

CS-236 Group Project Report, Phase III

Group number : 15

Fall 2025

This phase deals with creating a lightweight WEBUI or App to demonstrate the postgresql database. We used **Streamlit** for this and named the **application Stage3**.

Stage3 App:

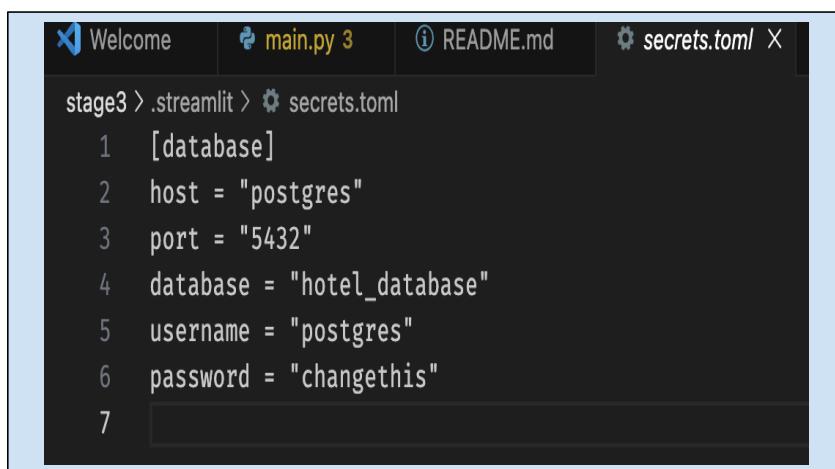
[Streamlit](#) is an open source Python framework that allows Python programmers to convert their Python data scripts into web apps quickly, without having to learn front-end development.

Stage3 is a Streamlit app that connects to the PostgreSQL database and presents a dropdown of tables in the public schema. When a table is selected, the Streamlit app connects to the database, retrieves all the data and presents it via a grid that is searchable, sortable and filterable. This app is contained in the code – “**main.py**” of the attached folder.

Steps taken to Setup Streamlit and create the Stage3 app:

All steps and the codes are given in the attached “[README.md](#)” in the **folder stage3**.

In short we loaded the suitable packages, created the Streamlit configuration directory and set up secrets to connect to the PostgreSQL database. Then we opened the **secrets.toml** file and populated the credentials.



```
stage3 > .streamlit > secrets.toml
1 [database]
2 host = "postgres"
3 port = "5432"
4 database = "hotel_database"
5 username = "postgres"
6 password = "changethis"
7
```

Functions inside the Stage3 App:

Function 1:

```
def load_data_from_table(conn, table_name: str) → pd.DataFrame:  
    """  
    Loads and caches data from a database table.  
  
    Args:  
        conn (SQLAlchemy Connection): SQLAlchemy connection to database  
        table_name (str): Name of the table to retrieve data from  
  
    Returns:  
        pd.DataFrame: Pandas Dataframe that contains all data in the table  
    """
```

Function 2:

```
def split_frame(input_df: pd.DataFrame, num_rows: int) → list[pd.DataFrame]:  
    """  
    Split a Pandas Dataframe into a list of Dataframes each containing a maximum number of rows  
  
    Args:  
        input_df (pd.DataFrame): Original Dataframe to be split  
        num_rows (int): Maximum number of rows in each Dataframe to split input_df into  
  
    Returns:  
        list[pd.DataFrame]: List of dataframes each containing maximum num_rows  
    """
```

Function 3 : Creates the pages in the WEBUI – for easy viewing

```
def paginate_dataframe(filtered_df: pd.DataFrame) → list[pd.DataFrame] :  
    """  
    Add UI on dataframe to create pagination for viewers  
  
    Args:  
        filtered_df (pd.DataFrame): Original dataframe  
  
    Returns:  
        list[pd.DataFrame]: List of paginated dataframes  
    """
```

Function 4 : To filter the databases based on column_names, range etc

```
def filter_dataframe(df: pd.DataFrame) → pd.DataFrame:  
    """  
        Add UI to let viewers filter columns of a Dataframe  
  
        Args:  
            df (pd.DataFrame): Original dataframe  
  
        Returns:  
            pd.DataFrame: Filtered dataframe  
    """
```

For Function 4, we created a streamlit container “filterable” and did the filtering inside that.

Once the connection is created we first loaded the database then used Function 4 to filter it and finally used Function 3 to display it in the local port.

Run the Streamlit application:

In the workspace terminal of the docker we did the following:

```
$ cd stage3  
$ source .venv/bin/activate  
$ streamlit run main.py  
  
Collecting usage statistics. To deactivate, set browser.gatherUsageStats to false.  
  
You can now view your Streamlit app in your browser.  
  
Local URL: http://localhost:8501  
Network URL: http://172.19.0.2:8501  
External URL: http://104.48.80.228:8501
```

Open [<http://localhost:8501>] in a browser to access the Streamlit app.

