
**Greenwich Community Theatre: Systems Development Project
(Coursework 2: Part A)**

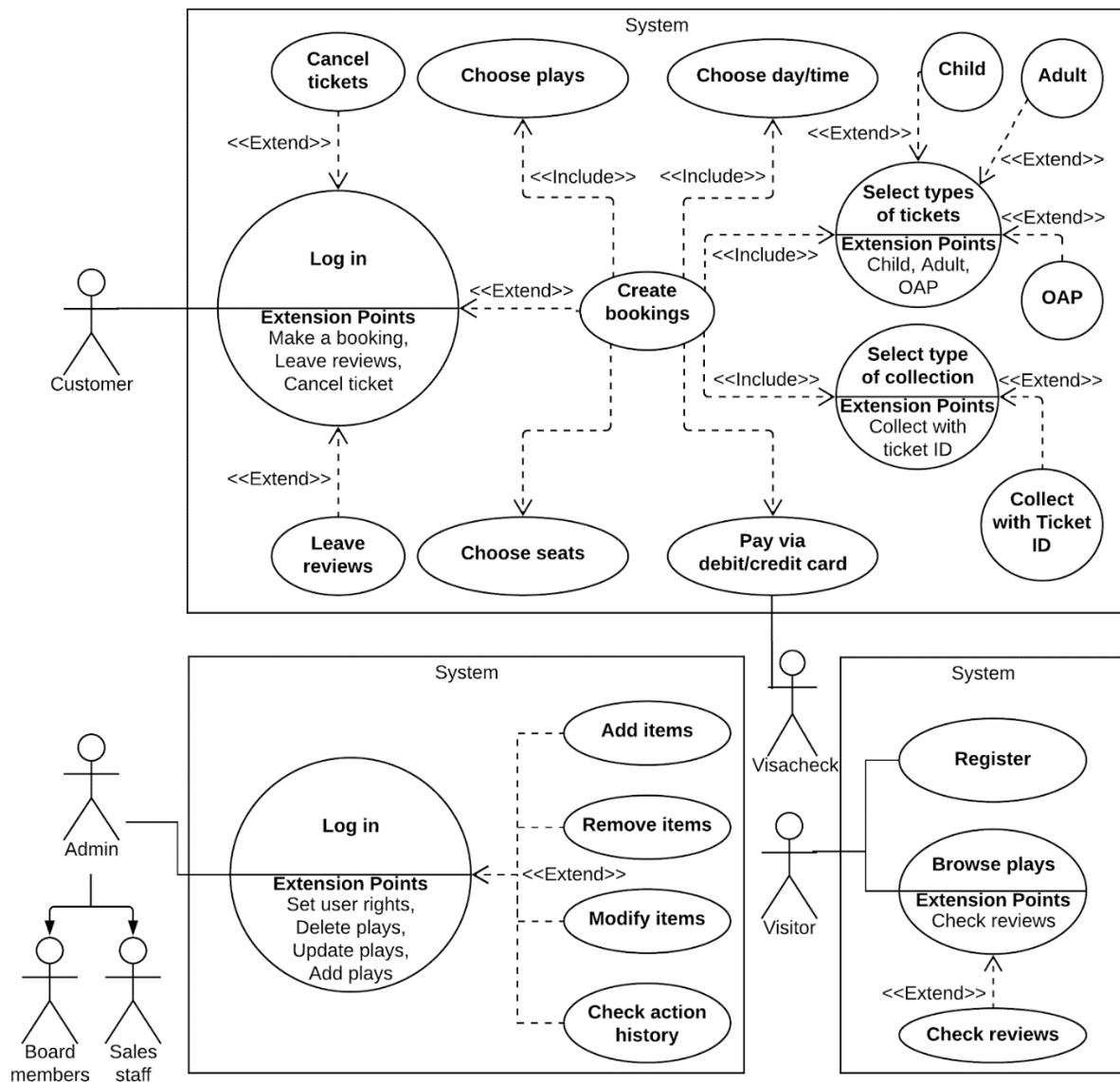
Kamilah Agbaje: 001004321
Abdullah Butt: 000993785
Trevor Kiggundu: 001001720
Mindaugas Lekavicius: 001001160
Marzieh Mohammadi: 000991406
April 2019

A report submitted in fulfilment of the requirements for the module, Systems Development Project, Computing and Information Systems Department, University of Greenwich.

Table of Contents

UML Use Case diagram:	3
Sequence/Interaction Diagrams:	4
State Chart Diagrams:	8
UML Class Diagrams:	11
Design Patterns:	13
Design problems encountered and how we solved them:	16
HCI factors:	17
Simplified user documentation (usability) to show how the program works:	19
Appendix:	25
References:	45
Personal Reflection:	47
Group members work contribution form:	47

UML Use Case diagram:



This use case was not fully implemented in the prototype: it is not possible for the user to select a ticket type that does not correspond to his/her age. In addition, “items” for administrators may mean a lot of different things - in particular, groups, bookings, carts, discounts, shipping, plays, reservations, seats, viewings, reviews, users, Visas). However, the actions needed for them are almost identical - aside from changes of passwords for users (that requires completing an additional form).

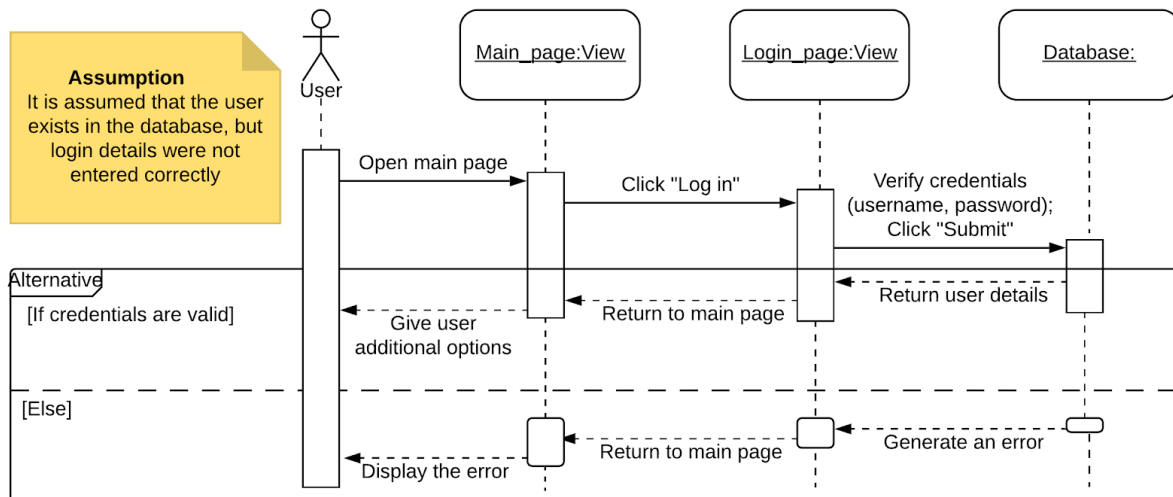
In addition, the ability for the customer to cancel purchased tickets on his own was not added to the prototype as well. However, an alternative option exists via the admin actions - therefore the customer still has access to this particular function, although indirectly and via a proxy.

Leaving reviews is possible - however, it is hard to find a way to do it on the current prototype. Also, reviews are not bound to watched plays - therefore it is possible for an unhappy customer to leave a lot of bad reviews for play he did not see. Yet, the admin action to moderate reviews allows to deal with this issue.

Sequence/Interaction Diagrams:

CUSTOMERS AND ADMINISTRATORS

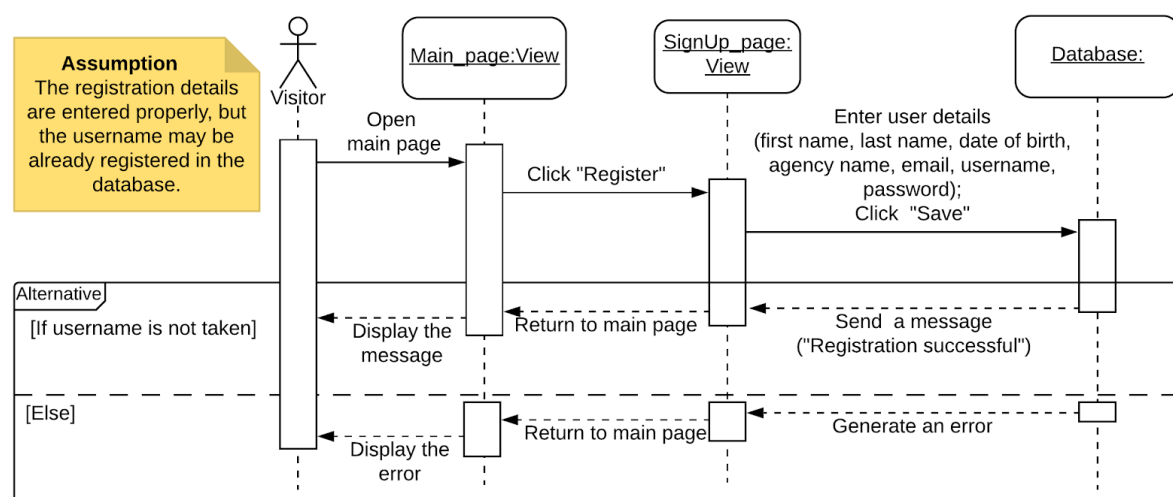
Login



Customers and administrators need to log in to use the program. For both categories, logging in is the same - however, administrators have different user details and therefore can access the admin page afterwards.

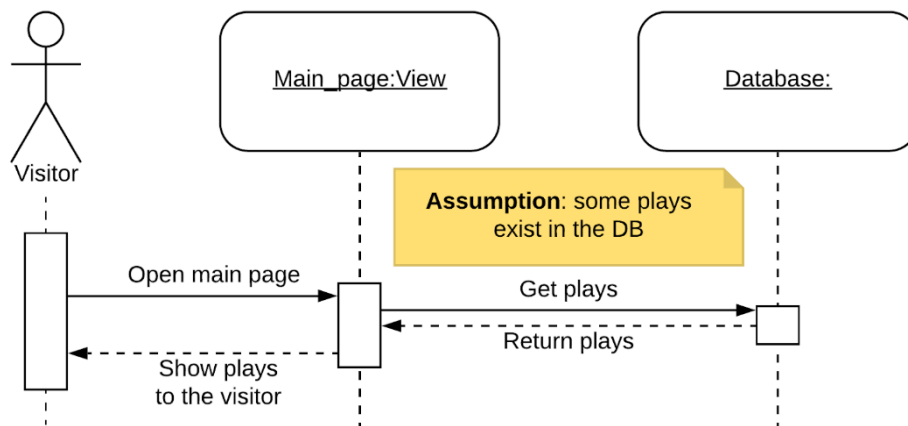
VISITORS

Register



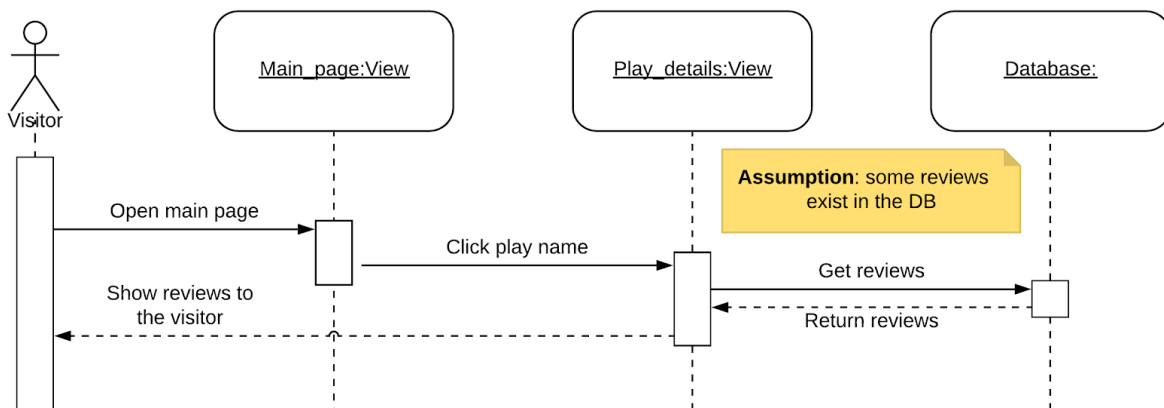
Visitors can register - via a registration form or via a proxy (admin). Administrators, however, can only be added by other administrators, restricting access to administrator status.

