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

# **Sprint Review Form**

| Group: | 18 |
| --- | --- |
| Sprint: | 5 |
| Members: | Benedict Ramage-Mangles, Ross Williams, Arjun Binning, Michael Duncan, Matthew Hill |

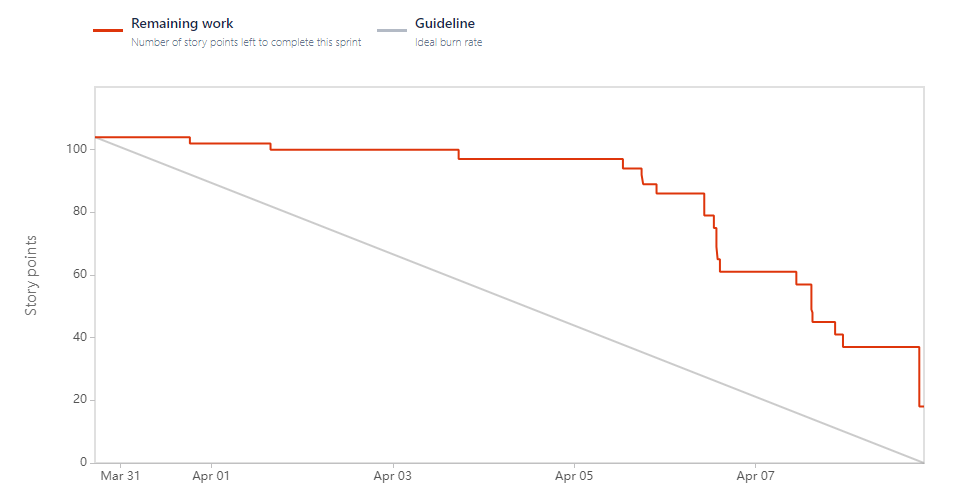
<<Sprint Planning>>

Sprint 5 Requirements:

**From the spec:**

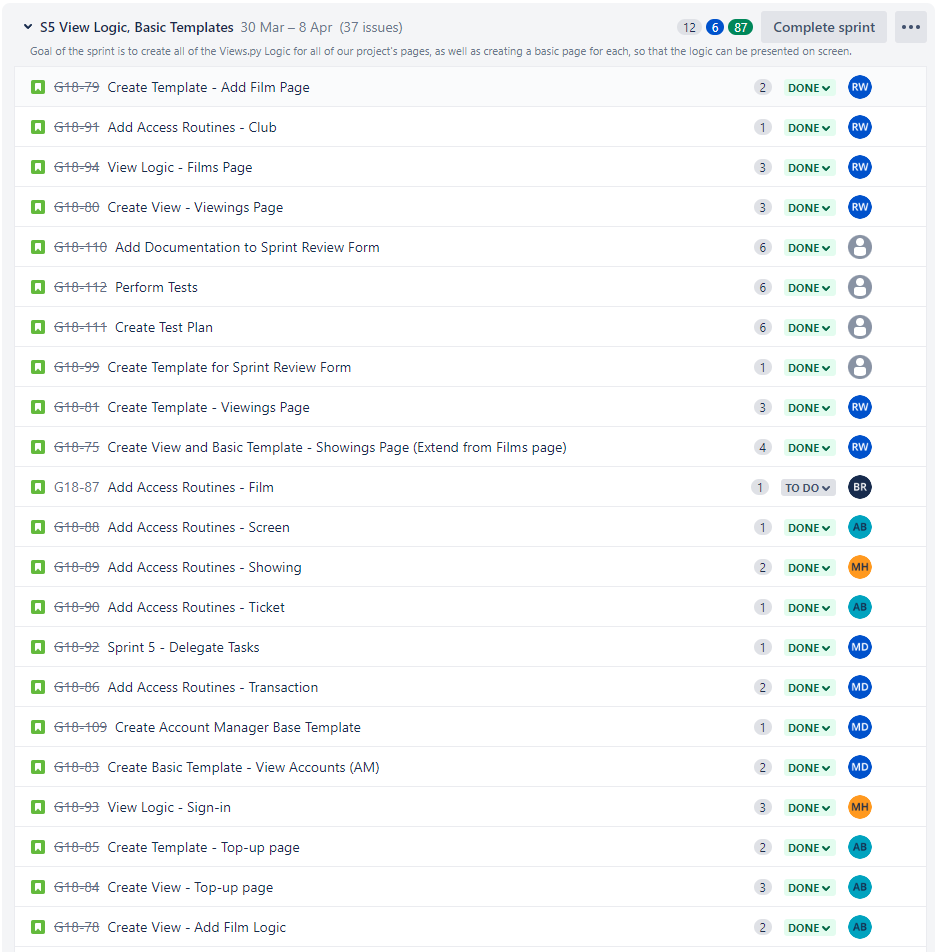
Completion of backend architecture and partial completion of front-end user interface. This requires, at least, half of the tasks to be completed. These tasks are expected to be completion of all model classes, some templates including Forms and building views.py. This sprint covers **Week 9** and **10**, its review is required at the end of Week 10, the demo and review will be assessed on **Week 11** in the same way as the previous sprint. This sprint weighs 30% of the overall mark.

<<Burn-down Chart>>



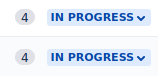
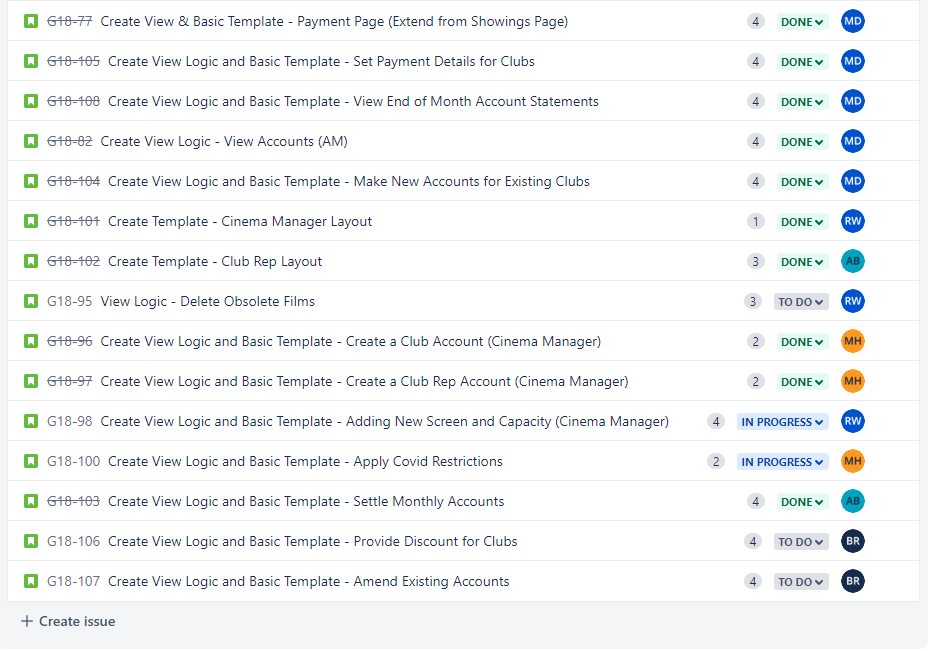
<<Backlog list>>

Sprint Backlog:

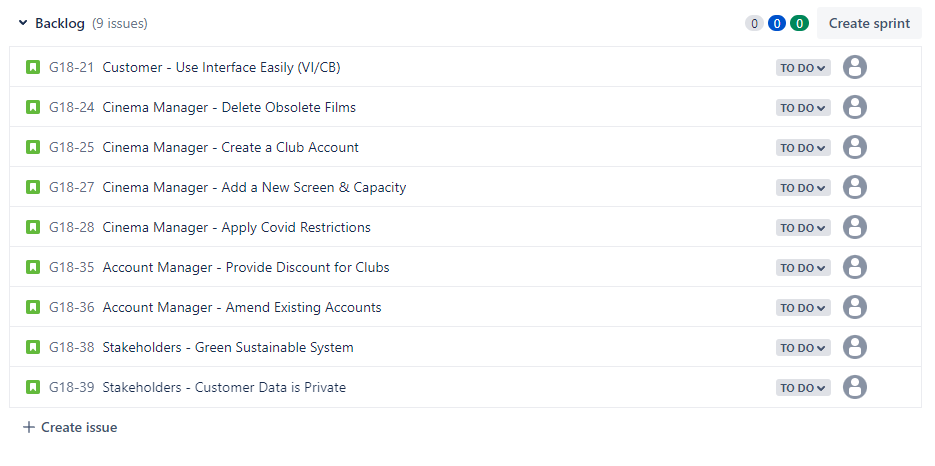


*Continued on Next Page…*

Sprint Backlog cont.



Product Backlog:



<<Communication Issues>>

No Communication issues to report. Everyone has attended weekly meetings and has communicated well.

<<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>

**TASKS TO BE DONE:**

Each task will include:

* Name
* Description
* Planning/Design
* Documentation
* Reflections

**Matthew Hill - Tasks**

**1 - Add Access Routines - Showing - Status DONE**

1. **Overview**

This task involves adding backend functionality to the models themselves, to handle logic (particularly CRUD - Create, Read, Update, Delete) which can be called from the views.py class to present business logic on the front end.

1. **Requirements**

*If unsure about any of the Showing model ‘required arguments or parameters’, please see the previous sprint report regarding the design of this model, or refer to the models.py file in the UWEFlix project.*

The Showing model must:

* Have a function to **create** a brand new Showing object, if given all of the non-null parameters as arguments. If they are all valid, the object can be created and returned.
* **Retrieve/Read** an existing object. This will be done using the object’s id, as we can guarantee that it is unique and therefore will return one and only one object, the one that we require.
* **Update** an existing object. This can take one or more of the required arguments from the model, depending on how many require updating. This function should change the old values with the ones given, and return the object.
* **Delete** an existing object. Given the id as an argument, of which if it exists, will be deleted from the database.

1. **Design**

Pseudocode for the required functions:

**Create**

BEGIN

screen AS Customer object

film AS Customer object

time AS Today's time

remaining \_tickets AS int

TRY

CREATE new Showing object(screen, film, ticketsLeft, socialDis)

RETURN Showing

CATCH

PRINT “Error - Showing object could not be created!”

END

**Retrieve/Read**

BEGIN

Id AS integer

TRY

GET Showing object from model with PK=id

RETURN Showing

CATCH

PRINT (“Error - Showing exists with that showing ID!”

END

**Update**

BEGIN

Id AS Integer

showing\_data AS Optional Arguments List

TRY

GET Showing object from model with PK=id

FOR each item in transaction\_data

IF item is a screen

UPDATE Showing.screen WITH item

ENDIF

IF item is a Date

UPDATE Showing.film WITH item

ENDIF

IF item is a Float

UPDATE Showing.ticketsLeft WITH item

ENDIF

IF item is a Boolean

UPDATE Showing.socialDis WITH item

ENDIF

ENDFOR

RETURN Showing object PK=id

CATCH

PRINT “Data item does not conform to any of the required input types/ this value could not be updated”

END

**Delete**

BEGIN

Id AS Integer

TRY

GET Showing object from model WHERE PK=id

DELETE Showing object

PRINT “Showing deleted successfully.”

CATCH

PRINT “Showing not found, or could not be deleted!”

END

1. **Implement**

*Code snippet from finished implementation:*

**

1. **Testing**

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

1. **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.

**2 - Create View Logic and Basic Template - Create a Club Account (Cinema Manager) - Status DONE**

1. **Overview**

This task involves creating the view logic and a basic template for creating a club account. This page will be used by the Cinema Manager to create a club account. On this page the Cinema Manager can enter the title, card number, expiry date, discount rate and the club. The ‘Add account’ button can be clicked, all the information in the fields will be added.

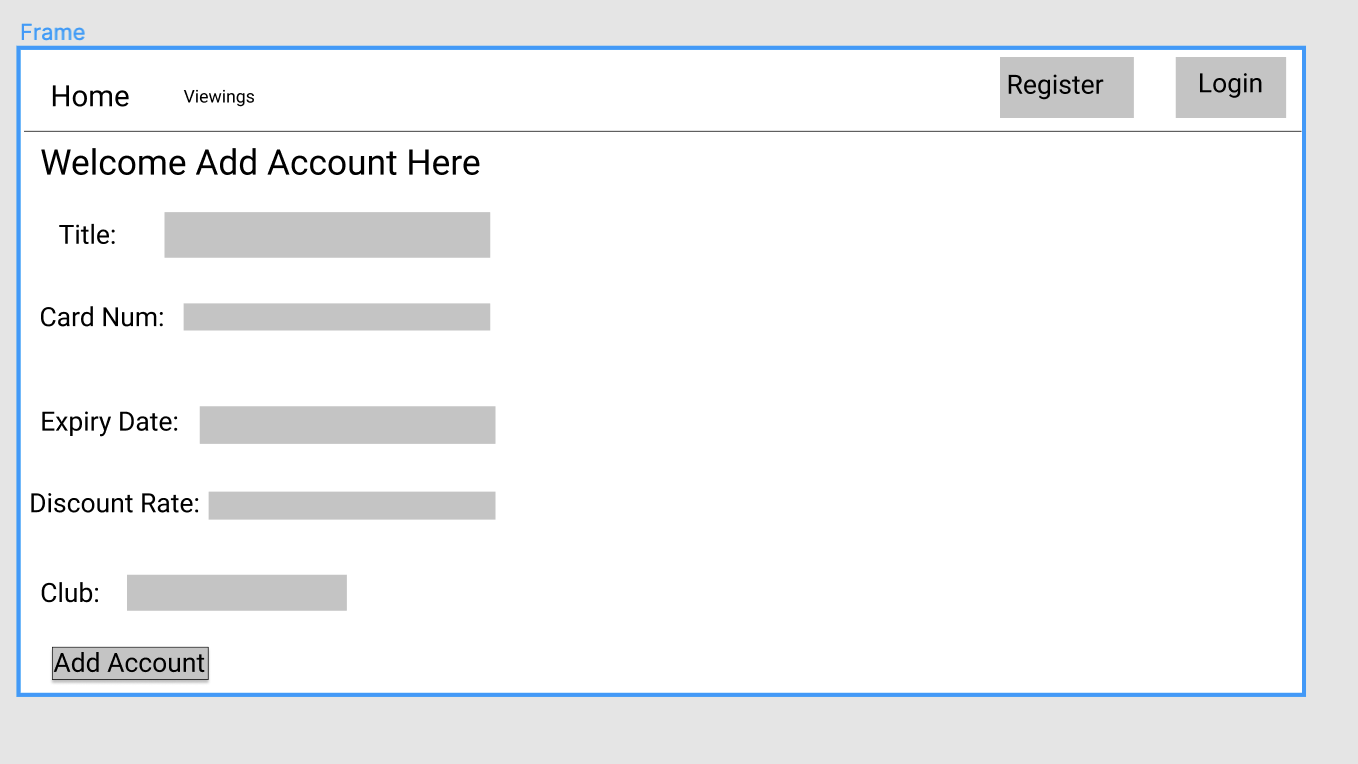
1. **Requirements**

* Provide Cinema Manager the functionality to set up and create a club account
* Form must include all relevant fields.
* The Rep ID and password should be unique and randomly generated
* Once created the system should tell you that a club account has been successfully created.

