This Interaction Overview Diagram shows how all the components are interconnected and how they interact with each other to work seamlessly in harmony and provide a GOOD User Experience.

There are 6 major layers where the **“SYSTEM”** is the main layer representing the parent container with one sub-container named **“WebApp”** which has other 4 sub-layers or sub-containers namely **“DATABASE”, “PYTHON”, “STATIC FILES” AND “TEMPLATES”.** Apart from this, the **“SYSTEM”** container also has its **“STORAGE”** named **“DEFAULT DOWNLOAD LOCATION”** and there is a **“STORAGE”** named **“/static/res”** in **“STATIC FILES”.**

Here is a brief description of all the components involved in the working of the web app:

1. **SYSTEM:**
   * It refers to the Machine on which the **WebApp** is running. The Machine or the **SYSTEM** is on the same network as of the server on which the **WebApp** is hosted.
2. **WebApp:**
   * It refers to the instance of the WebApp running on the system which has various components working in the backend. These components are hosted on a server away from the **SYSTEM** but on the same network as the **SYSTEM**.
3. **DATABASE:**
   * It refers to the MySQL database where all the necessary tables related to the functionality of the **WebApp** are stored.
4. **PYTHON:**
   * It refers to all the Python files coded to perform various functionalities namely API development, Analysis and Backend logic using Flask.
5. **STATIC FILES:**
   * These are components of a Flask App. These refer to all the js files and css files that support the HTML files in the **TEMPLATES** folder.
6. **TEMPLATES:**
   * These are components of a FlaskApp. These refer to all the frontend files like HTML.
7. **STORAGE:**
   * It refers to a storage space such as hard disks.

Now let's see the internal components of the **WebApp** in detail.

1. **DATABASE**
   * **emp\_exit\_db**(Employee Exit Data)
   * **users\_db**(Users’ Login Data)
   * **feedback\_db**(Feedback of Users)
2. **PYTHON**
   * **Connection.py(file):**

It consists of the Connection Logic

* + **Methods.py(file):**

It consists of Query functions used for communication with the database.

* + **Anlaysis.py(file):**

It consists of functions that are used to perform preprocessing, Analysis, creating graphs and data tables.

* + **App.py(file):**

It consists of the complete backend logic which is programmed in Flask.

* + **Selected Values of Filters(variable):**

It is a variable that points to **App.py**. It’s value is fetched from HTML files via a JS file that is related to the respective HTML.

* + **Login Credentials(variable):**

It is a variable that points to **App.py**. It’s value is fetched from l**ogin.html.**

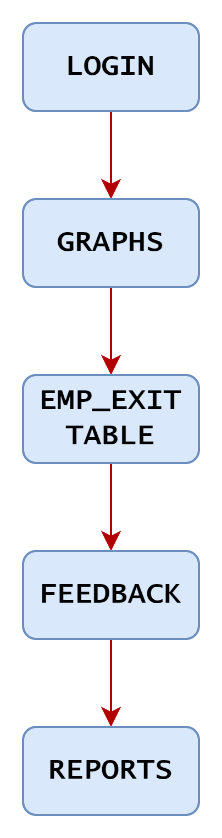
* + **e\_id, usr, feedbk(variable):**

It is a variable that points to **App.py**. It’s value is fetched from **feedback.html**.

1. **STATIC FILES**
   * **style.css**(Provides design to all the HTML files)
   * **emp\_exit.js**(JS file for **emp\_exit.html**)
   * **reports.js**(JS file for **reports.html**)
   * **graphs.js**(JS file for **graphs.html**)
2. **TEMPLATES**
   * **feedback.html**(Displays feedbacks from database)
   * **emp\_exit.html**(Displays Data Table from database)
   * **reports.html**(Displays reports generated via **Anlaysis.py(file)**)
   * **graphs.html**(Displays graphs generated via **Anlaysis.py(file)**)
   * **login.html**(Used to take login credentials inputs)

