Team 28 Group Project

Release 0.0.1

Team 28

Mar 24, 2023

CONTENTS:

CHAPTER

ONE

FLASKR PACKAGE

1.1 Submodules

1.2 flaskr.app module

flaskr.app.**about**()

Render the about page.

flaskr.app.**add_item**(*menu_item*)

Gets passed a menu item as an object and adds it to the database.

flaskr.app.**add_menu_item**()

Render the page to add a menu item. Collects item info from an HTML form, checks the data is valid, and adds it to the database.

flaskr.app.**ans_call**()

flaskr.app.**assign_table**(*table_num*, *waiter_id*)

Assign a waiter to a table.

flaskr.app.**calculate_total_price**(*order_details*)

Calculates the total price for a given order.

flaskr.app.**call**()

Assigns a waiter to a customer table, should they need help.

flaskr.app.**call_waiter**()

flaskr.app.**calling_confirm**()

flaskr.app.**create_account**(*user_account*)

Creates a user account with a provided user. Logs that user in after account creation.

flaskr.app.**create_app**()

Creates and configures the flask app.

flaskr.app.**create_login**()

Loads the page to create a login. Fetches entered details, creates the account, and logs them in.

flaskr.app.**cust_menu**()

Displays the menu.

flaskr.app.**customer_orders**()

Shows a customer the details of their orders, along with their current statuses.

flaskr.app.**delete_item**(*item*)

Delete a menu item from the database.

flaskr.app.**edit_menu_item**()

Displays all menu items, and allows you to select and delete them.

flaskr.app.**fetch_login**()

Loads the login page. Fetches the user details and logs them in,

flaskr.app.**filter_menu**()

Filters the menu by allergens, price, or calories, or orders by price or calories.

flaskr.app.**format_customer_orders_as_dictionary**(*orders*)

Format a list of orders into a dictionary with the information that customers would need to see.

flaskr.app.**format_kitchen_orders_in_dictionary**(*rows*, *sql_connection*)

Receives the orders to be made, and an sql cursor to fetch the name to add to a dictionary.

flaskr.app.**format_waiter_orders_in_dictionary**(*rows*)

Receives the orders taken, then formats them in a dictionary with the information that the waiters need. Returns the dictionary.

flaskr.app.**get_all_users**()

Returns all the users registered with the system.

flaskr.app.**get_menu**()

Fetches every row of the menu from the database.

flaskr.app.**get_orders_to_make**()

Fetch the orders that need to be made in the order they were placed, then return as a dictionary.

flaskr.app.**get_users_with_name**(*first_name*, *last_name*)

Returns all the users registered with the system matching a specific first and last name.

flaskr.app.**get_waiter_orders**(*state*)

Gets all the orders of a given state.

flaskr.app.**home**()

Render the home page.

flaskr.app.**index**()

Navigate to the home page.

flaskr.app.**kitchen_orders**()

Displays the orders that kitchen staff need to work on.

flaskr.app.**login**(*user_account*)

Logs in a provided user.

flaskr.app.**logout**()

Logs out the current logged-in user.

flaskr.app.**manage_accounts**()

Managers can change the admin level of accounts here to hire or fire kitchen staff, waiters, and managers. This displays all accounts and their current privilege level.

flaskr.app.**manage_accounts_edit**()

Allows managers to edit the admin level of an account.

flaskr.app.**menu**()

Filters the menu page. Get menu items from the database.

flaskr.app.**order**()

Takes the details of a customer's order.

flaskr.app.**order_confirmation**()

Tells the customer their order has been confirmed.

flaskr.app.**order_payment**()

Summarises the customer's order, provides a total cost, and requests payment details.

flaskr.app.**setup_order**(*table_num*)

Insert the table number into the database, then set an order session using the ID of the order.

flaskr.app.**summarise_order**()

Fetches a customer's order, and returns the contents with the total price to pay.

flaskr.app.**take_order**()

From the HTML order form, insert every item with a quantity > 0 into the database for the order in the current session.

flaskr.app.**update_order_status**()

flaskr.app.**update_state_to**(*state*, *order_id*)

Receives an order ID, and the state to update that order to.

flaskr.app.**update_user**(*user_id*, *role*)

Update the role of a user with a given user_ID to a specified role.

flaskr.app.**waiter_order_cancel**()

Cancels the selected order.

flaskr.app.**waiter_order_confirm**()

Shows the waiters the orders taken, and lets them confirm them.

flaskr.app.**waiter_order_delivered**()

Shows the orders to be delivered, and allows them to be marked as such.

1.3 flaskr.init_db module

class flaskr.init_db.**DBManager**(*app*)

Bases: object

Handles database connections and cursors.

close()

Close the cursor and the connection.

get_connection()

Gets a cursor to communicate to the database with.

get_db()

Gets the database connection.

1.4 flaskr.menu_item_model module

class flaskr.menu_item_model.**MenuItemModel**(*name*, *price*, *category*, *calories*, *allergen*)

Bases: object

Validates, then represents a new menu item.

static format_allergens(*allergen*)

Formats the allergens as comma-separated-values in a string.

static validate_calories(*calories*)

Throws an exception if the calories are left blank, or are not a valid integer.

static validate_name(*name*)

Throws an exception if the item name has been left blank.

static validate_price(*price*)

Throws an exception if the price is blank, or not a valid integer or decimal to 2 decimal places.

1.5 flaskr.user_account_model module

class flaskr.user_account_model.**UserAccountModel**(*first_name*, *last_name*, *password*, *role*)

Bases: object

Represents a user account for creating an account, or logging in.

static hash_password(*password*)

Hash a given password using Python's hashlib module. The hash is returned as a string of characters.

static validate_name(*first_name*, *last_name*)

Throws an exception if the first or last name fields are left blank.

static validate_password(*password*)

Throws an exception if the password field is left blank.

static validate_role(*role*)

Throws an exception if the role is an invalid number. Role can be blank so that the model can be used to log in as well as create an account.

1.6 Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

f

INDEX

O

S

T

U

V

W