Browse plays



While visitors can see the plays in the theatre, without registering, they cannot make bookings or review.

Check reviews

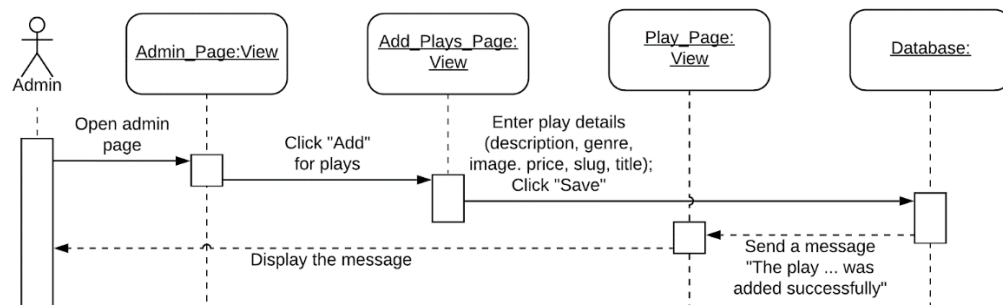


Visitors can get more details about plays by clicking in their names - amongst those details, the reviews left by customers are shown.

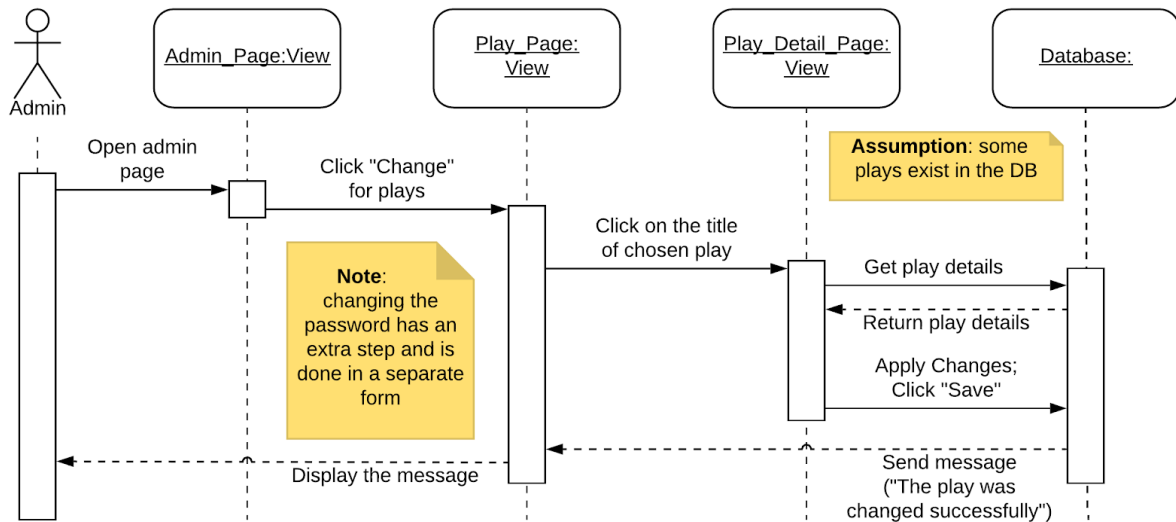
ADMINISTRATORS

Administrators can add, change and delete a lot of information. Here, plays are used as an example - however, extremely similar actions are possible for other categories of data.

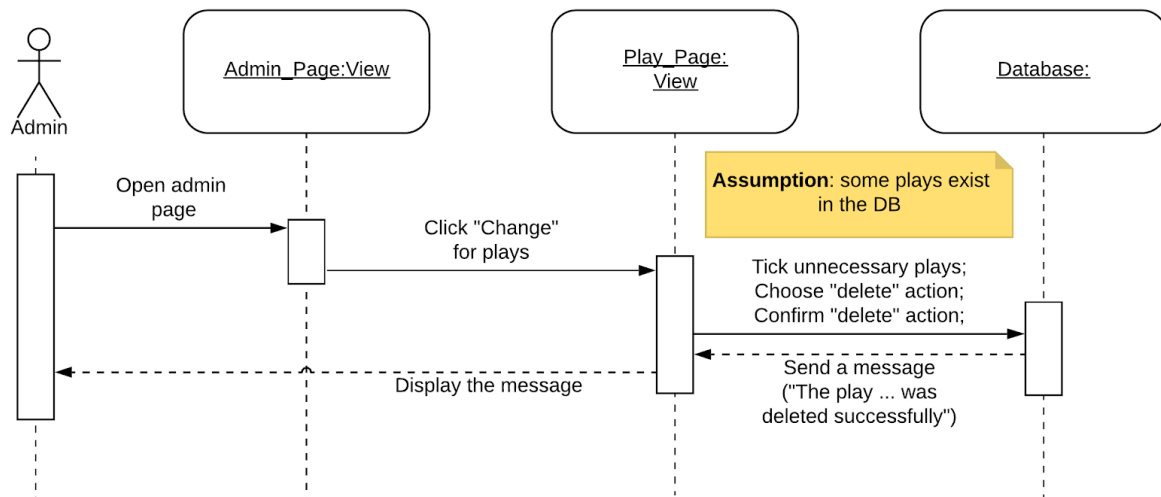
Add plays



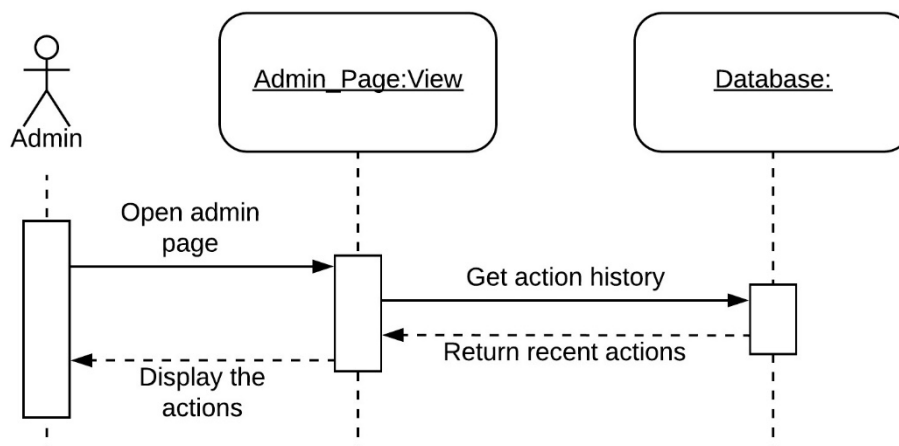
Modify plays



Delete plays

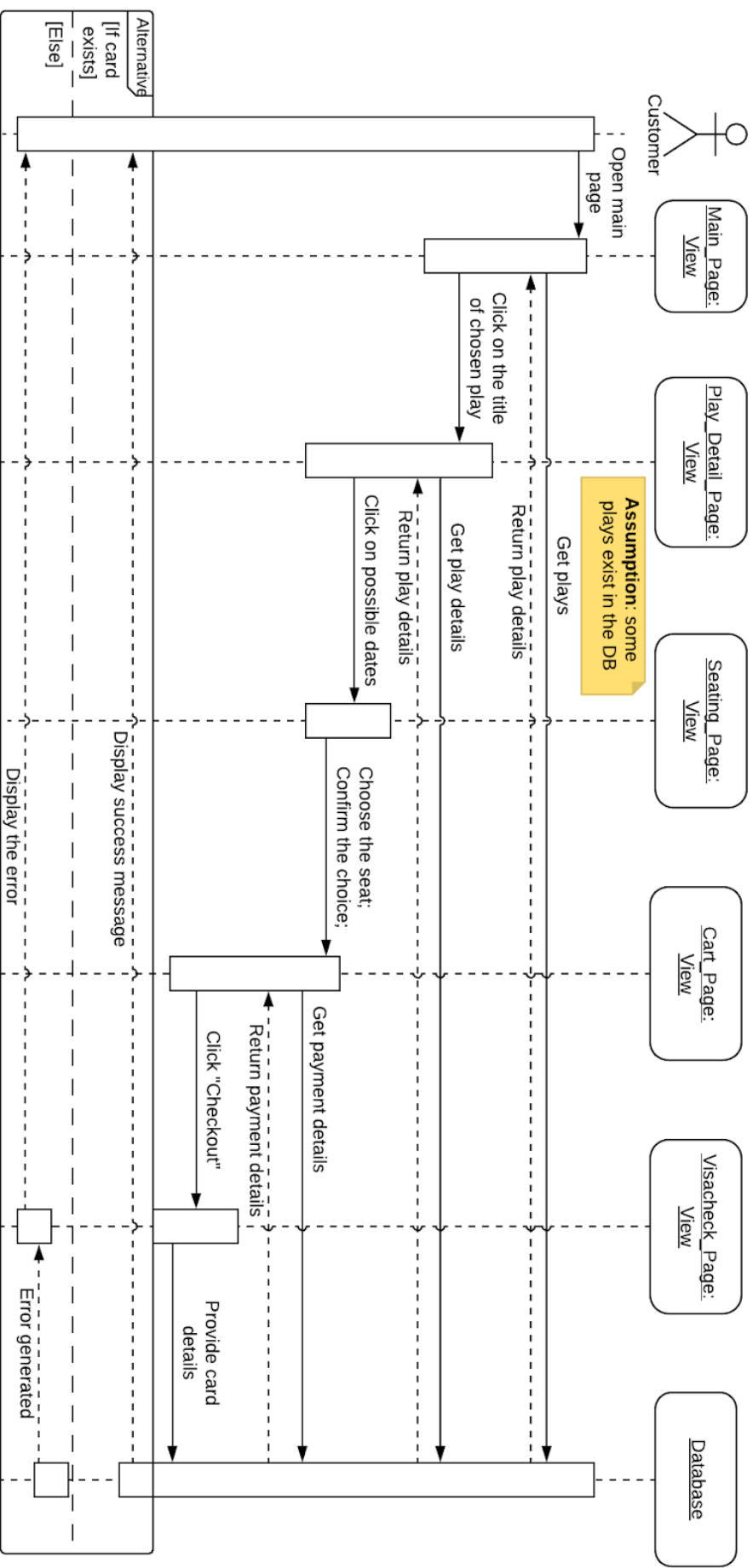


Check action history



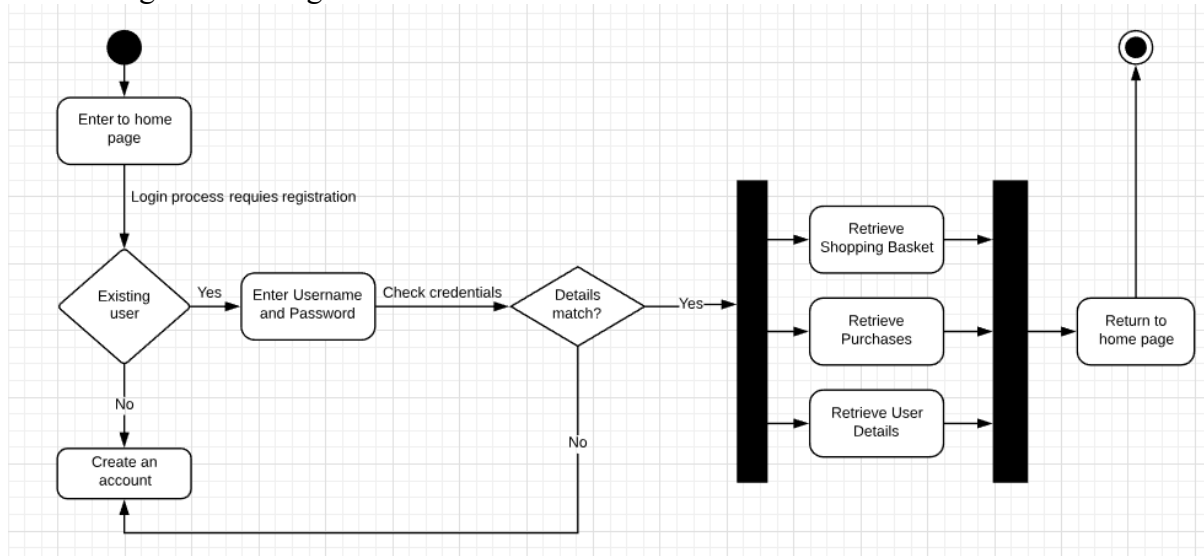
Administrators can see their most recent actions in a list to ensure that they accidentally do not change something they did not intend to change.

