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

# **Sprint Review Form**

| Group: | 18 |
| --- | --- |
| Sprint: | 4 |
| Members: | Benedict Ramage-Mangles, Ross Williams, Arjun Binning, Michael Duncan, Matthew Hill |

**SPRINT PLANNING:**

Sprint 4 Requirements:

* Creating the back-end Databases required for the system-to-be
  + Create Model Design, including all of their relations and dependencies
  + Implement the models in the project
  + Integrate all models into build, and test
* Implement Authentication for all users on the system
  + Create Template and View files for Login page
  + System accepts different user types, depending on access privileges
  + Send user to correct destination, depending on user type
  + Each login should be a Session, which ends after 20 minutes of inactive time.
  + Integrate and Test once implemented

**TASKS TO BE DONE:**

**Matthew Hill - Tasks**

I was in tasked with creating the models for film, screen, showing and club, as well as setting up the login page to accept different users

1. Model Film

The Film model holds the data for all the information regarding the Film, the data items are:

* **Title (String)**: A variable to hold the title of the film
* **Age\_rating (String):** A variable to hold the age rating of the film
* **Duration (String):** A variable to hold the length and duration of the film
* **Trailer\_Description (String):** A variable to hold/store the description of the film trailer

1. Model Screen

The screen model holds the data for all the information regarding the screen, the data items are:

* **Capacity(IntField):** A variable to store the capacity for each screen.
* **Apply\_Covid\_restrictions(Boolean):** True means covid restrictions have been applied, False means no covid restrictions are in place

1. Model Showing

The showing model holds the data for all the information regarding the showing, the data items are:

* **Screen (Foreign Key)**: A reference to another model (Screen) using a Foreign Key. This will show the user what screen the film will be in.
* **Film (Foreign Key):** A reference to another model (Film) using a Foreign Key. This will allow the retrieval of the film the user has picked
* **Time (DateTimeField):** The time of the showing

1. Model Club

The club Model holds the data for all the information regarding the club and the account details, the data items are:

* **Name (String)**: A variable to hold/store the name of the account
* **Card\_Number (IntField):** A variable to hold/store the card details, the max\_length=16, this is the standard length for card numbers
* **Card\_expiry\_date (DateField):** A variable to determine/store the card expiry date details
* **Discount\_rate (IntField):** A IntField(intrager) to determine if/ what discount rate is relevant

Each model has been created and added to the Model.py file

1. Login Page to accept Different Users

This final task was to create the logic for the login page to be able to accept different users and to differentiate between each one, the logic distinguishes what user has logged in and redirects them to the correct page/pages giving them the relevant permissions associated with the users log in.

Example:

Account login - student log in confirmed via account\_type = st

Redirects to the film page, only allowing the student to see and pick their desired film to then continue on the process.

This has been done by checking the account type of the user i.e:

if (acc.account\_type == 'cm'): # Cinema Manager

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

if (acc.account\_type == 'am'): # Accounts Manager

return render(request, 'uweflix/view\_accounts.html')

if (acc.account\_type == 'cr'): # Cinema Manager

return render(request, 'uweflix/viewings.html')

if (acc.account\_type == 'st'): # Student

return render(request, 'uweflix/viewings.html')

Once the user has been checked and verified, is then redirected to there relevant .html file. For all this to happen modifications were made to the view.py file.

**Michael Duncan - Tasks**

1. Sprint 4 - Allocate Tasks

I will have the role of deciding which tasks require completion, and by whom. We will have a group meeting to discuss strengths and who would like to put their name forward to do certain tasks. After which, we can decide the best way to allocate the workload across the group.

