CODE DESCRIPTION

We decided to use python which led us to using Flask as our framework for the web application. For the front-end we used HTML, CSS, and JavaScript.

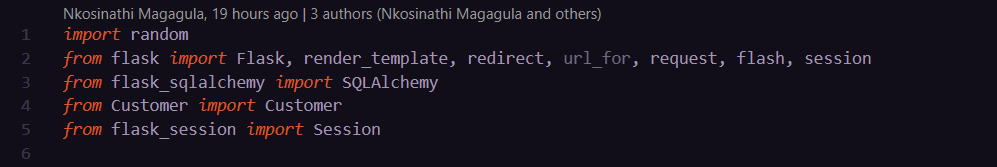


Figure 1

Figure 1 above shows all the necessary libraries we used or imported. We used random module for generating the unique number for the ticket id of the user or customer that is purchasing a ticket. We user the flask\_sqlalchemy library for creating and querying the database. The Customer class was for the customers that purchase a ticket. We used the flask\_session library which allows us to move variables between different web pages as shown in Figure 2 below and we used the secret key to protect the session cookies.

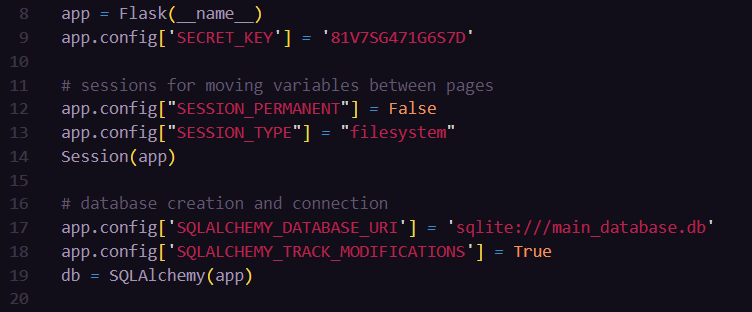


Figure 2

In Figure 2 above we created our flask app, then created our sqlite database named main\_database.db using sqlalchemy and connected it to our app.

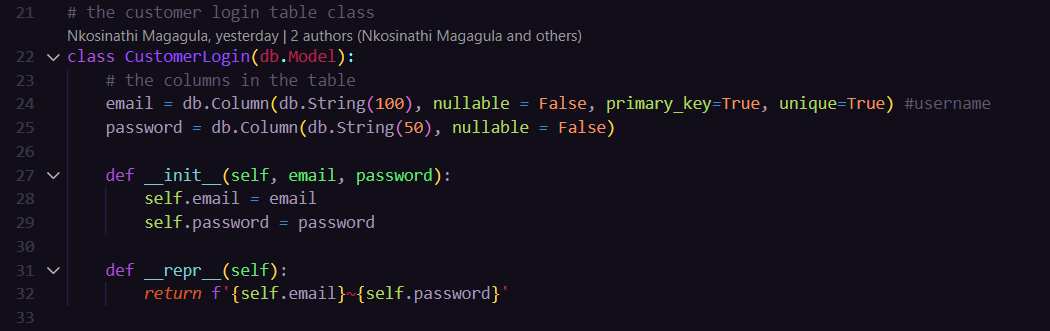


Figure 3

Figure 3 has the CustomerLogin class, which is used to create the customer\_login table in our main database. It has two columns, email which is the primary key and password. This is used to log in a user. The user can only login when they exist in our database. If they don’t exist, they must sign up. The class object is represented by a string with the email and corresponding password.