(As this is only a template, not all of the requirements for the functionality are listed )

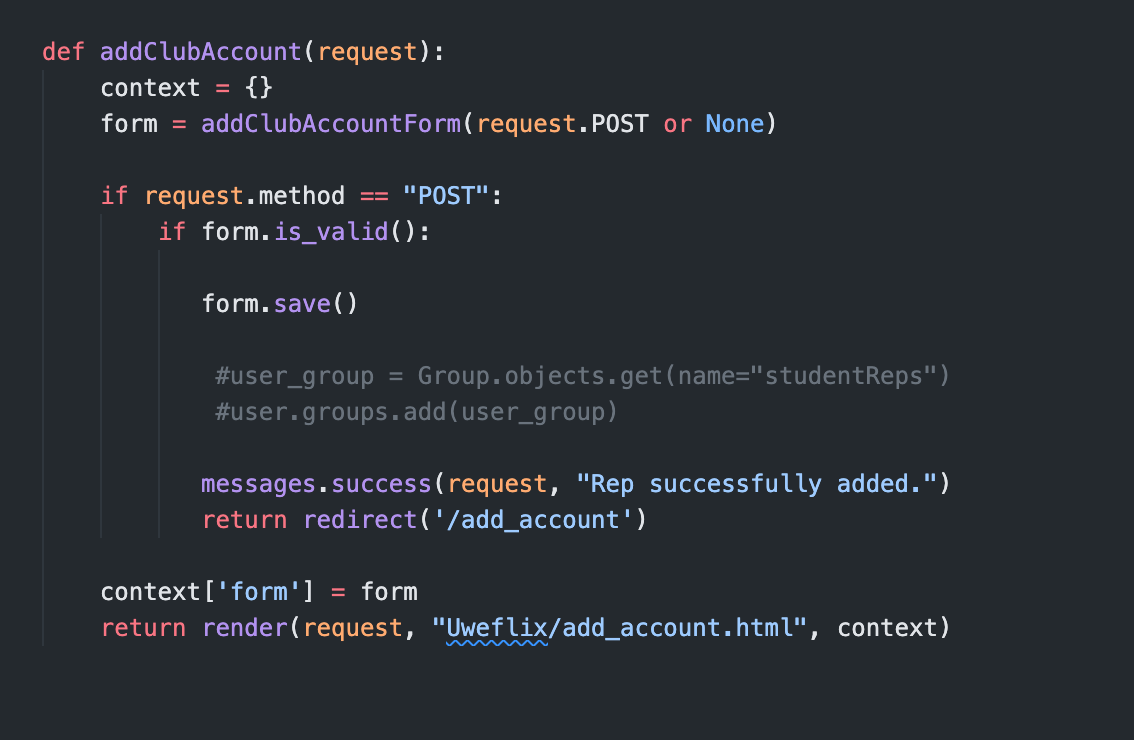
1. **Design**

Wireframe for the Club Account:

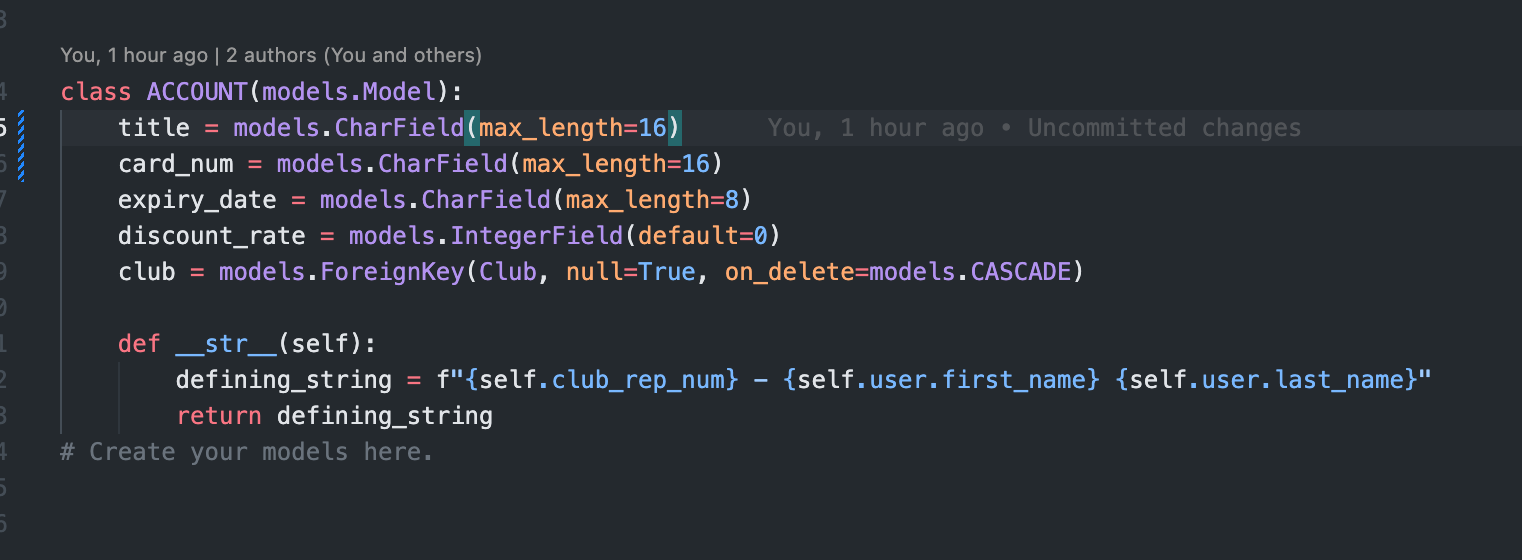


1. **Implementation**

Code Snippet of Views.py logic:

****

Code snippet Model.py logic



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

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

{% load static %}

{% block title %}

Add Rep

{% endblock %}

{% block content %}

<h2>Welcome Add Rep Here<h2>

<form method="post" class="form-group">

{% csrf\_token %}

{{ userform.as\_p }}

{{ form.as\_p }}

<button type="submit" class="btn btn-success">Add Rep</button>

</form>

{% endblock %}

**Here is the snippet in use once converted (Cinema Manager Home Page):**

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

{% load static %}

{% block title %}

Cinema Manager Home Page

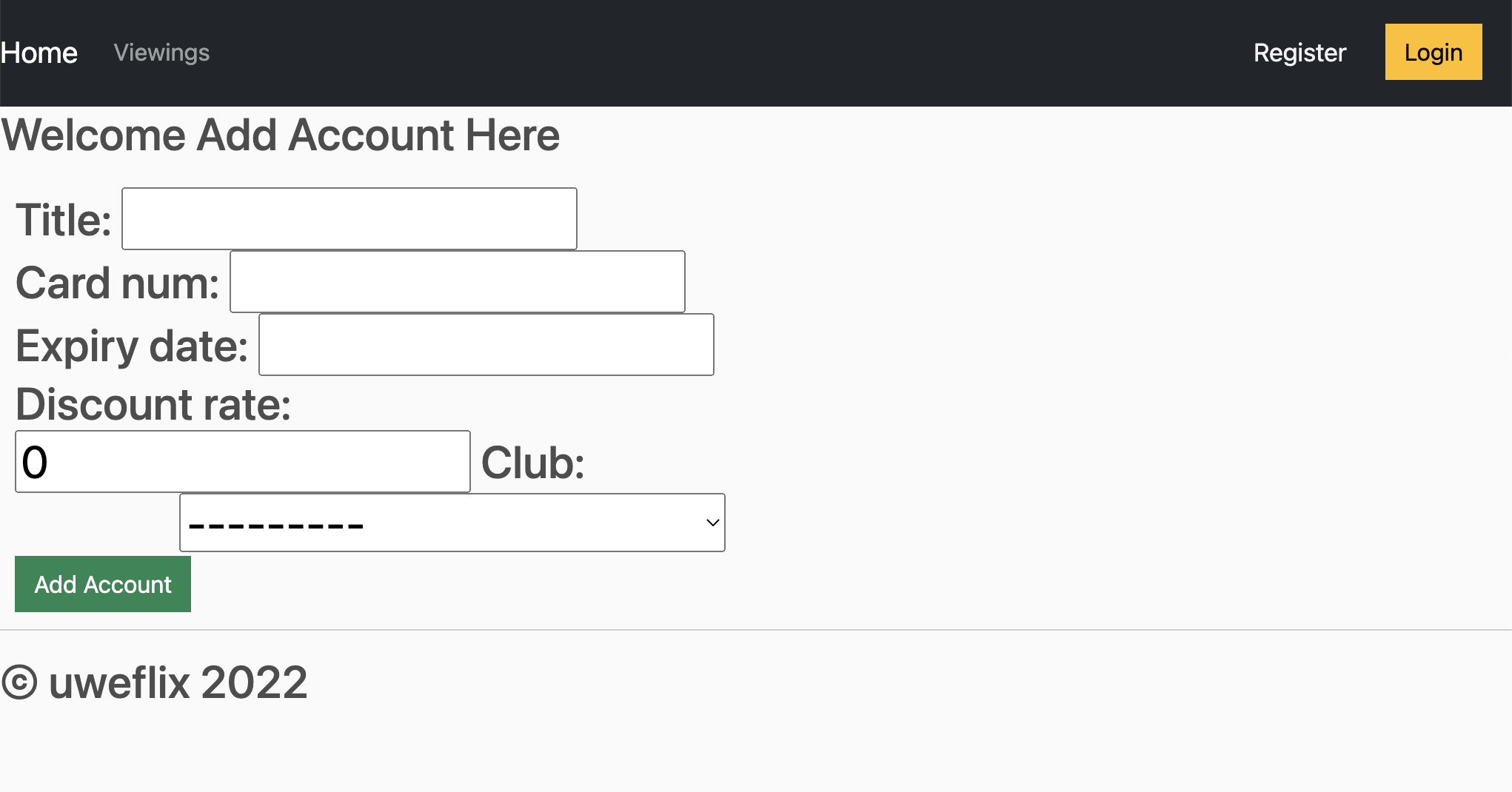
{% endblock %}

{% block content %}

<h2>Welcome Account Manager</h2>

{% endblock %}

Picture of end result below:



1. **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 will be worked on in Sprint 6 as well as other further functionality.

**3 - Create View Logic and Basic Template - Create a Club Rep Account (Cinema Manager) - Status DONE**

**a. Overview**

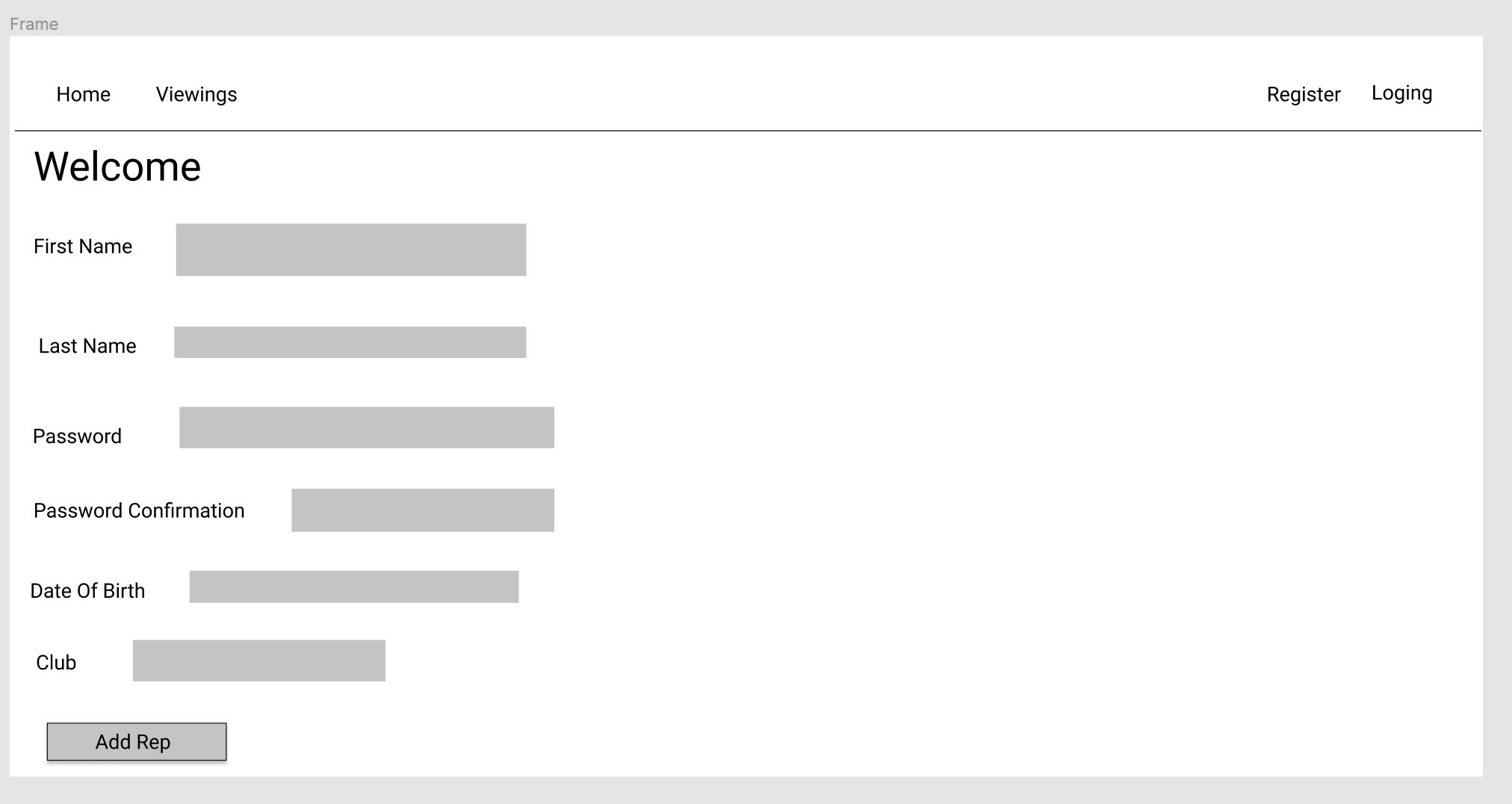
As explained in the specification, Club Managers are required to have the functionality to create a new club rep account. Our assumption of this information is that a club manager will create the account and give each club a unique ID and password to be able to log in, once created the club rep will have access to proceed as needed.

**b. Requirements**

* Once signed in, the Club Manager (**only**) should be able to create a Club Account for a Club Account
* Form must include an Employee/Club name, DOB and club
* The Rep ID and password should be unique and randomly generated
* Once treated the system should tell you that a reap account has been successfully created.

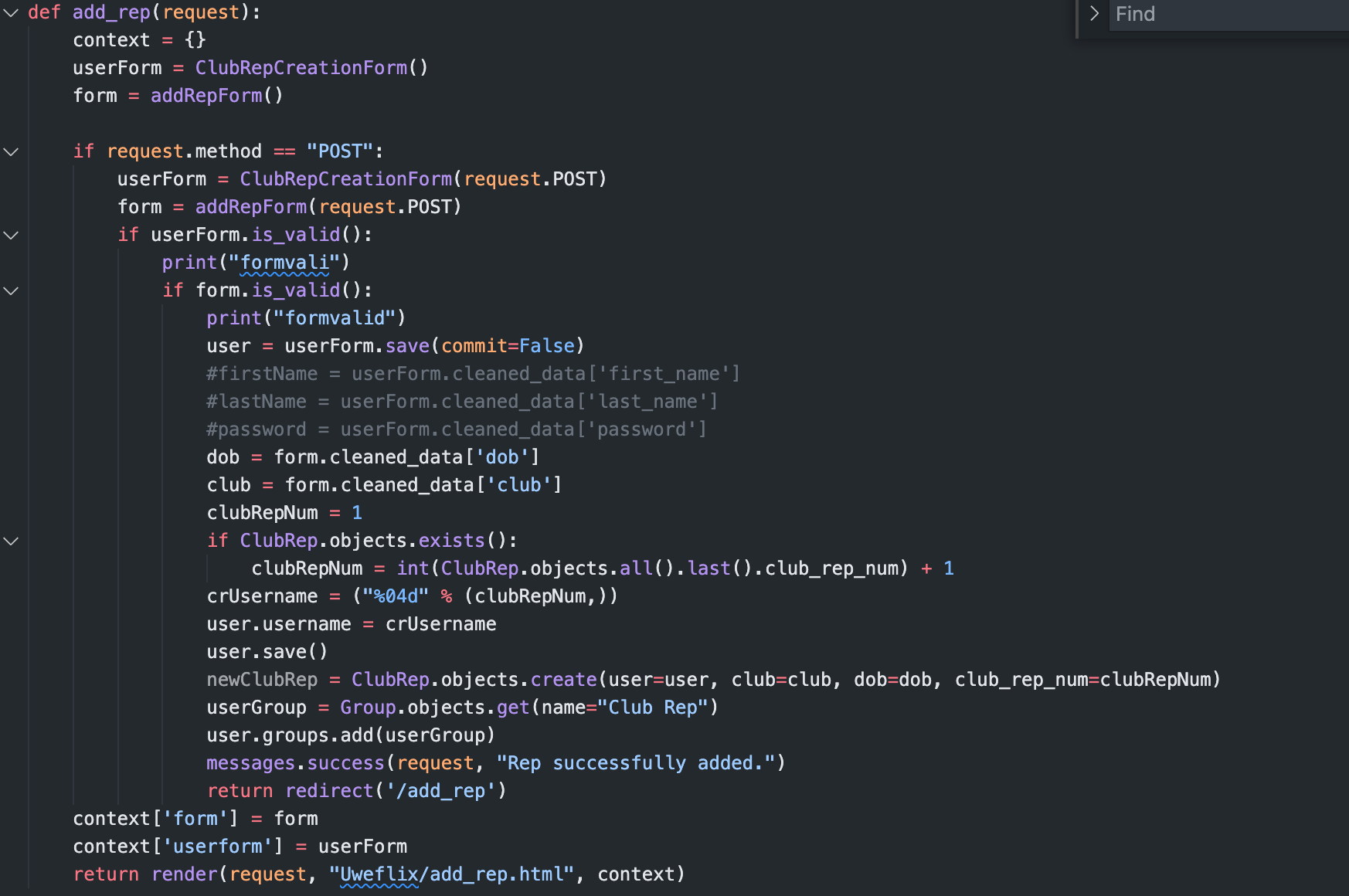
**c. Design**

Wireframe for the Club Rep Account:

