README for Hermosa The Hotel: Bookkeeping and Product Management Website

This is a ReadMe file for the Bookkeeping and Product Management Website that I developed for a fictitious client—Hermosa The Hotel. It includes several features (success criteria for the project):

- Admin authentication to ensure encapsulation and that the admin-functions are not accessible without authentication.
- A Product Catalogue page that stores essential information about all products and filters that could help in product analysis for the client or for ease in order-taking.
- A particular product could be viewed from the Product Catalogue page to be read out by the staff to a customer for order-taking purposes.
- Products could be added/ removed from the Product Catalogue and existing products could be edited.
- New customer registration and existing customer management.
- Order-management by being able to create new bills, reviewing stored bills, etc.

Technologies used:

- Flask
- Python3
- Jinja2
- HTML
- CSS
- MySQL
- JSON

Libraries and functions used:

- From flask:
  - .request()
  - .render_template()
  - .session()
  - .url_for()
  - .redirect()
- From json:
  - .load()
  - .dump()
- From mysql.connector:
  - Error
  - IntegrityError
  - .connect()
  - .cursor()
  - .execute()
  - .fetchall()
  - .fetchone()
  - .commit()
  - .close()

For convenience and ease of management and expansion of this project:

- The embedded-SQL functions concerning products, customers, and billing are stored in different files and called using the main file—***app.py***.
- Endpoints (logically named and not after functions) are used for each Flask route.
- ***Port 9000*** is used for testing of the product.
- A ***layout.html*** file is used as a template for other HTML pages using Jinja2.
- A ***static/css/style.css*** is used to style all common elements in all HTML pages.

The following are features and how they were developed using the technologies and methodologies:

- **Admin Authentication**:
  - *The default route ('/') of the web-application is set to the admin-authentication page*. Additionally, a *session variable is used* to ensure authenticated access to the homepage, which provides access to other admin functions.

```
@app.route('/', methods = ['GET', 'POST'], endpoint = 'Logout')
def admin_authenticate():
    if session:
        if session["admin"]=="True":
            return( redirect (url_for("Home")))
    else:         ⋯
```

  - *Accessing any other routes without having bypassed the admin authentication would lead to redirecting to the page*.

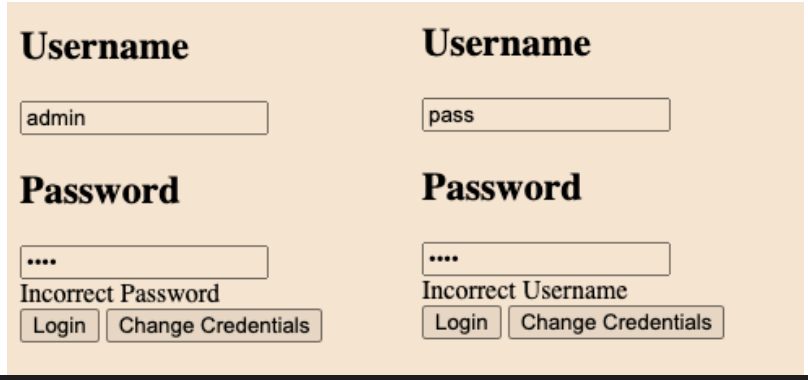
```
@app.route('/home', methods = ['GET' , 'POST'], endpoint = "Home")
def admin_homepage():
    session["phno"] = ""
    if session["admin"] == "True": ⋯
    else:
        return(redirect(url_for('Logout')))
```

  - Making changes to the admin credentials is possible only through success in the admin authentication.
  - A *locally stored JSON file contains the key-value pairs for the username and password* which *helps in keeping track of any changes in either of the values*. This is an *unconventional and thus more secure approach since the credentials are locally stored*.

```
{} admin_login_info.json > abc password
  1    {"username": "admin", "password": "pass"}


# using a dynamic json file to store changes to admin credentials
admin_info_path = 'admin_login_info.json'
def provide():
    with open(admin_info_path, 'r') as file:
        return json.load(file)
def put(log_info):
    with open(admin_info_path, 'w') as file:
        return json.dump(log_info, file)
```