CUSTOMERS - create booking



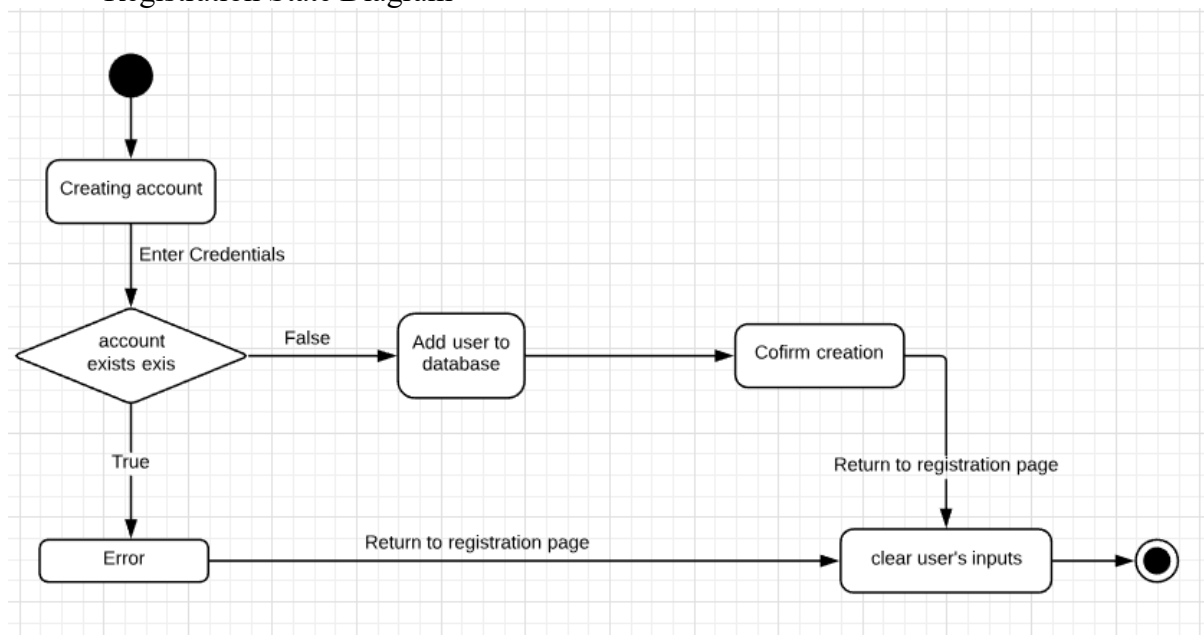
State Chart Diagrams:

Login State Diagram



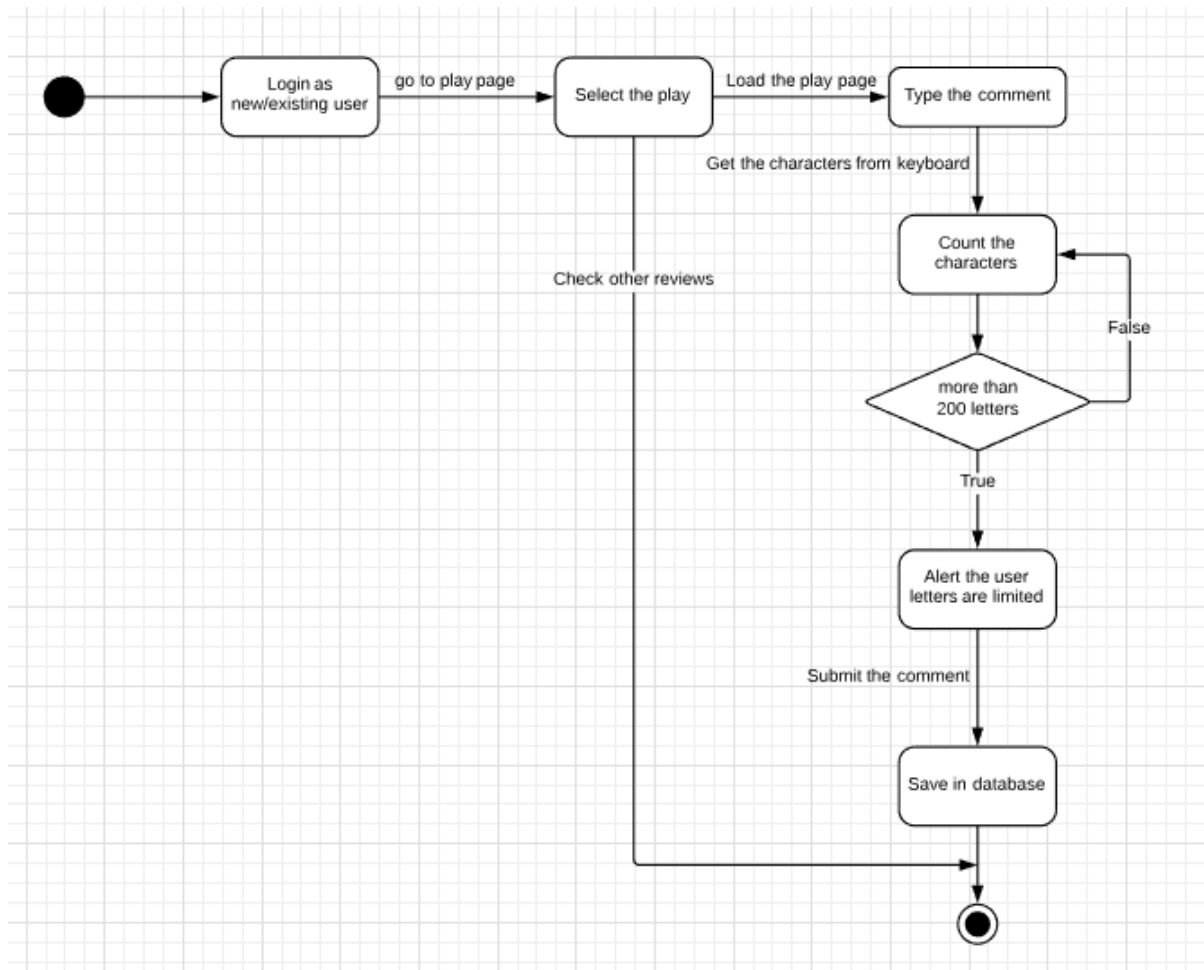
Logging in to the system requires registration. So, user needs to create an account first. If it's pre-done, user can access to the profile data as well as do shopping. Log out will return the user to home page.

Registration State Diagram



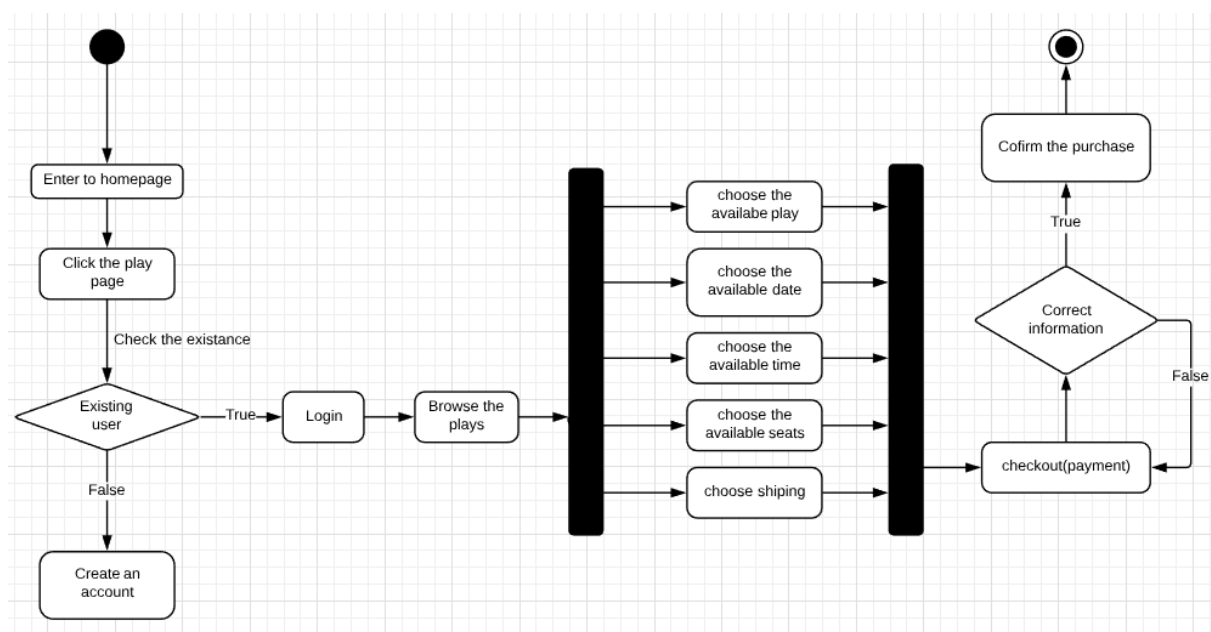
While doing the registration, system checks if entered data exists or not. If it doesn't exist, saves data to database and clears user's input after logout.

Review State Diagram



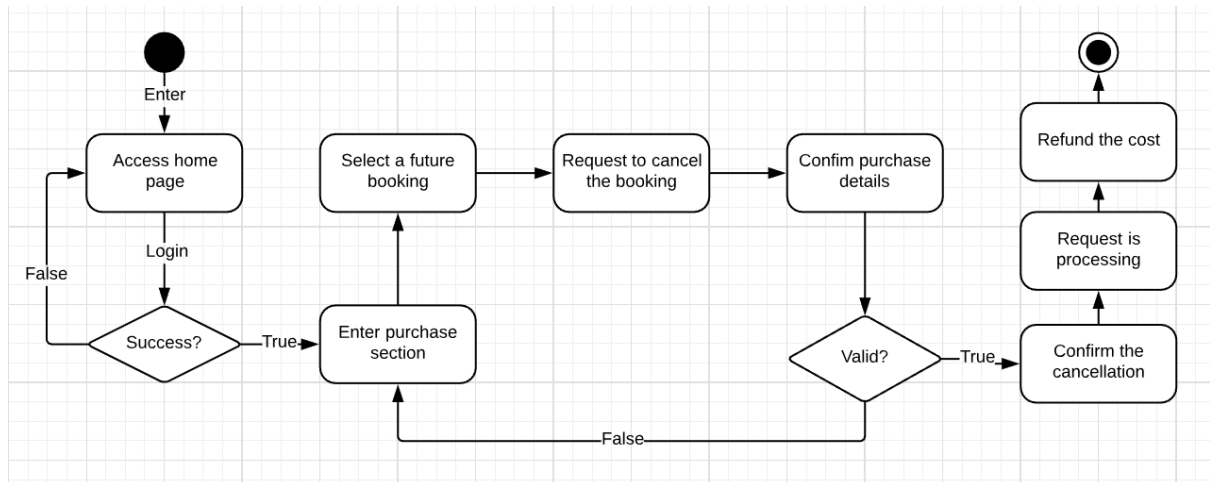
For review section, we have designed a loop to count number of characters to alert the reviewer of limited characters. In order to comment for each play, user selects the proper one and types the opinion there.