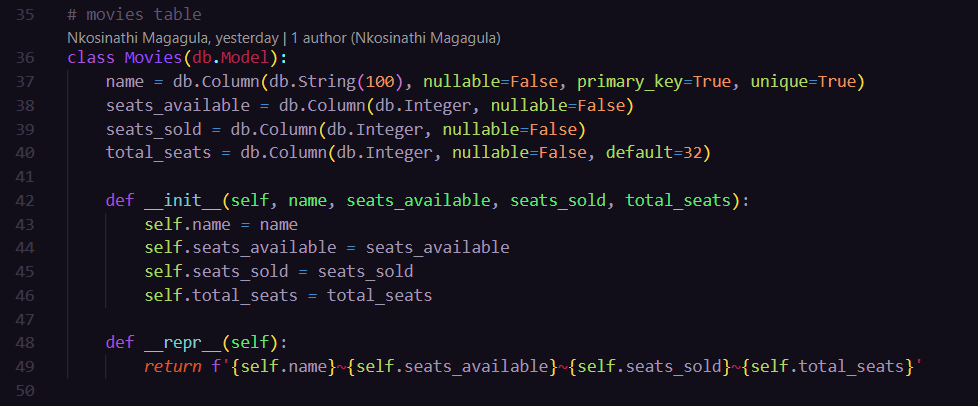


Figure 4

In Figure 4 there is a Movie class which creates the movie table in our main database. The table has 4 columns with ‘name’ which is a string and is used as the primary key, ‘seats\_available’ which is an integer, ‘seats\_sold’ which is an integer, ‘total\_seats’ which is also an integer. The class object is represented by a string with the name of the movie, seats available, seats sold, and total seats.

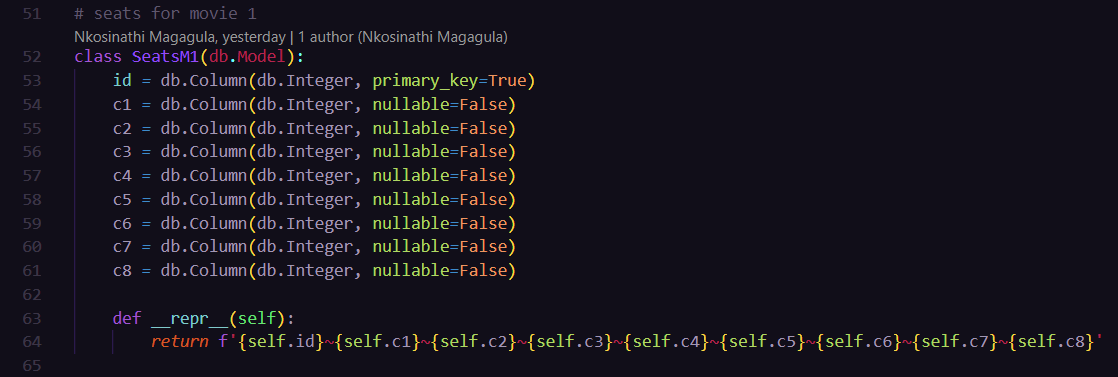


Figure 5

Figure 5 has a class which represents the seats for movie 1. We have three movies and all the class implementations for movie seats are the same. They have 8 columns and 4 rows as shown in Figure 6 below. 0 represent that the seat is available and 1 represent that the seat is sold.

