('age\_rating')

duration = request.POST.get('duration')

trailer\_desc = request.POST.get('trailer\_desc')

if (duration.isdigit()):

if(age\_rating in ages):

film = Film()

film.title = title

film.age\_rating = age\_rating

film.duration = duration

film.trailer\_desc = trailer\_desc

try:

image = request.FILES.get('image')

film.image = image

except:

pass

film.save()

else:

print("Invalid Age Rating")

else:

print("Duration is not a valid number")

if 'delete\_film' in request.POST:

form = deleteFilmForm(request.POST)

if form.is\_valid():

film\_id = form.cleaned\_data['film']

Film.removeFilm(film\_id)

context['form'] = form

return render(request, 'uweflix/add\_film.html', context)

1. **Testing**

Please refer to [90-91] in the Testing Document.

1. **Evaluation**

Overall, the functionality of this task was implemented well. It covers the requirements of deleting a film. There was slight trouble with managing multiple forms on one page, use of Django’s forms made this much easier. The styling of the page is quite basic and could’ve been improved further, however due to time constraints this wasn’t looked at.

**2. Cinema Manager – Adding New Screen (View Logic and Template)**

1. **Overview**

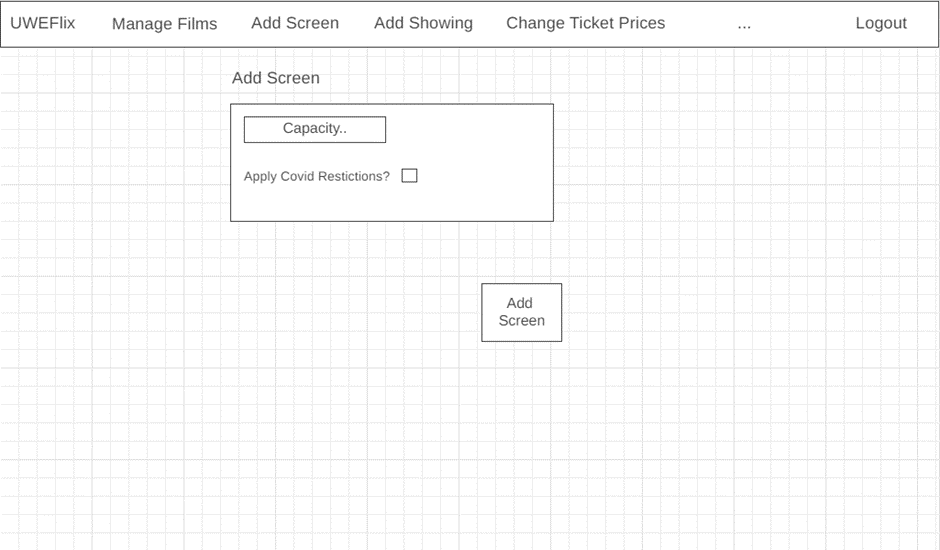
This task will implement the functionality of adding a new screen, this is added to allow the Cinema Manager control of this action. The Cinema Manager can set a capacity for the screen and also apply covid restrictions. This task involves creation of a template and view logic.

1. **Requirements**

* The Cinema Manager must be able to add a new screen, including the capacity of the screen and whether covid restrictions apply.
* When the ‘Add Screen’ button is pressed, the screen is added.

1. **Design**

A wireframe of the proposed design:

****

1. **Implementation**

**HTML code:**

{% extends "uweflix/layout.html" %}

{% load static %}

{% block title %}

Add a screen

{% endblock %}

{% block content %}

<h1>Add Screen</h1>

<style>

.submit-button {

margin: 10px 10px 10px 10px;

}

</style>

<form method="post" class="form-area" autocomplete="off">

<hr>

{% csrf\_token %}

{{ form.as\_p }}

<hr>

<button id="confirm-btn" type="submit" class="btn btn-warning">Add Screen</button>

</form>

{% endblock %}

Views.py code:

def add\_screen(request):

context = {}

form = addScreenForm

if request.method == "POST":

form = addScreenForm(request.POST)

if form.is\_valid():

capacity = form.cleaned\_data['capacity']

covid\_restrictions = form.cleaned\_data['apply\_covid\_restrictions']

Screen.newScreen(capacity, covid\_restrictions)

context['form'] = form

return render(request, 'uweflix/add\_screen.html', context)

1. **Testing**

Please refer to [92-93] in the Testing Document.

1. **Evaluation**

Overall, the functionality of this task was implemented well. It covers the requirements of adding a screen. The styling of the page is quite basic and could’ve been improved further, however due to time constraints this wasn’t looked at.