Booking State Diagram



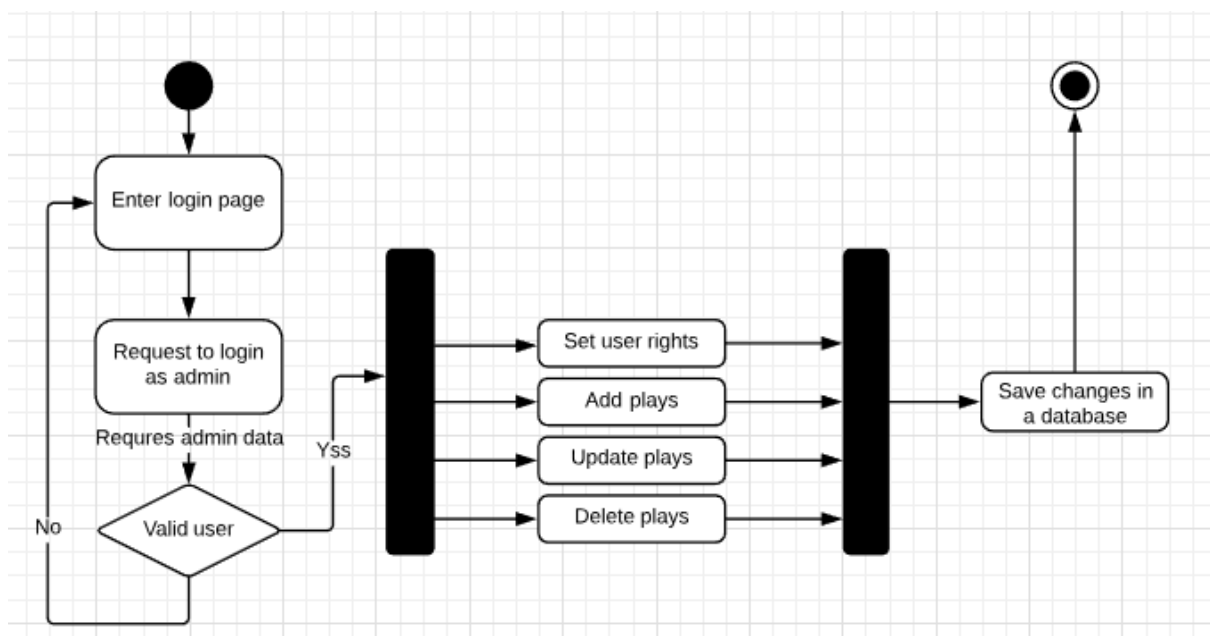
To book a ticket, first step is to login to the system by created account. Then user can browse the plays and after choosing the play and proper date and time (if available), it goes for payment section. If the cart data is valid, ticket can be booked for user.

Cancelation State Diagram



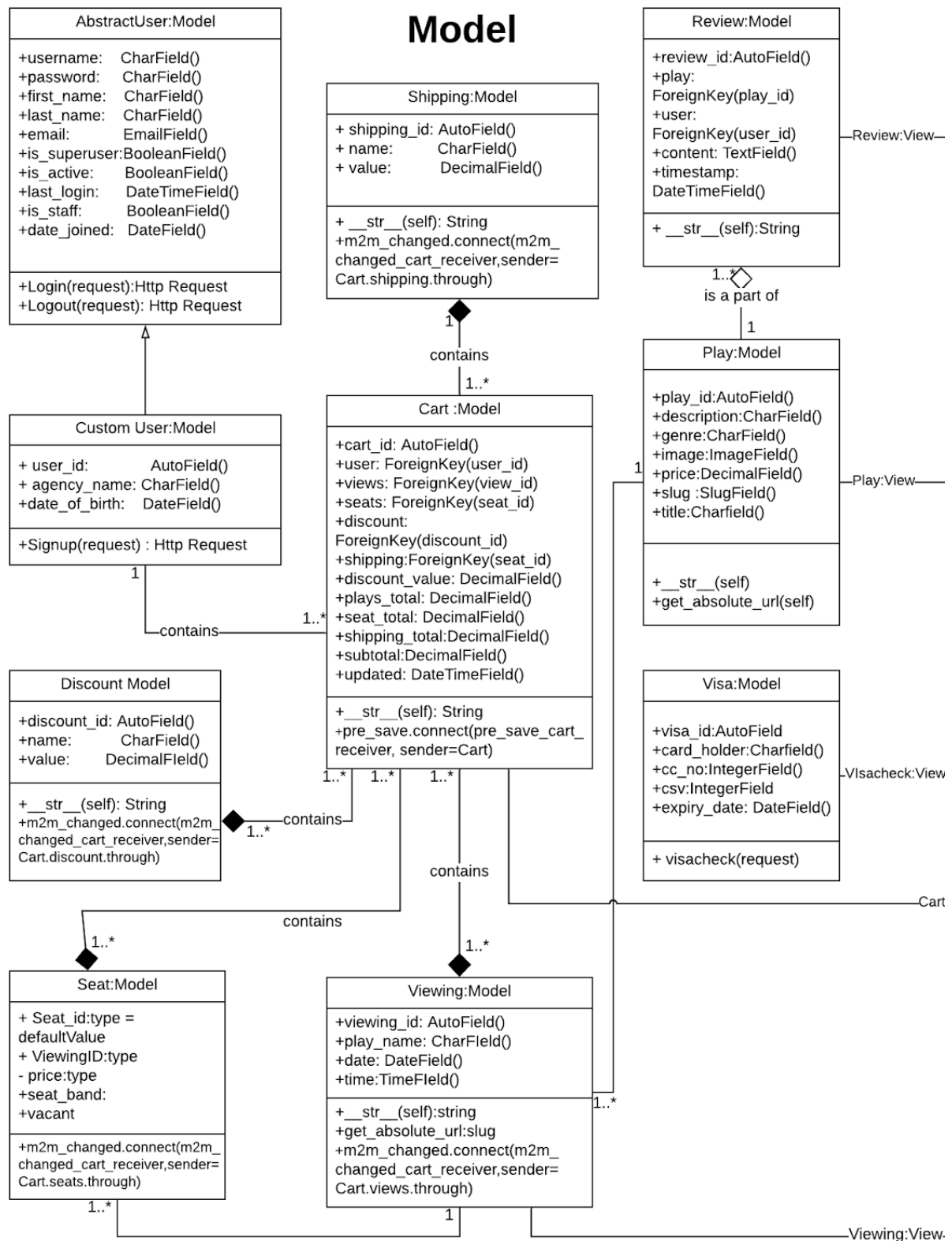
In order to cancel a booked ticket, after logging in user can enter the purchase information and if the data is still valid, cost will be returned to their account after a few days.

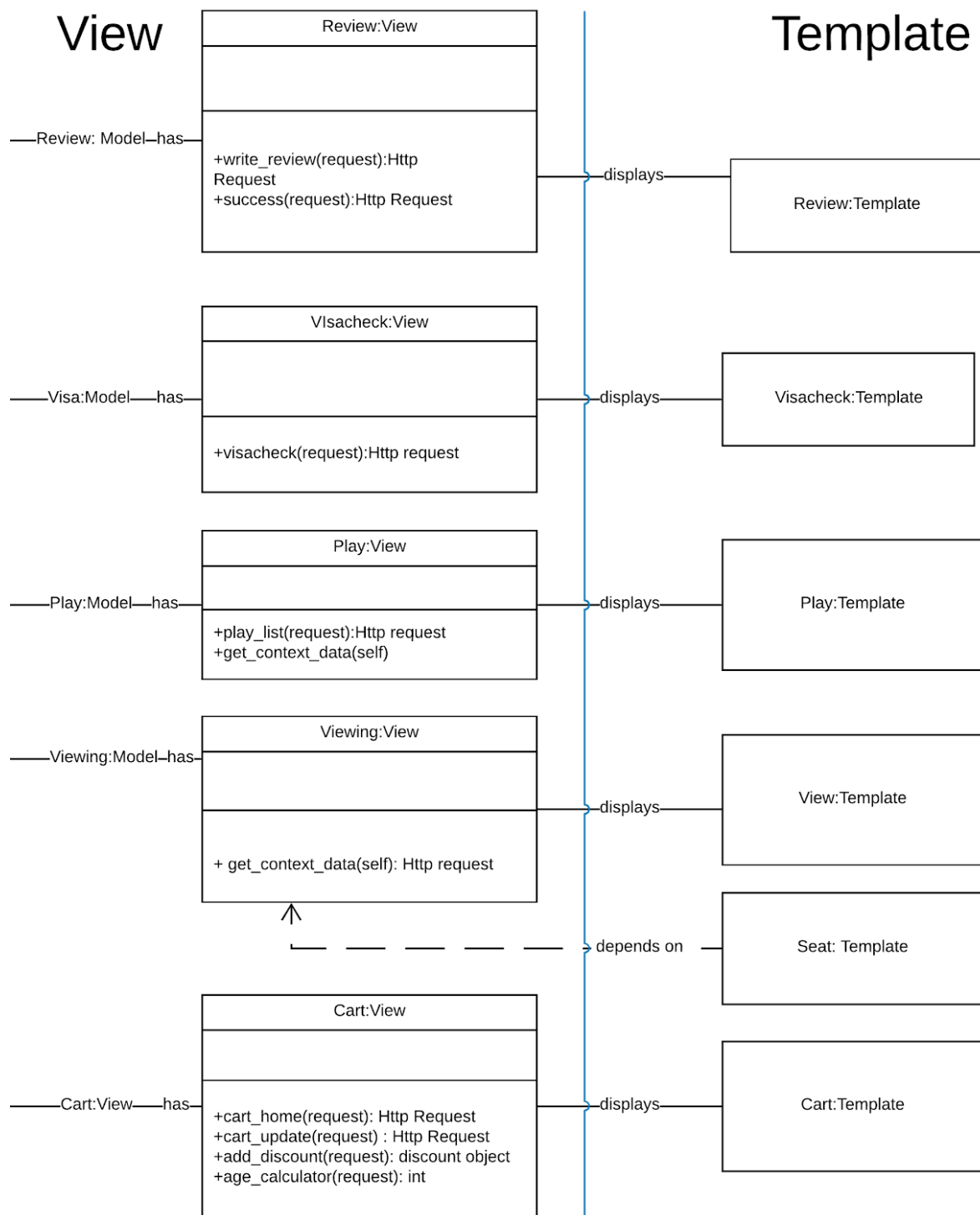
Admin Login State diagram



This system has some users as administrators who can add, delete or update the website which requires valid and specific username and password.