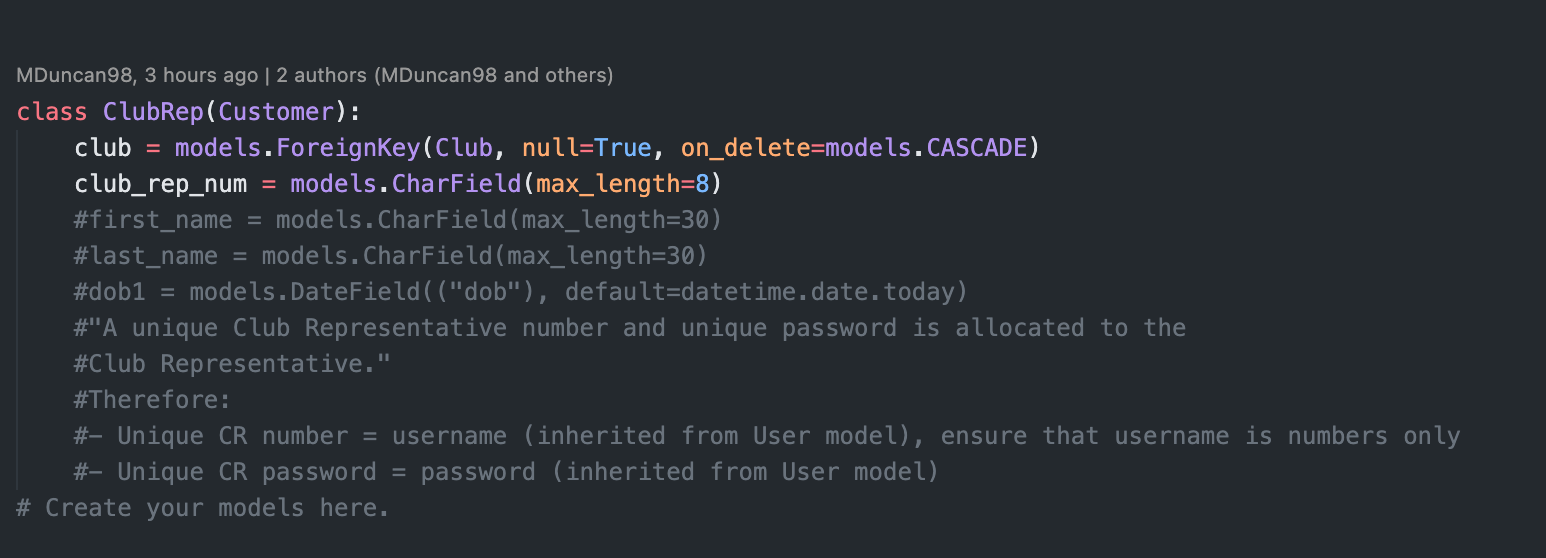
****

**d. Implementation**

Code snippet of Views.py logic



Code snippet of Model.py logic



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

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

{% load static %}

{% block title %}

<form method="post" class="form-group">

{% csrf\_token %} {{ form }}

<button type="submit" class="btn btn-success">Add Account</button>

</form>

{% endblock %}

{% block content %}

<h2>Welcome Add Account Here<h2>

<form method="post" class="form-group">

{% csrf\_token %} {{ form }}

<button type="submit" class="btn btn-success">Add Account</button>

</form>

{% endblock %}

**Here is the snippet in use once converted (Cinema Manager Home Page):**

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

{% load static %}

{% block title %}

Cinema Manager Home Page

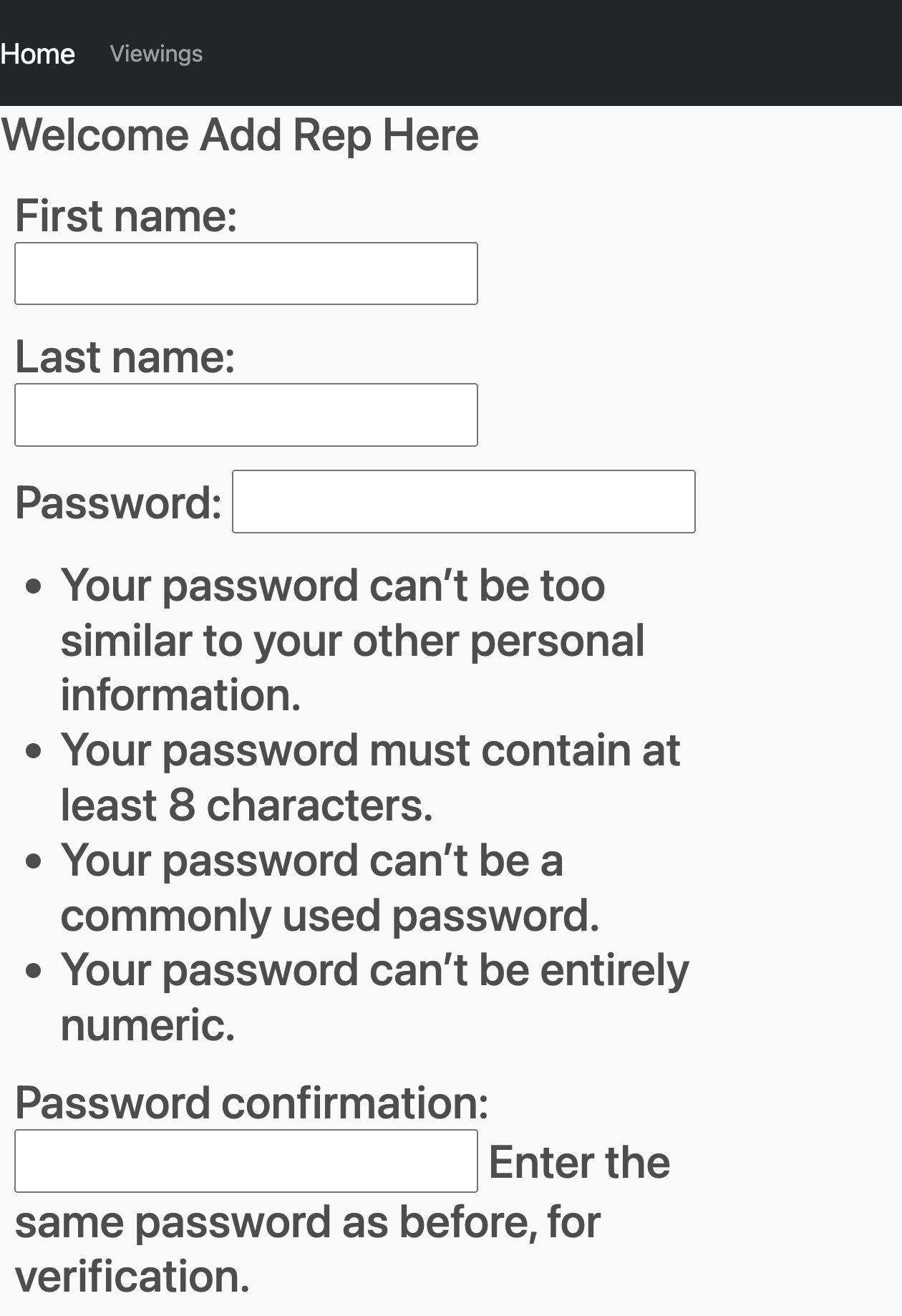
{% endblock %}

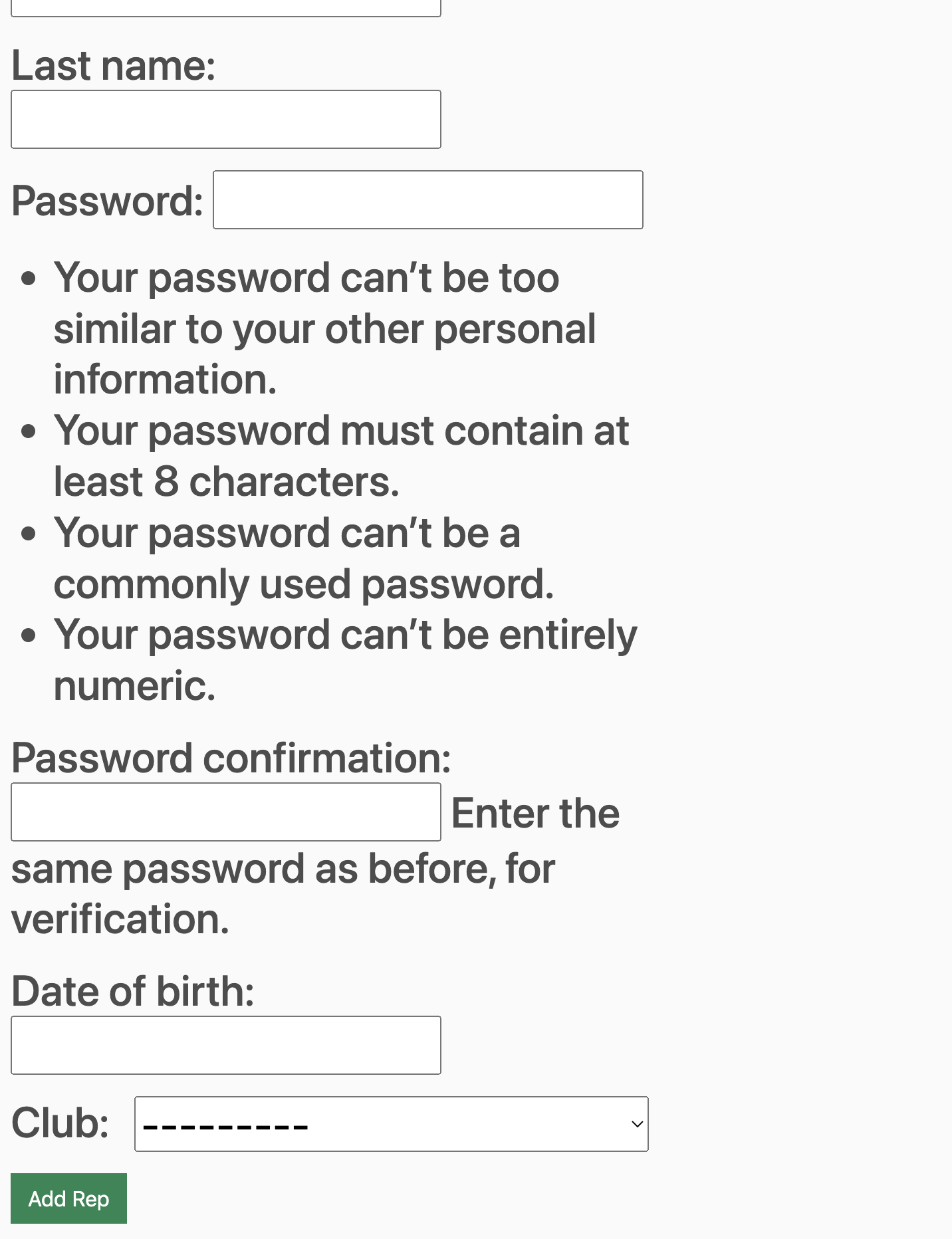
{% block content %}

<h2>Welcome Account Manager</h2>

{% endblock %}

Picture of 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 will be worked on in Sprint 6 as well as other further functionality.

**Michael Duncan - Tasks**

1. **Add Access Routines for the Transaction Model**
   1. **Overview**
   2. **Requirements**

*If unsure about any of the Transaction model ‘required arguments or parameters’, please see the previous sprint report regarding the design of this model, or refer to the models.py file in the UWEFlix project.*

The transaction model must:

* Have a function to create a brand new transaction object, if given all of the non-null parameters as arguments. If they are all valid, the object can be created and returned.
* Retrieve an existing object. This will be done using the object’s id, as we can guarantee that it is unique and therefore will return one and only one object, the one that we require.
* Update an existing object. This can take one or more of the required arguments from the model, depending on how many require updating. This function should change the old values with the ones given, and return the object.
* Delete an existing object. Given the id as an argument, of which if it exists, will be deleted from the database.
  1. **Design**

Pseudocode for the required functions:

**Create**

BEGIN

customer AS Customer object

Cost AS Float

Is\_paid AS boolean

Date AS Today’s date

TRY

CREATE new Transaction object(customer, date, cost, is\_paid)

RETURN Transaction

CATCH

PRINT “Error - Transaction could not be created!”

END

**Retrieve**

BEGIN

Id AS integer

TRY

GET Transaction object from model with PK=id

RETURN Transaction

CATCH

PRINT (“Error - Transaction could not be retrieved!”

END

**Update**

BEGIN

Id AS Integer

Transaction\_data AS Optional Arguments List

TRY

GET Transaction object from model with PK=id

FOR each item in transaction\_data

IF item is a Customer

UPDATE Transaction.customer WITH item

ENDIF

IF item is a Date

UPDATE Transaction.date WITH item

ENDIF

IF item is a Float

UPDATE Transaction.cost WITH item

ENDIF

IF item is a Boolean

UPDATE Transaction.is\_settled WITH item

ENDIF

ENDFOR

RETURN Transaction object

CATCH

PRINT “Error - could not update this Transaction object!”

END

**Delete**

BEGIN

Id AS Integer

TRY

GET Transaction object from model WHERE PK=id

DELETE Transaction object

PRINT “Transaction deleted successfully.”

CATCH

PRINT “Transaction not found, or could not be deleted!”

END

* 1. **Implementation**

*Code snippet from finished implementation:*

**

Each function is wrapped inside a try/catch block, to ensure that in the event of invalid or improper inputs, the system can handle the error and continue running even when an error is encountered.

Future improvements could include handling specific errors in the try/catch block, rather than generic ‘except’, which will except any exception. However, I felt that this use case would not greatly benefit from handling specific errors, as there is only one exception that is ever really raised in this scenario.

* 1. **Testing**

Please refer to tests [60-67] in the Testing Document (Link in ‘Relevant Links’).

* 1. **Conclusion**

Each function has been tested thoroughly and works as is expected. Therefore, I can confidently say that this task has been delivered as promised.

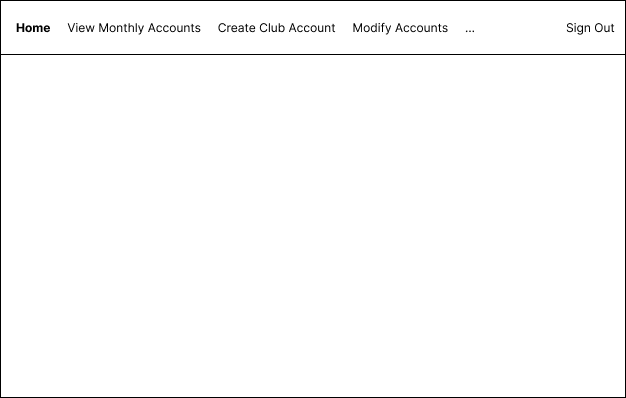
1. **Create Account Manager Base Template**
   1. **Overview**

As we want to restrict functionality of the website to those with the correct privileges, we will be creating a “base template” HTML page for each one of the user types. This will be the page that all of their other pages will inherit from, and therefore will allow us to only include relevant pages on their navigation bar, for example.

In this case, this base template will appear once the Accounts Manager has logged in, and has been redirected to their home page, and will be apparent for all other pages they have access to.

* 1. **Requirements**
* This page should only be visible once the Accounts Manager has signed in, and been authenticated.
* Once requirement 1 has been realised, This base template should appear on all pages accessed by the Account Manager.
* This base template should be removed once the Account Manager signs out, or their session is terminated.
* The base template should include all pages that the Account Manager can access in its navigation bar.
  1. **Design**

This base template will be created using Django’s built-in code snippet creator. This will allow us to create a base HTML page which can be inherited each time that we require it. This saves a lot of time writing duplicate code and reduces the chances of adding errors to our code.