1. Create Superuser

Superuser is built-in functionality in Django that allows those with the username and password to access back-end functionality and create new items, such as users, model objects, etc. I will be implementing this using django’s commands on the command line, which is:

*python manage.py createsuperuser --username=<username> --email=<email>*

I will use the following username and password:

Username: admin

Password: password

1. Create Model - Ticket

I will be responsible for creating the Ticket model, which will hold all information about each individual ticket sold by the UWEFlix system. After some consideration we have decided that this model must include:

* **Transaction (Foreign Key)**: A reference to another model using a Foreign Key
* **Showing (Foreign Key):** A reference to another model using a Foreign Key. This will allow the retrieval of which screen, film and time for the ticket.
* **Ticket\_type (String):** A variable to determine which type of ticket the ticket is. This can be one of three options as per the UWEFLix spec; “Adult”, “Student”, or “Child”.

1. Create Model - Account

I will also be creating the Account model, which will hold account data for each user on the system. Data items are:

* **Email (EmailField):** The email address that the user registers with, and is used to sign in. Django’s built in EmailField provides verification that the input is in the form of a valid email address.
* **Password (String):** The users password that they use with their email to sign in. While this is only stored as a string in the database, it will appear in a securefield at the front-end, meaning that the password will not be visible on screen. Passwords must be a minimum of 8 characters, and a maximum of 50.
* **Account\_type (String):** This will be how users are authenticated, and therefore given different privileges depending on which account they hold. There will be 4 classifications of account\_type:
  + “cm”: Cinema Manager
  + “am”: Account Manager
  + “cr”: Club Representative
  + “st”: Student

1. Create View/Template - View Accounts page (Login redirect for Account Manager)

This task is to create a page that the Accounts Manager can be redirected to, to provide early verification that the login system is working correctly. However, while I am creating this page, I will also be adding some basic functionality from the specification: *“the Account Manager requests to view end of month account statements and the new system supplies a list of all possible end of month account statements”.*

I will be adding a search function where the Accounts Manager can search through all Club Representatives via club\_rep\_id (primary key of ClubRep model), which will then bring up a table of all of the transactions that the Club Rep has made in the last *calendar* month (not last 30/31 days).

1. Create View/Template - Payment Page

Taken from the User Stories from Club Representatives, Students (and Customers), they must all be able to purchase (or book) tickets, using the new online system. Therefore, this task is to create the final page that users will see before confirming their booking.

This takes the Showing data from the previous page, which is required as a parameter before the page can be shown. This then allows the system to present an order summary, of which the user can then select how many of each ticket type they would like. There will be verification in place to ensure that the user selects *at least* one ticket type, and also only tickets they are eligible for (e.g. regular Customers cannot buy Student tickets, etc.).

Once this has all been done, the final cost will be calculated, and the user will be given the option to pay with their account credit, pay on the day at the cinema (therefore just booking the ticket), or in the event of the Club Representative, add to their monthly accounts bill.

**Ross Williams - Tasks**

1. Create Model - Transaction

The Transaction model will hold data related to a transaction. These include:

* **Customer (Foreign Key):** Reference to the Customer model (customer who is responsible for the transaction)
* **Date (DateField):** The date of the transaction
* **Cost (Float):** The cost of the transaction
* **Is\_settled (Boolean):** True means the transaction has been paid. False means the transaction has not been paid. This keeps track of transactions that have not been payed enabling users to pay at a later date.

1. Create Template - Add Film

This template allows the Cinema Manager to add films. There are four input fields: title, age rating, duration and trailer description. An ‘add film’ button can be clicked to submit the data. The data is sent as an HTTP post transaction.

1. Create Template/View - Viewings

This template shows all current films that can be seen. Here the film title can be seen along with a button ‘book’. When this button is clicked the user will be redirected to the showings page for that specific film.

In views.py the viewings function uses the film model to grab all the data so it can be displayed on the viewings.html page.

1. Create Template/View - Showings

This template shows all current showings for a particular film. The user can see more in-depth information about the movie including a brief description. Below the film information for different showings can be seen, the screen number, date and time can be seen for each showing. The user can select a showing and will be directed to the payment page when clicked. This template and view needs more work and will be looked at further in an upcoming sprint.

Some modifications were made to views.py and urls.py for all tasks where necessary.