UML Class Diagrams:





Design Patterns:

Design patterns are generally implemented into software solutions, these create patterns that can be easily understood and read from an outside observer. Furthermore, it makes it significantly easier to work with other programmers as the layout is linear and consistent. This is illustrated by the Gang of Four (GoF), whereby the authors explained 23 commonly used design patterns in the field of software engineering. These 23 patterns can be broadly separated into 3 different categories, these include Creational, which include the patterns that offer the best ways to instantiate classes. Structural, which include the patterns that offer the best way to organize class hierarchy. Finally, behavioural which offers the best way to handle communication between objects.

One such example of a design pattern that we had used was MVT. MVT is an acronym for Model View Template, this is also known as the 'Django' design pattern. MVT differs to the more commonly used MVC due to the interactions between the system and the users. In MVT a request to a URL is dispatched to the 'View'. The 'View' then calls to the 'Model', which in turn manipulates the data and prepares it for output, the data is then passed to the 'Template' as an emitted response. MVT is the main form of design pattern that we had included in our project.

MVC is another design pattern that has been implemented in our project. Which is included within the 23 patterns explained in GoF. MVC stands for Model View Controller. Model, the first of these would be used to represent single objects or even a structure of objects within a system. View is used to visually represent the model, it would ordinarily highlight certain attributes and hide others, thus it could be defined as a presentation filter. The Controller is the connection between the user and the system itself. It manipulates what can be seen on the screen to provide the user with appropriate input, i.e. Menus or any other form of giving commands data.

The following code has been created for the 'Plays' section of our project. Both MVT and MVC can be illustrated through this code as they are both extremely similar in structure.

Model- object in database contains fields. The 'title, slug, description, price, image, genre' fields define models within the project.

```
class Play(models.Model):
    title = models.CharField(max_length=120)
    slug = models.SlugField(default='abc', unique=True)
    description = models.TextField()
    price = models.DecimalField(decimal_places=2, max_digits=20)
    image = models.ImageField()
    genre = models.CharField(max_length=120)

    def __str__(self):
        return self.title

    def get_absolute_url(self):
        return "plays/{slug}/".format(slug=self.slug)
```

View- collects the list of plays available, which in turn also collects the results of the query and sends it to a key that can be used at a later point called 'Context'.

```
def play_list(request):
    queryset = Play.objects.all()
    context = {
        'play_list':queryset
    }

    return render(request,"plays/list.html", context)
```

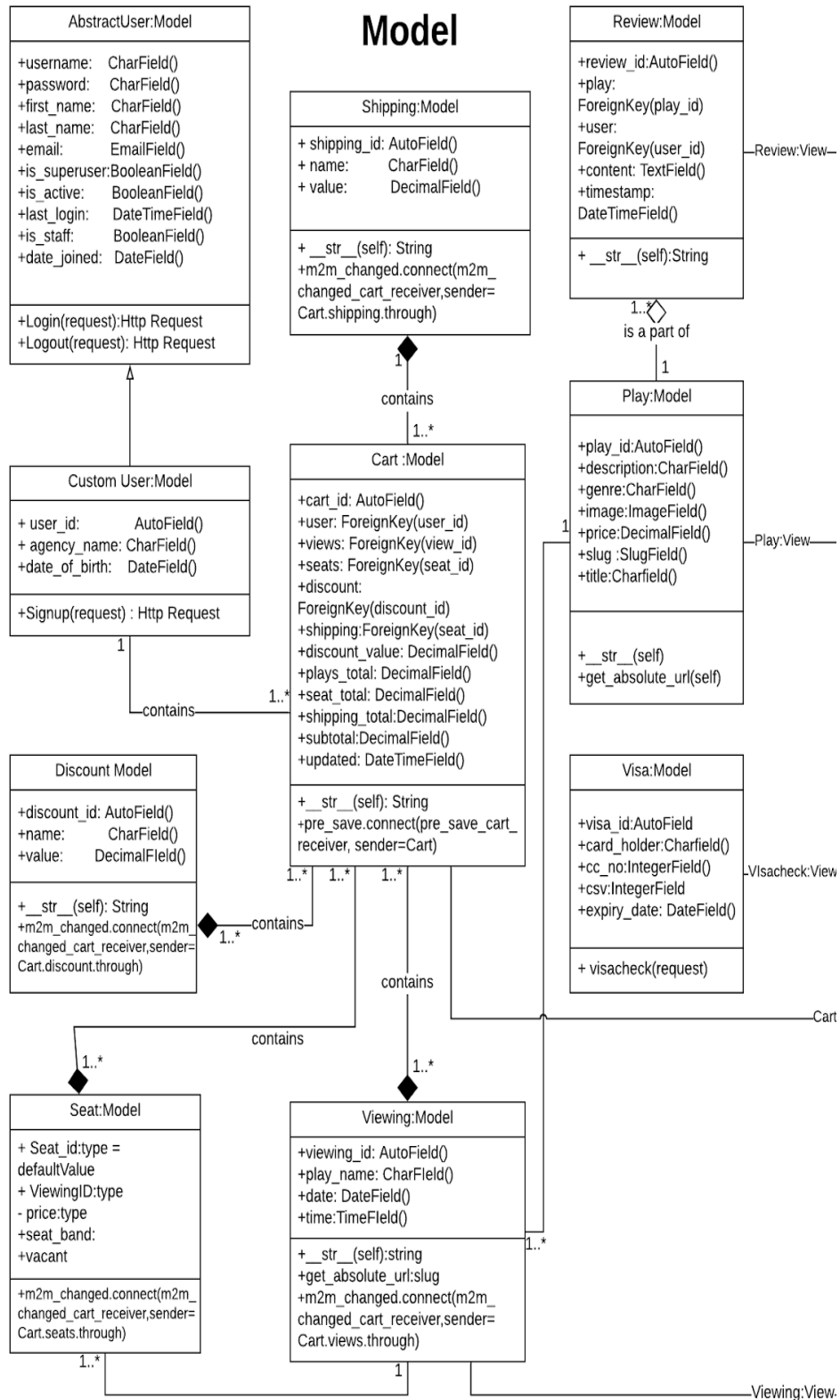
Template- the context key is passed into the template which contains a ‘for’ loop which counts through the list.

```
{% for play in play_list %}

<a href= '{{play.get_absolute_url}}'>{{ play.title }} </a> </br>
{{ play.image }} </br>
{{ play.genre }} </br>
{{ play.description }}</br>

{% endfor %}
```

Another GOF behavioural design pattern we implemented is the Observer pattern. Whereby an object often called a subject. maintains observers or listeners and notifies them of state changes of the object by calling one of their methods which will also result in the listeners being updated automatically.. We implemented this in Django with the use of signals. Our subject in this scenario is the Cart and the registered observers are the seating, discount and viewing models. When the state of the Cart changes the observers are notified by having their common method called for. ‘m2m_changed.connect()’. This will result in the cart total being recalculated. Below, is an illustration of how this has been implemented in our UML class diagram.



Design problems encountered and how we solved them:

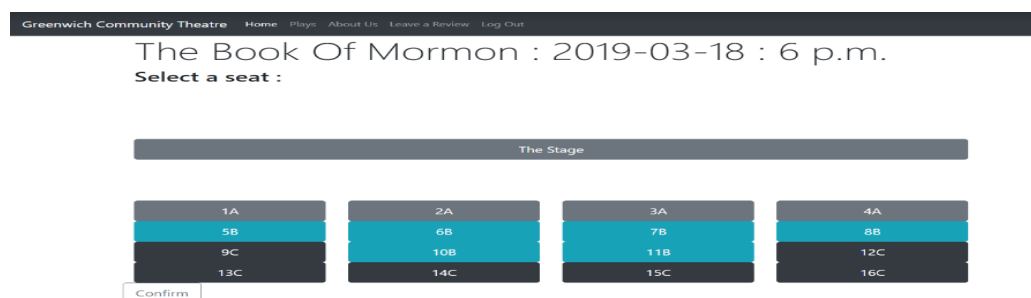
Our prototype was implemented using Django, a Python framework which is based on the Model View Controller design architecture. This served as the backend of our site alongside a Sqlite database. For the frontend, we made use of HTML and CSS with the help of Bootstrap. The core functionality of the theatre system is handled by the backend and often times we encountered situations whereby we had to think of the best approach to handle interactions with the client side.

A notable example of such a situation was when the graphical seating reservation system had to be implemented.

Although the database had been set up to handle the seats, their viewings and vacancy. It was difficult to come up with a solution to allow users to interact with the database seat object, given the fact that it also had to be laid out in a specific manner.

This issue was overcome with the use of forms, as forms are known to be the intermediary between the client and server side. However, it was not just a regular form with text boxes and widgets. Our form was completely made up of buttons that represented each seat in the database and the clicking of that button passed resulted in a POST request which sent the 'seat Id' to the database to handle whatever the user required. We also made use of html 'div' tags and table widgets to solve our issue with the layout of the buttons/seats.

Below is a screenshot of the final product.



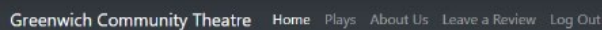
Another design/implementation problem we encountered was regarding the allocation of seats to customers. The Greenwich Community Theatre only contains one auditorium in which there are a number of seats which are physically reused per viewing. However, since the website requires users to have the ability to book a vacant seat per viewing it seemed difficult to cater one seat to many customers per viewing.

This problem was overcome by redesigning our database. Each seat object in the database has a Viewing Id Foreign key attached to it. The result is a one to many relationship in which one seat has many views. Although there are 16 physical seats in our theatre, the database illustrates 16 seats per viewing in the theatre. This way, when a customer is booking a seat online they are really just doing so for that particular session and not forever. This multiplicity between the models is described in our UML Class diagram also.

HCI factors:

HCI Factors considered:

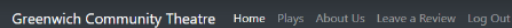
-Organizational Factors: We strived to make a system that was sequential enough that the user would not be confused when using it. A lot of this was achieved simply by modelling our theatre website after real-life systems such as ODEON and Cinemax cinema websites. A header at the top of the page helps to advertise to the user that they are indeed using the 'Greenwich Community Theatre' website. The various headings in the header, (such as plays) also aid in the organization of the website, as the user is given conventional yet useful shortcuts to navigate to other parts of the website.

A dark grey header bar with white text. On the left is 'Greenwich Community Theatre'. To its right are links: 'Home', 'Plays', 'About Us', 'Leave a Review', and 'Log Out'.

-Task Factors/System Functionality: Task allocation was the biggest feature regarding task factors that we had to implement for our cinema system. We did this by making sure that each button does what its supposed to do, (ex: 'Checkout' button taking the user to the checkout page.) This also involves doing productivity checks that are mentioned down below.

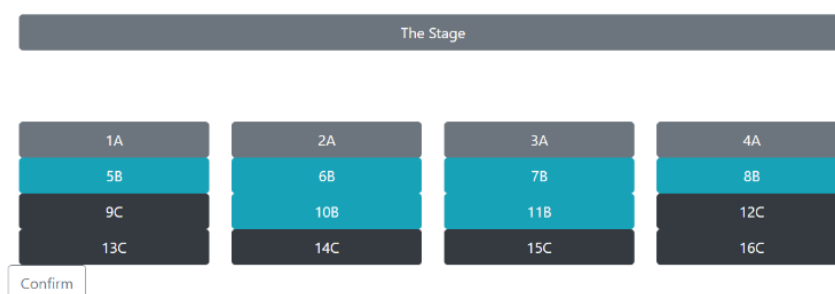
-Comfort Factors: Simplicity is key when deciding comfort factors, and we tried our best to provide that for the user. Simple additions such as a visual seating chart instead of a drop down menu, and graphics shown next to each play during play selection help to make the user feel comfortable when using our system. The user also gains trust in our system after being given access to previous theatre reviews.