Wireframe for the Base Template:

* 1. **Implementation**

*After some more thought about how to implement this functionality, we found out that it would be much simpler to use conditional chaining in one overall base layout, rather than creating individual ones. This is especially apparent when two user types require access to the same page, e.g. Viewings for Students, Customers, and Club Reps. Therefore, this Implementation reflects this change.*

**The entire base layout, will be available to view in the appendix of this document, or on the Github link in “UWEFlix\_django/uweflix/templates/uweflix/layout/html”. However, this documentation will only focus on the functionality required to render the Account Manager template in the desired way.**

**Here is the base template code:**

<!DOCTYPE html>

{% load static %}

<html>

<head>

**…**

</head>

<body>

**…**

{% if request.session.user\_id is None or request.session.user\_group == "Student" or request.session.user\_group == "Club Rep"%}

**…**

{% elif request.session.user\_group == "Account Manager" %}

<li class="nav-item active">

<a class="nav-link" href="{% url 'view\_accounts' %}">View Accounts</a>

</li>

<li class="nav-item active">

<a class="nav-link" href="{% url 'set\_payment' %}">Set Payment Details</a>

</li>

**<!--Future pages to be added here–>**

{% endif %}

{% if request.session.user\_id is None %}

**…**

{% else %}

**…**

<div class="topnav-right">

<a class="nav-link" href="{% url 'home' %}">{{ request.session.user\_group }}</a>

</div>

<div class="form-inline my-2 my-lg-0">

<a href="{% url 'logout' %}"class="btn btn-warning">Logout</a>

</div>

{% endif %}

**…**

</body>

</html>

**Here is the snippet in use once converted (Account Manager Home Page):**

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

{% load static %}

{% block title %}

Account Manager Home Page

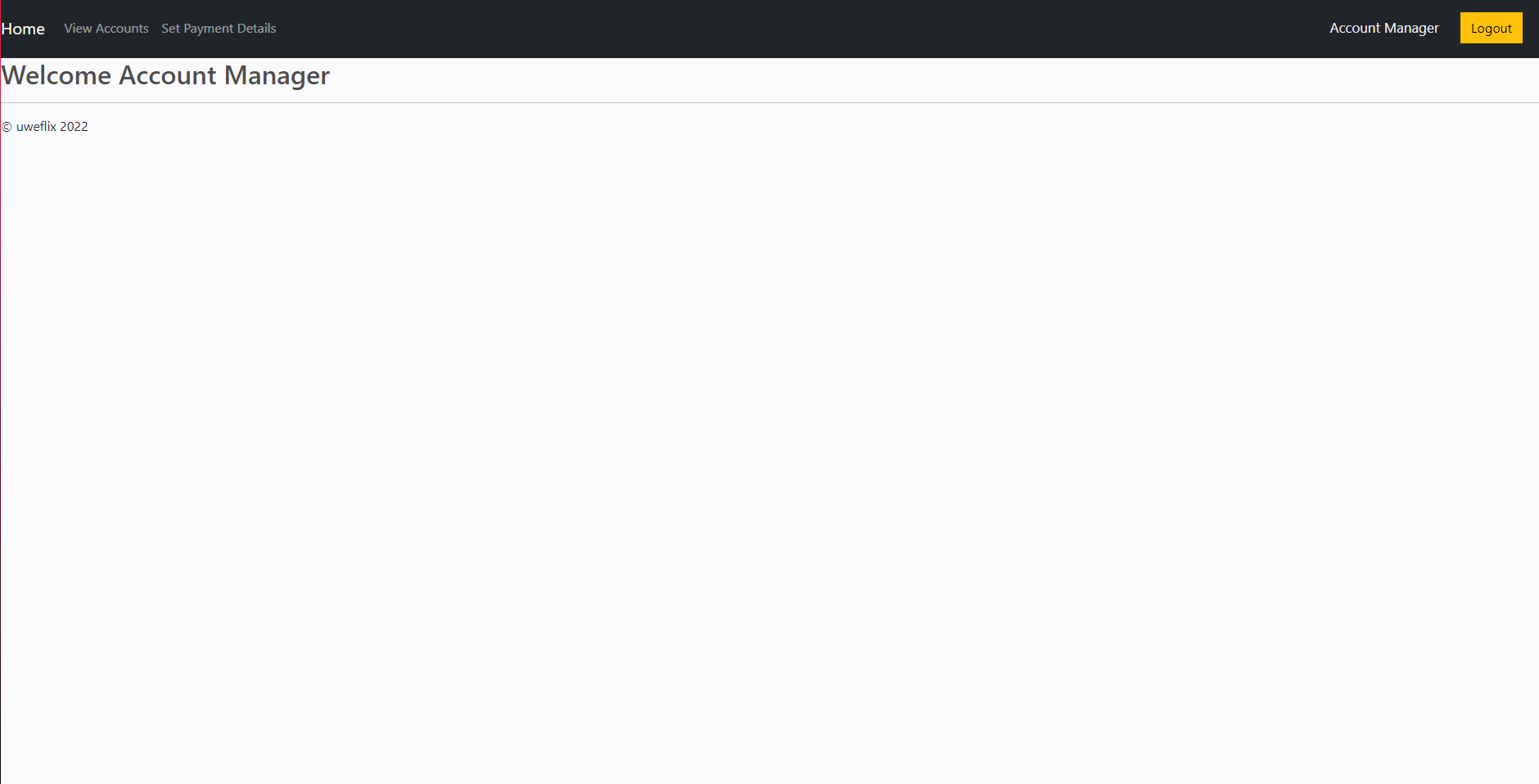
{% endblock %}

{% block content %}

<h2>Welcome Account Manager</h2>

{% endblock %}

Here is the view of the base template used in the Account Manager home page:



* 1. **Testing**

Please refer to [56-59] in the Testing Document.

1. **Create View Logic and Basic Template for Viewing End of Month Account Statements (Account Manager Page)**
   1. **Overview**

As described in the specification:

*“The UWEFlix Accounts Department stated that the new system is required to display account statements, to be provided at the end of every calendar month.”*

As the only customer who may make payments in advance is the Club Manager/Representative, we are assuming that this is the only customer type that requires statements to be generated.

As part of the Account Manager functionality, they should have a page where they are able to search for a specific customer, and they will be presented with all of the transactions that they have made in that **calendar** month.

* 1. **Requirements**
* Provide Account Manager the functionality to search through a specific customer’s transactions through the entire **calendar** month.
* A search should provide all transactions from the 1st of the current month, until the current date.
* The list should specify whether each of the transactions have been paid or not.
  1. **Design**

**Pseudocode for the views.py logic:**

BEGIN

CRForm AS DjangoForm

CR AS ClubRep

Transaction\_list AS ALL Transaction objects

OUTPUT CRForm

IF CRForm is Completed AND valid:

GET CR from CRForm

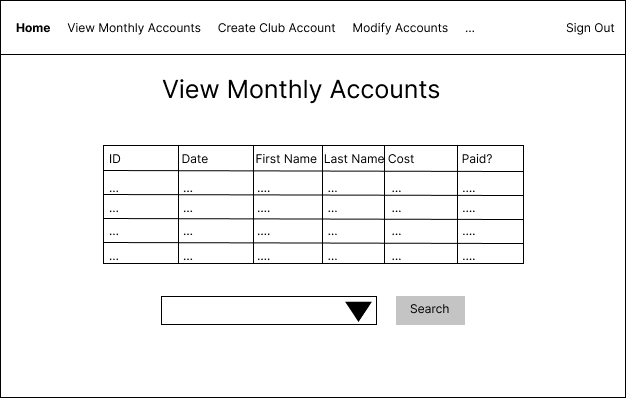
Transactions AS - SEARCH Transaction\_list for all Transaction objects for CR

OUTPUT Transactions AS Table

ENDIF

END

**Wireframe for the View Accounts page:**

****

* 1. **Implementation**

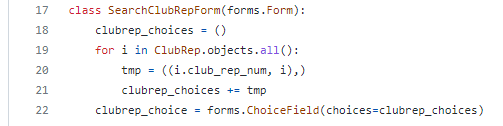
*Code snippet of views.py logic:*

**

This implementation was developed by following the designs as closely as possible.

Other related code:

* **SearchClubRepForm(): forms.py**

****

* **Dt: Imported datetime module**
* **View Accounts HTML Page:**

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

{% load static %}

{% block title %}

Viewings

{% 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>Transaction Viewing System</h1>

<hr/>

{% if transaction\_list %}

<h3>All transactions for Club Rep ({{club\_rep\_num}}) for the last calendar month:</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>Please select an account statement to view:</p>

{% endif %}

<hr/>

<form action="{% url 'view\_accounts' %}" method="post">

{% csrf\_token %}

Select a Club Rep to view: {{ form.clubrep\_choice }}

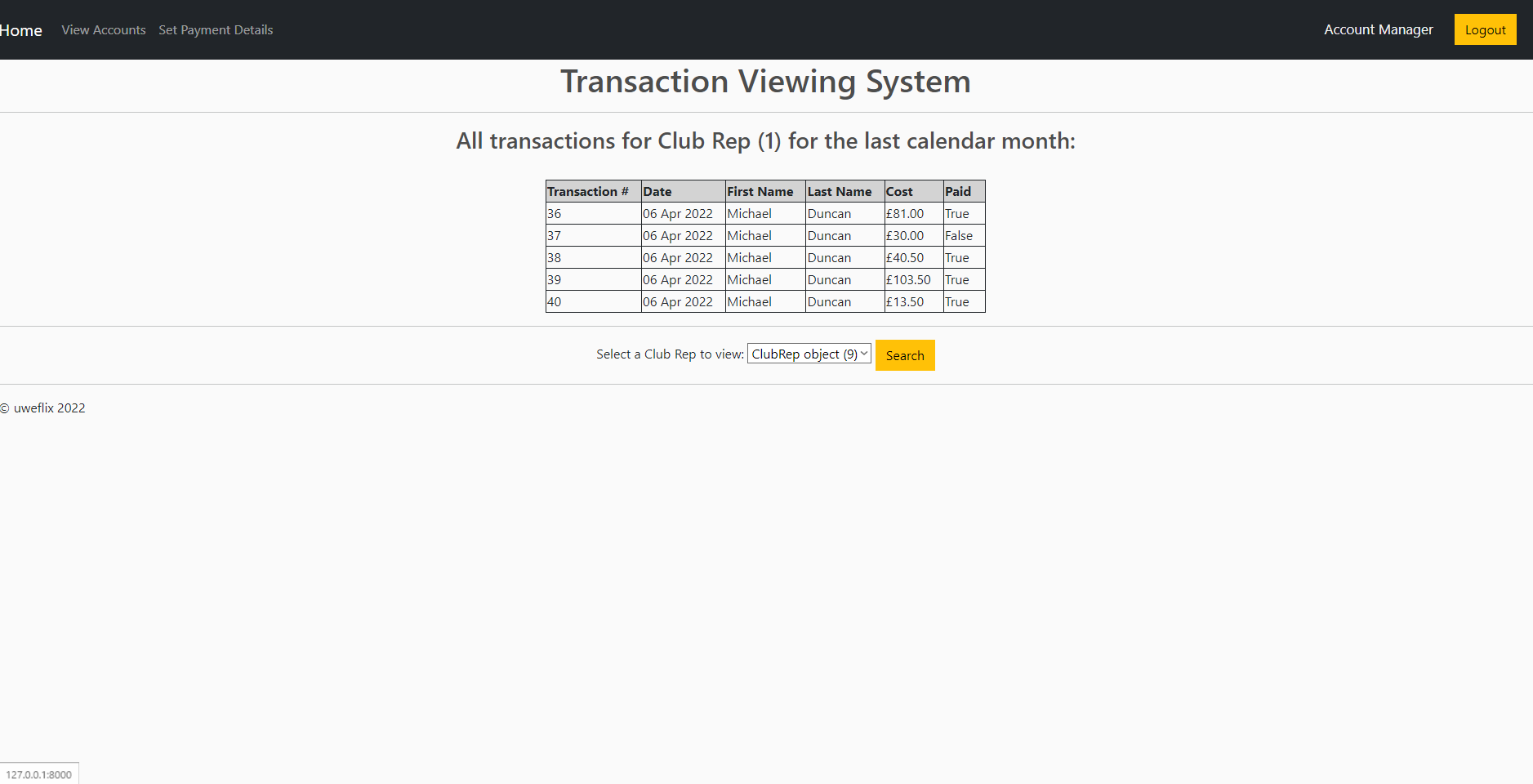
<button type="submit" class="btn btn-warning">Search</button>

</form>

</center>

{% endblock %}

**Example view of the View Accounts page:**

******

* 1. **Testing**

Please refer to [55] in the Testing Document.

1. **Create View & Template for the Payment Page (To follow on from the showings page)**
   1. **Overview**

For the general public and students, this page is the final part in the online booking process, where users will select how many tickets they require, and their payment details, as well as discount code if applicable.

All other booking information, such as the film and showing, will be retrieved from the previous pages in the booking process.

If the form is valid, and the payment is accepted, then the order will be processed, and credit will be removed from the user's account, or by other means if chosen (settle monthly, pay on the day, etc.). If not, payment will not be accepted and an error will be shown.

* 1. **Requirements**
* This page will require a customer of one of the following types (customer, student, club rep) to complete a purchase of the showing they have selected.
* For regular customers, they should only be allowed to make reservations, with payment being made on the day.
* For students, they should have the option of the payment option above, or to pay with the credit on their account. If their total credit is equal to or exceeds the cost of the booking, it will be accepted.
* For club reps, they have the two options above, as well as setting the payment at a later date. This transaction will be added to their monthly bill.
* Club reps will also have the option to apply a discount to their order. If a discount rate is set on their Club account, it will be applied at that rate.
* The number of tickets in a booking **cannot** exceed the total number of tickets remaining in the showing. If it does, the booking will not be accepted.
* If all the criteria are met, then the payment will be accepted, and a booking will be made. All of the tickets will be generated onto a transaction, and the number of tickets purchased will be removed from the remaining tickets for that showing.
  1. **Design**

**Pseudocode:**

BEGIN

Showing AS Showing Object

paymentForm As DjangoForm

IF paymentForm is Completed AND Valid

TotalCost AS Float

AdultTickets AS Int

StudentTickets As Int

ChildTickets As Int

TotalTickets As Int = (AdultTickets + StudentTickets + ChildTickets)

TotalCost As Float

PaymentOption As String

Paying As Boolean

IF Remaining tickets for showing < TotalTickets

CANCEL Booking

DISPLAY error

ELSE

IF User is signed in

UserType AS String

IF UserType is Student or Club Rep

User AS Customer Object

IF PaymentOption is Credit AND User.credit >= cost

User.credit = User.credit - cost

Paying = true

ELSE IF UserType is Club Rep AND PaymentOption is Pay Later

Paying = false

ELSE

CANCEL Booking

DISPLAY error

ENDIF

ELSE IF Payment Option is Pay on the Day

User = null

Paying = false

ELSE

CANCEL booking

DISPLAY error

ENDIF

newTransaction AS Transaction Object (User, Today’s Date, TotalCost, Paying)

FOR all adultTickets

Ticket Object (newTransaction, Showing, Adult)

ENDFOR

FOR all studentTickets

Ticket Object (newTransaction, Showing, Student)

ENDFOR

FOR all childTickets

Ticket Object (newTransaction, Showing, Child)

ENDFOR

UPDATE Remaining Tickets for Showing

PRINT Confirmation Message

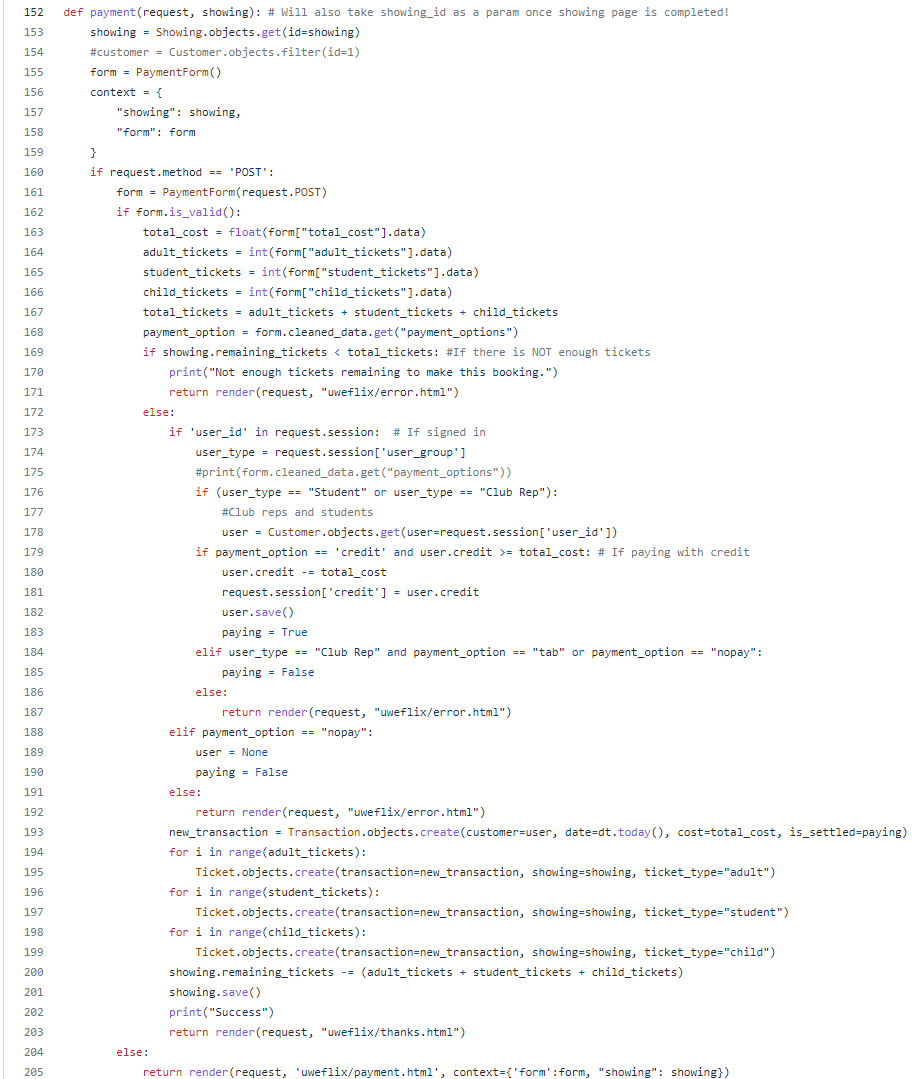
END

**Wireframe for Payment Page:**



* 1. **Implementation**

**Views.py Code for Payment Page:**



**PaymentForm() from forms.py:**

****

**HTML Template for Payment Page:**

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

{% load static %}

{% block title %}

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 '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">