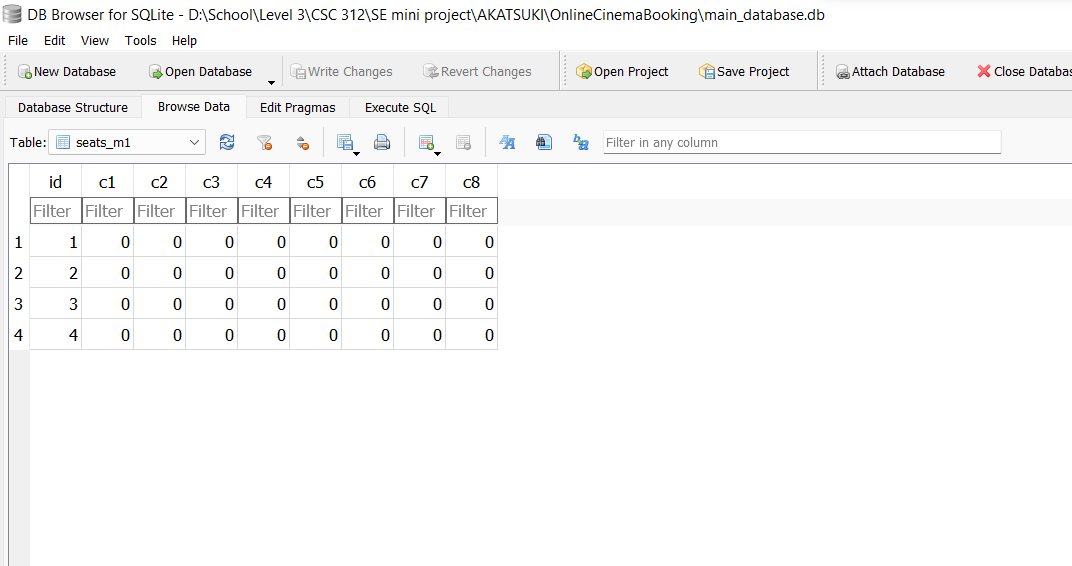


Figure 6

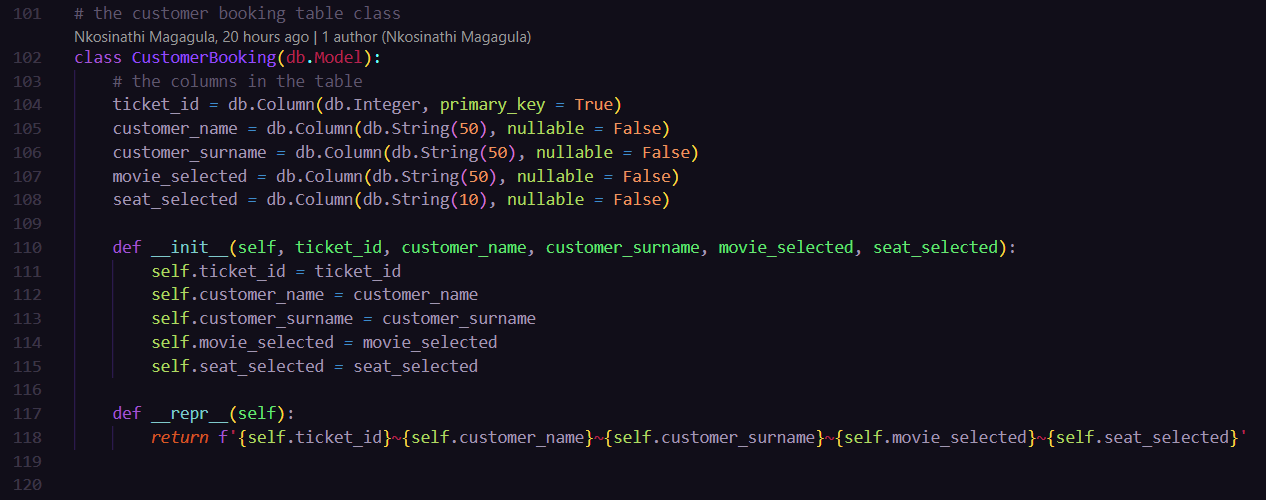


Figure 7

Figure 7 has the CustomerBooking class which creates the customer\_booking table in our main database. It has the ticket id as the primary key, the customer name, customer surname, movie selected, and the seat selected.



Figure 8

Figure 8 has the login logic. The username and password are taken from the form in the login page. We the query the customer\_login table using the username since its our primary key (email). If it doesn’t exist in the database, it will flash a message in the page to tell the user to sign up. If it exists, it will check if the password entered matches the one in the database that is associated with the username/email. If they match, the user will be redirected to the home page, otherwise it will flash a message in the page to tell the user that the password is wrong and ask them to try again.



Figure 9

Figure 9 has the sign-up logic. The user will be directed to this page when they click on the sign-up button on the login page. The signup function will therefore take the username, password, and password confirmation from the form and get them assigned to variables. It will then check if the password that is entered is the same as the confirmation password that is entered. If they are not the same, it will flash a message on the page telling the user that the passwords are not matching. If they match it will add the user to the customer\_login table, let the user know that they have successfully signed up, and redirect to the login page. If a user enters a username that already exist, there will be an exception. We flash a message to let them know if they enter a username that already exists.

Text

Description automatically generated

Figure 10

The function in Figure 10 is for the home page. It queries the Movies table in our database to get the number of seats sold or available for each movie. This is then rendered to the html home page. When the table in the database is updated, the html page is also updated.

