Method for creating database in SQL for Seal Images

Creating a Database in PostgreSQL is a complicated process that requires a lot of steps in setting up, all the queries are premade but need to be implemented holistic manner in which order is essential. The folder that is available in Github should include this file as well as a folder called: **Project\_DB2**. This folder contains all the queries, functions, and files required to build this database with some contained data.

An important note something I just figured out, you are able to download table right away from postgresql, so I have created a new folder with some files that are difinitly compatible with the table presets.

(This guide assumes that PostgreSQL has already been installed and that pgAdmin is accessible)

1. You want to create a database with the corresponding name. The default server information:
   1. **Hostname**: localhost
   2. **Port**: 5432
   3. **Password**: ‘password’
   4. **User**: posture
2. Next is the step is creating the required tables:
   1. **data\_tags\_table.sql** is the first table that needs to be created so that all the tags can be properly imported later on. The names of the column MUST correspond exactly to that of the CSV file.
      1. Importing **data\_tags.csv** which should be located in files, will be imported into PostgreSQL, it is important that ‘header’ is enabled. (I choose to call the file and the table by the same name for clarity).
      2. You will also need to add two columns after the CSV file has been imported, the first one is **logic\_tag** (serial primary key) and the other is the **picture\_number** column.
3. The next step is to add the randomly generated numbers to the **picture\_number** column which is done by executing the query called **picture\_number\_generator.sql**. This query uses the function called **generate\_unique\_random\_number.sql**.
4. There needs to be a table in which user data is stored (UUID, usernames, password), this will be done in the **user\_data** table, to which the query is **user\_data\_UUID.sql**.
   1. First add the extension to the query **uuid-ossa\_extension.sql**
   2. After you run this code you might see in your function dropdown that a lot of UUID functions got added, I believe they are extensions from the UUID-ossa package which allows them to become a global entity.
5. Creating the user functions that will be n used for creating a user in the database
   1. There is an **admin\_create().sql**
      1. Admin is a SUPERUSER that can change any feature within the database
      2. Later in the UI there might be an option to create additional admins if the database becomes scalable.
   2. And **standard\_create().sql**, this function is not a superuser so the user needs to be granted permission to execute certain queries, and only has access to the table **data\_tags**
      1. Insert, update, and delete rows
      2. Usage of all sequences in the schema
      3. Grant executes all functions in the public schema
   3. Then using the **insert\_user\_data.sql** function to take input of a user that has just created a username and ‘password’ as an argument, the random correspondant UUID will generated in this function.
6. Next is building the **user\_documentation.sql** table, this will record the changes being made to the table **data\_tags**. First, you create the table from the file then run the function **perform\_user\_action\_with\_documentation().**sql which takes the user that is editing it and the details also the **picture\_number** specifying the row that was edited. **user\_documentation** has these columns:
   * 1. **Documentation\_id** which is the serial primary key
     2. **User\_id** is **UUID** from the **user\_data** table
     3. **Action\_type** is a VARCHAR
     4. **Affected\_row\_id** is INTEGER
     5. **Timestamp** gives data and time
7. First create a table with **user\_data\_UUID.sal** rname and password of the current user. Since there was a choice between making 7 functions or making one function that will handle the documentation by itself: **perform\_user\_action\_with\_documentation().**
   1. What this function does is takes user information to shows what a user has done by time stampin and dcumenting the sort of action, this system records all the changes made to the system