**Arjun Binning - Tasks**

1. Create Template - Login

The template is ‘view’ that users will see when they attempt to log in. This consists of two input fields taking in the username and the password, and a submit button in order to submit the input data. This was done using HTML forms. When submitted, the method of the form must be ‘POST’ in order to work alongside Django, and inside the form, ‘**{% csrf\_token %}**’had to be added to avoid any csrf errors. From there, any input can be processed in a login method of the views.py file.

Alongside this, a url to the page had to be created in the urls.py file. This was done by adding **‘path("login/", views.login, name="login"**)’ to the urlpatterns part of the file, referencing the login method created in views.py .

1. Create Sessions - Expire after 20 minutes.

As per the UWEFlix Requirements, each Login had to work with sessions, and these sessions expire after 20 minutes of inactivity. This was achieved by firstly adding 3 lines into settings.py file.

The first line added was: ‘**SESSION\_EXPIRE\_AT\_BROWSER\_CLOSE = True**’ which allows for a session to expire whenever the browser is closed.

The second line added was: ‘**SESSION\_COOKIE\_AGE = 20\*60**’ which allows for a session to expire after 20 \* 60 seconds (20 minutes). However this line makes a session expire after 20 minutes regardless of if there is activity or not.

Therefore, the third line added was ‘**SESSION\_SAVE\_EVERY\_REQUEST = True**’ which allowed a session to be saved every interaction, and therefore only expire after 20 minutes of inactivity.

From there, session variables were able to be created. This is done as: ‘**request.session['username'] = Username**’ which for example stores the username of a user within a session and can be accessed anywhere whilst the session is still active.

1. Create Topup Page

Firstly the template was created - being the ‘view’ that users see when they access the page. This was done using HTML, and consisted of a paragraph element and multiple buttons the user can select from. The buttons allow the user to select how much they want to top up, and Javascript code is used so when the button is pressed, the paragraph element changes the values to whatever button was clicked. Alongside the premade buttons, a ‘custom text’ field was implemented allowing the user to input however much credit they wanted to. From there, Javascript validation was used to ensure the input was 2 decimal places(if not, it would ensure it was by adding 0’s if lower, or by rounding if it was higher), and it checked if the value entered was a number.

Alongside this, a custom form was created. Again this form required the method of it to be ‘POST’ and implemented ‘**{% csrf\_token %}**’ to avoid any csrf errors. The attribute of the form was ‘hidden’ so it was unable to be seen, and everytime a credit button was pressed, the Javascript code would implement that value into the form. This was done so when the information was submitted, the credit value was able to be processed by Django.

From there, the ‘Topup’ method was created in views.py. Firstly this checks if a login session is valid by the following if statement: ‘**if 'accountType' in request.session:**’. From here, if there is a valid session, the account type was tested to see if it was ‘cr’ (Club Rep) as they were the only users who could add credit to themselves at the time. From there, the form from the template was read, and the Club Rep account was found using ‘**ClubRep.objects.get(username = request.session['username'])’**, and the requested credit was added.