Text

Description automatically generated

Figure 11

In the home page the user will select a movie and be directed to the page of selecting a seat. This will work the same way for all the 3 movies. They will just be using 3 different tables that represent the seats for each movie. When the user selects a seat, this function in Figure 11 will record which seat is being selected. The seats are represented in a letter and a number, this function formats the selected seat number to a list of two numbers representing the row and column of the seats. This makes it easier for us to change the value later when indicating that the seat is taken. Session cookies are used hold the seat number for the selected seat. The user cannot select a seat that has already been selected. To prevent that we query the seats table and check if the seat that is being selected is represented by a 1 in the seats table. If there is a 1, we let the user known and prompt them to select another seat. If it is not a 1 it means there is a 0, which means the user should be directed to the checkout page.



Text

Description automatically generated

Figure 12

The above figure has the checkout function. All the movies have the same checkout functions, the only difference is the movie seats table that is used. We get the movie selected, then get the seat selected from the session cookies. The checkout function generates a random number to represent the ticket id. The details that are entered in the checkout form are assigned to variables so that they can be added to the customer\_booking table in our database. The customer class is used to create a customer instance, which is later used to create a CustomerBooking instance that is going to be added in the database. The seats table is then indexed using the list representing the seat selected. The seats table is the updated at the appropriate column and row by putting 1, showing that the seat is sold. This function also queries the movies table to update the number of seats sold or available for the movie that was selected. After that the CustomerBooking instance with all the required details is added to our table in the database and commit. All this happens when the user clicks the checkout button. After that the user is directed to the thank you page which displays the ticked id of the user which can be used to cancel the ticket later. The ticket id is rendered with the thankyou html page.

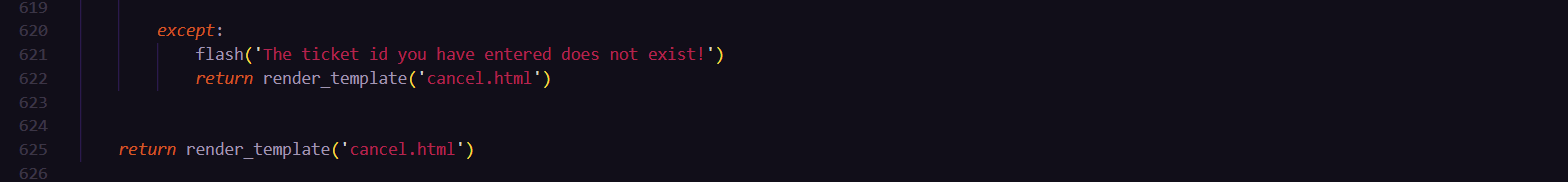


Text

Description automatically generated

Text

Description automatically generated



The above code is the cancel function for cancelling a ticket booked. The user will be directed to a page where they will need to enter the ticket id for the ticket that they want to cancel. That ticket id is read from that page and stored to a variable. We query the database with the ticket id, if it is not available, we flash a message saying the ticket id does not exist. If it exists, we get which movie does it belong to, and the seat number associated with it. If and elif statements are used to get which seats table we will be updating. We use these details to update the seats table. The seats table is updated by putting a 0 on the row and column that was indicated by the seat number, meaning that the seat is now available. The movies table is also updated. The number of seats available or sold are updated (+1 on available seats, -1 on seats sold). After that the entry is deleted and we commit. We the direct to another page showing the user that the ticket is successfully cancelled.

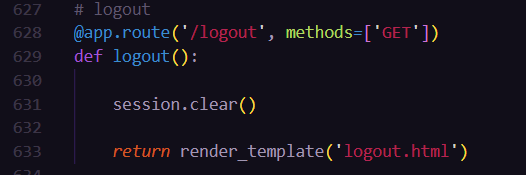


Figure 13

The above figure has a logout function which clears all the session cookies and logs the user out. This happens after the user clicks the logout button in the home page. They are taken to another page that shows them that they have successfully logged out.

Graphical user interface, text

Description automatically generated

Figure 14

The code in Figure 14 runs the application.

{% if movie1[1] == '0' %}

{% else %}

{% endif %}

We used the above if else statements in the html page to make the user to be able to select a movie only if there are still seats available. If there are no seats available for a certain movie, that movie cannot be selected.