**----------------------------------------------------------------------------------------------------------------------------**

Let’s assume the navigation of the user through the **WebApp** as follows:



**Steps:**

1. The process starts with the implementation of “Connection.py” which helps to establish the connection with the database.
2. In the front end, the user will 1st see the “login.html” where s/he has to enter the credentials.
3. Credentials are sent to “App.py” which will validate the credentials with the database using the query functions provided in “Methods.py”.
4. Invalid credentials will lead to an Alert Box on the “login.html”. Valid credentials will redirect the user to “graphs.html”.
5. When user is redirected to “graphs.html” in backend, “App.py” will execute the following:
   1. A list of possible filter options based on ‘department’ will be created in “graphs.html” using the “filters” functions in “Analysis.py”.
   2. The default value of filter will be “All”.
   3. “graphs.js” will use ajax to send the current value of the filter to “App.py” in the designated route.
   4. “App.py” will generate graphs based on the complete data available and store them in location “/static/res” in png form.
   5. After this the png images will open in “App.py” using file handling for jsonification.
   6. This json dump is now again sent back to “graphs.js” which will use ajax to interpret these json dumps and place the graphs in their designated <div>.
   7. If user selects a different filter value, steps (c) to (f) will be repeated.
6. Now user goes to “emp\_exit.html” and “App.py” will perform the following functions in backend:
   1. A list of possible filter options based on multiple fields will be created in “emp\_exit.html” using the “filters” functions in “Analysis.py”.
   2. The default value of all the filters will be “All”.
   3. “emp\_exit.js” will use ajax to send the current value of all the filters to “App.py” in the designated route.
   4. “App.py” will now use the dynamic function of retrieving data form “emp\_exit\_db” coded in the file “Methods.py”.
   5. After data retrieval, the dataset is converted into json dumps.
   6. These json dumps are now again sent back to “emp\_exit.js” which will use ajax to interpret these json dumps and place the data in a tabular format.
   7. A ‘Feedback’ button is also added in the rightmost cell of each record which will be used to redirect to “feedback.html”.
   8. If user selects a different value in filters, steps (c) to (g) will be repeated.
7. Now using the “Feedback” button, user will be redirected to “feedback.html” and “App.py” will execute the following:
   1. First of all, the user is redirected to the feedback page of the employee record whose respective “Feedback” button is clicked.
   2. In “App.py”, the function receives the “emp\_id” of the respective “Feedback” button clicked by the user.
   3. Now, the feedback query logic coded in “Methods.py” is used to fetch the feedback of the particular “emp\_id” using a foreign key.
   4. This feedback data is sent directly to “feedback.html”.
   5. User can also add feedback here using the textbox.
   6. The following data is sent to “App.py” : ‘e\_id’, ‘usr’, ‘feedbk’.
   7. Query logic to add feedback coded in “Methods.py” is used with the above inputs to add the record to the database.
   8. Using the “Back” button, user will be redirected to “emp\_exit.html”.
8. At last user will go to “reports.html” and following actions are performed by “App.py” in backend:
   1. A list of possible filter options based on ‘department’ will be created in “reports.html” using the “filters” functions in “Analysis.py”.
   2. The default value of filter will be “All”.
   3. “reports.js” will use ajax to send the current value of the filter to “App.py” in the designated route.
   4. “App.py” will generate graphs & data tables based on the complete data available.
   5. The graphs are stored in location “/static/res” in png form and the data tables are converted into json form.
   6. After this the png images will open in “App.py” using file handling for jsonification.
   7. This json dumps of graphs and data tables are now again sent back to “reports.js” which will use ajax to interpret these json dumps and place them in their designated <div>.
   8. If user selects a different filter value, steps (c) to (g) will be repeated.
   9. User can also download the report using “Download” button.
   10. JSPDF is used to set a custom styling and arrangement of pages in pdf.
   11. The PDF will be stored in the default download location of the system on which the “WebApp” is running.