Deliverable 2 Report

CSI 2132

Paul Barasa (300055931)

Josh Bassett (8179179)

Ahmed Khalil Merchaoui (300040178)

Our original intention was to have a fully functional webpage that mimics the likes of AirBnB. Due to circumstances outside of our control, the webpage is partly functional. Our database is populated with thousands of mock houses to resemble an AirBnB database. Users can also create accounts and sign in. Users can search for properties to book, but unfortunately the functionality to view the results was not implemented due to various complications. Users can input their desired house details on the Host a Home page. Input validation was set up for each page.

All the HTML pages we intended to integrate with our backend were built and can be viewed in chrome. They are styled and partly functional. Pages such as Booked Home and List Home were not implemented due to backend complications.

Many queries for this project were created and are functional, however, the raw data retrieved from Kaggle (other sources were checked as well and were equivalent) was simply scraped from the AirBnB website rather than being complete data. The queries created for the project that won't be demonstrated in the app are as follows:

[TEN QUERIES FROM PROJECT DESCRIPTION]

```
query1 = """CREATE VIEW rental_guests AS
        SELECT u.name 'Guest Name', p.property_type 'Rental Type', b.amount_paid 'Rental
Price',
                b.booking_date 'Signing Date', br.name 'Branch', pt.name 'Payment Type',
t.payment_status 'Payment Status'
        FROM AirBnB_Group_40.bookings_table b
        INNER JOIN AirBnB_Group_40.properties_table p ON p.id = b.property_id
        INNER JOIN AirBnB_Group_40.users_table u ON u.id = b.user_id
        LEFT JOIN AirBnB_Group_40.branches_table br ON br.address_id = p.address_id
        LEFT JOIN AirBnB_Group_40.transactions_table t ON t.booking_id = b.id
        LEFT JOIN AirBnB_Group_40.payment_type_table pt ON pt.id = t.payment_type_id
        ORDER BY pt.name ASC
        ORDER BY b.booking_date DESC;"""

query2 = """CREATE VIEW GuestListView AS
        SELECT u.*, br.id
        FROM AirBnB_Group_40.users_table u
        INNER JOIN AirBnB_Group_40.bookings_table b ON u.id = b.user_id
        INNER JOIN AirBnB_Group_40.properties_table p ON p.id = b.property_id
        LEFT JOIN AirBnB_Group_40.branches_table br ON br.address_id = p.address_id
        ORDER BY br.id ASC
        ORDER BY u.id ASC;"""
```

```
query3 = """SELECT MIN(price_per_day)
        FROM AirBnB_Group_40.bookings_table;"""

query4 = """CREATE VIEW Properties AS
        SELECT p.*, br.id
        FROM AirBnB_Group_40.users_table u
        INNER JOIN AirBnB_Group_40.bookings_table b ON u.id = b.user_id
        INNER JOIN AirBnB_Group_40.properties_table p ON p.id = b.property_id
        LEFT JOIN AirBnB_Group_40.branches_table br ON br.address_id = p.address_id
        ORDER BY br.id ASC
        ORDER BY p.rating ASC;"""

query5 = """CREATE VIEW UnrentedProperties AS
        SELECT p.*
        FROM AirBnB_Group_40.properties_table p
        LEFT OUTER JOIN AirBnB_Group_40.bookings_table ON p.id = b.property_id;"""

query6 = """CREATE VIEW RentalsOn10thDay AS
        SELECT p.*
        FROM AirBnB_Group_40.properties_table p
        INNER JOIN AirBnB_Group_40.bookings_table b ON p.id = b.property_id
        WHERE EXTRACT(DAY FROM b.check_in_date)<=10 AND EXTRACT(DAY FROM
b.check_out_date)>=10;"""

query7 = """CREATE VIEW EmployeesMakingOver15k AS
        SELECT e.id, br.manager_id, e.name, br.id, br.name, e.salary
        FROM AirBnB_Group_40.employees_table e
        LEFT JOIN AirBnB_Group_40.branches_table br ON e.branch_id = br.id
        WHERE e.salary>15000;"""

query8 = """SELECT pt.name 'property type', p.host_id 'host', ad.street 'street', ad.city 'city',
ad.province 'province', b.amount_paid 'price', pt.name 'payment type'
        FROM AirBnB_Group_40.bookings_table b
        LEFT JOIN AirBnB_Group_40.properties_table p ON p.id = b.property_id
        LEFT JOIN AirBnB_Group_40.properties_type_table pt ON p.property_type_id = pt.id
        LEFT JOIN AirBnB_Group_40.addresses_table ad ON p.address_id = ad.id
        LEFT JOIN AirBnB_Group_40.transactions_table tr ON tr.booking_id = b.id
```

```
	LEFT JOIN AirBnB_Group_40.payment_type_table pt ON tr.payment_type_id =
pt.id;"""

query9 = """UPDATE AirBnB_Group_40.users_table
	SET phone_number = 6134354040
	WHERE id=1234"""

query10 = """CREATE FUNCTION AirBnB_Group_40.ConcatNames (@firstName
VARCHAR(250), @lastName VARCHAR(250))
	RETURNS VARCHAR(250)
	AS BEGIN
	   DECLARE @fullName = CONCAT(@firstName, '', @lastname)
	   RETURN @fullName
	END"""
```