  - The admin is informed in the admin authentication page of the issue with logging in (wrong username/password), if any, using a dynamic Jinja2 variable.




```python
@app.route('/', methods = ['GET', 'POST'], endpoint = 'Logout')
def admin_authenticate():
    if session: …
    else:
        if request.method=='POST':
            admin_log_info = provide()
            action = request.form.get("action")
            if action=='Login':
                u = request.form.get("usrname")
                p = request.form.get("passw")
                if u == admin_log_info["username"]:
                    if p == admin_log_info["password"]:
                        session["admin"] = "True"
                        return (redirect(url_for('Home')))
                    else:
                        return(render_template('admin-auth.html', log_message="Incorrect Password"))
                else:
                        return(render_template('admin-auth.html', log_message="Incorrect Username"))
```

templates > admin-auth.html > ...

```
{%extends 'layout.html'%}
{%block content%}

<form action="" method="POST">
    <h2>Username</h2>
    <input name="usrname">
    <h2>Password</h2>
    <input type="password" name="passw">
    <br>
    {{log_message}}
    <br>
    <input type="submit" name="action" value="Login">
    <input type="submit" name="action" value="Change Credentials">
</form>
```

- **Product Management**:
    - The client can choose to *add products* to, *make changes* (description, price, renaming, changing the image, etc.) to, and remove products from, the Product Catalogue. This is carried out using H*TML forms* (defined *using a POST request* that is *received by the backend using the 'request' library offered by Flask*) and *committing changes to the Database through embedded MySQL* (using the *mysql-connector library for Python3*). There is appro*priate 'error handling' for each database commit*

Catalogue

Product Page

Name : Piña Colada

Price : 270

Description : A refreshing tropical blend of creamy coconut, tangy pineapple juice, and smooth rum, served chilled and garnished with a slice of pineapple.

Type : Beverage

Ingredient : Vegetarian

EDIT DELETE

templates > admin-add.html > ...

```
<form action="" method="POST">
    <label for="name">Product Name:</label><br>
    <input type="text" id="name" name="name" placeholder="Butter Chicken" required><br><br>
    <label for="price">Price(₹):</label><br>
    <input type="number" id="price" name="price" step="10" placeholder="600" required><br><br>
    <label for="isVeg">Ingredients Information:</label><br>
    <select id="veg" name="veg" required>
        <option value="Choose" selected disabled>Choose</option>
        <option value="Vegetarian">Vegetarian</option>
        <option value="Non-vegetarian">Non-vegetarian</option></select><br><br>
    <label for="type">Type:</label><br>
    <select id="type" name="type" required>
        <option value="Choose" selected disabled>Choose</option>
        <option value="Food">Food</option>
        <option value="Beverage">Beverage</option>
        <option value="Dessert">Dessert</option></select><br><br>
    <label for="description">Description:</label><br>
    <textarea id="description" name="description" rows="10" cols="50" required></textarea><br><br>
    <label for="image_link">Image Link:</label><br>
        <input type="url" id="image_link" name="image_link" style="width: 40%;" required><br><br>
    <input type="submit" value="Submit" >
</form>
```

```
@app.route('/add', methods = ['GET', 'POST'], endpoint= 'Add Products')
def add_product_page():
    if session["admin"] == "True":
        if request.method=='POST':
            if request.form.get('action')=='Home': …
            else:
                P_details = request.form.to_dict()
                o_p = add_product(P_details["name"], P_details["price"], P_details["description"],
                                  P_details["veg"], P_details["type"], P_details["image_link"])
                return(redirect(url_for("Home", messg=o_p)))
        else: …
    else: …
```

admin_functions.py > ...

```
def add_product(p_name, price, desc, veg, type, imgL):
    db = mysql.connector.connect(host = "localhost",
        user = "root", password = "cincy@2024",database = "banerjee_kitchen"
    )
    cursor = db.cursor()
    sql = f'''INSERT INTO product_info
    (`p_name`, `price`, `description`, `isVeg`, `type`, `imageLink`)
    VALUES (%s, %s, %s, %s, %s, %s)'''
    try:
        cursor.execute(sql, (p_name, price, desc, veg, type, imgL))
        result = cursor.fetchall()
        print(result)
        db.commit()
        cursor.close()
        db.close()
        return("Product Added Successfully!")
    except Error as err:
        cursor.close()
        db.close()
        return(f"Error in Adding Product: {err}")