<div class="form-group">

Adult tickets (£5): {{ form.adult\_tickets }}

</div>

<br/>

<div class="form-group">

Student Tickets (£4): {{ form.student\_tickets }}

</div>

<br/>

<div class="form-group">

Child Tickets (£3): {{ form.child\_tickets }}

</div>

<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>

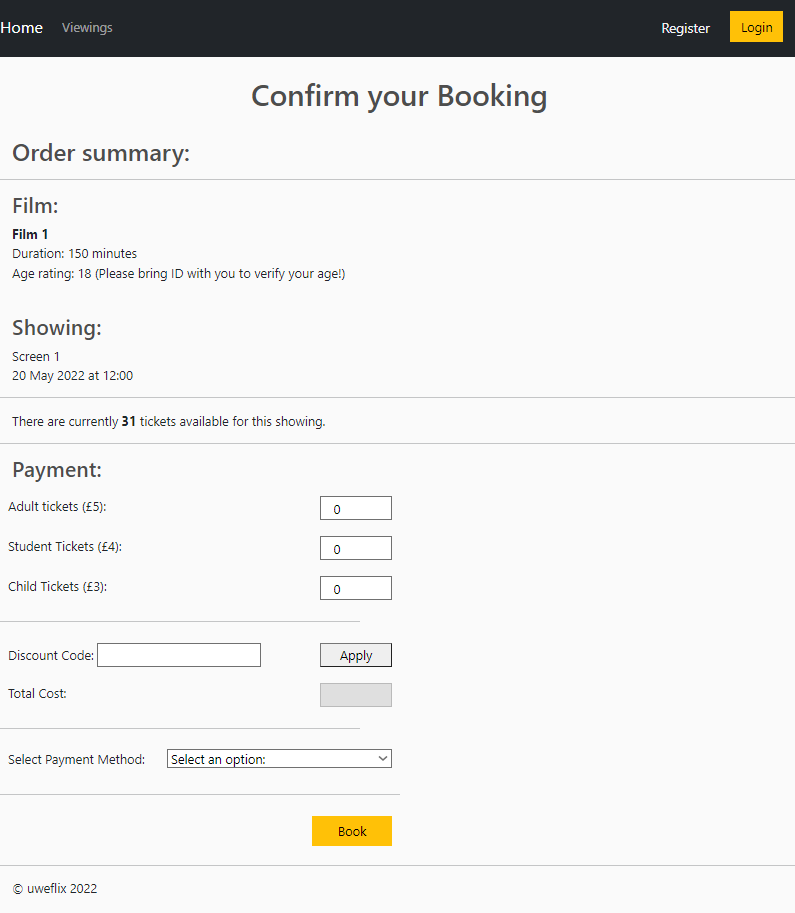
</form>

</div>

</body>

{% endblock %}

**View of Payment Page:**

****

* 1. **Testing**

Please refer to [33-46] in the Testing Document.

1. **Set Payment Details for Clubs (Account Manager)**
   1. **Overview**

As explained in the specification, Account Managers are required to have the functionality to set payment details for a Club Account. Our assumption of this information is that a Club will have one Payment method assigned to it, which all Club Representatives of that Club will have access to in order to make block bookings.

* 1. **Requirements**
* Once signed in, the Account Manager (**only**) should be able to set payment details for a Club Account
* Form must include an Employee/Club name, Card Number and Expiry Date
* Card Number must be exactly 16 digits long
* Expiry Date must be some time in the future
  1. **Design**

**Pseudocode:**

BEGIN

AccessClubForm AS DjangoForm

IF AccessClubForm is Completed AND Valid

ClubID AS Int

Club AS Club Object

CardNumber AS BigInt

ExpiryDate AS Date

IF CardNumber is Valid

Club.card\_number = CardNumber

ENDIF

IF ExpiryDate is Valid

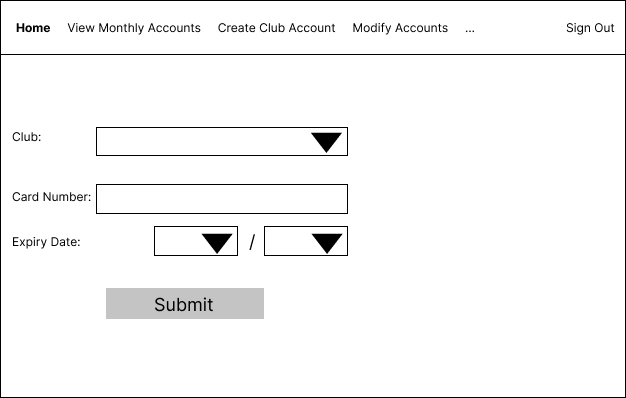
Club.expiry\_date = ExpiryDate

ENDIF

ENDIF

END

**Wireframe:**

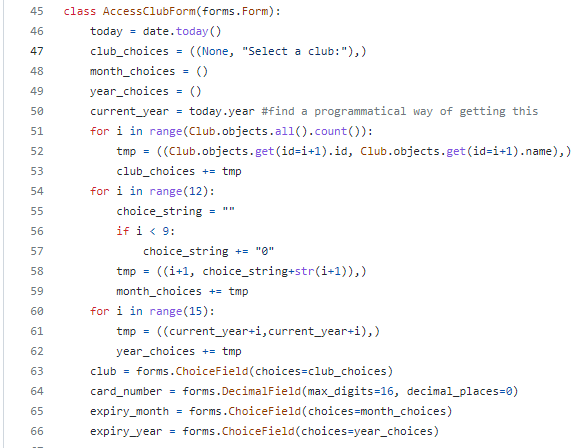


* 1. **Implementation**

**Views.py Code for Set Payment Details Page:**



**AccessClubForm() from Forms.py:**



**HTML Template for Set Payment Page:**

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

{% load static %}

{% block title %}

Set Payment Details

{% endblock %}

{% block content %}

<h1>Set Club Payment Details</h1>

<form method="POST">

{% csrf\_token %}

<div class="form-area">

<div class="alert-error">

{{ form.non\_field\_errors }}

</div>

<div class="form-group">

<p>Club: {{ form.club }}</p>

</div>

<hr/>

<div class="form-group">

<p>Card Number: {{ form.card\_number }}</p>

</div>

<div class="form-group">

<p><div class="expiry-label">Expiry Date: </div><div class="expiry-section">{{ form.expiry\_month }} / {{ form.expiry\_year }}</div></p>

</div>

<hr/>

<div class="form-group">

<input id="set-payment-btn" class="btn btn-warning" type="submit" value="Confirm">

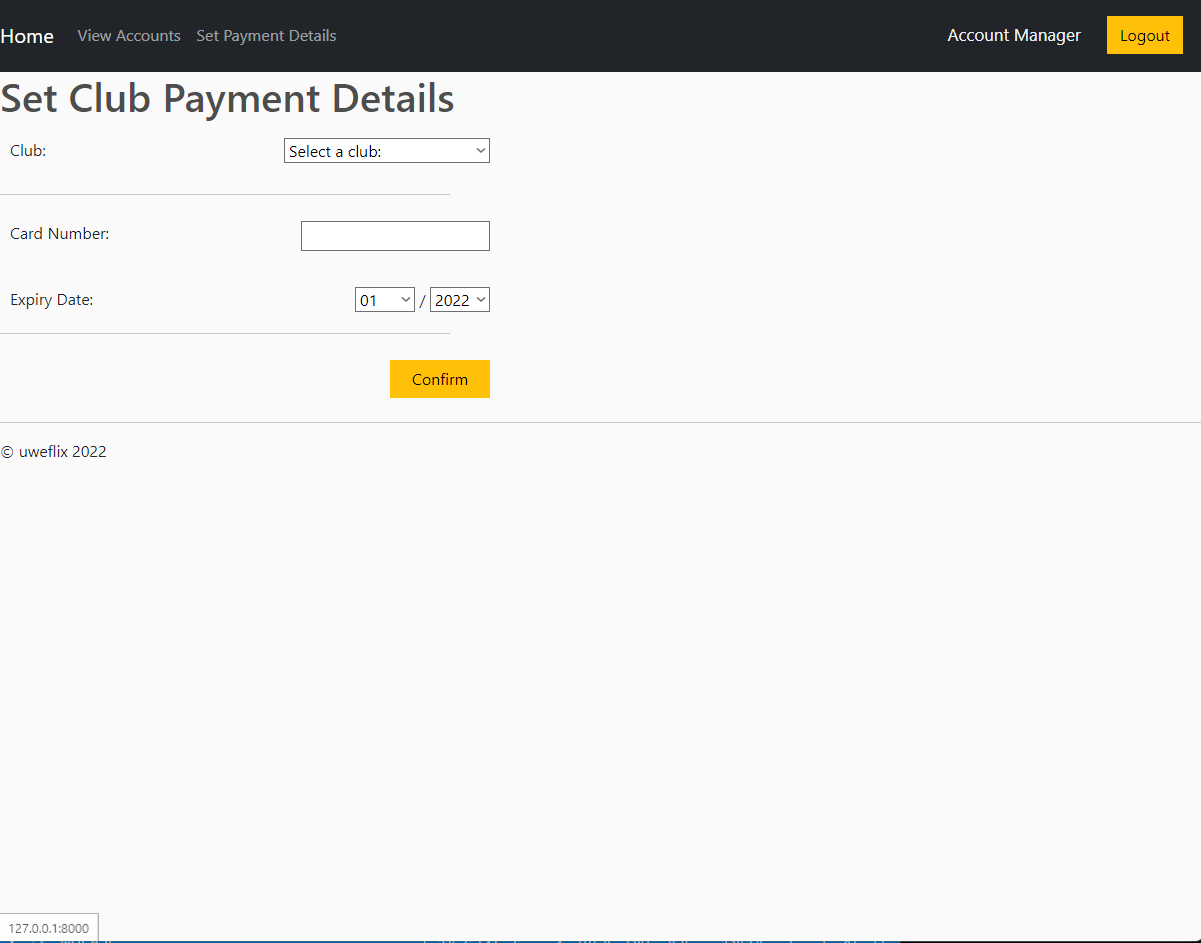
</div>

</div>

</form>

{% endblock %}

**View of Set Payment Page:**

****

* 1. **Testing**

Please refer to [47-54] in the Testing Document.

**Ross Williams - Tasks**

1. **Create Template - Add Film - DONE**

**a. Overview**

This task involves creating a basic template for the add film page. This page will be used by the Cinema Manager to add films. On this page the Cinema Manager can enter the title, age rating, duration and trailer description. The ‘Add Film’ button can be clicked, all the information in the fields will be added.

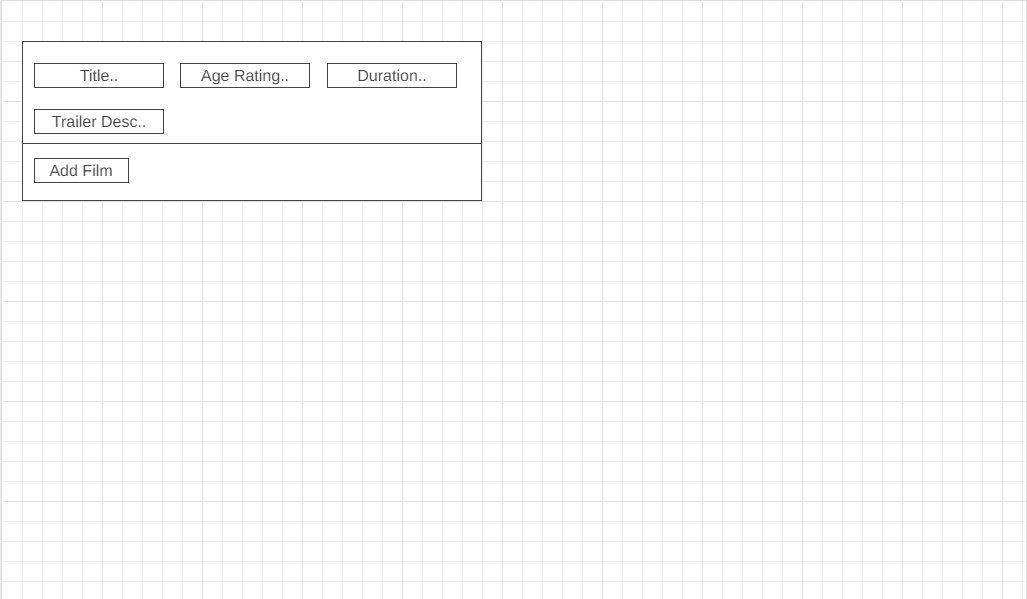
**b. Requirements**

* The cinema manager must be able to enter information about the film in the specific fields.

(As this is only a template, the requirements for the functionality are not listed.)

**c. Design**

**Wireframe for the Add Film page:**

****

Above is the wireframe for the add film page. The navbar is not included.

**d. Implementation**

**HTML for Add film template:**

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

{% block title %}

Add a film

{% endblock %}

{% block content %}

<div class="row">

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

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

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

{% 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>

<hr>

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

</form>

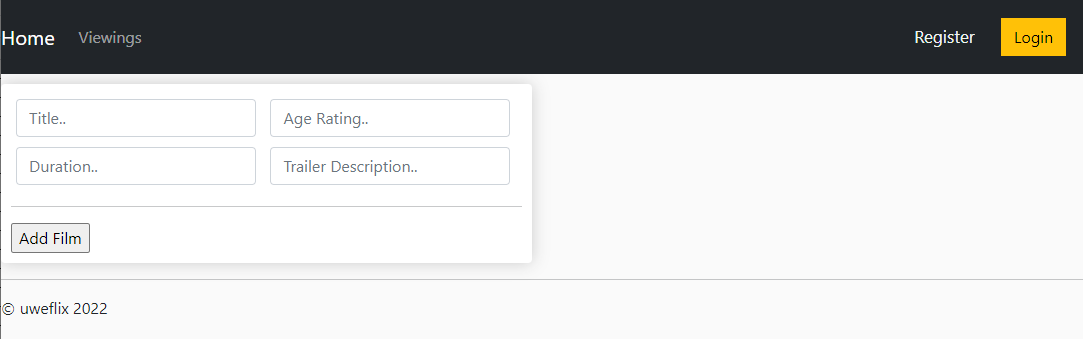
</div>

</div>

</div>

{% endblock %}

**Add film page webpage below:**



**e. Testing**

The template for this task matches the wireframe. Testing the logic/functionality will be done with the view task.

For further testing, please refer to tests in the Testing Document (Link in ‘Relevant Links’).

**f. Reflections**

Task went well with no issues. As this is a basic template it will be improved upon in an upcoming sprint, therefore this basic template meets the design specification.

1. **Create View/Template - Viewings Page - DONE**

**a. Overview**

This task involves creating a basic template and view logic for the viewings page. This page will be used by all users to see the films available. On this page all films will be displayed with a title, and will have a ‘book’ button that will redirect the user to the showings page for that specific film.