TEST CASES

**Signup & login**

[admin@gmail.com](mailto:admin@gmail.com) 123456

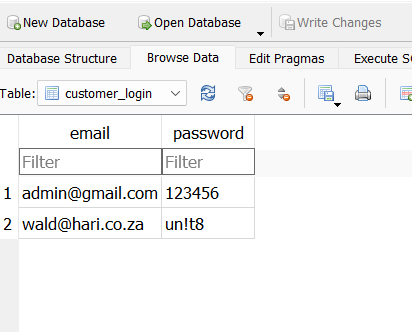
[wald@hari.co.za](mailto:wald@hari.co.za) un!t8

**The database was empty at the beginning.**

Graphical user interface, text, application

Description automatically generated

**After signing up with both test cases.**



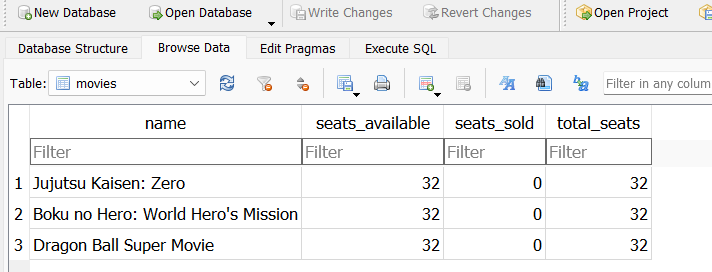
**Booking**

wald hari

james wells

oliver simmons

**before buying tickets.**

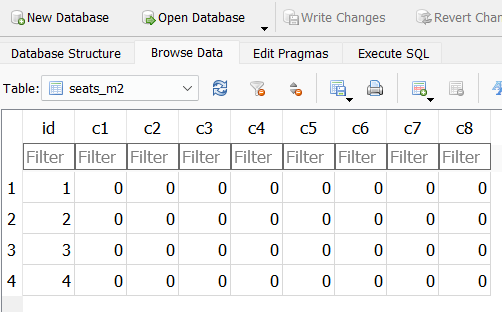


Graphical user interface, text, application

Description automatically generated

Text

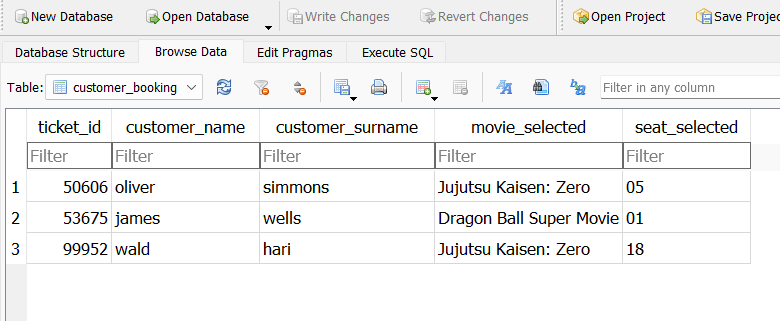
Description automatically generated with medium confidence

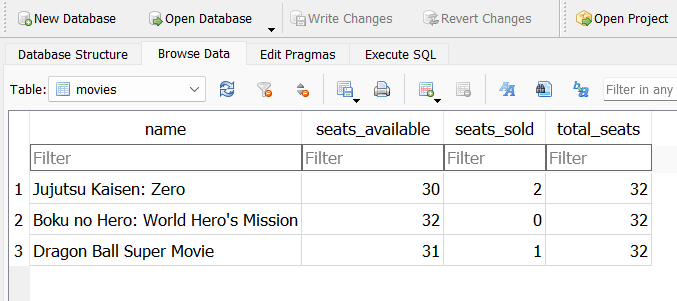


Text, table

Description automatically generated with medium confidence

**After buying 3 tickets**

****

****

**Table

Description automatically generated**

**Table

Description automatically generated**

**Text

Description automatically generated**

**After deleting 2 tickets with the ticket id 53675 and 99952**

**Graphical user interface, text, application, email

Description automatically generated**

**Graphical user interface, text, application, email

Description automatically generated**

**Text

Description automatically generated with low confidence**

**Table

Description automatically generated**

**Table

Description automatically generated**