```

- The Product Catalogue page uses an *HTML table*, that comprises of values *obtained using embedded-MySQL* and *parsed into the table using Jinja2 templating*.

```
@app.route('/view', methods = ['GET', 'POST'], endpoint = 'Manage Products')
def view_page(messg="Welcome"):
    if session["admin"] == "True":
        if request.method =='POST':⋯

        else:
            if request.args.get("messg") is None:
                return render_template(f'admin-view.html', P_table=show_products(),
                                       msg = "Greetings!")
            else:
                return render_template(f'admin-view.html', P_table=show_products(),
                                       msg = request.args.get("messg"))
    else:
        return(redirect(url_for('Logout')))
```

```
{% for item in P_table %}
<tr>
  <td>
    <form method="POST">
      <input type="submit" name="send" value="{{item[0]}}">
    </form>
  </td>
  <td>{{ item[1] }}</td>
  <td>{{ item[2] }}</td>
  <td>{{ item[3] }}</td>
  <td>{{ item[4] }}</td>
  <td>{{ item[5] }}</td>
  <td>{{ item[6] }}</td>
  <td>
    <form method="POST" action="">
    <input type="hidden" name='product_id' value='{{item[0]}}'>
    <input type="submit" name="send" value="edit">
    </form>
  </td>
```

- The Product Catalogue page also consists *filters* that can be applied to produce a special list of products that are either *sorted using a condition* and/ or *satisfy a constraint*. *Multiple filters can be chosen at once*. This filter was developed using *HTML forms using a POST request* that sends an appropriate value to the *Flask server to help construct a dynamic MySQL query*. In a similar manner, the filtered list of products is displayed using another *embedded-MySQL query and Jinja2 templating*.

Home

Product Catalogue

Product ID	Product Name	Price	Rating	Frequency	Ingredient Information	Type	Edit	Delete
None	None	Lower-Higher	Higher-Lower	None	Veg	Food	APPLY CLEAR	
53	Vegetable Spring Rolls	600	4.3	25	Vegetarian	Food	edit	delete
113	Pani Puri	800	4.5	35	Vegetarian	Food	edit	delete
126	Vegetable Bonda	800	4.4	23	Vegetarian	Food	edit	delete
64	Pumpkin Soup	800	4.4	24	Vegetarian	Food	edit	delete
120	Vegetable Pakora	800	4.3	25	Vegetarian	Food	edit	delete

```
<form action="" method="POST">
<td><select name="frequency">
      <option value="" selected>None</option>
      <option value="DESC">Higher-Lower</option>
      <option value="ASC">Lower-Higher</option></select></td>

<td><select name="isVeg">
      <option value="" selected>None</option>
      <option value="Vegetarian">Veg</option>
      <option value="Non-vegetarian">Non-Veg</option></select></td>

<td><select name="type">
      <option value="" selected>None</option>
      <option value="Food">Food</option>
      <option value="Beverage">Beverage</option>
      <option value="Dessert">Dessert</option></select></td>

<td><input type="submit" name="action" value="APPLY">
  <input type="submit" name="action" value="CLEAR"></td></form>
```

app.py > view_page

```
    def view_page(messg="Welcome"):
        if request.method =='POST':
            P_id = request.form.get("product_id")
            send = request.form.get("send")
            action = request.form.get("action")
            if send == 'delete':
                return(redirect(url_for('Delete', pid=P_id)))
            elif send == 'edit':
                return(redirect(url_for('Edit', pid=P_id)))
            elif send=='Home':
                return(redirect(url_for('Home')))
            elif action == 'APPLY':
                res = show_products(p_name=request.form.get("pname"),
                                    price=request.form.get("price"),
                                    rating=request.form.get("rating"),
                                    frequency=request.form.get("frequency"),
                                    isVeg=request.form.get("isVeg"),
                                    type=request.form.get("type"))
                return(render_template('admin-view.html', P_table=res))
            elif action=='CLEAR':
                res = show_products()
                return render_template(f'admin-view.html', P_table=res)
            # View Page
            else:
              return (redirect (url_for ('product_details', pid = send)))
```

- The dynamic SQL query is constructed using several logical elements including *Python lists* that count and store the “ORDER BY” and “WHERE” conditions. This *minimises the potential errors in the SQL query completely*.