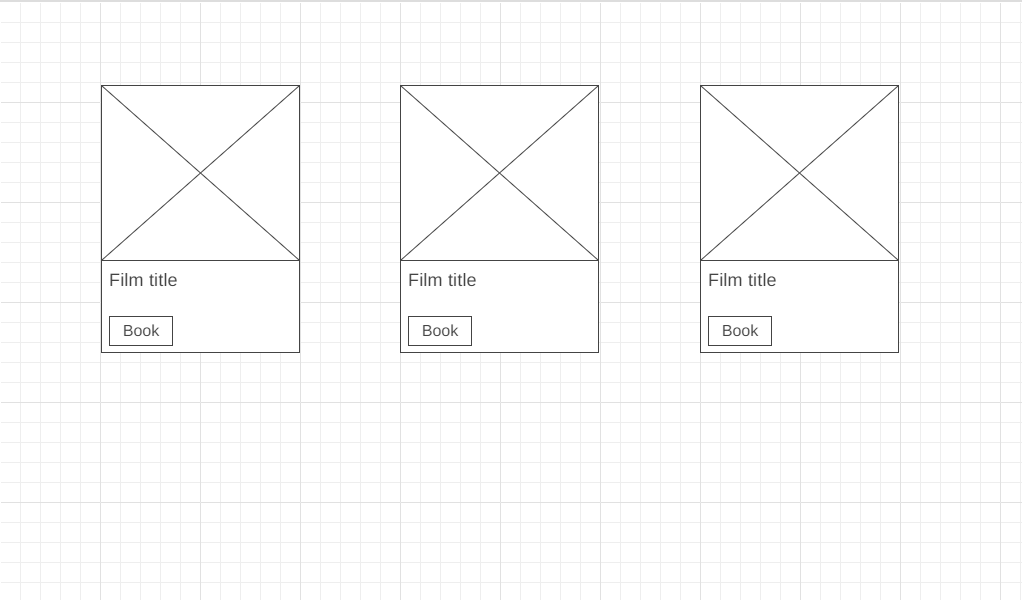
**b. Requirements**

* All users must have access to the page
* All films must be displayed along with their title.
* User must be able to click on the ‘book’ button to be redirected to the showings for that specific film.

**c. Design**

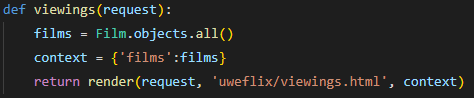
1. All film objects passed in views.py
2. Information about the film will be displayed dynamically using django in viewings.html

**Wireframe for the Viewings page:**

****

Above is the wireframe for the viewings page. The navbar is not included.

**d. Implementation**

**Views.py** 

**HTML for Viewings template:**

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

{% load static %}

{% block title %}

Viewings

{% endblock %}

{% block content %}

<div class="row">

{% for film in films %}

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

<img class="thumbnail" src="{% static 'uweflix/images/placeholder.png' %}">

<div class="box-element film">

<h6><strong>{{film.title}}</strong></h6>

<hr>

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

</div>

</div>

{% endfor %}

</div>

{% endblock %}

**e. Testing**

Please refer to [68] in the Testing Document.

**f. Reflections**

Task went well with no issues. Image is a placeholder at the moment, this will need to be worked on further. However, the basic design has been met and all tests have passed.

1. **Access Routines - Club - DONE**

**a. Overview**

This task involves adding backend functionality to the club model, to handle logic (particularly CRUD - Create, Read, Update, Delete) which can be called from the views.py class.

**b. Requirements**

The club model must have functions that allow for:

* Creation of a new club object using the valid parameters.
* Read an existing club object using the object’s id.
* Update an exisiting object using the object’s id.
* Delete an existing object using the object’s id.

**c. Design**

Pseudocode for the required functions:

**Create**

BEGIN

club AS Club object

Name as String

Card\_number as Integer

Card\_expiry\_date as date

Discount\_rate as Integer

TRY

CREATE new Club object(Name, Card\_number , Card\_expiry\_date , Discount\_rate)

RETURN Club

CATCH

PRINT “Error - Club can’t be created!”

END

**Retrieve**

BEGIN

Id AS Integer

TRY

GET Club object from model with id

RETURN Club

CATCH

PRINT “Error - Club can’t be found!”

END

**Update**

BEGIN

Id AS Integer

Club\_data AS Optional Arguments List

TRY

GET Club object from model with id

FOR each item in club\_data

IF item is a Name

UPDATE Club.name WITH item

ENDIF

IF item is a Card\_number

UPDATE Club.card\_number WITH item

ENDIF

IF item is a Card\_expiry\_date

UPDATE Club.card\_expiry\_date WITH item

ENDIF

IF item is a Discount\_rate

UPDATE Club.discount\_rate WITH item

ENDIF

ENDFOR

RETURN Club object

CATCH

PRINT “Error - could not update this Club object!”

END

**Delete**

BEGIN

Id AS Integer

TRY

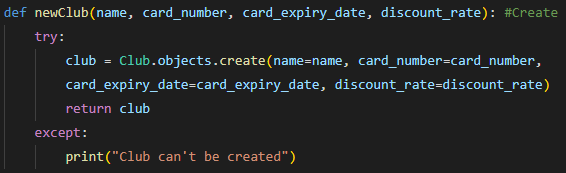
GET Club object from model with id

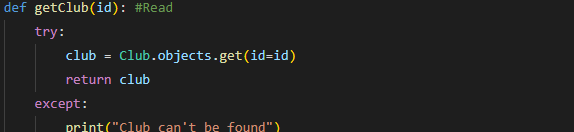
DELETE Club object

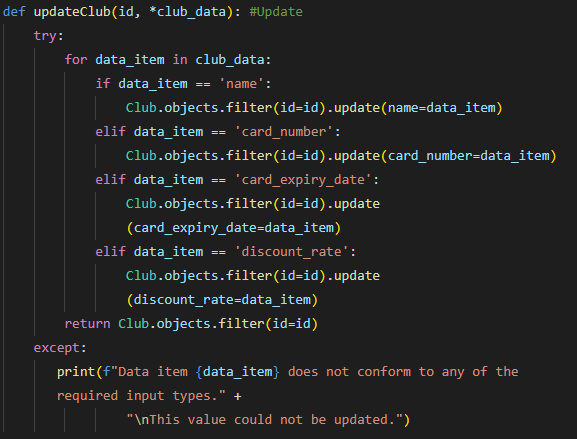
CATCH

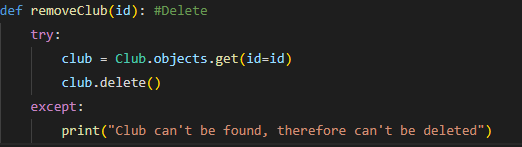
PRINT “Club can’t be found, therefore can’t be deleted”

END

**d. Implementation**







**e. Testing**

Please refer to [69-76] in the Testing Document.

**f. Reflections**

Task went well with no issues. All tests have passed.

1. **Create View/Template - Showings Page - DONE**

**a. Overview**

This task involves creating a basic template and view logic for the showings page. This page will be used by all users to see the showings available for a specific film. On this page more information about the film will be shown, including a brief trailer description, along with all showings for that film ordered by date. The user can then click on a specific showing to be directed to the payment page.

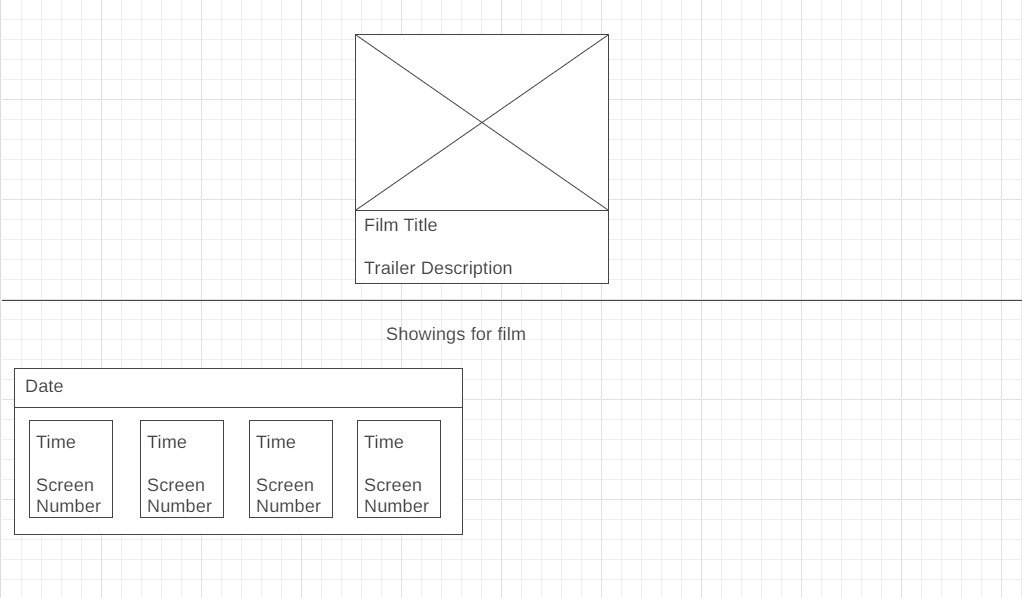
**b. Requirements**

* All users must have access to the page
* The specific film must be displayed along with its title, trailer description and age rating.
* Showings must be displayed in order, with the earliest showings displayed first.
* User must be able to click on the showing to be redirected to the payment page.

**c. Design**

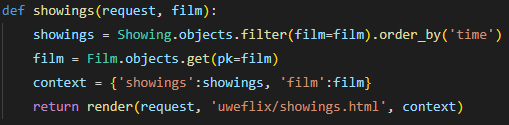
1. All showings objects passed in views.py for a specific film ordered by date.
2. Information about the film and its showings will be displayed dynamically using django in showings.html

**Wireframe for the Showings page:**

****

Above is the wireframe for the showings page. The showings are ordered by date and every showing for a specific date is shown below that date. The navbar is not included.

**d. Implementation**

**Views.py:**

**HTML for Showings page:**

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

{% load static %}

{% block title %}

Showings

{% endblock %}

{% block content %}

<style>

form {

display: inline-block;

}

h3 {

text-align: center;

}

h2, h5, p, .col-lg-4 {

padding-left: 15px;

}

.showing-box {

padding-left: 10px;

padding-top: 10px;

}

</style>

<div class="row">

<div class="center">

<img class="thumbnail" src="{% static 'uweflix/images/placeholder.png' %}">

<div class="box-element film">

<h6><strong>{{film.title}}</strong></h6>

<br>

<p>{{film.trailer\_desc}}</p>

</div>

</div>

<div class="showings">

<hr>

<h3>Showings for {{film.title}}</h3>

{% for showing in showings %}

<div class ="showing-box">

<div class="box-element film">

<h6><strong>{{ showing.time | date:'j M Y' }}</strong></h6>

<hr>

<div>

<a class="btn btn-outline-secondary add-btn" href="{% url 'payment' showing.pk %}">

<h6><strong>{{ showing.time | time:'G:i'}}</strong></h6>

<h6><strong>Screen {{showing.screen.id}}</strong></h6>

</a>

</div>

</div>

</div>

{% endfor %}

</div>

</div>

{% endblock %}

**e. Testing**

Please refer to [77] in the Testing Document.

**f. Reflections**

Task went well with no issues. Template will need to be worked on further. However, the basic design has been met and all tests have passed.

**Additional Comments:**

Tasks that were not completed this sprint roll over to sprint 6.

**Arjun Binning - Tasks**

1. **Add Access Routines – Screen – Status Done**

**a. Overview**

This task involves adding backend functionality to the models themselves, to handle logic (particularly CRUD - Create, Read, Update, Delete) which can be called from the views.py class to present business logic on the front end.

*If unsure about any of the Screen model ‘required arguments or parameters’, please see the previous sprint report regarding the design of this model, or refer to the models.py file in the UWEFlix project.*

**b. Requirements**

The Showing model must:

* Have a function to **create** a brand-new Screen object, if given all of the non-null parameters as arguments. If they are all valid, the object can be created and returned.
* **Retrieve/Read** an existing object. This will be done using the object’s id, as we can guarantee that it is unique and therefore will return one and only one object, the one that we require.
* **Update** an existing object. This can take one or more of the required arguments from the model, depending on how many require updating. This function should change the old values with the ones given and return the object.

· **Delete** an existing object. Given the id as an argument, of which if it exists, will be deleted from the database.

**c. Design**

The Pseudocode for the required functions:

**Create**

BEGIN  
capacity AS Integer  
covid\_restrictions AS Boolean  
TRY

CREATE new Screen object(capacity, covid\_restrictions)

RETURN Screen

CATCH

PRINT “Error – Screen could not be created!”

END

**Retrieve**

BEGIN

id as Integer

TRY

GET Screen object from model with PK=id

RETURN Screen

CATCH

Print “Error – Screen could not be retrieved!”

END

**Update**

BEGIN

id AS Integer

fieldToEdit AS Optional Arguments List

TRY

GET Screen object from model with PK=id

FOR each item in fieldToEdit

IF item is Integer

UPDATE capacity WITH item

ENDIF

IF Item is Boolean

UPDATE covid\_restrictions WITH item

ENDIF

ENDFOR

RETURN Screen Object

CATCH

PRINT “Error – Could not update this Screen!”

END

**Delete**

BEGIN

id AS Integer

TRY

GET Screen Object from model where PK=id

Delete Screen Object

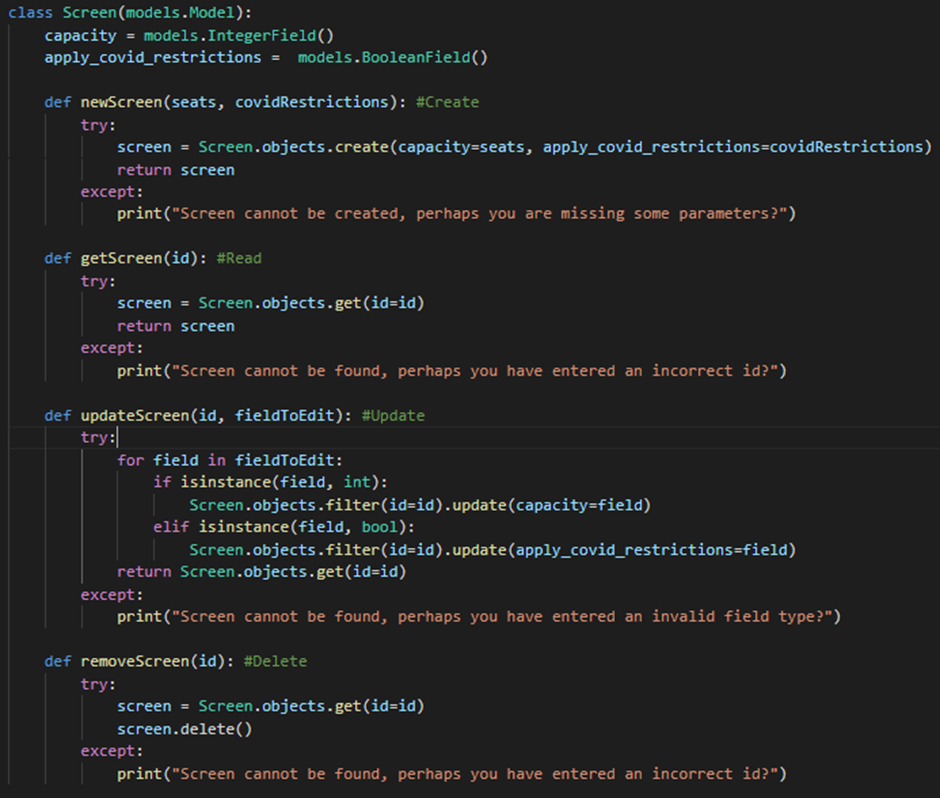
PRINT “Screen Deleted Successfully”

CATCH

PRINT “Error – Could not delete Screen”

END

**e. Implementation**



**f. Testing**

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

1. **Add Access Routines – Ticket – Status Done**

**a. Overview**

This task involves adding backend functionality to the models themselves, to handle logic (particularly CRUD - Create, Read, Update, Delete) which can be called from the views.py class to present business logic on the front end.

*If unsure about any of the Ticket model ‘required arguments or parameters’, please see the previous sprint report regarding the design of this model, or refer to the models.py file in the UWEFlix project.*

**b. Requirements**

The Ticket model must:

* Have a function to **create** a brand-new Ticket object, if given all of the non-null parameters as arguments. If they are all valid, the object can be created and returned.
* **Retrieve/Read** an existing object. This will be done using the object’s id, as we can guarantee that it is unique and therefore will return one and only one object, the one that we require.
* **Update** an existing object. This can take one or more of the required arguments from the model, depending on how many require updating. This function should change the old values with the ones given and return the object.