Streamlit output:

Following are some snippets of the streamlit outputs:

Snippet 1: Showing the combined database table 'cr_hb'

A screenshot of a Streamlit application running on localhost:8501. The title bar says "cr_hb". The interface includes a dropdown for "Filter on Column(s)" with "Choose options" selected, and a "Sort Data:" section with "No" selected. Below is a table with columns: booking_status, market_segment_type, stays_in_weekend_nights, lead_time, avg_price_per_room, arrival_month. The data shows various combinations of these variables.

| booking_status | market_segment_type | stays_in_weekend_nights | lead_time | avg_price_per_room | arrival_month |
|----------------|---------------------|-------------------------|-----------|--------------------|---------------|
| 1 | Offline | 1 | 224 | 65 | |
| 1 | Online | 2 | 5 | 106.68 | |
| 0 | Online | 2 | 1 | 60 | |
| 0 | Online | 0 | 211 | 100 | |
| 0 | Online | 1 | 48 | 94.5 | |
| 0 | Online | 0 | 346 | 115 | |
| 1 | Online | 1 | 34 | 107.55 | |
| 1 | Online | 1 | 83 | 105.61 | |
| 1 | Offline | 0 | 121 | 96.9 | |
| 1 | Online | 0 | 44 | 133.44 | |
| ... | | | | | |

Snippet 2: 'customer_reservation' and how filtering working on its columns

A screenshot of a Streamlit application running on localhost:8501. The title bar says "customer_reservation". The interface includes a dropdown for "Filter on Column(s)" with "Booking_ID" selected. Below is a table with columns: stays_in_week_nights, lead_time, arrival_year, arrival_month, arrival_date, market_segment_type, avg_price_per_room. The data shows various combinations of these variables, filtered by Booking_ID.

| Booking_ID | stays_in_week_nights | lead_time | arrival_year | arrival_month | arrival_date | market_segment_type | avg_price_per_room |
|------------|----------------------|-----------|--------------|---------------|--------------|---------------------|--------------------|
| INN00066 | 0 | 3 | 30 | 2018 | 10 | | |
| INN00067 | 0 | 2 | 179 | 2018 | 6 | | |
| INN00068 | 0 | 2 | 26 | 2018 | 4 | | |
| INN00069 | 0 | 1 | 55 | 2018 | 4 | | |
| INN00070 | 1 | 2 | 74 | 2018 | 4 | | |
| INN00071 | 0 | 3 | 143 | 2018 | 8 | | |
| INN00072 | 1 | 0 | 34 | 2018 | 11 | | |
| INN00073 | 1 | 1 | 30 | 2018 | 8 | | |

Additional results:

By connecting streamlit with pgsql, we are able to view the datasets and tables in a direct and convenient way. By our code, we can do filtering and sorting efficiently. One great thing about streamlit is its live data update from the server, which means any changes in the database will at once update to the frontend.

The figure shown below is a demo of the toy database. We imported the dataframes processed in Phase 1 and Phase 2 into PostgreSQL for further processing. For easily using search method, in this version all the data type we make it text so it supports global search (for years, agents...). Also, it is also quite convenient to store what we found is useful, just by clicking the download button, the page will be download to local. If you find the rows in one page is too small, you can also config in the side bar.

Toy Database Demo (PostgreSQL + Streamlit)

We load the dataframes processed in Phase 1 and Phase 2 into PostgreSQL for further processing and interactive exploration.

Data

Columns to display:

booking_id x stays_in_weekend_nights x stays_in_week_nights x lead_time x arrival_year x arrival_month x arrival_date_day_of_month x market_segment_type x avg_price_per_room x booking_status x arrival_date_week_number x hotel x country x email x

Filters

Filter on column(s):

Choose options

Global search (applied to text columns):

2017

Sorting

Sort data?

No Yes

Sort by

booking_id

Direction

Ascending Descending

Total rows after filtering: 6526

Page

Total pages: 262

1 - +

Download current page as CSV

| | booking_id | stays_in_weekend_nights | stays_in_week_nights | lead_time | arrival_year | arrival_month | arrival_date_day_of_month | market_segment_type | avg_price |
|-----|------------|-------------------------|----------------------|-----------|--------------|---------------|---------------------------|---------------------|-----------|
| 46 | INN00047 | 0 | 2 | 32 | 2017 | 11 | 20 | Offline TA/TO | 73.0 |
| 53 | INN00054 | 0 | 4 | 51 | 2017 | 11 | 11 | Offline TA/TO | 60.0 |
| 63 | INN00064 | 0 | 1 | 2 | 2017 | 9 | 10 | Complementary | 0.0 |
| 74 | INN00075 | 2 | 3 | 34 | 2017 | 10 | 25 | Offline TA/TO | 75.0 |
| 103 | INN00104 | 0 | 1 | 16 | 2017 | 10 | 31 | Online TA | 95.0 |
| 104 | INN00105 | 0 | 3 | 24 | 2017 | 10 | 8 | Online TA | 107.0 |
| 106 | INN00107 | 1 | 4 | 10 | 2017 | 11 | 2 | Online TA | 82.95 |
| 107 | INN00108 | 1 | 3 | 0 | 2017 | 9 | 21 | Online TA | 155.0 |
| 109 | INN00110 | 2 | 1 | 32 | 2017 | 9 | 5 | Online TA | 94.5 |
| 119 | INN00120 | 2 | 3 | 2 | 2017 | 11 | 14 | Online TA | 103.0 |

Screen shot of the database config:

Database Config

Select table: mytable

Page size: 25

| | booki |
|----|-------|
| 14 | INNOC |
| 16 | INNOC |
| 17 | INNOC |
| 18 | INNOC |
| 19 | INNOC |
| 20 | INNOC |
| 26 | INNOC |
| 28 | INNOC |
| 29 | INNOC |
| 30 | INNOC |

Download c

Project Contribution :

Phase 1:

Bufan: did the initial coding in .py file, did and wrote the “Installation of Pyspark : Bufan” in the report, generated new datasets in .xlsx format.

Sayantika: made the .ipynb notebook, wrote the report, generated new datasets in .csv format.

Phase 2:

Bufan: Made the video, did the .py file and checked Sayantika’s codes.

Sayantika: did the initial coding(Spark Analysis) and docker setup, made the .ipynb notebook, wrote the report.

Phase 3:

Bufan: Made the video/presentation, did the app.py file and checked Sayantika’s codes. Wrote additional results in the report

Sayantika: did the initial coding, made stage3 app, wrote the report, submission