- Visual Seating Chart:

A dark grey header bar with white text. On the left is 'Greenwich Community Theatre'. To its right are links: 'Home', 'Plays', 'About Us', 'Leave a Review', and 'Log Out'.

The Book Of Mormon : 2019-03-18 : 6 p.m.

Select a seat :



- Review System:

Greenwich Community Theatre Home Plays Cart Leave a Review Log Out

Play*

The Lion King

Content*

This is a test review

Submit

-The User: Similar to the organizational factors, we made sure to create a system that the user was able to cognitively comply with. Assumptions were made that the user would have used a website before, and ordered cinema/movie tickets before, and based off those assumptions, we created a system similar to other mediums that provide the same services (ODEON, Cinemax etc.) These systems have provided satisfactory customer reviews throughout the years so a comparison to them was necessary to create the best possible system.

-Productivity factors: This was by far our biggest concern, as this was the section that required error-checking on the simplest level. Things such as making sure the user's credit card is valid, making sure the user has created an account to buy tickets, password verification, making sure the buttons all function correctly etc. These all play a role in increasing the overall quality and output of the system.

- Password Verification:

Password*

.....

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation*

.....

Enter the same password as before, for verification.

-Health and Safety factors: A VISACheck system ensures that customer information is not compromised at any time during booking, also checking that their payment information is valid and consistent. A straight forward, easy-flowing website, as discussed in the organizational factors, ensures that the customer experiences no stress while using the service.

Simplified user documentation (usability) to show how the program works:

Upon first opening the program, the customer will come across a webpage asking them to either login or sign up for a 'Greenwich Community Theatre' account. If the user already has an account and logs in using the 'Login' button, then they will be redirected to the play selection page (further down below). However, if the user does not have an account, then they should select the 'Sign Up' option at the bottom of the page.

The screenshot shows the top navigation bar of the Greenwich Community Theatre website with links: Home, Plays, Cart, and Leave a Review. Below the navigation bar is the 'Login' section. It features a 'Username:' label followed by a text input field, a 'Password:' label followed by a text input field, and a 'Login' button. Below the button is the text 'Don't have an account ?' followed by a blue 'Sign Up' link.

After selecting the 'Sign Up' option, the user will be directed to a page that will prompt the user to create a username and a password, as well as add their email address and date of birth.

The screenshot shows the 'Sign up' page of the Greenwich Community Theatre website. The navigation bar includes links: Home, Plays, Cart, Leave a Review, and Log In. The 'Sign up' section contains several form fields: 'Username*' with a text input field and a note 'Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.'; 'Email address' with a text input field; 'Date of birth' with a text input field showing 'dd/mm/yyyy'; a checkbox labeled 'Are you a representing an agency?'; 'Password*' with a text input field and a list of password requirements: 'Your password can't be too similar to your other personal information.', 'Your password must contain at least 8 characters.', 'Your password can't be a commonly used password.', and 'Your password can't be entirely numeric.'; and 'Password confirmation*' with a text input field and a note 'Enter the same password as before, for verification.' Below the form fields is a 'Sign up' button.

If all the parameters are confirmed by the system, then clicking the 'Sign Up' button will create a new account.

Sign up

Username*

anothertest

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Email address

another@test.com

Date of birth

13/04/2019

☐ Are you representing an agency?

Password*

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation*

Enter the same password as before, for verification.

After a successful login attempt/account creation, the user will find a selection of plays to choose from, each offered by the Greenwich Theatre. Each play will have a graphic showing what the play will look like as well as a short description of that specific play underneath it. Alternatively, each play will also be displayed in a slide show, and the user can select which play they would like to view more information about. Both of these actions can be invoked by selecting the 'Book Now' button.

- Slide Show method:





- Playlist method:

Waitress

The musical is based on the 2007 film of the same name, written by Adrienne Shelly. It tells the story of Jenna Hunterson, a waitress in an abusive relationship with her husband Earl.

[Book Now](#)

The Lion King

This Disney feature follows the adventures of the young lion Simba, the heir of his father, Mufasa. Simba's wicked uncle, Scar, plots to usurp Mufasa's throne by luring father and son into a stampede of wildebeests. But Simba escapes, and only Mufasa is killed.

[Book Now](#)

Selecting a specific play will then lead the user to a separate page where they can view more information about the play. They will then select which day and time to view the specific play they have chosen. Previous customer reviews of that specific play will also be displayed underneath the description and date/time selection. The play selected below is 'The Book of Mormon'.

Greenwich Community Theatre Home Plays Cart Leave a Review Log Out

THE BOOK OF MORMON

The Book Of Mormon

The play makes light of various Mormon beliefs and practices, but ultimately endorses the positive power of love and service.

[Book Now](#)

- Date/Time selection, along with previous customer reviews; For this demo, the date of March 18th has been selected, with a show start time of 6 pm.

The Book Of Mormon

THE BOOK OF MORMON

The play makes light of various Mormon beliefs and practices, but ultimately endorses the positive power of love and service.

[March 18, 2019 6 p.m.](#)

[March 21, 2019 6 p.m.](#)

1 Review

I really enjoyed this play

— by Admin

After date and time selection, the user is then prompted to select which seat they would like to book and sit in. A seating chart is shown as a visual aid. After the user has selected their seat(s), they should press the ‘Confirm’ button.

Greenwich Community Theatre Home Plays About Us Leave a Review Log Out

The Book Of Mormon : 2019-03-18 : 6 p.m.

Select a seat :

The Stage

1A	2A	3A	4A
5B	6B	7B	8B
9C	10B	11B	12C
13C	14C	15C	16C

[Confirm](#)

After seat selection, the user is taken to the ‘Cart’ page, where the customer’s receipt is shown. A summary of the play name, date, prices, seat selection, subtotal, discount, shipping and total overall cost are shown. If the user is satisfied with all the details, then they should press the ‘Checkout’ button to continue.

Cart

#	Play Name	Play Date	Play Time
1	The Book Of Mormon	March 18, 2019	6 p.m.
			Plays Total: £20.00
#	Seat Number	Seat Band	Seat Price
1	1A	A	30.00
			Seat Total £30.00
<input type="radio"/> Standard Shipping <input checked="" type="radio"/> First Class <input type="radio"/> Pick up Add Shipping			
			Subtotal £50.00
			Discount* 1.00
			Shipping £5.99
			Total £55.99

* discount value is automatically calculated and multiplied to total before shipping

[Checkout](#)

The Checkout page consists of a “VISACheck” system that allows the user to provide their bank/card details in order to successfully checkout and receive their tickets. The user enters their name, card number, CVV number and expiry date. If the user is satisfied with all the details, then they should press the ‘Submit’ button to continue.

Checkout

Card holder*

Cc no*

Expiry date*

Csv*

After completing checkout, a message is displayed thanking the user for their purchase. They are also notified that they will be receiving their tickets based on their desired form of

collection. A button labelled 'Book another play?' is also displayed in case the customer would like to make another purchase.

Thank you for your order!
Your Ticket will be ready for shipment/
collection shortly.

Book another play?

The review page is a separate entity that allows a customer to leave a review on a play that they have watched before. This page will appear after a successful checkout.

Greenwich Community Theatre Home Plays Cart Leave a Review Log Out

Play*

The Lion King

Content*

This is a test review!

Submit

After leaving a review, the customer should press the 'Submit' button. Once that button is pressed, another message thanking the customer for their cooperation is displayed. Once again, A button labelled 'Book another play?' is also displayed in case the customer would like to make another purchase. Clicking that button will return the user to the home page.

Thank you for your review !

Book another play?

Appendix:

Listings of any code:

Models.cart

```
from django.conf import settings
from django.db import models
from django.db.models.signals import pre_save, post_save, m2m_changed
from plays.models import Play, Seat, Viewing
from decimal import Decimal

from django.shortcuts import render, redirect

User = settings.AUTH_USER_MODEL
# Create your models here.
class Discount(models.Model):
    name = models.CharField(max_length=120)
    value = models.DecimalField(default=0.00, max_digits=100, decimal_places=2)

    def __str__(self):
        return str(self.name)

class Shipping(models.Model):
    types = [('Standard Shipping', 'Standard Shipping'), ('First Class', 'First Class'), ('Pick up', 'Pick up')]
    name = models.CharField(choices=types, default=None, null=True, max_length=100,)
    value = models.DecimalField(default=0.00, max_digits=100, decimal_places=2)

    def __str__(self):
        return str(self.name)

class CartManager(models.Manager):

    def new_or_get(self, request):
        cart_id = request.session.get("cart_id", None)
        qs = self.get_queryset().filter(id=cart_id)
        if qs.count() == 1:
            new_obj = False
            print('Cart id exists')
            cart_obj = qs.first()
            if request.user.is_authenticated and cart_obj.user is None:
                cart_obj.user = request.user

            cart_obj.save()
        else:
            cart_obj = Cart.objects.new(user=request.user)
            new_obj = True
```

```

        request.session["cart_id"] = cart_obj.id
    return cart_obj, new_obj

```

```

class Cart(models.Model):
    user = models.ForeignKey(User, null=True, blank=True, on_delete=models.CASCADE)
    #plays = models.ManyToManyField(Play, blank=True)
    views = models.ManyToManyField(Viewing, blank=True)
    seats = models.ManyToManyField(Seat, blank=True)
    discount = models.ManyToManyField(Discount, blank=True)
    shipping = models.ManyToManyField(Shipping, blank=True)

    playstotal = models.DecimalField(default=0.00, max_digits=100, decimal_places=2)
    seattotal = models.DecimalField(default=0.00, max_digits=100, decimal_places=2)
    discountvalue = models.DecimalField(default=0.00, max_digits=100, decimal_places=2)
    shippingtotal = models.DecimalField(default=0.00, max_digits=100, decimal_places=2)

    subtotal = models.DecimalField(default=0.00, max_digits=100, decimal_places=2)
    total = models.DecimalField(default=0.00, max_digits=100, decimal_places=2)
    updated = models.DateTimeField(auto_now_add=True)

    objects = CartManager()

    def __str__(self):
        return str(self.id)

def m2m_view_cart(sender, instance, action, *args, **kwargs):
    if action == 'post_add' or action == 'post_remove' or action == 'post_clear':
        #plays = instance.plays.all()
        views = instance.views.all()
        total = 0
        for x in views:

            #for x in plays:
                total += Decimal(20.00)
            instance.playstotal = total

        instance.save()

m2m_changed.connect(m2m_view_cart, sender=Cart.views.through)

def m2m_seat_cart(sender, instance, action, *args, **kwargs):
    if action == 'post_add' or action == 'post_remove' or action == 'post_clear':

        seats = instance.seats.all()

```

```

total = 0

for x in seats:
    total += x.price
instance.seattotal = total

instance.save()

m2m_changed.connect(m2m_seat_cart,sender=Cart.seats.through)

def m2m_discount_cart(sender, instance, action, *args, **kwargs):
    if action == 'post_add' or action == 'post_remove' or action == 'post_clear':

        discount = instance.discount.all()

        total = 0

        for x in discount:
            total += x.value
        instance.discountvalue = total

        instance.save()

m2m_changed.connect(m2m_discount_cart,sender=Cart.discount.through)
#
def m2m_changed_cart_receiver(sender, instance, action, *args, **kwargs):
    if action == 'post_add' or action == 'post_remove' or action == 'post_clear':

        shipping = instance.shipping.all()

        total = 0

        for x in shipping:
            total += x.value
        instance.shippingtotal = total

        instance.save()

m2m_changed.connect(m2m_changed_cart_receiver,sender=Cart.shipping.through)

def pre_save_cart_receiver(sender, instance, *args, **kwargs):
    instance.subtotal = instance.playstotal + instance.seattotal
    instance.total = ((instance.subtotal)*instance.discountvalue) + instance.shippingtotal
    # instance.total = instance.subtotal #+seats #-discounts
pre_save.connect(pre_save_cart_receiver, sender=Cart)