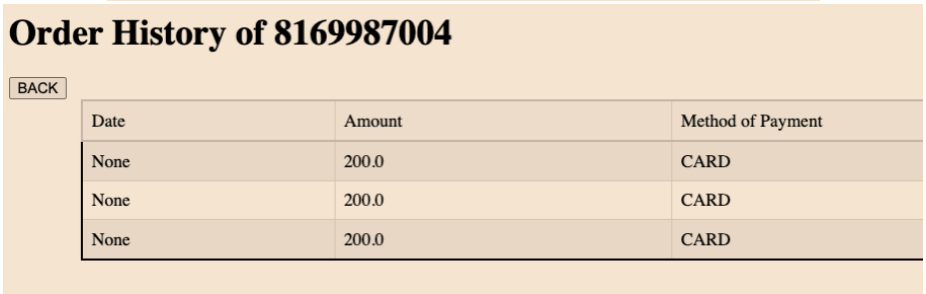
admin_functions.py > show_products

```
def show_products(pid='', p_name='', price='', rating='', frequency='', isVeg='', type=''):
    db = mysql.connector.connect(
        host = "localhost",
        user = "root",
        password = "cincy@2024",
        database = "banerjee_kitchen"
    )
    q1 = "SELECT * FROM product_info"
    q2 = ""
    q3 = ""
    sequences = []
    if pid:
        sequences.append(f"product_id {pid}")
    if p_name: ...
    if price: ...
    if rating: ...
    if frequency: ...
    if len(sequences)>0:
        for seq in sequences:
            q2 = q2+' '+seq+','
    else:
        q2="  "
    q2 = q2[:-1]
    if q2[-1]=='C':
        q2 = "ORDER BY"+q2
```

```
    conds = []
    if isVeg:
        conds.append(f"isVeg='{isVeg}' and ")
    if type:
        conds.append(f"type='{type}' and ")
    for c in conds:
        q3 += c
    if len(q3)>0:
        q3 = q3[:-4]
        q3 = "WHERE "+q3
    else:
        q3 = ""
    sql = q1+' '+q3+' '+q2+';'
    cursor = db.cursor()
    cursor.execute(sql)
    result = cursor.fetchall()
    cursor.close()
    db.close()
    return(result)
```

- **Customer Management:**
    - New customers can be registered using their phone number used while generating their bill. This is done using the fore-mentioned *HTML forms and embedded-MySQL* technique of data collection and storing. It is informed whether the phone number input has previously been registered or no through *MySQL error management*.

    - A particular user's details and purchase history can be viewed by clicking on the user's phone number in the bill history. This could be achieved using a *session variable that stores the concerned customer's phone number*.

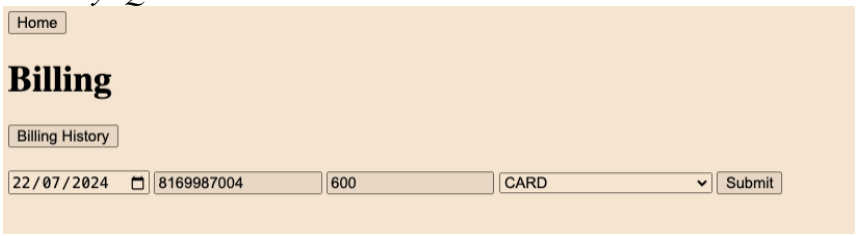


```python
@app.route('/customer/history', methods=['GET', 'POST'], endpoint = 'History')
def view_customer():
    if request.method == 'POST':
        if request.form.get("page") == 'BACK':
            return(redirect(url_for('Customer')))
    else:
        return(render_template('customer-history.html', phno=session["phno"],
                               P_table=userHistory(session["phno"])))
```

```python
def getInfo(phno):
    db = mysql.connector.connect(
        host = "localhost",
        user = "root",
        password = "cincy@2024",
        database = "banerjee_kitchen"
    )
    cursor = db.cursor()
    sql = (f"SELECT * FROM customer_info WHERE ph_no = {int(phno)};")
    cursor.execute(sql)
    result = cursor.fetchone()
    cursor.close()
    db.close()
    return(result)
```

- The details of existing customers too, can be changed using an *HTML forms* and *POST request* managed by the backend. In this case too, it is informed if any errors have taken place while *committing the database transaction*.

- **Bill Management:**
  - Bills can be added using an *HTML form* and be *securely stored on the MySQL database*.



  - Bill history can be viewed by the admin and filters can be applied for convenience and analysis. This uses the same technique as the *Product Catalogue* Page.