· **Delete** an existing object. Given the id as an argument, of which if it exists, will be deleted from the database.

**c. Design**

The Pseudocode for the required functions:

**Create**

BEGIN

transaction AS Transaction Object

showing AS Showing Object

ticket\_type AS String

TRY

CREATE new Ticket object (transaction, showing, ticket\_type)

RETURN Ticket

CATCH

PRINT “Error – Ticket could not be created!”

END

**Retrieve**

BEGIN

id as Integer

TRY

GET Ticket object from model with PK=id

RETURN Screen

CATCH

Print “Error – Screen could not be retrieved!”

END

**Update**

BEGIN

id AS Integer

fieldToEdit AS Optional Arguments List

TRY

GET Ticket object from model with PK=id

FOR each item in fieldToEdit

IF item is Transaction Object

UPDATE transaction WITH item

ENDIF

IF Item is Showing Object

UPDATE showing WITH item

ENDIF

IF Item is String

UPDATE ticket\_type WITH item

ENDIF

ENDFOR

RETURN Ticket Object

CATCH

PRINT “Error – Could not update this Ticket!”

END

**Delete**

BEGIN

id AS Integer

TRY

GET Ticket Object from model where PK=id

Delete Ticket Object

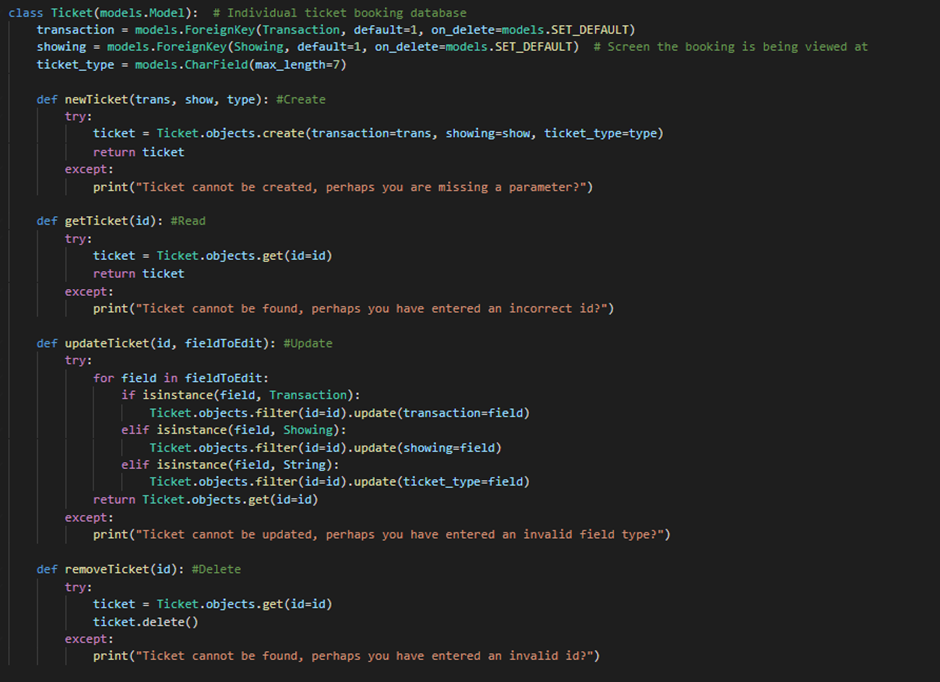
PRINT “Ticket Deleted Successfully”

CATCH

PRINT “Error – Could not delete Ticket”

END

**e. Implementation**



**f. Testing**

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

1. **Create View – Add Film Logic – Status Done**

**a. Overview**

This task involves reading information from the form in the Add Film Template and creating a new Film Object using the data given, and validating it to ensure it is correct.

**b. Requirements**

This Includes

* Read the information from the form and sort them into different variables.
* **Validate** this information, ensure the Age Rating fits into one of the rating categories, and ensure the duration is completely numerical.
* **Create** a Film Object using this information and save it into the table.

**c. Design**

The Pseudocode for the required function:

**Create Film**

BEGIN

age\_ratings AS String List

film\_title AS String

film\_age\_rating AS String

film\_duration AS Integer

trailer\_description AS String

IF film\_duration IS Integer

IF film\_age\_rating IN age\_ratings

new Film Object(film\_title, film\_age\_rating, film\_duration, trailer\_description)

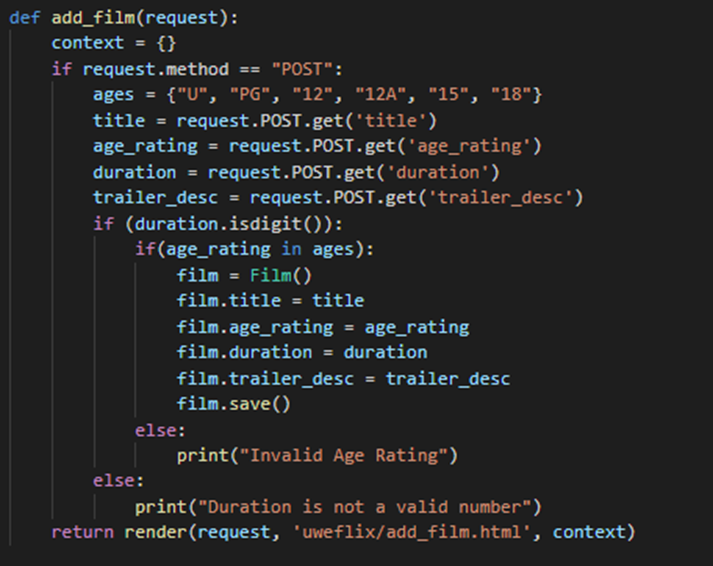
SAVE Film

ENDIF

ENDIF

END

**e. Implementation**



**f. Testing**

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

1. **Create Template – Topup Page – Status Done**

**a. Overview**

Club Representatives and Students should be able to access a page where they can top up the credit on their account. This will be done through the Topup Page.

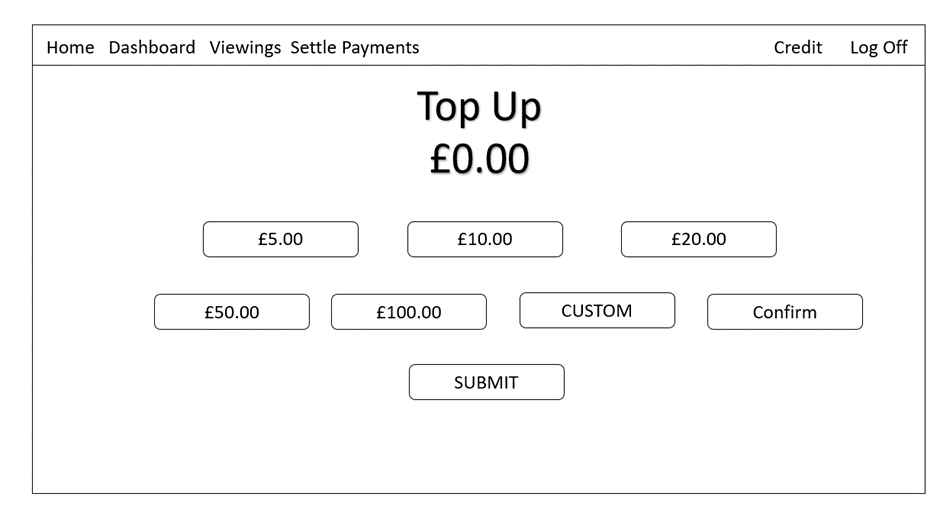
**b. Requirements**

This Includes

* Have a numerical figure at the top of the screen which changes depending on how much credit is to be added.
* Have multiple buttons that can change the credit to a default value on the screen.
* Have a custom box where the user can manually input credit amounts.
* Have a hidden form so that the input credit can be used in the Views page.

**c. Design**

This is an example Wireframe for the Top Up Page.



**e. Implementation**

The page was implemented through HTML and was created using this code:

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

{% block title %}

Topup

{% endblock %}

{% block content %}

<style>

h1 {

text-align: center;

margin: 20px;

}

#topupAmount {

text-align: center;

margin: 10px;

font-size: 50px;

}

.topupAmountButton {

display: inline-block;

width: 200px;

padding: 5px;

margin: 25px 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);

}

.topupAmountButton::placeholder {

color: white;

text-align: center;

opacity: 0.8;

}

.flex-container {

display: flex;

justify-content: center;

}

.flex-container > div {

margin: 10px;

padding: 20px;

font-size: 30px;

}

.submitTopUp {

width: 200px;

padding: 5px;

margin: 25px 0;

border-radius: 8px;

background-color: #F0AD4E;

font-weight: bold;

}

</style>

<h1>Topup Credit</h1>

<p id = "topupAmount">£0.00</p>

<div class="flex-container">

<div><button onclick="change\_topupAmount('5.00')" class="topupAmountButton">£5.00</button></div>

<div><button onclick="change\_topupAmount('10.00')" class="topupAmountButton">£10.00</button></div>

<div><button onclick="change\_topupAmount('20.00')" class="topupAmountButton">£20.00</button></div>

</div>

<div class="flex-container">

<div><button onclick="change\_topupAmount('50.00')" class="topupAmountButton">£50.00</button></div>

<div><button onclick="change\_topupAmount('100.00')" class="topupAmountButton">£100.00</button></div>

<div><input type = "text", name = "customTopUpAmount", class="topupAmountButton", placeholder="Custom"></div>

<div><button onclick="change\_topupAmount('Custom')" class="topupAmountButton">Confirm</button></div>

</div>

<div class="flex-container">

<form method="POST">

{% csrf\_token %}

<input type="hidden" name="topUpValue" id="topUpValueForm" />

<div><button class="submitTopUp">Submit</button></div>

</form>

</div>

<script>

function change\_topupAmount(price) {

if (price == "Custom") {

var custom = parseFloat(document.getElementsByName("customTopUpAmount")[0].value, 10);

if (isNaN(custom) != true) {

document.getElementById("topupAmount").innerHTML = "£" + custom.toFixed(2);

document.getElementById("topUpValueForm").value = custom.toFixed(2);

}

} else {

document.getElementById("topupAmount").innerHTML = "£" + price;

document.getElementById("topUpValueForm").value = price;

}

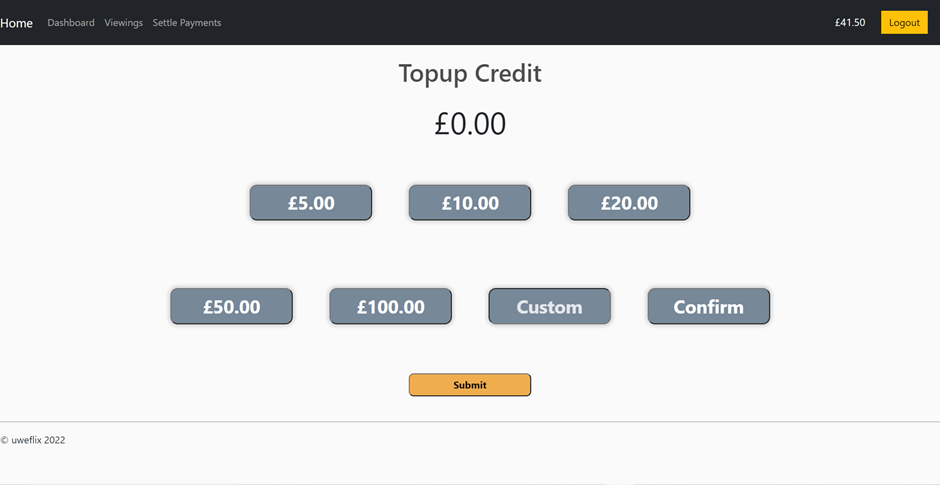
}

</script>

{% endblock %}

**f. Testing**

The Page looked similar to what was requested on the design and the outcome was:



For further testing, please refer to tests in the Testing Document (Link in ‘Relevant Links’).

1. **Create View – Topup Page – Status Done**

**a. Overview**

Club Representatives and Students should be able to access a page where they can top up the credit on their account. This will be done through the Topup Page.

**b. Requirements**

This Includes:

* Ensure the user is logged in and that they are a Club Rep or a Student.
* Read the value of the credit to be added through the top up form.
* Since Float validation is already completed, ensure the value is 2 decimal places.
* Added the credit into the logged Users account.

**c. Design**

The Pseudocode for the required function:

**Create Film**

BEGIN

userObject as Object

IF user\_group IN Session

IF Sessions user\_group IS Club Rep:

userObject is ClubRep

ELIF Sessions user\_group IS Student:

userObject is Student

ELSE:

RETURN HOME

END IF

TopUpValue as Float.

user = Get User Object from Model where id=user\_group

user.Credit = user.Credit + Rounded to 2 Decimal Places(TopupValue).

ENDIF

END

**e. Implementation**



**f. Testing**

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

1. **Create Template – Club Rep and Student Home Page – Status Done**

**a. Overview**

Club Representatives and Students should have their own respective home pages where they can be redirected to in Login, and access their main functionalities..

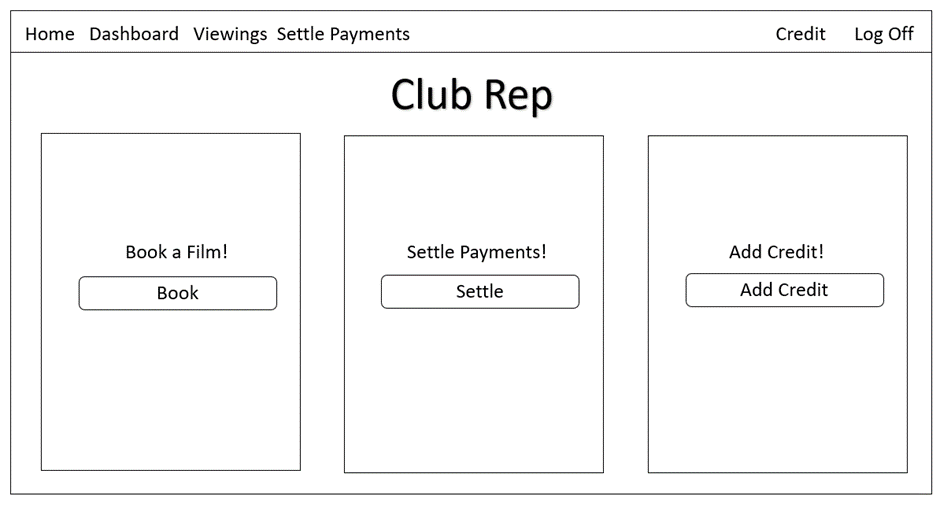
**b. Requirements**

This Includes

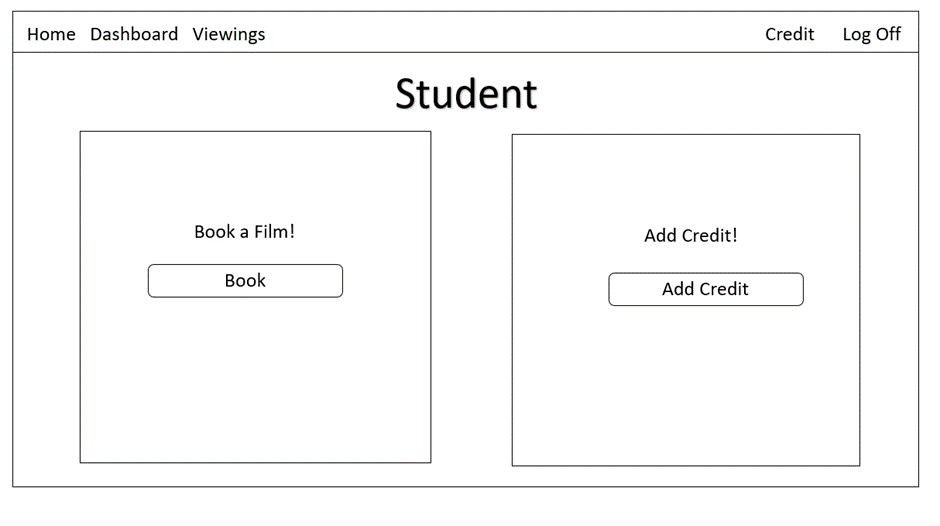
* Club Rep Home Page must have 3 sections – Redirect to Book a film, Redirect to settle their payments, Redirect to Topup Credit.
* Student Home Page must have 2 sections – Redirect to Book a film, Redirect to Topup Credit.

**c. Design**

This is an example Wireframe for the Club Rep Home Page.



This is an example Wireframe of the Student Home Page.