**3. Cinema Manager – Add Showing (View Logic and Template)**

1. **Overview**

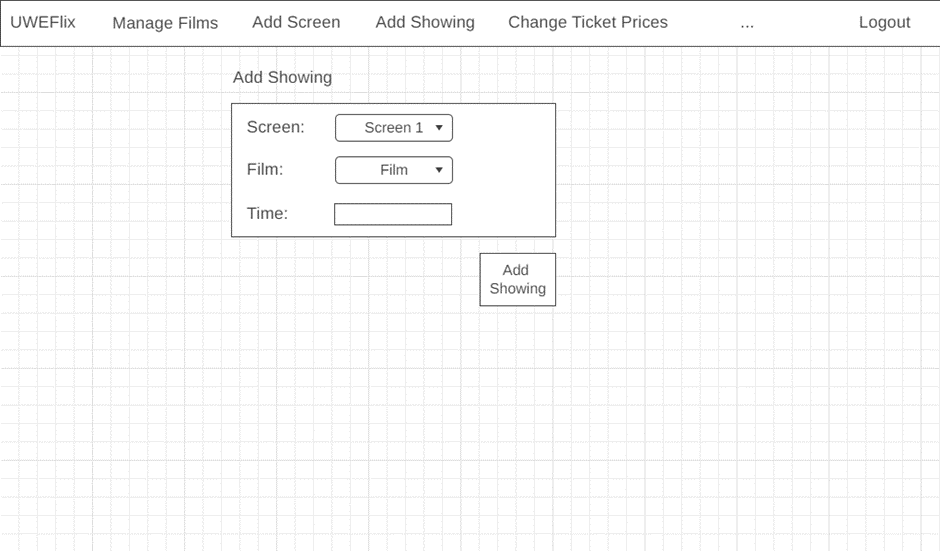
**This task will implement the functionality of adding a new showing, this is added to allow the Cinema Manager control of this action. The Cinema Manager can select a screen, a film to shown and a time to be shown. This task involves creation of a template and view logic.**

1. **Requirements**

* The Cinema Manager must be able to add a new showing, including what screen, what film and what time to be shown.
* When the ‘Add Showing’ button is pressed, the showing is added.

1. **Design**

A wireframe of the proposed design:



1. **Implementation**

**HTML code:**

{% extends "uweflix/layout.html" %}

{% load static %}

{% block title %}

Add a showing

{% endblock %}

{% block content %}

<h1>Add Showing</h1>

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/jqueryui/1.12.1/jquery-ui.min.css" />

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/jquery-ui-timepicker-addon/1.6.3/jquery-ui-timepicker-addon.min.css" />

<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/jqueryui/1.12.1/jquery-ui.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery-ui-timepicker-addon/1.6.3/jquery-ui-timepicker-addon.min.js"></script>

<script>

jQuery(function($) {

$("#id\_time").datetimepicker();

});

</script>

<form method="post" class="form-area" autocomplete="off">

<hr>

{% csrf\_token %}

{{ form.as\_p}}

<hr>

<button id="confirm-btn" type="submit" class="btn btn-warning">Add Showing</button>

</form>

{% endblock %}

Views.py code:

def add\_showing(request):

context = {}

form = addShowingForm

if request.method == "POST":

form = addShowingForm(request.POST)

if form.is\_valid():

screen = form.cleaned\_data['screen']

film = form.cleaned\_data['film']

time = form.cleaned\_data['time']

Showing.newShowing(screen,film,time)

context['form'] = form

return render(request, 'uweflix/add\_showing.html', context)

1. **Testing**

Please refer to [94-95] in the Testing Document.

1. **Evaluation**

Overall, the functionality of this task was implemented well. It covers the requirements of adding a showing. Implementation of the datetime picker was quite tricky as I have not used JavaScript or its libraries before. The styling of the page is quite basic and could’ve been improved further, however due to time constraints this wasn’t looked at.

**4. Base Layout - Footer (Content and Styling)**

1. **Overview**

This task will style the footer and add contents to make the site more user-friendly.

1. **Requirements**

* The footer should be styled and contain user-friendly options for colour-blind and visually impaired users.

1. **Design**

A wireframe of the proposed footer:



1. **Implementation**

**HTML code:**

<div class="footer">

<footer class="navbar navbar-expand-lg navbar-dark bg-dark">

<hr/>

<div class="form-area">

<p class="navbar-brand">Options:</p>

<p style="color: lightgray;">Colour-blind friendly mode:

<label class="switch">

<input type="checkbox" id="colourBlindBox" onclick="setColourBlindOptions()">

<span class="slider round"></span>

</label>

</p>

<p style="color: lightgray;">Font size:

<input type="button" id="increase" class="btn btn-success" value=" + ">

<input type="button" id="resetMe" class="btn btn-secondary" value=" = ">

<input type="button" id="decrease" class="btn btn-danger" value=" - "/></p>

<p class="navbar-brand">© UWEFlix 2022</p>

</div>

</footer>

</div>

1. **Testing**

No testing needed for styling of footer, for functionality of the content see other tests.

1. **Evaluation**

Overall, this task went well, the styling of this footer matches other areas of the website (navbar) and its contents are easily accessible for users.

**Benedict Ramage-Mangles - Tasks**

1. Discount for clubs
   1. Template
   2. View Logic
   3. Account Manager functionality
2. Amend Accounts
   1. Template
   2. View Logic
   3. Account Manager functionality

**References**

Tableau (2022) *5 tips on designing colour-blind friendly visualizations.* Available from: <https://www.tableau.com/en-gb/about/blog/examining-data-viz-rules-dont-use-red-green-together#:~:text=Use%20a%20colour-blind-friendly%20palette%20when%20appropriate&text=For%20example%2C%20blue%2Forange%20is,blue%20to%20someone%20with%20CVD>. [Accessed 02nd May 2022].

**Appendix**