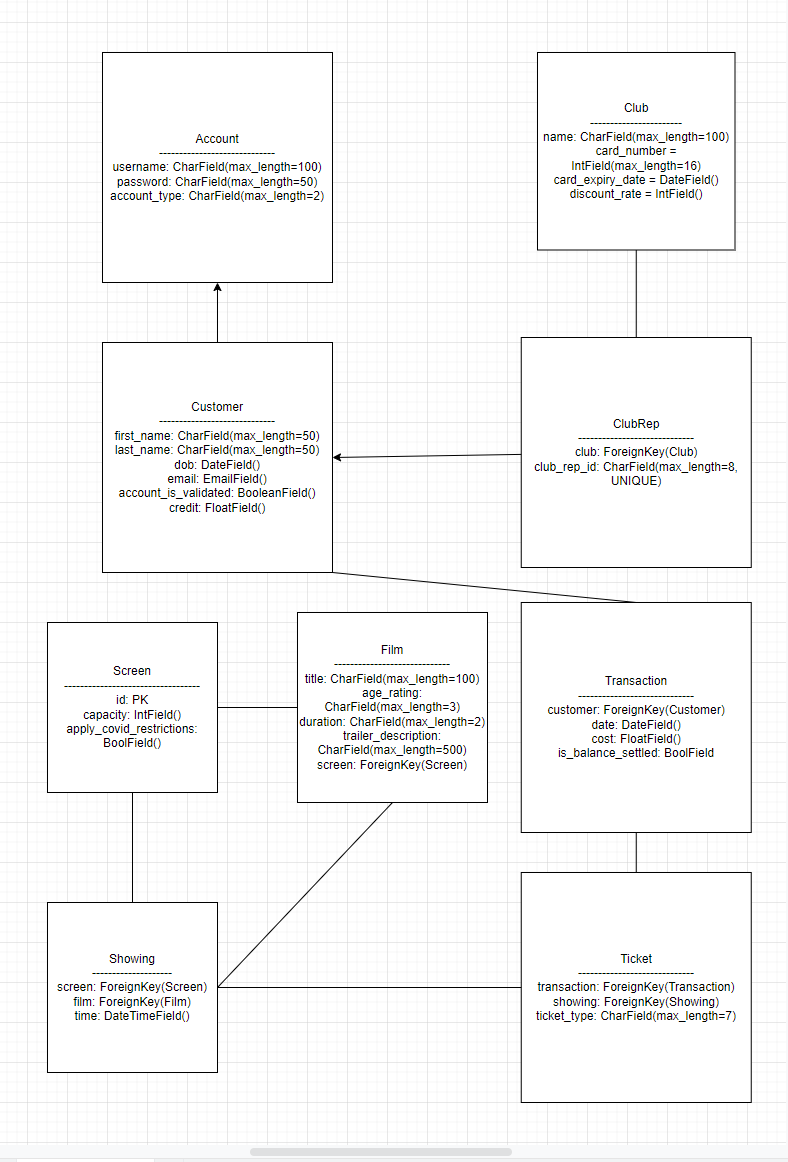
**Benedict RM - Tasks**

* Plan the testing for our user story. I have a test plan that involves:
  + Admin login
  + Session timeout
  + Account manager login
  + Club rep login
  + Book ticket
  + Bad password

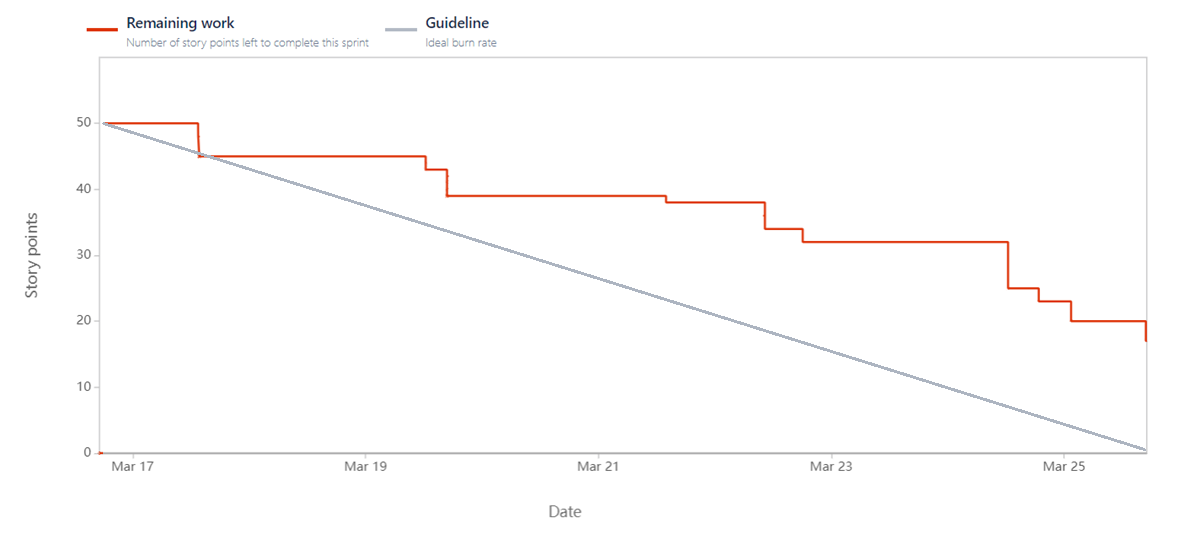
On top of the user story, fields need to be tested in various ways: I’ll look at what the expected bounds are and test things like what happens when things are outside them. Also worth considering is to make sure that the system can’t be crashed in any way, for instance by leaving a field blank. This may be a problem due to our system for authentication. The test plan document can be found here