There is also the query from host.php which is not fully functional, therefore the query is shown below:

```
“””INSERT INTO AirBnB_Group_40.propreties_table (id, name,
description, host_id, bedroom_count, bathroom_count, rate, start_date,
end_date, created_at, modified_at, status) VALUES ('$randomId', '$title',
'$description', '$host_id', '$bedrooms', '$bathrooms', '$rate',
                                                               '$startDate',
                                                              '$endDate',
                                                              '$timestamp',
                                                              '$timestamp',
                                                              active)
```

Code Necessary To Implement Interfaces

All interface code has been uploaded to the GitHub repository found under the following hyperlink: https://github.com/P4ul1029/Webpage

The DBMS used is pgAdmin 4 as this is that which was used throughout the course. The programming languages used for the application and interfacing are as follows:

1. Python (with psycopg2) for DDL and populating schemas with entries
2. PHP for backend functionality, taking in requests from the front-end application and communicating them as queries to the database

3. HTML, CSS, and Javascript for front-end functionality and to provide an interface for users of the application

The data used to populate the database was found using Kaggle to search for scraped AirBnB data. Due to the lack of access to what would obviously be proprietary data, many attributes within tables of the database could not feasibly be populated. This includes, most notably, branch information, complete user information, in-depth reviews data, etc. One will notice of course many NULL entries for certain attributes. Consider instead the functionality of the programs in these cases rather than the results which were hindered simply due to lack of resources.

DDL INSTALLATION

To create schemas, tables, and populate the database we used Python3 with a package, psycopg2, to interface with the postgresql database. All the files pertaining to the creation and population of the database can be found within the directory: “.\Webpage\Database”.

The CSV files containing the raw AirBnB data cannot be included in the github repository due to size and therefore will strictly be included in the zipped folder. These will be found in the directory: “.\Webpage\Database\KaggleData”.

The installation of the python program for database creation and population is described in the README.md file found in the main directory. However, it is also outlined below (Note that one must have Python3 and pip installed prior to following these instructions):

In command prompt or Windows Powershell

1. Navigate to project directory

```
cd .\Webpage\Databases
```

2. Configure PYTHONPATH. Set the PYTHONPATH to point to this project's directory.

```
set PYTHONPATH="$PWD"
```

~~~
  ```
~~~

3. Create virtual environment

~~~
 ```
  python -m venv venv
  ```
~~~

4. Activate virtual environment

~~~
 ```
  venv/bin/activate
  ```
~~~

5. Install requirements to the virtual environment

~~~
 ```
  pip install -r requirements.txt
  ```
~~~

6. Create `database.ini` file inside the `db` directory with the following format