```

Views.cart

```

from django.shortcuts import render, redirect
from django.conf import settings
from datetime import date
# Create your views here.
from plays.models import Play, Viewing, Seat
from .models import Cart, Discount, Shipping

def cart_home(request):
    cart_obj, new_obj = Cart.objects.new_or_get(request)
    ship = Shipping.objects.all()
    context = {
        'cart': cart_obj,
        'ship': ship
    }
    return render(request, "carts/home.html", context)

def cart_update(request):
    print(request.POST)
    # play_id = request.POST.get('play_id') #gets play to put in basket
    view_id = request.POST.get('view_id')
    seat_id = request.POST.get('seat')
    # ship_id = request.POST.get('ship_id')

    #if play_id is not None:
    #if view_id is not None:

        view_obj = Viewing.objects.get(id=view_id)
        seat_obj = Seat.objects.get(id=seat_id)
        # ship_obj = Shipping.objects.get(id=ship_id)

        #play_obj = Play.objects.get(id=play_id)
        cart_obj, new_obj = Cart.objects.new_or_get(request)

        if view_obj in cart_obj.views.all():
            #cart_obj.views.remove(view_obj)
            cart_obj.views.add(view_obj)
            cart_obj.seats.add(seat_obj)

            cart_obj.discount.add(add_discount(request))

        else:

            cart_obj.views.add(view_obj)
            cart_obj.seats.add(seat_obj)

```

```

        cart_obj.discount.add(add_discount(request))
        request.session['cart_items'] = cart_obj.views.count()

#cart_obj.plays.remove(play_obj)
return redirect('cart:home')

def cart_update_again(request):
    print(request.POST)
    ship_id = request.POST.get('ship')

    #if play_id is not None:
    if ship_id is not None:

        ship_obj = Shipping.objects.get(id=ship_id)

        print(ship_obj)

        #play_obj = Play.objects.get(id=play_id)
        cart_obj, new_obj = Cart.objects.new_or_get(request)

        if ship_obj in cart_obj.shipping.all():
            # cart_obj.shipping.remove(ship_obj)
            cart_obj.shipping.add(ship_obj)

        else:
            cart_obj.shipping.add(ship_obj)

        request.session['cart_items'] = cart_obj.views.count()

#cart_obj.plays.remove(play_obj)
return redirect('cart:home')

def add_discount(request):

    #1- OAP 2-child 3-none 4-as1 5-as2 6-hr
    if request.user.is_authenticated:
        # today = date.today()
        # birthday = request.user.date_of_birth
        # days_in_year = 365.2425
        age = age_calculator(request)

        if age >= 65 and request.user.agency_name == False:
            discount_obj = Discount.objects.get(pk=1)
            # return discount_obj

```

```

elif age <= 18 and request.user.agency_name == False:
    discount_obj = Discount.objects.get(pk=2)
    # return discount_obj
elif age >18 and age < 65 and request.user.agency_name == False:

    discount_obj = Discount.objects.get(pk=3)
    # return discount_obj
elif request.user.agency_name == True:
    discount_obj = Discount.objects.get(pk=4)
    # return discount_obj
else:
    discount_obj = Discount.objects.get(pk=3)
else:
    discount_obj = Discount.objects.get(pk=3)

return discount_obj

```

```

def age_calculator(request):
    today = date.today()
    birthday = request.user.date_of_birth
    days_in_year = 365.2425
    age = int((date.today() - birthday).days / days_in_year)
    print (age)
    return age

```

Home.html

```
{%extends "base.html"%}
```

```
{% block content %}
```

```
<h1>Cart</h1>
```

```
{% if cart.views.exists and cart.seats.exists %}
```

```
<div class="container">
```

```
<table class="table cart-table">
```

```
<thead>
```

```
<tr>
```

```
<th>#</th>
```

```
<th>Play Name</th>
```

```
<th>Play Date</th>
```

```
<th>Play Time</th>
```

```
</tr>
```

```
</thead>
```

```
<tbody class='cart-body'>
```

```
{% for view in cart.views.all %}
```

```
<tr class='cart-product'>
```

```

        <th scope="row">{{ forloop.counter }}</th>
        <td>{{ view.play_name }}

    </td>

    <td>{{ view.date }}</td>

        <td>{{ view.time }}</td>
    </tr>
    {% endfor %}

<tr>
    <td colspan="3"></td>
    <td><b>Plays Total:</b> £<span class='cart-total'>{{ cart.playstotal }}</span></td>
</tr>

</tbody>
</table>

<div class='cart-item-remove-form' style='display:none'>

</div>

```

```

<table class="table">
    <thead>
        <tr>
            <th scope="col">#</th>
            <th scope="col">Seat Number</th>
            <th scope="col">Seat Band</th>
            <th scope="col">Seat Price</th>
        </tr>
    </thead>
    <tbody>
        {%for seat in cart.seats.all%}

        <tr>

            <th scope="row">{{ forloop.counter }}</th>
            <td>{{ seat.number }}</td>
            <td>{{ seat.seat_band }}</td>
            <td>{{ seat.price }}</td>
        </tr>

```

```

{%endifor%}

<tr>
  <td colspan="3"></td>
  <td><b>Seat Total</b> £<span class='cart-total'>{{ cart.seattotal }}</span></td>
</tr>
</tbody>
</table>

<div class="col">
<form method='POST' action='{% url "cart:updatea" %}' data-endpoint='{% url
"cart:updatea" %}' class="form" {% if request.user.is_authenticated %}data-user='abc'{%
endif %}> {% csrf_token %}

{% for o in ship %}

  <input type="radio" name="ship" value="{{ o.id }}">{{ o.name }}

{%endifor%}
  <button type='submit' class='btn btn-link'>Add Shipping</button></div>

</form>
</div>

  <th scope="row"></th>

<table class="table">
  <tbody>
    <tr>

      <td colspan="1"></td>
      <td><b>Subtotal</b> £<span class='cart-subtotal'>{{ cart.subtotal }}</span></td>
    </tr>
    <tr>
      <td colspan="1"></td>
      <td><b>Discount*{{ cart.discountvalue.name }}</b> <span class='cart-total'>{{
cart.discountvalue }}</span></td>
    </tr>
    <tr>
      <td colspan="1"></td>
      <td><b>Shipping</b> £<span class='cart-total'>{{ cart.shippingtotal }}</span></td>
    </tr>
    <tr>
      <td colspan="1"></td>
      <td><b>Total</b> £<span class='cart-total'>{{ cart.total }}</span></td>
    </tr>
  </tbody>
</table>

```



```

<tr>
  <td colspan="2"></td>

</tr>
</tbody>
</table>

```

```

<form method='POST' action='{% url "visacheck:visacheck" %}' data-endpoint='{% url
"visacheck:visacheck" %}' class="form" {% if request.user.is_authenticated %}data-
user='abc'{% endif %}> {% csrf_token %}
<p> * discount value is automatically calculated and multiplied to total before shipping
costs</p>

```

```

  <button type='submit' class='btn btn btn-primary'>Checkout</button></div>

```

```

</form>

```

```

</div>
{% else %}
<p class='lead'>Cart is empty</p>
{% endif %}
{% endblock %}

```

Plays.model

```

from django.db import models

```

```

# Create your models here.

```

```

class Play(models.Model):
    title = models.CharField(max_length=120)
    slug = models.SlugField(default='abc', unique=True)
    description = models.TextField()
    price = models.DecimalField(decimal_places=2, max_digits=20)
    image = models.ImageField()
    genre = models.CharField(max_length=120)

```

```

    def __str__(self):
        return self.title

```

```

    def get_absolute_url(self):
        return "plays/{slug}/".format(slug=self.slug)

```

```

class Viewing(models.Model):
    play_name = models.ForeignKey(Play, on_delete=models.CASCADE)
    date = models.DateField()

```

```

time = models.TimeField()

def __str__(self):
    return str(self.date)

def get_absolute_url(self):
    return "plays/seat/{pk}".format(pk=self.pk)

class Seat(models.Model):
    bands = [('A', 'A'), ('B', 'B'), ('C', 'C')]

    number = models.CharField(max_length=100, default='0')
    viewing = models.ForeignKey(Viewing, null=True, on_delete=models.CASCADE)
    seat_band = models.CharField(choices=bands, default=None, null=True, max_length=100,)
    price = models.DecimalField(decimal_places=2, max_digits=20)
    vacant = models.BooleanField(default=True)

    def __str__(self):
        return str(self.number)

Plays.views
from django.shortcuts import render, get_object_or_404
from django.http import Http404, HttpResponse, HttpResponseRedirect
from django.contrib.auth.mixins import LoginRequiredMixin
from django.views.generic import ListView, DetailView
from django.views.generic.edit import FormMixin
from django.contrib.auth.decorators import login_required
from django.utils.decorators import method_decorator

# Create your views here.
from cart.models import Cart
from .models import Play, Viewing, Seat
from reviews.models import Review

#change slider to home page, add album to play list
def play_list(request):
    queryset = Play.objects.all()
    context = {
        'plays':queryset
    }

    return render(request,"plays/list.html", context)

def home_slider(request):
    queryset = Play.objects.all()

```

```

context = {
    'plays': queryset
}

return render(request, "home.html", context)

```

```

class ViewDetailView(DetailView):
    queryset = Viewing.objects.all()
    template_name = "plays/seat.html"

    def get_context_data(self, *args, **kwargs):
        context = super(ViewDetailView, self).get_context_data(*args, **kwargs)
        pk = self.kwargs.get('pk')
        instance = Viewing.objects.get(pk=pk)
        context['s'] = Seat.objects.filter(viewing=instance)
        #context['form'] = BookingForm(initial={'views': instance, 'plays': instance.play_name})

        return context

```

```

class PlayDetailView(DetailView):
    context_object_name = 'book'
    queryset = Play.objects.all()
    template_name = "plays/detail.html"

    def get_context_data(self, *args, **kwargs):
        context = super(PlayDetailView, self).get_context_data(*args, **kwargs)

        cart_obj = Cart.objects.new_or_get(self.request)
        new_obj = Cart.objects.new_or_get(self.request)

        #pk = self.kwargs['pk']
        #instance = Play.objects.get(pk=pk)
        slug = self.kwargs.get('slug')
        instance = Play.objects.get(slug=slug)
        context['v'] = Viewing.objects.filter(play_name__title=instance)
        context['r'] = Review.objects.filter(play__title=instance).order_by('-id')
        context['cart'] = cart_obj
        return context

```

Detail.html

```
{%extends "base.html"%}
```

```
{% block content %}
```

```
<div>
```

```
<h1 class="display-3">{{object.title}}</h1>
```

```
<img src = '{{object.image.url}}' class = 'rounded float-left' style="max-width:500px; max-height:500px !important;"/>
```

```
Genre: {{object.genre}}</br>
```

```
{{object.description}}</br>
```

```
</div>
```

```
<div>
```

```
<ul class="list-group">
```

```
{% for i in v %}
```

```
<li class="list-group-item"> <a href= '{{i.get_absolute_url}}'>{{ i.date}} {{i.time}} </a></li>
```

```
{% endfor %}
```

```
</ul>
```

```
</div>
```

```
<div>
```

```
{{r.count}} Review{{ r|pluralize }}
```

```
{% for i in r %}
```

```
<blockquote class="blockquote">
```

```
<p class="mb-0">{{i.content}}</p>
```

```
<footer class="blockquote-footer">by <cite title="Source Title">{{i.user|capfirst}}</cite></footer>
```

```
</blockquote>
```

```
</div>
```

```
{% endfor %}
```

```
</div>
```

```
{%endblock%}
```

