CISC 327 – Group 11
Nicole Osayande
Melissa Zhu
Teaghan Laitar
Aubrey McLeod

Assignment 2 Design Document and Test Plan

Design Document:

The overall approach to the project is to have each web page manage the user's interaction and call to backend functions when necessary. These pages are connected to the frontend through Flask blueprints. The back end will have access to the data base which stores all the user and ticket information. Below are all the components of our program and what their individual purpose is.

Class/Method Name	**Description of intention**
frontend	Blueprint: Imports and links all routes
models	Contains the database model specifications for User and Ticket and creates the gives access to the database
User	Defines the SQL table to include id, email, password, name, and balance
Ticket	Defines the SQL table to include id, ticket_name, quantity, price, expiration, and owners_email
home	This is a landing page that the user sees
profile()	Ensures the user is logged in and gets all tickets and returning the user template
login	This will redirect a user to their profile page once logged in
login_get()	Display the login route, if the user a user is logged in will redirect to home
login_post()	Gets the user input from the form and validates it (see validation), will log in the user and redirect to home if successful
logout	This will redirect the user to the home page when they've logged out
logout()	Cancels the user's session
register	This will handle the user's interaction with the /register page of the program using the methods detailed bellow
register_get()	This will display the register page, or redirect to the home page if a user is logged in
register_post()	This will get all the user's input from the form and ensure that all specifications were met for name, email, and password then register the user if all is correct
buy	This will ensure that a user can buy a valid ticket
buy_get()	This method is intended to return the ticket(s) that the user bought and redirects the user to the homepage
buy_post()	This method allows the user to input the ticket name and quantity for the ticket(s) they want to purchase
sell	This will post a valid ticket(s) a user wants to sell
sell_get()	This method is intended to return the ticket to be sold and redirects the user to the homepage

sell_post()	This method allows users to fill out a form with the ticket name, quantity, price and expiration date for the ticket(s) they wish to sell
update	This will handle updates to the ticket attributes
update_get()	This method is intended to return the updated ticket information and redirects the user to the homepage
update_post()	This method allows users to update ticket name, quantity, price and expiration date for tickets they have already posted
authenticate	Defines a wrapping layer that authenticates a user's session
authenticate(inner_function)	This method ensures that any logged in routes are only accessible to valid users
wrapped_inner()	This method verifies if a user is logged in and stores it in a session
tickets	This is used to initialize a ticket
get_all_tickets()	This method returns all the valid (not expired) tickets in the database
add_ticket(ticket_name, quantity, price, expiration, owners_email)	This method populates the list of valid tickets in the database with the ticket name, quantity, price, expiration date and owner's email
buy_ticket(ticket_name, quantity)	This method handles a ticket that is bought
update_ticket(ticket_name, quantity, price, expiration)	This method updates a valid ticket in the database
users	This file defines all the backend operations with the user object from the database
get_user(email)	This returns a user from the database
login_user(email, password)	This returns a user and then checks that the password matches the users stored password
register_user(email, name, password, password 2, balance)	This adds a user's registry information into the database
validation	This is used to validate the users email and password
validate_email_address(email)	This checks if the given address meets specifications using a third-party library and returns a Boolean value
validate_password(password)	This checks if the given password meets the requirements detailed for the project and returns a Boolean value

Test Plan:

The test case levels will be organized into backend, frontend, and integration. Backend will include tests for the database entries. This will include saving and accessing users and tickets information from the local database. Frontend will be organized by the different routes. Each route will be tested to ensure that they act as they should when proper input is given and when improper input is given. The navigation between pages will also be tested. Integration will be used to make sure the frontend and backend interact as they should. This will ensure that the information the user sees, and inputs is reflected properly in the database. The order these will be tested in will be frontend, backend, then integration. We will be using pytest to help with the automated testing so the code can be properly tested on a regular basis without too much manual actions from the program team. These tests files will be stored in gitHub,

but the test data stored in the database will be locally stored meaning the database will have to be remade each time the tests are run.

The testing will be broken down into the R1-R8 partitions from A1. Melissa will be responsible for testing R1 and R8. Teaghan will be responsible for R2 and R7. Nicole will be responsible for R5 and R6. And Aubrey will be responsible for R3 and R4. These assignments match the assignments from A1 for the test case descriptions. If an error is found with user registration Teaghan should be notified. In the case of an error pertaining to the user login or 404 errors Aubrey should be notified. If their error is an error in the homepage then Nicole and Melissa should be notified. As no other implementations have been altered by a specific team member all other errors should be reported to the group and assignment for who is responsible for the fix will be made from there.

To manage our action minute costs we will use input partition testing to test legal user inputs. Using the requirement partitions from A1 the program can be tested by simulating each partition using the simplest form of the user input that is included in the partition. This will allow us to be confident that the program is acting as it should in all cases but will be less costly than exhaustive testing and more structured than shotgun testing.