https://docs.google.com/spreadsheets/d/1OfF0PlnA7NxUuSMhXcDRh7qDHArg3XSYatILbvpZlVk/edit?usp=sharing

Database Design:



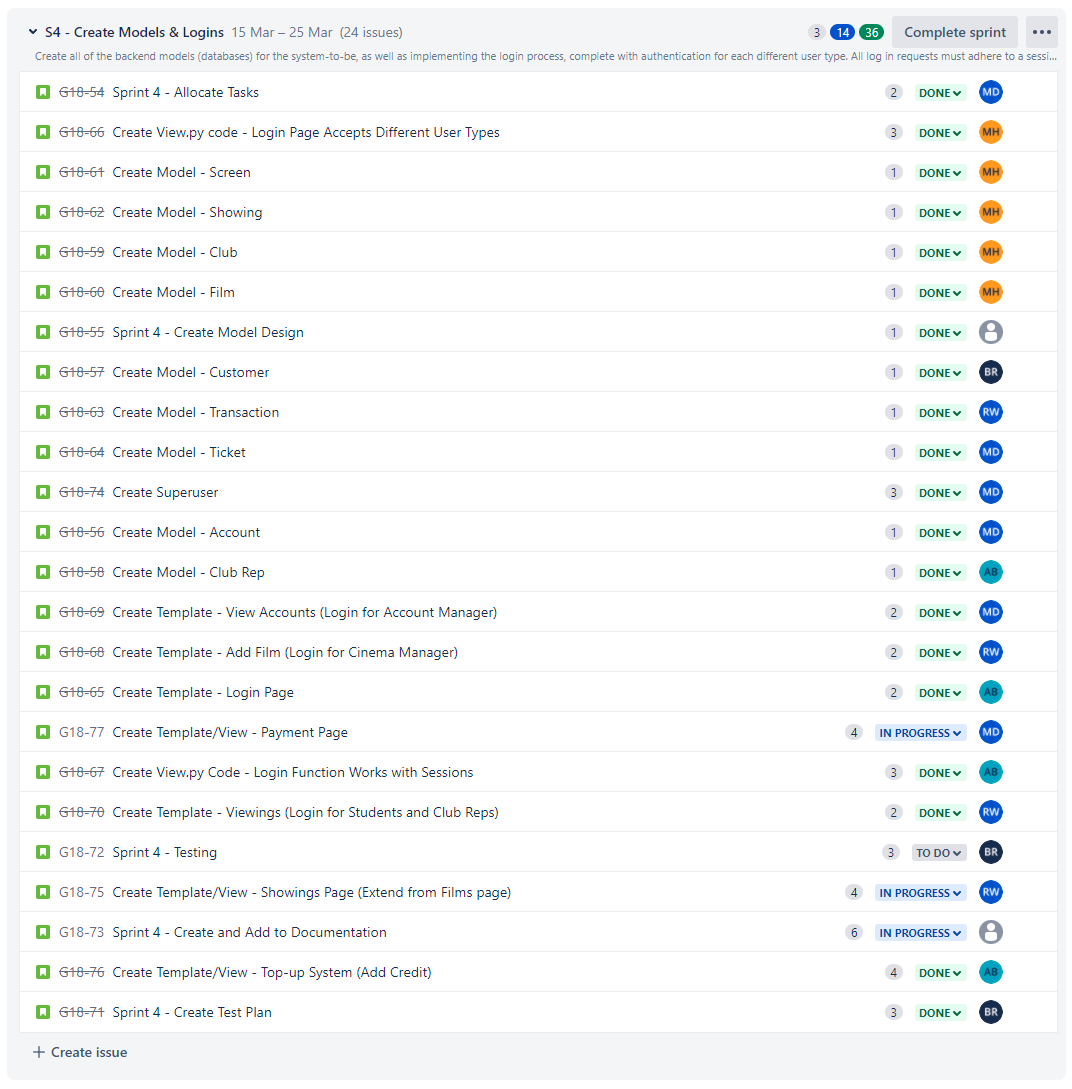
<<Burn-down Chart>>



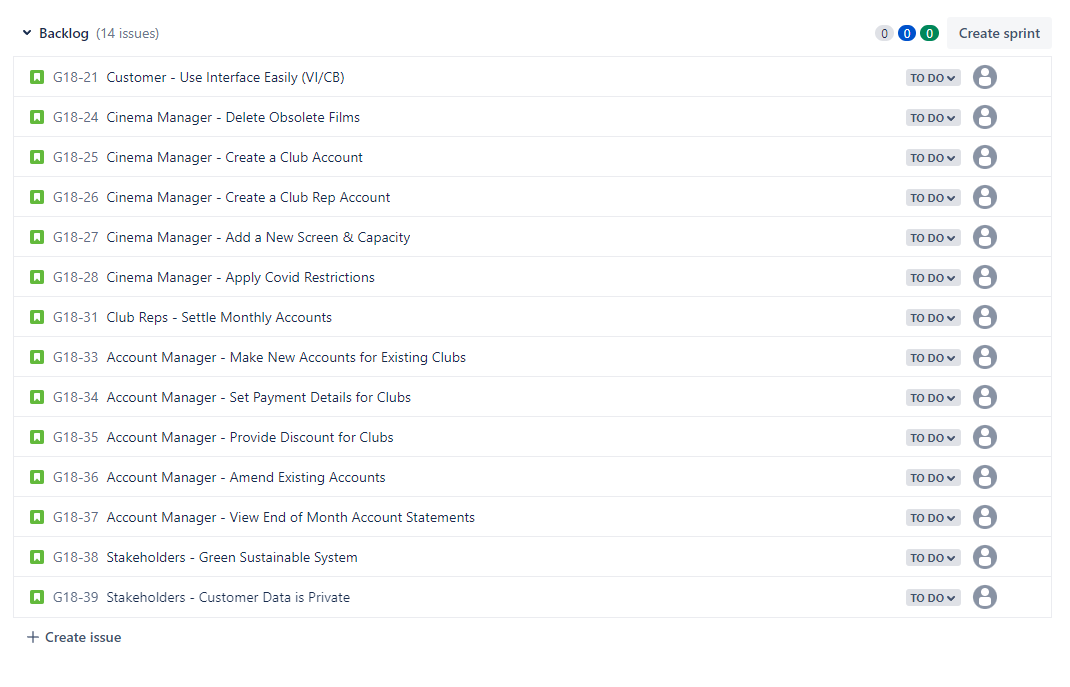
Above is the burndown chart for this sprint, and how many of the story point estimates were completed so far. The chart shows that the majority of the tasks were completed this sprint, however a few will be carried over into the next sprint and be completed before the next set of tasks begin.

<<Backlog list>>

Sprint Backlog:



Product Backlog:



<<Communication Issues>>

No Communication issues to report.