~~~
 ```
  [localhost]
  #REMOTE eecs connection
  database = <Your Database>
  user = <Your Username>
  password = <Your Password>
  host = web0.eecs.uottawa.ca
  port = 15432

  #LOCAL
  #database=postgres
  #host=localhost
  #port=5432

  [data_source_files]
  #Berlin Dataset
  Berlin_Calendar_Data=KaggleData\Berlin\calendar_summary.csv
~~~

```
Berlin_Listings_Data=KaggleData\Berlin\listings.csv
Berlin_Listings_Summary_Data=KaggleData\Berlin\listings_summary.csv
Berlin_Neighbourhoods_Data=KaggleData\Berlin\neighbourhoods.csv
Berlin_Reviews_Data=KaggleData\Berlin\reviews.csv
Berlin_Reviews_Summary_Data=KaggleData\Berlin\reviews_summary.csv

#Boston Dataset
Boston_Calendar_Data=KaggleData\Boston\calendar.csv
Boston_Listings_Data=KaggleData\Boston\listings.csv
Boston_Reviews_Data=KaggleData\Boston\reviews.csv

#Madrid Dataset
Madrid_Calendar_Data=KaggleData\Madrid\calendar.csv
Madrid_Listings_Data=KaggleData\Madrid\listings.csv
Madrid_Listings_Summary_Data=KaggleData\Madrid\listings_detailed.csv
Madrid_Neighbourhoods_Data=KaggleData\Madrid\neighbourhoods.csv
Madrid_Reviews_Data=KaggleData\Madrid\reviews.csv
Madrid_Reviews_Summary_Data=KaggleData\Madrid\reviews_detailed.csv

#Melbourne Dataset
Melbourne_Calendar_Data=KaggleData\Melbourne\calendar_dec18.csv
Melbourne_Listings_Data=KaggleData\Melbourne\listings_dec18.csv
Melbourne_Listings_Summary_Data=KaggleData\Melbourne\listings_summary_dec18.csv
Melbourne_Neighbourhoods_Data=KaggleData\Melbourne\neighbourhoods.csv
Melbourne_Reviews_Data=KaggleData\Melbourne\reviews_dec18.csv
Melbourne_Reviews_Summary_Data=KaggleData\Melbourne\reviews_summary_dec18.csv

#Seattle Dataset
Seattle_Calendar_Data=KaggleData\Seattle\calendar.csv
Seattle_Listings_Data=KaggleData\Seattle\listings.csv
Seattle_Reviews_Data=KaggleData\Seattle\reviews.csv
```

7. Run Application

```
python main.py
```

This concludes the instruction set for the running of the python scripts.

The application can be run on a local server which connects to the remote university's server and PostgreSQL database.

To run the application you will need to setup a local server on your machine (we used XAMPP). Then you will need to clone the repository or unzip the submitted file under this directory relatively to where xampp is installed (htdocs/temp/Webpage/).
Then you will need to change the username and password in database_config.php file.
And you should be good to go.

Webpages

All the HTML, CSS, and Javascript files that implement our webpages are separated into the following folders:

- Sign_Up_Page
    - The page where a user can make an account
- Sign_In_Page
    - The page where a user can sign in
- Search_Page
    - The page where a user can search for a house to book
- Results_Page
    - The page that displays a user's search results
- Host_a_Home
    - The page where a user can list their own property
- Listed_Home
    - The page where a user can view the details of a property they listed
- Booking_Page
    - The page where a user can booked a home they clicked on from the results page
- Booked_House
    - The page where a user can view the details of a property they booked

To access these webpages, simply open the Sign_Up_Page and as you interact with the site, you will naturally progress through the remaining pages.

Branch Employee Application

Instructions are based on using Eclipse IDE.

1. Download the following JDBC driver
   https://jdbc.postgresql.org/download.html

2. Open Branch_Employee_Application folder in Eclipse
3. From the Project Properties, click on Java Build Path > Libraries > Add External JARs
4. Select the downloaded JAR file
5. Now run the file in the Eclipse console or command line as a branch employee would