**e. Implementation**

The Club Rep page was implemented through HTML and was created using this code:

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

{% load static %}

{% block title %}

Club Rep Home Page

{% endblock %}

{% block content %}

<style>

.flex-container {

display: flex;

margin-top: 50px;

}

.flex-child {

flex: 1;

background: lightgrey;

margin-right: 20px;

margin-left: 20px;

text-align: center;

border-radius: 12px;

box-shadow: 0 0 5px 2px rgba(211,211,211,0.8);

}

.clubRepHomeButtons {

width: 250px;

padding: 5px;

margin: 25px 0;

border-radius: 12px;

background-color: lightslategrey;

color: white;

font-weight: bold;

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

}

.extraHeight {

height: 125px;

}

</style>

<div class="flex-container">

<div class="flex-child">

<div class="extraHeight"></div>

<h2>Book Film</h2></br>

<h3>Book a viewing!</h3></br>

<a href="{% url 'viewings' %}">

<input type="button" value="Book" class="clubRepHomeButtons" />

</a>

<div class="extraHeight"></div>

</div>

<div class="flex-child">

<div class="extraHeight"></div>

<h2>Settle Payments</h2></br>

<h3>Settle outstanding payments!</h3></br>

<a href="{% url 'settle\_payments' %}">

<input type="button" value="Settle" class="clubRepHomeButtons" />

</a>

<div class="extraHeight"></div>

</div>

<div class="flex-child">

<div class="extraHeight"></div>

<h2>Topup Credit</h2></br>

<h3>£{{ request.session.credit|floatformat:2 }} left!</h3></br>

<a href="{% url 'topup' %}">

<input type="button" value="Topup" class="clubRepHomeButtons" />

</a>

<div class="extraHeight"></div>

</div>

</div>

{% endblock %}

The Student Home page was also coded in HTML and coded as such:

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

{% load static %}

{% block title %}

Student Home Page

{% endblock %}

{% block content %}

<style>

.flex-container {

display: flex;

margin-top: 50px;

}

.flex-child {

flex: 1;

background: lightgrey;

margin-right: 20px;

margin-left: 20px;

text-align: center;

border-radius: 12px;

box-shadow: 0 0 5px 2px rgba(211,211,211,0.8);

}

.studentHomeButtons {

width: 250px;

padding: 5px;

margin: 25px 0;

border-radius: 12px;

background-color: lightslategrey;

color: white;

font-weight: bold;

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

}

.extraHeight {

height: 125px;

}

</style>

<div class="flex-container">

<div class="flex-child">

<div class="extraHeight"></div>

<h2>Book Film</h2></br>

<h3>Book a viewing!</h3></br>

<a href="{% url 'viewings' %}">

<input type="button" value="Book" class="studentHomeButtons" />

</a>

<div class="extraHeight"></div>

</div>

<div class="flex-child">

<div class="extraHeight"></div>

<h2>Topup Credit</h2></br>

<h3>£{{ request.session.credit|floatformat:2 }} left!</h3></br>

<a href="{% url 'topup' %}">

<input type="button" value="Topup" class="studentHomeButtons" />

</a>

<div class="extraHeight"></div>

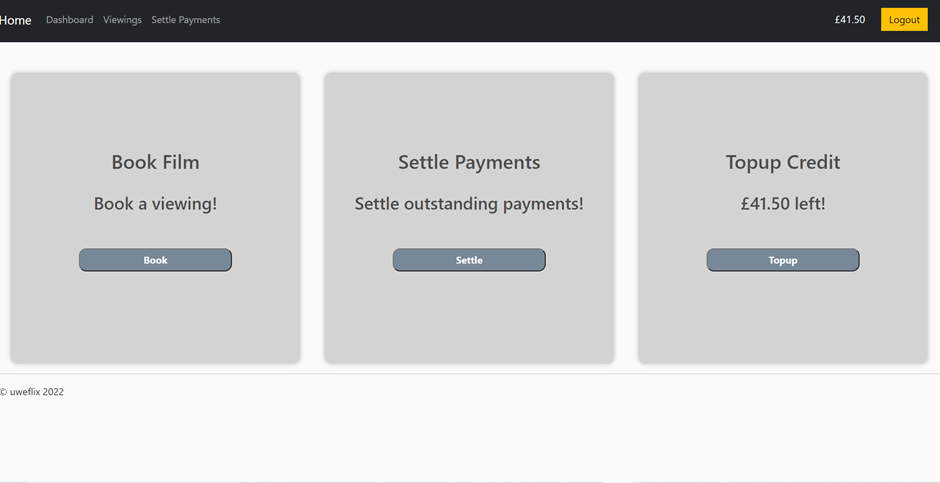
</div>

</div>

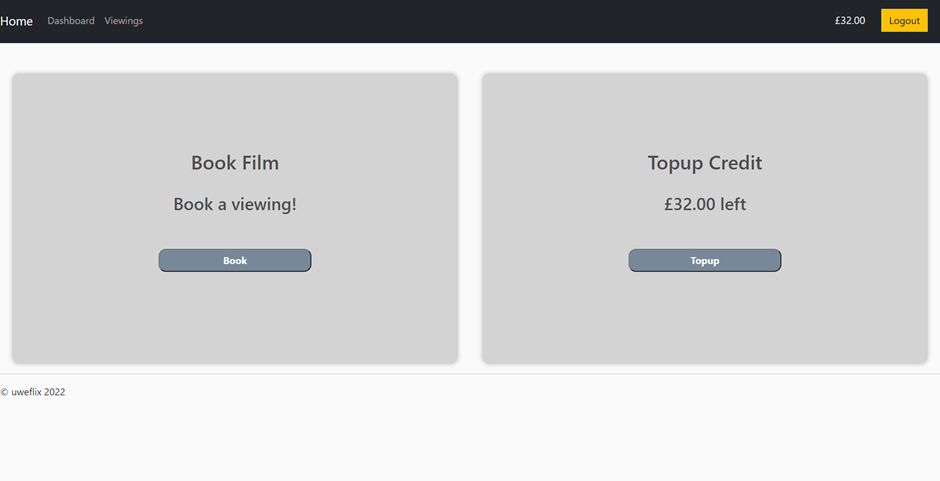
{% endblock %}

**f. Testing**

The Club Rep Page looked similar to what was requested on the design and the outcome was:



Likewise with the Student Page:



For further testing, please refer to tests in the Testing Document (Link in ‘Relevant Links’).

1. **Create Template and View – Settle Payments – Status Done**

**a. Overview**

At the end of every month, Club Representatives are confronted with their outstanding payments, and are required to settle them using the Bank Card linked in the system.

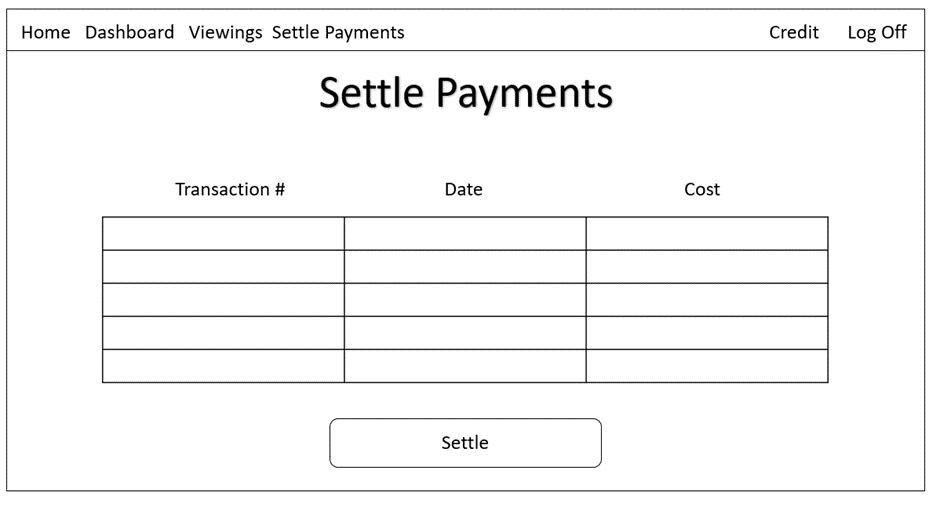
**b. Requirements**

This Includes

* Display only the outstanding payments for the logged in Club Rep
* Have a Button which sets the outstanding payments to settled, taking money off the bank card stored in the system.

**c. Design**

This is an example Wireframe for the Settle Payments Page Template.



And the Pseudocode for the View is as such:

BEGIN

club\_rep as ClubRep Object

GET Transactions Objects WHERE (club\_rep = customer, month = thisMonth, year = thisYear, settled = false).

amount\_owed as Integer = 0

FOR each Transaction is Transactions

amount\_owed = amount\_owed + Transaction.cost

transaction.is\_settled = true

transaction.save

ENDFOR

END

**e. Implementation**

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

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

{% load static %}

{% block title %}

Settle Payments

{% 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;

}

.settleButton {

width: 250px;

padding: 5px;

margin: 25px 0;

border-radius: 12px;

background-color: lightslategrey;

color: white;

font-weight: bold;

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

}

</style>

<center>

<h1>Settle Payments</h1>

<hr/>

{% if transactions %}

<h3>Outstanding transactions for Club Rep #{{club\_rep}} for the last month:</h3>

<br/>

<table class="transactions\_list">

<thead>

<tr>

<th>Transaction #</th>

<th>Date</th>

<th>Cost</th>

</tr>

</thead>

<tbody>

{% for outstanding\_transaction in transactions %}

<tr>

<td>{{ outstanding\_transaction.id }}</td>

<td>{{ outstanding\_transaction.date | date:'d M Y' }}</td>

<td>£{{ outstanding\_transaction.cost }}</td>

</tr>

{% endfor %}

</tbody>

</table>

<form method="POST">

{% csrf\_token %}

<input type="submit" value="Settle", class="settleButton">

</form>

{%else%}

<h3>No Outstanding Transactions!</h3>

{% endif %}

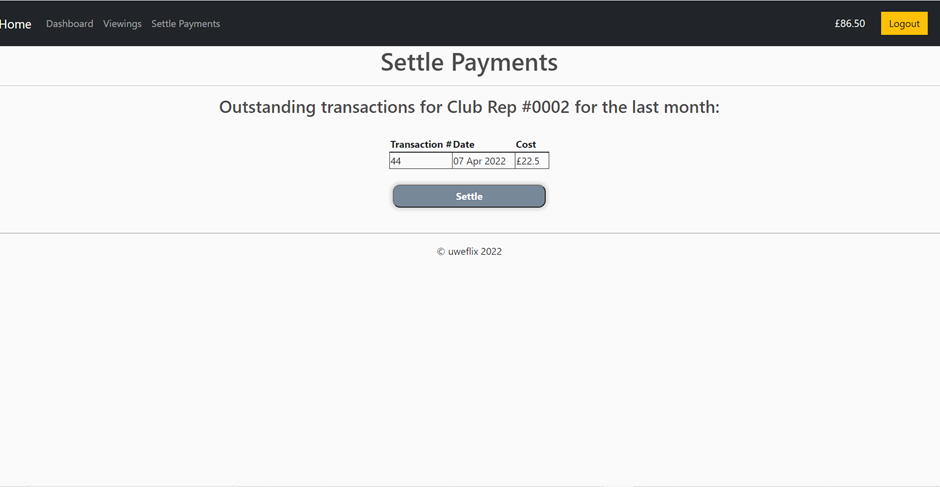
{% endblock %}

The Implementation for the View was as such:



**f. Testing**

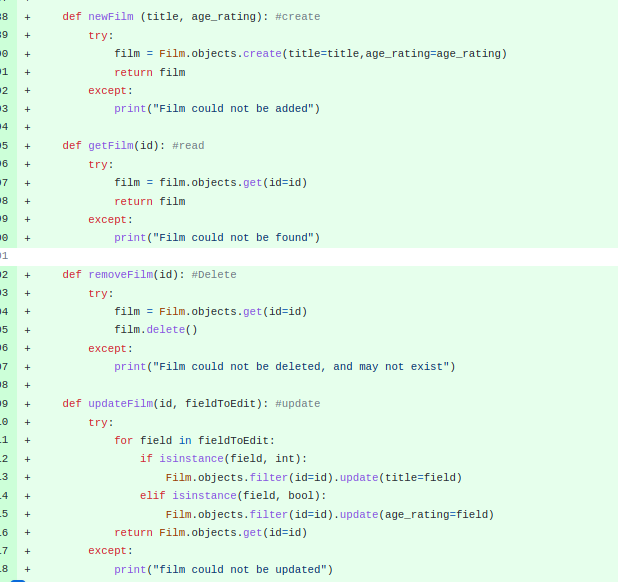
The Page looked similar to what was requested on the design and the outcome was:



For further testing, please refer to tests in the Testing Document (Link in ‘Relevant Links’).

**Benedict RM - Tasks**

In this sprint I worked on fixing an issue where some linux-based systems that use case-sensitive filenames were unable to correctly host the website, as well as the CRUD methods for films.

****

The specific problem was that log**I**n.html is referenced, and the file itself is named log**i**n.html. (note the lower/uppercase “i”s.)

**Appendix - Source Code**

HTML Base Layout:

<!DOCTYPE html>

{% load static %}

<html>

<head>

<meta charset="utf-8"/>

<title>{% block title %}{% endblock %}</title>

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">

<link rel="stylesheet" type="text/css" href="{% static 'uweflix/site.css' %}"/>

<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, minimum-scale=1" />

<script type="text/javascript" src="{{ STATIC\_URL }} /static/js/jquery-3.6.0.js"></script>

<script src="{{ STATIC\_URL }} /static/js/rango-jquery.js"></script>

</head>

<body>

<!-- <div class="navbar">

<a href="{% url 'home' %}" class="navbar-brand">Home</a>

<a href="{% url 'viewings' %}" class="navbar-item">Viewings</a>

</div> -->

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

<a class="navbar-brand" href="{% url 'home' %}">Home</a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarSupportedContent">

<ul class="navbar-nav me-auto">

{% if request.session.user\_id is None or request.session.user\_group == "Student" or request.session.user\_group == "Club Rep"%}

{% if request.session.user\_group == "Club Rep"%}

<li class="nav-item active">

<a class="nav-link" href="{% url 'club\_rep\_home' %}">Dashboard</a>

</li>

{% endif %}

{% if request.session.user\_group == "Student"%}

<li class="nav-item active">

<a class="nav-link" href="{% url 'student\_home' %}">Dashboard</a>

</li>

{% endif %}

<li class="nav-item active">

<a class="nav-link" href="{% url 'viewings' %}">Viewings</a>

</li>

{% if request.session.user\_group == "Club Rep"%}

<li class="nav-item active">

<a class="nav-link" href="{% url 'settle\_payments' %}">Settle Payments</a>

</li>

{% endif %}

{% elif request.session.user\_group == "Cinema Manager" %}

<li class="nav-item active">

<a class="nav-link" href="{% url 'add\_film' %}">Add Film</a>

</li>

{% elif request.session.user\_group == "Account Manager" %}

<li class="nav-item active">

<a class="nav-link" href="{% url 'view\_accounts' %}">View Accounts</a>

</li>

<li class="nav-item active">

<a class="nav-link" href="{% url 'set\_payment' %}">Set Payment Details</a>

</li>

{% endif %}

</ul>

{% if request.session.user\_id is None %}

<div class="topnav-right">

<a class="nav-link" href="{% url 'registerPage' %}">Register</a>

</div>

<div class="form-inline my-2 my-lg-0">

<a href="{% url 'login' %}"class="btn btn-warning">Login</a>

</div>

{% else %}

{% if request.session.user\_group == "Student" or request.session.user\_group == "Club Rep" %}

<div class="topnav-right">

<a class="nav-link" href="{% url 'topup' %}">£{{ request.session.credit|floatformat:2 }}</a>

</div>

{% else %}

<div class="topnav-right">

<a class="nav-link" href="{% url 'home' %}">{{ request.session.user\_group }}</a>

</div>

{% endif %}

<div class="form-inline my-2 my-lg-0">

<a href="{% url 'logout' %}"class="btn btn-warning">Logout</a>

</div>

{% endif %}

</div>

</nav>

<div class="body-content">

{% block content %}

{% endblock %}

<hr/>

<footer>

<p>© uweflix 2022</p>

</footer>

</div>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p" crossorigin="anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.10.2/dist/umd/popper.min.js" integrity="sha384-7+zCNj/IqJ95wo16oMtfsKbZ9ccEh31eOz1HGyDuCQ6wgnyJNSYdrPa03rtR1zdB" crossorigin="anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.min.js" integrity="sha384-QJHtvGhmr9XOIpI6YVutG+2QOK9T+ZnN4kzFN1RtK3zEFEIsxhlmWl5/YESvpZ13" crossorigin="anonymous"></script>

<script>

</script>

</body>

</html>