<Reflections>>≠

| Task | Person | What was expected? | What was actually delivered? | Additional Comments |
| --- | --- | --- | --- | --- |
| Create Model Screen | Matthew Hill | A model to hold all the functionality needed to hold all the data for a screen,  I.e. capacity(), covid\_restrictions() | Fully implement the required models with the functionality needed to store all of the required data. | As the program/website evolves there is scope to add additional data fields if needed. |
| Create Model Film | Matthew Hill | A model to hold all the functionality needed to hold all the data for a Film,  I.e. title(), age(), duration(), trailer(), screen() | Fully implement the required models with the functionality needed to store all of the required data. | As the program/website evolves with feed back there is scope to add additional data fields if needed. |
| Create Model Showing | Matthew Hill | A model to hold all the functionality needed to hold all the data for a Showing,  I.e. screen(), film(), time() | Fully implement the required models with the functionality needed to store all of the required data. | As the program/website evolves there is scope to add additional data fields if needed. |
| Create Model Club | Matthew Hill | A model to hold all the functionality needed to hold all the data for a Club,  I.e. name(), card\_number(), card\_expiry(), discount\_rate() | Fully implement the required models with the functionality needed to store all of the required data. | As the program/website evolves there is scope to add additional data fields if needed. |
| Create Login Page to accept Different User | Matthew Hill | When a user logs in the login page redirects them to the relevant page with the relevant permissions, I.e. the student logs in they only see the Films, showings ext | Managed to implement the required function to allow the user to be redirected to the relevelt pages. | This one was a little tricky, as I was unable to use the built in django authentication() function as we had created our own. |
| Create Login Template | Arjun Binning | The view of the login page, with a username and password field, alongside a submit button. | The view was created with a username and password field, with the username field showing text, and the password field showing a password. The submit button was also created and linked to the Django methods. | Design adjustment may be needed as more front-end ideas and feedback are given from the rest of the group. |
| Create Sessions which expire after 20 minutes | Arjun Binning | Logins are set to work within sessions, and these sessions are to expire after 20 minutes of inactivity. | The login works with sessions, and session variables are used to confirm the login for different pages. After 20 minutes of inactivity, the session expires and the user is logged out. | Existing views may need to include session variables as certain views should be restricted to different user types. |
| Create Topup Page | Arjun Binning | Top up view is to be created, alongside taking the data and adding it to the logged in user. | The template was created which had buttons to change the topup amount. The page was restricted to club rep (as only they could add credit to themselves so far) and once they submitted they could add credit to themselves successfully. | Design adjustments may be needed depending on feedback from the group. Page restrictions and accessibility may also need to be adjusted later on in the code. |
| Create test plan & Test system | Benedict RM | Plan and test the user stories we want to implement and also all the fields we need | The test plan is prepared for the user stories, but individual fields will need to be tested as well. | Need to look into the backend behind each input field and start producing legal and illegal inputs for each one, as well as edge cases |
| Create Model - Ticket | Michael Duncan | A model to hold all ticket information, so once they have been booked they are stored for later reference. | As Expected. | No additional comments to report. |
| Create Model - Account | Michael Duncan | A model to hold user information, such as username, password and account type. | As Expected. | It has come to my attention that there is an in-built Django model named User, which has pre-built functionality such as authenticate(), etc. As we were already in the process of making this model, we decided to continue with what we had rather than using what Django offered. |
| Create View/Template - View Accounts | Michael Duncan | A page where the Accounts Manager can view monthly accounts from Club Representatives. | As Expected. | Not all of the functionality is there at present, however the Accounts Manager can look up monthly reports based on club rep id. |
| Create View/Template - Payment Page | Michael Duncan | The final page in the booking process, where the user decides number of tickets and payment method | As Expected. | Only limited payment methods are working at present; Club Rep paying later, and Club Rep and Student paying with credit. Paying in advance not finished. |
| Create Superuser | Michael Duncan | Superuser functionality to access and modify back end data and functionality. | As Expected | Nothing to report. |