List.html

```
{%extends "base.html"%}
```

```
{% block content %}
```

```
<div class="container">
```

```
<div class="row">
```

```
<div class="card" style="width: 40rem;">
```

```
    {% for play in plays%
```

```
    
```

```
    <div class="card-body">
```

```
        <h5 class="card-title">{{play.title}}</h5>
```

```
        <p class="card-text">{{play.description}}</p>
```

```
        <a href="{{ play.get_absolute_url }}" class="btn btn-primary">Book Now</a>
```

```
    </div>
```

```
    {%endfor%}
```

```
</div>
```

```
</div>
```

```
</div>
```

```
{% endblock %}
```

Seat.html

```
{%extends "base.html"%}
```

```
{% block content %}
```

```
<h1 class="display-4 ">{{object.play_name}} : {{object}} : {{object.time}}</h1>
```

```
<h2> Select a seat : </h2>
```

```
</br>
```

```
</br>
```

```
</br>
```

```
</br>
```

```
</br>
```

```

<div class="row">
  <div class="col-sm-12">

    <button type="button" class="btn btn-secondary btn-lg btn-block">The
Stage</button>

  </div>
</div>

</br>
</br>
</br>
</br>

<form method='POST' action='{% url "cart:update" %}' data-endpoint='{% url "cart:update"
%}' class="form" {% if request.user.is_authenticated %} data-user='abc' {% endif %}> {%
csrf_token %}

<div class="row">
  <input type='hidden' name='view_id' value='{ { object.id } }' />
  {% for o in s %}
    {% if o.vacant == True %}

      <div class="col-sm-3">

        <div class="btn-group-toggle btn-block" data-toggle="buttons">
          {% if o.seat_band == 'A' %}
            <label class="btn btn-secondary btn-lg btn-block">
              {%elif o.seat_band == 'B' %}
                <label class="btn btn-info btn-lg btn-block">

                  {%elif o.seat_band == 'C' %}
                    <label class="btn btn-dark btn-lg btn-block">
                      {%endif%}

                      <input type="radio" name="seat" value="{ {o.id} }" autocomplete="off">
{{o}}
                    </label>
                  </div>
                </div>

              </div>
            </div>
          </div>
        </div>
      </div>
    {% endif %}
  {% endfor %}
</div>

```

```

        {% else %}
        <div class="col-sm-3">
        <div class="btn-group-toggle btn-block" data-toggle="buttons" >
            <label class="btn btn-light ">
                <input type="radio" name="seat" value="{{ o.id }}" autocomplete="off"
disabled> {{ o }}
            </label>
        </div>
        </div>

        {% endif %}
    {% endfor %}

</br>
</br>

    <button type="submit" class="btn btn-outline-secondary btn-lg">Confirm</button>

</div>

</form>

```

```
{%endblock%}
```

Reviews view

```

from django.shortcuts import render
from django.http import HttpResponseRedirect
from .forms import ReviewForm
from .models import Review
from plays.models import Play
# Create your views here.
def write_review(request):
    # if this is a POST request we need to process the form data
    if request.method == 'POST':
        # create a form instance and populate it with data from the request:
        form = ReviewForm(request.POST)
        # check whether it's valid:
        if form.is_valid():

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

```

```

the = Play.objects.get(pk=play)

content = request.POST.get('content')
review = Review.objects.create(play=the, user=request.user, content=content)
review.save()
# process the data in form.cleaned_data as required
# ...
# redirect to a new URL:
#return HttpResponseRedirect('/thanks/')

# if a GET (or any other method) we'll create a blank form
else:
    form = ReviewForm()

    return render(request, 'reviews/review.html', {'form': form})
def success(request,pk=None, *args, **kwargs):
    pass

    context = {

    }

    return render(request, "reviews/review_success.html", context)

```

Review models

```

from django.db import models
from plays.models import Play
from users.models import CustomUser
# Create your models here.

```

```

class Review(models.Model):
    play = models.ForeignKey(Play,on_delete=models.CASCADE)
    user = models.ForeignKey(CustomUser,on_delete=models.CASCADE)
    content = models.TextField(max_length=160)
    timestamp = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return str(self.play);

```

Review forms

```

from django import forms
from .models import Review

```

```

class ReviewForm(forms.ModelForm):
    class Meta:

```



```

        model = Review
        fields = ('play','content',)

review.html
{%extends "base.html"%}
{% load crispy_forms_tags %}

{% block content %}

<form method="post" action='{% url "reviews:success" %}' data-endpoint='{% url
"reviews:success" %}' class="form" {% if request.user.is_authenticated %} data-user='abc' {%
endif %}>
    <div class="form-group">
        {% csrf_token %}
        {{ form|crispy }}
    </div>
    {%if request.user.is_authenticated %}

    <button type="submit" value="post">Submit</button>
    {%else%}
        <button type="submit" value="post" disabled>Submit</button>
    {%endif%}
</form>
{%endblock%}

```

Visacheck models

```
from django.db import models
```

Create your models here.

```
class Visa(models.Model):
    card_holder = models.CharField(max_length=120)
    cc_no = models.IntegerField()
    expiry_date=models.IntegerField()
    csv=models.IntegerField()
```

```

    def __str__(self):
        return self.card_holder

```

Visacheck views

```

from django.shortcuts import render
from .models import Visa
from .forms import VisaForm
from django.contrib.auth.decorators import login_required
# Create your views here.
@login_required
def visacheck(request,pk=None, *args, **kwargs):
    #selection= None

```

```

if request.method == 'POST':
    form = VisaForm(request.POST)
    if form.is_valid():
        #selection = form.cleaned_data['play']
        pass # does nothing, just trigger the validation
    else:
        form = VisaForm()

context = {
    'form': form
}

return render(request, "visacheck/visacheck.html", context)

def success(request, pk=None, *args, **kwargs):
    pass

    context = {

    }

    return render(request, "visacheck/success.html", context)

```

Visacheck.html

```

{% load crispy_forms_tags %}
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-
fit=no">
    <title>Greenwich Community Theatre</title>
    {% include 'base/css.html' %}
    {% block base_head %} {% endblock %}
</head>
<body>

    <div class='container'>

        {% block content %}
        {% block title %} <h1 class="display-4">Checkout</h1> {% endblock %}

```

```

    <form method="post" action='{% url "visacheck:success" %}' data-endpoint='{% url
"visacheck:success" %}' class="form" {% if request.user.is_authenticated %}data-
user='abc'{% endif %}>
        {% csrf_token %}
        {{ form|crispy }}

    <button type="submit">Submit</button>
</form>

{% endblock %}
</div>

{% include 'base/js.html' %}
{% block javascript %}

{% endblock %}
</body>
</html>

```

Task allocation table:

Tasks	Group Members involved
System Analysis	Kamilah, Mindaugas, Abdullah, Marzieh, Trevor
System Design	Kamilah
System Development	Kamilah,
Produce Documentation	Kamilah, Mindaugas. Abdullah, Trevor
System Testing	Kamilah, Mindaugas, Trevor, Abdullah

Group members work contribution form:

Group/Team Name: Data Pirates

Team member name	Student ID	Individual overall work contribution (%)	Signature
Student: Kamilah Agbaje	001004321	20	K.A
Student: Abdullah Butt	000993785	20	A.B.
Student: Trevor Kiggundu	001001720	20	T.K.
Student: Mindaugas Lekavicius	001001160	20	M. L.
Student: Marzieh Mohammadi	000991406	20	M.M
Total 100%			

Minutes of team meetings:

Date	Members Present	Elapsed Time (minutes)
23/01/19	All	60
30/01/19	All	60
06/02/19	All	15
23/01/19	All	60
07/02/19	All	60
21/02/19	All	60
28/02/19	All	60
13/03/19	All	30
27/03/19	All	90

References:

- Anon (2017) Factors In Human Computer Interaction (HCI) | Articles, HCI, Literature | Leading UX in Bangladesh, *Userhub*, Userhub, [online] Available at: <https://www.theuserhub.com/literature/factors-in-human-computer-interaction-hci/> (Accessed April 1, 2019).
- Anon (n.d.) Cinemax, *Cinemax*, [online] Available at: <https://www.cinemax.com/> (Accessed April 1, 2019).
- Anon (n.d.) ODEON Cinemas - Book Film Tickets & Check Cinema Listings Now!, *ODEON Cinemas - Book Film Tickets & Check Cinema Listings Now!*, [online] Available at: <https://www.odeon.co.uk/> (Accessed April 1, 2019).
- Anon (n.d.) eCommerce: Coding for entrepreneurs, *eCommerce: Coding for entrepreneurs*, Coding for entrepreneurs, [online] Available at: <https://www.codingforentrepreneurs.com/courses/ecommerce> (Accessed April 1, 2019).
- Anon (n.d.) Django's Structure – A Heretic's Eye View - Python Django Tutorials, *The Django Book*, [online] Available at: <https://djangobook.com/mdj2-django-structure/> (Accessed April 1, 2019).
- Otto, M. and Thornton, J. (n.d.) Bootstrap, · *The most popular HTML, CSS, and JS library in the world.*, [online] Available at: <https://getbootstrap.com/> (Accessed April 1, 2019).

**Greenwich Community Theatre: Systems Development Project
(Coursework 2: Part B)**

Trevor Kiggundu: 001001720
April 2019

A report submitted in fulfilment of the requirements for the module, Systems Development Project, Computing and Information Systems Department, University of Greenwich.

Personal Reflection:

This course overall was a great addition to the systems development criteria we were taught during first year. As an aspiring software engineer, I found this module to be greatly helpful in my understanding of the subject. I was able to create a theatre system prototype on paper, and then with the help of my group members, implement it and make a user interactive system that could be fully tested. We were able to successfully do this as well as produce a well-documented report. Things that went well were the fact that we were able to complete the program, it was not an easy task to complete. Certain things that went badly were the fact that we were undecided on which program/IDE to use. We started off with an ASP.NET application using C# in Visual Studio, but that faltered after a while. I then personally tried to implement it in Java using NetBeans but that failed as well. A return to Visual Studios using winforms was on the table for a while, but that was a bust as well. We finally returned to one of Kamilah's original ideas in using Python (more specifically Django), and we were able to complete the application that way. I personally had numerous problems with my ASP.NET application, especially since it was so promising at the beginning. However, after failing to implement the visual seating chart and VISACheck, it was decided to abandon it completely. I felt like I did a lot of work, especially regarding the lab report as I modelled it after the one we did for the Computer Algorithms module during 1st term. I was happy with my role of coordinating the report and brainstorming user implementation and HCI factors, a big part of the report. My group could have worked a little bit better than we did, but in all fairness, it was hard to as we all had other deadlines to attend to as well. It was reaching a point where we had to ask for an extension in another module to accommodate for complicated allocation scenarios, but we are able to pull through in the end. The planning could have been better executed during the latter stages on the term, and leadership and management were split between three people in the group, it was always going to be hard to integrate everyone's thoughts in that regard. Some of our group members had never programmed in C# and Python especially, so we had to accommodate for their contribution as well. In the end however, we were all able to contribute to the final product and I am more than satisfied with the results of the task.

Group members work contribution form:

Group/Team Name: Data Pirates

Team member name	Student ID	Individual overall work contribution (%)	Signature
Student: Kamilah Agbaje	001004321	20	K.A
Student: Abdullah Butt	000993785	20	A.B.

Student: Trevor Kiggundu	001001720	20	T.K.
Student: Mindaugas Lekavicius	001001160	20	M. L.
Student: Marzieh Mohammadi	000991406	20	M.M
Total 100%			