| Task | Person | What was expected? | What was actually delivered? | Additional Comments |
| --- | --- | --- | --- | --- |
| Create Model Screen | Matthew Hill | A model to hold all the functionality needed to hold all the data for a screen,  I.e. capacity(), covid\_restrictions() | Fully implement the required models with the functionality needed to store all of the required data. | As the program/website evolves there is scope to add additional data fields if needed. |
| Create Model Film | Matthew Hill | A model to hold all the functionality needed to hold all the data for a Film,  I.e. title(), age(), duration(), trailer(), screen() | Fully implement the required models with the functionality needed to store all of the required data. | As the program/website evolves with feed back there is scope to add additional data fields if needed. |
| Create Model Showing | Matthew Hill | A model to hold all the functionality needed to hold all the data for a Showing,  I.e. screen(), film(), time() | Fully implement the required models with the functionality needed to store all of the required data. | As the program/website evolves there is scope to add additional data fields if needed. |
| Create Model Club | Matthew Hill | A model to hold all the functionality needed to hold all the data for a Club,  I.e. name(), card\_number(), card\_expiry(), discount\_rate() | Fully implement the required models with the functionality needed to store all of the required data. | As the program/website evolves there is scope to add additional data fields if needed. |
| Create Login Page to accept Different User | Matthew Hill | When a user logs in the login page redirects them to the relevant page with the relevant permissions, I.e. the student logs in they only see the Films, showings ext | Managed to implement the required function to allow the user to be redirected to the relevelt pages. | This one was a little tricky, as I was unable to use the built in django authentication() function as we had created our own. |
| Create Login Template | Arjun Binning | The view of the login page, with a username and password field, alongside a submit button. | The view was created with a username and password field, with the username field showing text, and the password field showing a password. The submit button was also created and linked to the Django methods. | Design adjustment may be needed as more front-end ideas and feedback are given from the rest of the group. |
| Create Sessions which expire after 20 minutes | Arjun Binning | Logins are set to work within sessions, and these sessions are to expire after 20 minutes of inactivity. | The login works with sessions, and session variables are used to confirm the login for different pages. After 20 minutes of inactivity, the session expires and the user is logged out. | Existing views may need to include session variables as certain views should be restricted to different user types. |
| Create Topup Page | Arjun Binning | Top up view is to be created, alongside taking the data and adding it to the logged in user. | The template was created which had buttons to change the topup amount. The page was restricted to club rep (as only they could add credit to themselves so far) and once they submitted they could add credit to themselves successfully. | Design adjustments may be needed depending on feedback from the group. Page restrictions and accessibility may also need to be adjusted later on in the code. |
| Create test plan & Test system | Benedict RM | Plan and test the user stories we want to implement and also all the fields we need | The test plan is prepared for the user stories, but individual fields will need to be tested as well. Some limited testing has been done for certain user stories. | Need to look into the backend behind each input field and start producing legal and illegal inputs for each one, as well as edge cases |
| Create Model - Transaction | Ross  Williams | A transaction model will hold data related to a transaction | As Expected. | No additional comments to report. |
| Create Template - Add Film | Ross Williams | A template that allows the Cinema Manager to add films. There will be four input fields: title, age rating, duration and trailer description. An ‘add film’ button can be clicked to submit the data. | Template that can be built upon. | No additional comments to report. |
| Create Template/View - Viewings | Ross Williams | A template that shows all current films that can be seen. Here the film title can be seen along with a button ‘book’. When this button is clicked the user will be redirected to the showings page for that specific film. | As Expected. When a new film is added it will be shown on the viewings page. | Some additional functionality can be added (no duplicate films etc.). |
| Create Template/View - Showing | Ross Williams | This template shows all current showings for a particular film. The user can see more in-depth information about the movie including a brief description. Below the film information for different showings can be seen, the screen number, date and time can be seen for each showing. The user can select a showing and will be directed to the payment page when clicked. | Difficulty getting it to work for a specific film. Currently shows all showings for all films. | This needs additional work. When a user clicks ‘book’ on the viewings page it will need to show the showings for that film and that film only. |

<<Relevant Links>>

github.com/bean64/Group-18-ESD-2022

https://discord.gg/zwjTNUzEhM

[Test Plan Sprint 4](https://docs.google.com/spreadsheets/d/1OfF0PlnA7NxUuSMhXcDRh7qDHArg3XSYatILbvpZlVk/edit?usp=sharing)